KubeSec: Automatic Detection of Takeover Risks Introduced by Third-Party Apps in the Kubernetes Ecosystem

Tao Zheng, Qiyu Hou, Hao Ren, *Member, IEEE*, Xingshu Chen, Gelei Deng, Tianwei Zhang, *Member, IEEE*, Guowen Xu, *Senior Member, IEEE*, and Hongwei Li, *Fellow, IEEE*

***Abstract*—Third-party applications (TPAs) are integral components of managed Kubernetes clusters, but are also frequently exploited in takeover attacks. Recent incidents have demonstrated that TPAs can be weaponized to gain control over clusters. Given their critical role within the Kubernetes ecosystem, it is essential to explore the potential attack surfaces associated with various types of TPAs. To address this, we propose KubeSec, a framework that systematically investigates these risks by analyzing application permission configurations and component code dependencies. This investigation revealed a significant number of insecure RBAC binding patterns, uncovering 562 such patterns and identifying 375 vulnerabilities linked to 134 CVEs. These vulnerabilities impact millions of users, with an average remediation time exceeding 10 months. All findings have been reported to the relevant teams, leading to the assignment of 21 new CVEs by the community. These results highlight substantial security risks associated with TPAs in Kubernetes clusters and emphasize the urgent need for further research to develop more secure cluster management practices.**

***Index Terms*—Third-party application, Kubernetes, takeover risks, Kubernetes ecosystem.**

I. Introduction

KUBERNETES is widely applied for orchestrating container clusters and managing the deployment and scheduling of containers. The Cloud Native Computing Foundation (CNCF) reports that 90% of container cloud service users and providers are adopting or evaluating it for automated deployment, scaling, and management [1]. Its flexibility and scalability have led Kubernetes to dominate nearly 80% of the market [2]. However, this widespread adoption has also raised concerns about security risks, particularly those introduced by third-party applications (TPAs). TPAs are external plugins and applications that extend Kubernetes' functionality but often introduce significant security vulnerabilities that can lead to cluster takeover risks, scenarios where attackers gain complete control over the cluster, compromising its confidentiality, integrity, and availability. A prominent example is the Tesla cluster compromise, where attackers exploited TPA vulnerabilities to leak millions of users' private data [3]. This incident underscores how takeover risks, enabled by insecure TPAs, can have severe real-world consequences.

The security risks posed by TPAs manifest in two primary ways: privilege escalation through insecure Role-Based Access Control (RBAC) configurations and third-party component (TPC) vulnerabilities. RBAC, a core security mechanism in Kubernetes, governs access to cluster resources. When TPAs are granted excessive permissions (e.g., unrestricted access to cluster secrets or administrative roles), attackers can hijack these privileges to escalate their access and take over the cluster. For example, a TPA with overly permissive *ClusterRole* bindings can allow an attacker to manipulate critical resources like *ClusterRoleBindings* or *Secrets*, leading to complete cluster compromise. Meanwhile, TPC vulnerabilities in TPAs, such as outdated libraries or known exploits in dependencies, provide alternative attack vectors. For example, a vulnerable TPC like *protobuf* (with a history of critical CVEs) can be exploited to execute malicious code within a TPA's *DaemonSet*, enabling attackers to pivot to cluster-wide control. These risks are exacerbated by the fact that cloud providers often reuse the same TPAs across clusters, accelerating vulnerability propagation.

Existing studies [4], [5], [6], [7], [8], [9], [10] have explored Kubernetes security but suffer from critical limitations. First, they lack a comprehensive analysis of privilege-binding vulnerabilities in TPAs, which offer attackers straightforward paths to cluster takeover (e.g., via misconfigured *ClusterRoleBindings*). Second, they focus on container or cluster configurations, overlooking TPC vulnerabilities in TPAs, which can be equally

Received 13 November 2025; revised 20 February 2026; accepted 1 March 2026. Date of publication 13 March 2026; date of current version 12 May 2026. This work was supported in part by the National Natural Science Foundation of China NSFC under Grant 62402331, and Grant 62502075, in part by the Fundamental Research Funds for the Central Universities under Grant YJ202429, in part by the General Program of the Sichuan Provincial Natural Science Foundation under Grant 2026NSFSC0431, and in part by the Fundamental Research Funds for the Central Universities under Grant SCU2024D012. *(Tao Zheng and Qiyu Hou contributed equally to this work.) (Corresponding author: Hao Ren.)*

Tao Zheng and Qiyu Hou are with the School of Cyber Science and Engineering, Sichuan University, Chengdu 610065, China.

Hao Ren and Xingshu Chen are with the School of Cyber Science and Engineering, Sichuan University, Chengdu 610065, China, and also with Cyber Science Research Institute (Sichuan University) and Key Laboratory of Data Protection and Intelligent Management (Sichuan University), Ministry of Education, Chengdu 610065, China (e-mail: hao.ren@scu.edu.cn).

Gelei Deng and Tianwei Zhang are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore 639815.

Guowen Xu and Hongwei Li are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China.

Digital Object Identifier 10.1109/TDSC.2026.3673764

destructive. For example, Yang [4] highlighted privilege escalation risks but did not address how TPC flaws in TPAs could link with RBAC misconfigurations. Third, scalability remains a challenge. Most tools rely on manual configurations for TPA analysis, hindering large-scale assessments. These gaps leave the Kubernetes ecosystem without a holistic understanding of takeover risks or practical detection methodologies.

To address these limitations, we propose KubeSec, the first automated framework to systematically detect the takeover risks introduced by TPA and TPC. KubeSec integrates two novel modules: (1) **RBAC Privilege Detection**. This module automates the analysis of privilege binding configurations in TPAs, identifying dangerous patterns (e.g., overly permissive $*$ verbs on sensitive resources) and mapping their relationships to service accounts and pods. By dynamically parsing both the YAML definitions and the live cluster states, it overcomes the static analysis limitations of previous tools [11], [12], [13]. Existing work only parses rendered `YAML` and does not handle `Helm` templates, aggregated *ClusterRoles*, or cross-namespace bindings, resulting in the inability to explore deeper takeover risks. (2) **TPC Vulnerability Detection**. Leveraging Software Composition Analysis (SCA) tools like *CycloneDx* and *Dependency-Track*, this module detects vulnerable third-party components in TPAs and assesses their exploitability in cluster takeover scenarios (e.g., a *protobuf* RCE flaw enabling *DaemonSet* hijacking).

The main contributions of this paper are summarized as follows:

- A systematic analysis of Kubernetes takeover risks, unifying privilege escalation and TPC exploit paths into a cohesive threat model.
- The design and implementation of KubeSec, which achieves 86.3% precision in privilege-binding detection, outperforming state-of-the-art tools like EPA [4].
- Large-scale empirical findings: We identified 562 insecure RBAC patterns and 375 TPC vulnerabilities (linked to 134 CVEs) across 227 TPAs, with an average remediation time exceeding 10 months. These results, validated by 21 newly assigned CVEs, highlight the urgent need for proactive TPA security measures. We also open-source our method at https://anonymous.4open.science/r/KubeSec-6EF3.

Organization. Section II reviews related work. Section III introduces the preliminaries, and Section IV gives the motivation and challenges. Section V introduces the design and implementation of KubeSec. Section VI discusses performance evaluation. Finally, Section VII concludes our work.

II. Related Work

A. Container Security

As the most critical basic component of the cluster, container security is directly related to the stability of the Kubernetes cluster state [14], [15], [16], [17], [18]. Researchers have conducted a series of large-scale empirical research works on container image security status from different security perspectives [19], [20], [21], [22], [23]. Tak [24] studied 10,000 images in DockerHub, and they found that a large number of containers had default settings that violated the security regulations. Zerouali

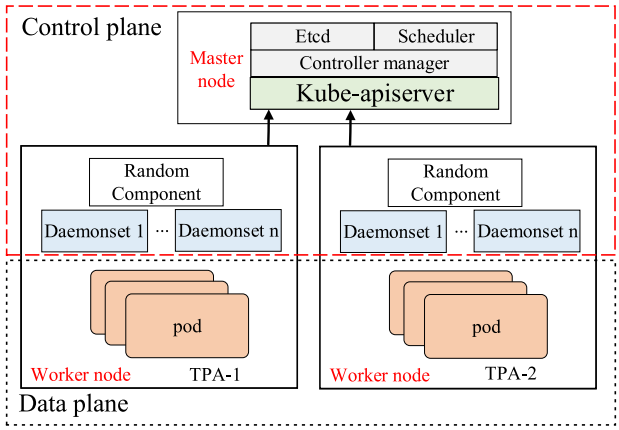


Fig. 1. The overview of Kubernetes architecture.

conducted vulnerability and third-party package analysis on 7380 mirrors [25], [26]. Wist [27] found that the more serious mirror vulnerabilities were mostly introduced by Python and JavaScript scripts. Liu [28] collected and analyzed malicious behaviors in more than 2 million mirrors from DockerHub. In addition, researchers have implemented a series of automated image security analysis tools [15], [29], [30], [31] and dynamic detection methods for container environments [32], [33], [34], [35].

B. Kubernetes Ecosystem Security

Cluster security is beginning to receive attention from researchers [36], [37]. Senjab [38] and Turin [39] studied cluster resource management and scheduling. Zhong studied the current status and future perspectives of machine learning applications to clusters [40], Shamim studied the recent practice of knowledge-based security for Kubernetes clusters [41], Sayagh [42] presented a method for generating recommended configuration policies for clusters, and Haque utilized knowledge graphs to automate the generation of cluster security configuration policies [9], [43]. In addition, there are researchers who utilize graph mining and deep learning methods to study the detection methods of cluster insecure policies [8], [44], [45], [46]. In terms of dynamic risk detection, Yang [4] proposed for the first time a cluster takeover method based on excessive privileges, and He proposed a cluster attack method using ebpf, which is capable of realizing cross-cloud attacks on clusters [5].

However, these security analyses are based on specific attack scenarios, and our work can confidently provide TPAs with comprehensive vulnerability detection that can inspire more in-depth analysis of takeover risks in the Kubernetes ecosystem.

III. Preliminaries

A. Third-Party Apps of Kubernetes

As shown in Fig. 1, the architecture of the Kubernetes system involves two key dimensions: the control plane and the data plane. Specifically, within the control plane, the Master

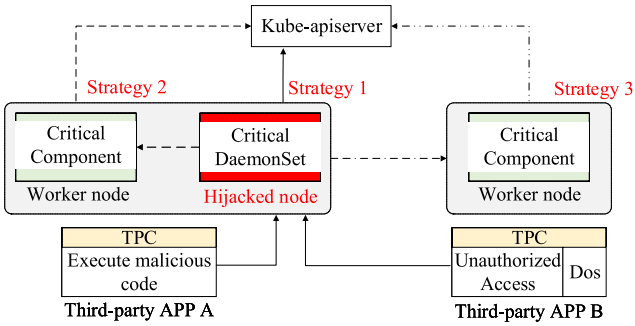


Fig. 2. Three Kubernetes takeover strategies.

node is responsible for managing the overall state and configuration of the cluster. The Master node comprises the following components: **kube-apiserver**. Serving as the front end for the Kubernetes API, it handles API requests and executes changes to the cluster's state. **kube-scheduler**: Responsible for scheduling newly created Pods onto nodes within the cluster. **kube-controller-manager**. Executes controllers, monitors the cluster's state, and adjusts it to ensure the desired state matches the actual state. **etcd**. A distributed key-value store used to store all configuration data and status information of the cluster. In the data plane, Worker nodes are responsible for running containerized applications. This primarily includes: **kubelet**. Communicates with the Master node, manages Pods on the node, and ensures that Pods run according to specifications. **kube-proxy**. Maintains network rules on the node, enabling service discovery and load balancing. **Container Runtime**. Responsible for running containers, such as Docker or containerd.

The TPAs in Kubernetes primarily consist of Dynamic Workloads and DaemonSets. Dynamic Workloads refer to application-level components implemented via standard controllers (e.g., Deployments or StatefulSets), where Pods are scheduled non-deterministically across worker nodes by the Kubernetes scheduler. In contrast, a DaemonSet is a controller ensuring a Pod replica runs on every node or a selected subset. DaemonSets are commonly used to deploy node-level services and often operate with elevated access to node resources. With node-wide deployment and elevated privileges, compromised DaemonSets can provide attackers with persistent access across nodes and enable cluster-wide takeover.

B. Takeover Risk of Kubernetes

As shown in Fig. 2, there are privilege-level risk and code-level risk of TPAs in Kubernetes. An attacker can directly take over the Kubernetes cluster through the risk of excessive privileges of TPAs, or he can use the TPC vulnerability in the source code to construct an attacking chain to control *DaemonSet* or other privileges to achieve the takeover of the Kubernetes cluster [4].

1) Permission-Level Risk: The two TPAs running in the cluster have a critical *DaemonSet* and critical components, respectively. The three takeover strategies are briefly described as follows:

- *Strategy 1 (Direct takeover):* The key daemon in TPA-A has excessive privilege to bind the cluster, and once the attacker's controlled worker nodes contain such DaemonSet pods, the attacker is able to abuse the service account of TPA-A to directly take over the control privileges of the cluster.
- *Strategy 2 (Indirect takeover):* The key component of TPA-A has the authority to manage the cluster, the attacker hijacks this component through the key daemon of TPA-A and forces the key component of TPA-A to run in the controlled working node, so as to obtain the cluster control authority.
- *Strategy 3 (Combination takeover):* The key component of TPA-B has the privilege to manage the cluster, and the attacker hijacks this component through the key daemon of TPA-A, or combines more TPA excessive privileges together to accomplish the privilege escalation.

2) Code-Level Risk: When a vulnerable TPC is introduced during TPA development, an attacker can escalate privileges via various attacks. The main types are described below:

- *Execute malicious code:* An attacker can exploit a code vulnerability in a component (e.g., CVE-2024-35182) to inject malicious code via techniques such as code injection, cross-site scripting (XSS), or remote command execution, thereby gaining control of a *DaemonSet*.
- *Unauthorized Access:* Vulnerable components could lead to access control flaws that could be exploited by an attacker to bypass authentication and authorization mechanisms to gain unauthorized access (e.g., CVE-2024-3744).
- *Denial of Service:* Certain vulnerabilities could lead to denial-of-service (DoS) attacks that make applications unavailable (e.g., CVE-2024-32476). An attacker could cause a cluster takeover by exploiting these vulnerabilities to send malicious requests or exhaust resources.

When an attacker obtains the cluster secret information by abusing the excessive privileges of the key *DaemonSet*, the attacker uses the Token corresponding to the *Service Account* to authenticate with the *kube-apiserver*. At the same time, he directly obtains all the secrets of the cluster, including those of the cluster administrator who creates it, and escapes from the working node to takeover the whole cluster by using the tainting mechanism. Note that only CVEs with CVSS $\geq$ 5.0 and CWEs related to privilege escalation or remote code execution are retained to ensure that each TPC edge can form a takeover path.

It can be seen that no matter it is a privilege-level vulnerability or a code-level vulnerability, an attacker can take over the control privileges of Kubernetes by constructing a suitable attack chain. Therefore, systematically analyzing the privilege binding relationships and code components of TPAs is essential for securing clusters.

IV. Motivation and Challenges

A. Threat Model

We assume a scenario where an attacker has compromised a Kubernetes worker node and aims to gain control over the

```
kind: DaemonSet
metadata:
  name: hwameistor-local-disk-manager
  namespace: {{ .Release.Namespace}}
spec:
  template:
    spec:
      serviceAccountName: hwameistor-admin
---
kind: ClusterRoleBinding
metadata:
  name: hwameistor-admin-binding
roleRef:
  kind: ClusterRole
  name: hwameistor-role
subjects:
- kind: ServiceAccount
  name: hwameistor-admin
  namespace: {{ .Release.Namespace}}
---
kind: ClusterRole
metadata:
  name: hwameistor-role
rules:
  - apiGroups:
      - apiextensions.k8s.io
    resources:
      - '*'
    verbs:
      - '*'
```

Fig. 3. The permission-level takeover risk of *Hwameistor*. It has the -∗- verb, which means that it can perform any operation on the resource.

entire cluster. In a real-world context, if the attacker gains access to the applications running within the cluster, they could: (1) compromise the applications running inside the containers, (2) execute container escapes to compromise the underlying worker nodes, and (3) eventually take control of the entire cluster. Consistent with the approach outlined in the literature [4], our focus is on the third step, where we assume that the attack proceeds under the principle of least privilege. Specifically, the TPA is deployed in accordance with official documentation, and all other Kubernetes security measures are implemented following best practices.

Next, we will demonstrate through a motivation example that even with these stringent security measures in place, an attacker can still exploit the excessive privileges granted to the TPA to compromise the entire cluster, thereby posing a substantial threat to cluster security.

B. Motivation Example

As shown in Fig. 3, *Hwameistor* is a high-availability local storage system designed for cloud-native stateful workloads. It enhances service stability by deploying a *DaemonSet* named *hwameistor-local-disk-manager*, which ensures that a replica pod controller runs on each node. The *hwameistor-local-disk-manager* utilizes a service account called *hwameistor-admin*, which is bound to a *ClusterRole* named *hwameistor-role* through a *ClusterRoleBinding*. This binding grants the service account the permissions defined in *hwameistor-role*. Notably, the *hwameistor-role* has been configured with permissions to perform all operations on all resources, meaning the *hwameistor-local-disk-manager DaemonSet* has the capability to perform any action on any resource within the cluster, such as listing cluster secrets, creating pods, and modifying node configurations. However, these permissions are excessive for *Hwameistor*'s needs, which only require specific permissions, such as *get, list*, and *watch* on node resources, and *list, get, create, delete, update*, and *patch* on **persistent volume resources**.

This over-authorization significantly expands the potential attack surface of the Kubernetes cluster, making it more vulnerable to malicious exploitation. If a malicious actor were to obtain a token with these elevated permissions, they could leverage multiple pathways to escalate privileges at the cluster level, potentially leading to a full takeover of the cluster. For example, after hijacking the *Hwameistor* node, the attacker uses its critical *DaemonSet* (*hwameistor-local-disk-manager*) to obtain **Service Account** (*hwameistor-admin*) and assigns a cluster control role via *ClusterRoleBinding*. At the same time, it combines the token corresponding to **Service Account** to authenticate with *Kube-apiserver*, obtains all the secret information of the cluster, and thus takeover the whole cluster. In addition, the attacker's takeover process based on TPA code-level risk mirrors the procedure outlined above and will not be reiterated here.

Therefore, the rapid, automated, and systematic discovery of takeover risks associated with TPAs in Kubernetes clusters, particularly in environments with frequent configuration changes, is crucial for safeguarding the confidentiality, integrity, and overall security of Kubernetes clusters.

C. Challenges

Although Kubernetes is exposed to various takeover risks, there are significant challenges in systematically identifying these risks in production environments. Three major challenges are summarized below.

Challenge 1. TPAs dataset construction: To facilitate our research, the primary hurdle lies in establishing a vast and all-encompassing dataset of TPAs, encompassing diverse types of TPAs from multiple vendors. This endeavor is crucial in order to yield persuasive findings regarding the prevailing security concerns associated with TPAs. Regrettably, a publicly accessible dataset of TPAs for research purposes remains absent. Moreover, an additional challenge arises as an increasing number of vendors are now impeding public access to TPAs, resorting to anti-scraping techniques to counter hostile web crawlers. Consequently, this exacerbates the difficulty in collecting requisite data.

Challenge 2. RBAC privilege binding: The dynamic nature of RBAC privilege bindings in TPAs introduces considerable complexity into Kubernetes environments. These bindings are influenced not only by the initial security configurations of both the cluster and the TPAs but also by ongoing changes within the system. While existing tools [11], [12], [13] capture static privilege bindings from API requests, they overlook bindings derived directly from the source code. These source-specific

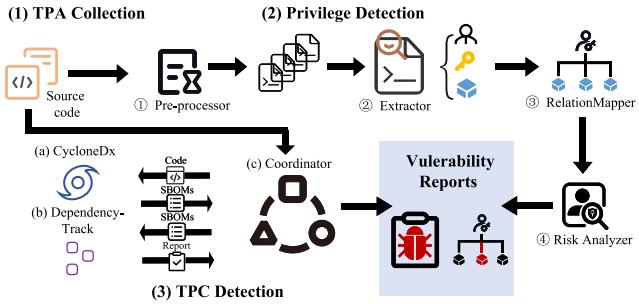


Fig. 4. The architecture overview of KubeSec, which mainly includes three modules: (1) TPA collection module, which automatically crawls the samples to be analyzed. (2) RBAC privilege detection module automatically analyzes the static and runtime RBAC permission binding relationships in TPA. (3) The TPC detection module utilizes two optimal tools to conduct third-party component security analysis for TPA.

bindings are selectively applied based on scenarios dictated by the *Kube-apiserver*, necessitating a more nuanced analysis. Furthermore, the absence of a standardized method to assess the risks associated with RBAC privileges complicates the evaluation process. Without such standards, it becomes challenging to quickly and automatically determine the security risks of TPAs once their privilege bindings are resolved, potentially leading to overlooked vulnerabilities amid configuration changes.

Challenge 3. TPC-VEX Association and Scalable Orchestration: Kubernetes third-party components are updated continuously and often lack lock-files; their SBOMs must be cross-referenced with vulnerability feeds without build-time context. The challenge is to orchestrate `CycloneDX-generated` bills of materials against `NVD, GitHub Advisories`, and `OSS Index` while preserving version-level traceability across 227 Helm charts and to do so in a reproducible, stateless pipeline that completes SBOM to CVE to VEX in minutes rather than hours. Transitive Go modules, omitted version ranges, and the absence of a unified VEX format make large-scale, Kubernetes-specific vulnerability association both essential and non-trivial.

V. Design and Implementation

To address the above challenges and comprehensively automate the identification of takeover risk of TPAs, we designed KubeSec. As shown in Fig. 4, KubeSec consists of three key components: **TPA collection module**, **privilege detection module**, and **TPC detection module**.

A. TPAs Collection

To address challenge 1, we first design a TPA collection module that is responsible for collecting TPAs to build a large-scale dataset. We develop a web crawler to collect TPA samples from official websites to address the scarcity of TPA samples. We collect 177 TPAs from CNCF, including 115 sandbox project samples, 37 incubation project samples, and 25 graduation project samples. In conjunction with previous work [4], we manually organize and collect 50 pre-installed TPAs from the official mirrors of *Google GKS, Microsoft AKS, Amazon EKS* and *Alibaba ACK*. In addition, we maintain the latest version of each sample in the TAPs database with a customized crawler. This initiative ensures that we have comprehensive coverage of mainstream TPA projects in the current Kubernetes ecosystem. In the end, we formulate a dataset that contains 227 TPAs as the analysis source, and we have included them in our open-source dataset.

B. RBAC Privilege Detection

To tackle Challenge 2, we propose a privilege detection module comprising four key components: Pre-processor, Extractor, Relation Mapper, and Risk Analyzer.

1) Pre-Processor: The configuration of TPA can be specified either through YAML files or using the *Helm Charts* framework. The primary challenge for this module is the automatic recognition and extraction of TPA configuration information. Our approach, KubeSec, begins by locating and parsing the *templates* directory, *chart.yaml*, and *values.yaml* files within the source code to capture metadata and configuration details. It then processes the default values from *values.yaml*, using a template engine to replace variable references (e.g., . *Values.clusterRole.name*) in the templates with actual values. This process results in fully-rendered Kubernetes resource configuration files, akin to those directly defined in YAML files, thereby simplifying the subsequent mapping of authorization relationships. For each high-risk permission tuple, KubeSec additionally records the runtime `can-i` result in field *runtimePermissions*; entries absent from this list are marked `unused` to allow post-deployment filtering of excess privileges.

2) Extractor: The *Extractor* employs `ConfigLocator` and `ConfigFetcher` components to retrieve resource description information from both open-source and closed-source TPAs, facilitating an analysis of privilege risks.

1) *For open-source TPAs:* Kubernetes supports merging multiple YAML configuration files into a single file and allows simultaneous application of these configurations, which poses a challenge for configuration file analysis. To address this issue, *ConfigLocator* customizes the data structure for each resource object to accurately store its descriptive information. *ConfigLocator* first identifies and defines the boundaries of each configuration file and performs an accurate split of the merged file. It locates the list of all resource configurations and parses the *YAML* files one by one, using file suffix matching. For key resource objects, such as *ClusterRoleBinding, ClusterRole, DaemonSet*, *ConfigLocator* extracts their resource metadata and organizes it into a structured representation.
2) *For closed-source TPAs:* Although *kubectl*, the official tool of Kubernetes, provides rich cluster query capabilities, its commands are often inefficient for authorization analysis involving multiple resource objects, and their default output formats are difficult to use directly for automated processing. To solve this problem, *ConfigFetcher* first scans the local directory of the master node to extract the necessary authentication information, such as the cluster administrator's certificate and token, from the *kubeconfig* configuration file. It then uses this information to

gain full access to cluster resources. It invokes Kubernetes' REST API to retrieve configuration information for various resources. For example, by sending a *get* request to */apis/rbac.authorization.k8s.io/v1/clusterroles*, *ConfigFetcher* can directly retrieve *ClusterRole* resources in their original configuration format.

Similarly, it can also obtain configuration information for other resources such as *ServiceAccount, ClusterRoleBinding*, and *DaemonSet*. Finally, the obtained resource configuration information is stored structurally to form different resource description information sets.

3) Relation Mapper: Although Kubernetes *ClusterRoleBinding* configuration explicitly defines the binding relationship between *ClusterRole* and *Service Account*, it does not directly reveal the specific permissions held by *ServiceAccount* or the actual users of *ServiceAccount*, which significantly limits the efficiency of permission analysis work.

To achieve the precise extraction of these relationships, *Relation Mapper* retrieves all original *ClusterRoleBinding* data from the structured storage of resource description information. It extracts the corresponding *ServiceAccount* and *ClusterRole* names through the Subject field and roleRef field within each binding. Subsequently, it uses these names to search and extract the complete description information of *Service Account* and *ClusterRole* from their respective resource description information collections, thereby establishing a complete binding relationship between *ClusterRoles* and *ServiceAccounts*.

Then, considering the dependency relationship between resource objects such as Pods, Deployments, DaemonSets, and StatefulSets on *ServiceAccounts*, we specifically focus on the *serviceAccountName* field in the configuration of these resource objects. By matching this field with the *Service Account* names we have identified, we can determine the *ServiceAccounts* used by these resource objects and simultaneously extract their namespace fields to identify the namespace where the resource objects are located. Finally, we integrate the complete information of users, *ServiceAccounts*, and *ClusterRoles* into a single JSON data record, achieving precise extraction of permission binding relationships. Concurrently, to enhance the contextual information of the data, we supplement the JSON record with additional metadata such as project name and namespace, so that subsequent analysis work can more comprehensively understand the full picture of permission relationships.

4) Risk Analyzer: The *Risk Analyzer* is used to identify dangerous authorizations that pose a risk of cluster takeover. In the Kubernetes environment, the granting of permissions typically involves the ability to perform specific operations on specific resources, e.g., *list secrets* denotes the permission to enumerate Secrets resource. Therefore, KubeSec builds a dataset that can fully express TPAs authorization relationships. In the process of risk identification, we recognize that both operations and resources are of non-negligible importance. In order to comprehensively and accurately identify potentially dangerous authorizations, *Risk Analyzer* first summarizes the resource categories that have a significant impact on cluster security, such as *Secrets*, *ClusterRoleBindings*. At the same time, we also sort out a series of risky operations acting on these resources, such as *get, list* and *patch*. Based on the characteristics of these resources and operations, *Risk Analyzer* obtains the permission risk profile of the TPAs based on the impact scope and sensitivity of the operations. We will analyze this in detail in Section VI-B.

C. TPC Vulnerability Detection

To address challenge 3, this module integrates two leading TPC analysis tools, *CycloneDx* [47] and *Dependency-Track* [48]. To optimize the efficiency of the automated analysis, we developed a *Coordinator* to enable seamless collaboration between these tools.

CycloneDX provides a powerful capability to accurately parse the TPAs' source code in order to identify software components and their dependencies, and to generate a detailed, machine-readable Software Bill of Materials (SBOM). The SBOM list not only covers detailed information about the components, but also clearly shows the hierarchical relationships between them, providing us with a comprehensive and accurate insight into the software components. *Dependency-Track* is equipped to handle SBOMs and Vulnerability Exploitability Exchanges (VEX) in *CycloneDx* format. It is capable of identifying multiple forms of risk including, but not limited to, components with known vulnerabilities, outdated components, tampered components, and license risks. By integrating multi-source vulnerability intelligence (e.g., National Vulnerability Database, GitHub Advisories, Sonatype OSS Index, etc.), *Dependency-Track* is able to detect possible vulnerabilities in the TPC on top of *CycloneDx*-generated SBOMs.

The *Coordinator* module aims to automate the entire analysis process and improve analysis efficiency by intelligently connecting all links. The entire automated process consists of the following steps: ① Extracting and generating the SBOM from TPAs using *CycloneDx*. ② Creating project instances in Dependency-Track for each TPA. ③ *Coordinator* module automatically uploads SBOM to the corresponding Dependency-Track project instance for vulnerability detection. ④ Exporting a VEX report with rich vulnerability information, such as vulnerability details, severity, affected version, etc., from *Dependency-Track*.

VI. Empirical Evaluation

To provide more in-depth insights and evaluate the effectiveness of KubeSec in uncovering cluster-takeover paths introduced by TPAs, we conduct four complementary experiments in the Kubernetes ecosystem from the following aspects.

- *RQ1:* What is the performance of KubeSec in large-scale RBAC privilege parsing tasks?
- *RQ2:* What are the types of permissions and the status quo of permissions assigned to TPAs in the Kubernetes ecosystem?
- *RQ3:* What is the real-world impact of TPAs with privilege vulnerabilities?
- *RQ4:* What is the distribution of TPC vulnerabilities types in TPAs, and what is the average time to fix the vulnerabilities?

TABLE I
THE DATASET COMPOSITION, WHICH WE ILLUSTRATE WITH THE EXAMPLE OF *HWAMEISTOR*

Type	Meaning	Example
Project	The TPA's name	*hwameistor-main*
Kind	Role Binding Type	*ClusterRoleBinding*
Name	Role Binding Name	*hwameistor-role*
NameSpace	The TPA's namespace	*dynamic_parameters*
Subject	Role Binding Objects	*hwameistor-admin*
Pod	Service account users	*hwameistor-controller*
Label_class	The TPA's label	*hwameistor-pod*

TABLE II
PERFORMANCE COMPARISON OF RBAC ANALYSIS TOOLS

Tools	Precision (%)	R1 (%)	R2 (%)	F1
rback	46.2	2.2	27.1	0.04
audit	23.7	1.1	10.4	0.02
rbac-police	38.7	6.3	31.5	0.11
EPA	82.4	72.5	73.1	0.79
KubeSec	**86.3**	**80.2**	**80.7**	0.84

TABLE III
THE MEANING OF THE SYMBOLS

Notions	Type	meanings
V1	Verbs	* *means user can perform any operation*
V2	Verbs	*get the resource*
V3	Verbs	*list the resource*
V4	Verbs	*watch the resource*
V5	Verbs	*update the resource*
V6	Verbs	*patch the resource*
V7	Verbs	*escalate the resource*
R1	Resources	* *means user can perform any resource*
R2	Resources	*user can perform secrets resource*
R3	Resources	*ClusterRoleBinding resource*
R4	Resources	*ClusterRole resource*
R5	Resources	*Node resource*

A. *RQ1: Performance of* KubeSec

To answer RQ1, we demonstrate that KubeSec is effective for scanning large-scale databases by constructing the ground truth for evaluation. It is worth noting that since KubeSec accomplishes the TPC vulnerability analysis task through two state-of-the-art tools, CycloneDx and Dependency-Track, and the performance analysis of the related tools is fully discussed in the literature [49], [50], [51], we focus on the discussion of KubeSec's performance in the RBAC analysis task. Specifically, we compare KubeSec with four representative RBAC detection tools. These include three widely-used open-source scanners—rback [12], audit [11], and rbac-police [13]—as well as EPA, a state-of-the-art tool for detecting RBAC privilege binding relationships [4].

1) Ground Truth Construction: Since there is no public test data for RBAC binding detection in TPAs, we need to build a base fact dataset containing the labeled mappings of RBAC mapping relationships. We used 177 TPA projects from CNCF and 50 pre-installed TPAs from real-world Kubernetes services. These items originate from the dataset presented in previous work [4] and are used in conjunction with our KubeSec. To mark dependencies, we manually checked all files in the projects, both source and non-source files, to find RBAC dependencies. At the same time, we installed and ran all TPAs projects in the local Kubernetes cluster and manually organized to get the RBAC binding relationships. After 27 days of manual checking, we obtained 2372 RBAC bindings dependencies as base facts. As shown in Table I, the dataset contains 6 elements and 1 label that can reflect the details of permission binding for TPAs.

To assess inter-rater reliability, 20% of the bindings (476 entries) were randomly selected and independently re-annotated by a second reviewer, yielding Cohen's $\kappa = 0.87$ (95% CI: 0.82–0.92), indicating almost perfect agreement; discrepancies were resolved through discussion. Subsequently, the consensus samples were cross-validated in a Kind reference cluster by executing `kubectl auth can-i -list -as=system:serviceaccount:<ns>:<sa>`, which confirmed 98.3% of the entries against the API server responses; the eight mismatches were corrected, resulting in a final labeling accuracy of 98%.

2) Performance Analysis: We evaluate the KubeSec and the baselines and calculate the precision and recall according to the method described in [52]. Where precision $P = TP/(TP + FP)$, and $R1 = TP/(TP + FN)$ refers to the recall against the full ground truth. We compute R2 for each baseline based on the dependencies that the baseline supports to detect. As can be seen from the results in Table II, KubeSec outperforms all the baselines in terms of F1 score ($F1 = 2P * R1/(R1 + P)$).

KubeSec's R1 and R2 are almost the same, which shows that KubeSec's detection capability can cover most of the elements in the ground truth. More than 85% of KubeSec's erroneous results are caused by TPA privilege naming duplicates, e.g., the critical DaemonSet *kubevirt-operator* of Kubevirt is duplicated in the Kind and Subject elements. These code-caused problems are difficult to avoid. Nevertheless, our results demonstrate that KubeSec's permission detection module significantly outperforms other tools. As described in Section V, this is because KubeSec's *ConfigLocator* and *ConfigFetcher* modules are able to obtain accurate permission bindings from both the TPA's source code and the Kube-apiserver dimensions, whereas the other tools [4], [11], [12], [13] can only obtain permission bindings from the kube-apiserver because of the complexity of the authentication process and the limited information they can extract.

Answer to RQ1: KubeSec has demonstrated superior performance on RBAC privilege parsing tasks and is able to support large-scale analysis efforts.

B. *RQ2: Permission Vulnerability Type*

To answer RQ2, we conducted a large-scale analysis of the types of privilege bindings in the collected TPAs using KubeSec, and tried to communicate with all TPA development teams to clarify the types of privileges that cause the risk of cluster takeovers and to explore the root causes of privilege vulnerabilities.

As shown in Table III, we identified 4,659 permission types from 227 TPAs. Among them, we find that high-level permissions (that is, permissions granted to more than 100 TPAs)

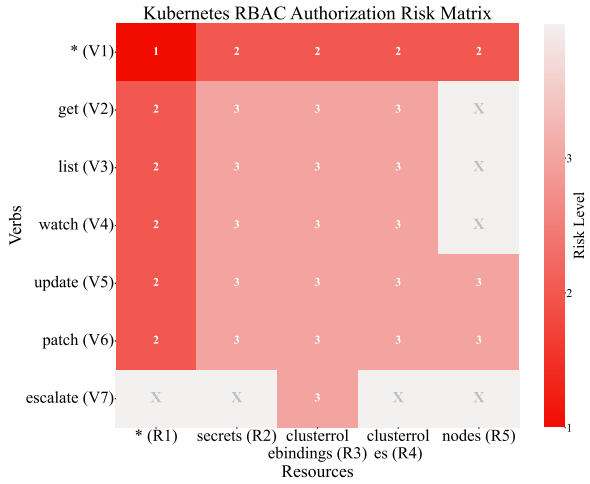


Fig. 5. Types of RBAC Privilege Bindings with Takeover Risk in Kubernetes. 1, 2, and 3 indicate high, medium, and low level of risk, respectively.

are mainly concentrated in 9 types of permissions for pods resources, nodes resources, and services resources, such as $list$, get, and $watch$ operations. This reflects the fact that monitoring and access to underlying resources is a core requirement of TPA functionality. Mid-level permissions (granted TPA permissions between 80 and 100) include 20 types of permissions for the $create$ and $patch$ operations on the events resource, the $list$, get, and $watch$ operations on the configmaps and namespace resources. Low-level permissions cover the $list$, get, and $watch$ operations of ingresses and serviceaccounts, the create operations of pods, and the $delete$, $update$, and $patch$ operations of secrets resources. It also includes the $*$ and $list$ operations on any resource, a total of 40 permission types.

We also found 3,822 permission types that were customized by each TPA for their specific functional requirements. These permissions are usually used only by themselves, such as $create$, $delete$, $list$, $patch$, $update$, and $watch$ operations on routes resources, and $list$, get, and $watch$ operations on commands resources. This reflects the flexibility and extensibility of the Kubernetes privilege system, but also increases the complexity and potential risks of privilege management.

Based on the three cluster takeover strategies in Section III-B1, we systematically identify critical verbs and sensitive resources (e.g., *Secrets*, *Nodes*, and *RBAC bindings*) based on their impact scope and operational sensitivity. These elements form the coverage boundaries of our risk matrix, ensuring the classification is anchored to concrete takeover mechanics. As shown in Fig. 5, we classify these permissions into three danger levels: ① Arbitrary operation–arbitrary resource level. This level represents the highest risk, as such permissions are widely recognized as sufficient to enable Direct Takeover (Strategy 1). Granting unrestricted privileges (e.g., $* - *$) allows an attacker to bypass authorization boundaries and obtain cluster-wide administrative capabilities. ② Arbitrary operation–sensitive resource or sensitive operation–arbitrary resource level. This level is commonly associated with Indirect Takeover (Strategy 2). For example, arbitrary operations on sensitive resources (e.g., $* - secrets$) may expose credentials, while sensitive operations across broad resource scopes (e.g., $list - *$) can facilitate cluster-wide resource discovery and subsequent privilege escalation. ③ Sensitive operations–sensitive resource level Level. Typically leveraged in Combination Takeover (Strategy 3), these permissions can also trigger rapid escalation. For example, `list-secrets` enables an attacker to harvest administrator tokens and utilize tainting mechanisms to escape nodes and seize control.

In addition, we discussed these authorizations with numerous TPA maintainers. We submitted risk reports to the TPA community where dangerous authorizations exist and received responses from 167 development teams. The following three examples represent the most representative views. 1) First, some teams emphasize that certain permissions are assigned out of functional necessity. For example, the **Eternal-Secrets** project security team explicitly states that *'I am aware how RBAC works - however this is a secret management project. It needs list secrets in order for it to do its own job, I.e. for the operator to work.'* 2) Second, some teams felt that while the assignment of certain permissions may not be technically necessary, they chose to grant them for specific usage scenarios. For example, the **Kuma** security team mentioned this in response to their list secrets permission risk, explaining: *'We need to read this to configure builtin Gateway with those secrets via XDS. Gateway secret can be in any namespace hence we need to access all secrets. If you don't use Gateway API you can set experimental.gatewayAPI to false to mitigate this'*, and 3) finally, there are some teams that recognize that permissions may be over-configured in some cases. For example, the **spiderpool** security team responded to all of the *get, list*, and *watch* permissions it grants on resources ($*$) with *'I just noticed the spiderpool-admin has configured resources: [$*$], this is unexpected. Thanks for the catch!'*.

Nevertheless, through communication and validation analysis, we believe that the following two aspects are the root causes of the risk of cluster takeover introduced by TPAs: 1) In the process of TPA development, insufficient anticipation of possible application scenarios and lack of fine-grained consideration of the necessity of privilege assignment led to many TPAs being given excessive privileges that are dangerous. This enables the execution of takeover **strategies 1 and 2**. 2) The lack of a dynamic management mechanism for permissions during the operation of TPAs leads to the long-term exposure of TPAs' administrative permissions in different namespaces, thus providing an execution prerequisite for takeover **strategy 3**.

Answer to RQ2: We identified 4,659 privilege binding types in 227 TPAs, of which 562 binding types are at risk of cluster takeover. Meanwhile, the lack of lifecycle management of privileges from creation to destruction is the root cause of cluster takeover.

C. RQ3: Impact of Privilege Vulnerability

To answer RQ3, we use KubeSec to detect privilege vulnerabilities in 227 TPAs. The Fig. 6 illustrates the RBAC privilege

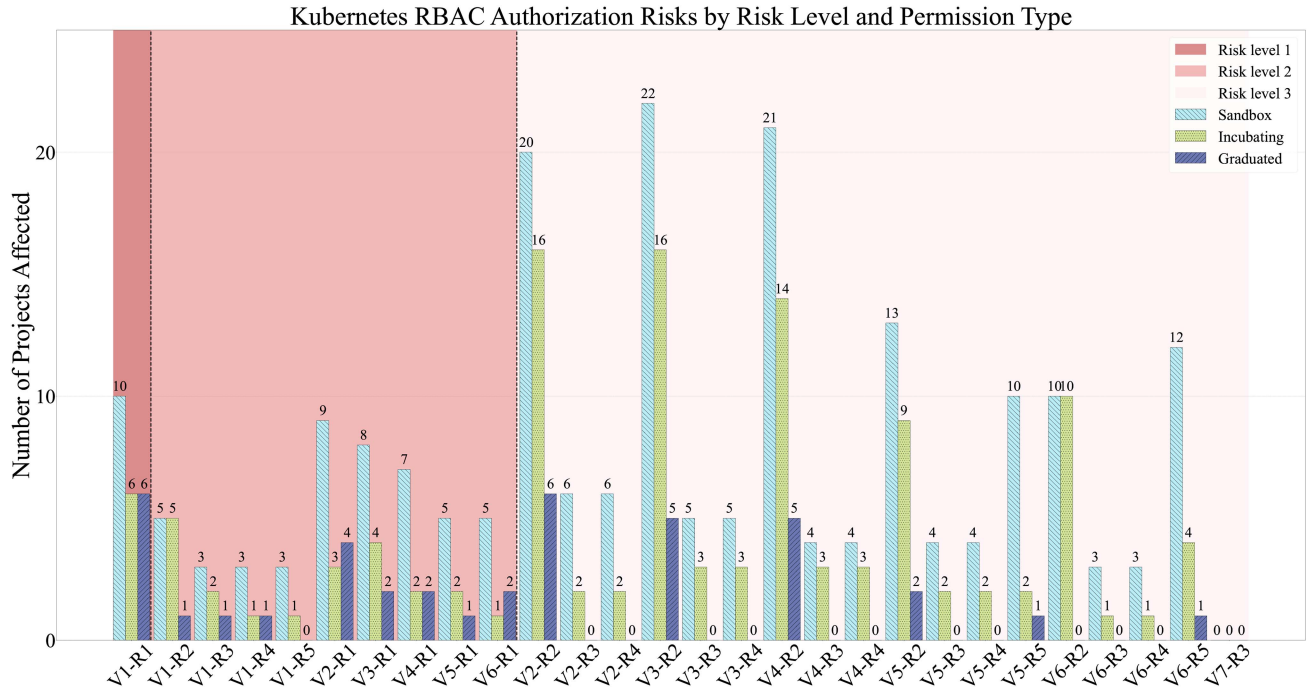


Fig. 6. The Privilege Vulnerability in three type Project.

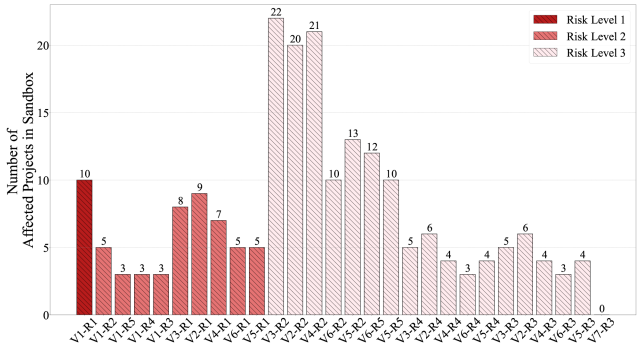


Fig. 7. Privilege vulnerability in sandbox project.

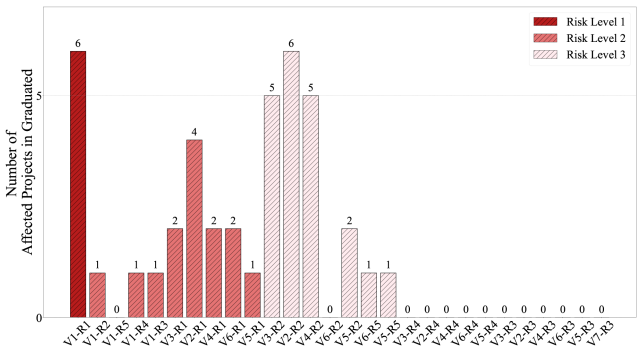


Fig. 9. Privilege vulnerability in graduated project.

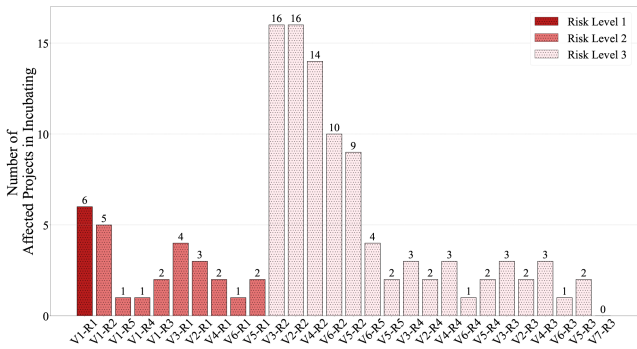


Fig. 8. Privilege vulnerability in incubating project.

vulnerability and risk level profile for the three types of projects in the CNCF. The Figs. 7, 8, and 9 show the distribution of the three permission risks in CNCF projects, with the horizontal coordinate showing the combinations type of permissions that are at risk of takeover, and the vertical coordinate indicating the frequency of occurrence of each combination of permissions. The same *Service Account* in many projects has multiple risky permissions, for example, (1) there exists a service account named *carina-csi-node* in the **Carina** project, which has both *list* and *watch* permissions on secrets resources and patch and update permissions on Nodes resources. After stealing the token of this *Service Account*, the attacker can directly control the whole cluster with three takeover strategies. (2) **Kanister** owns a *Service Account* named *kanister-operator*, which has *get, list, watch, create*, and *update* permissions on *ClusterRole* and *ClusterRoleBinding* sensitive resources. An attacker can use its create *ClusterRole* privilege to create a new *ClusterRole*, and define arbitrary permissions, and further utilize the create, update privileges on sensitive resources of the *ClusterRoleBinding*, the attacker can create or modify the *ClusterRoleBinding*, the high privilege *ClusterRole* binding to their own account or other

TABLE IV
KUBERNETES' TPA TAKEOVER VULNERABILITY IDENTIFIED BY KubeSec. √: FIXED, ✗: CONFIRMED.

TPA name	CVE-ID	CVSS Scores	CWE ID	Status	Risk Authority	Project
Carina	CVE-2024-32359	6.9	CWE-926	✓	*V2, V3 of R2; V6 of R5*	Sandbox
spiderpool	CVE-2024-33393	6.2	CWE-530	✓	*V2, V3, V4 of R1*	Sandbox
kubevirt	CVE-2024-33394	5.9	CWE-319	✓	*V1 of R1*	Incubating
karmada	CVE-2024-33396	8.4	CEW-275	✓	*V1, V2, V3 of R2*	Incubating
piraeus-operator	CVE-2024-33398	7.5	CWE-926	✓	*V2, V3, V4 of R2; V2, V3, V4, V5, V6 of R3, R4*	Sandbox
Kruise	CVE-2024-36532	**10.0**	CWE-749	✓	*V2, V3, V4, V5 of R2; V3 of R1*	Incubating
volcano	CVE-2024-36533	**9.8**	CWE-732	✓	*V2, V3, V4, V5 of R2; V5 of R5*	Incubating
hwameistor	CVE-2024-36534	**10.0**	CWE-312	✓	*V1 of R1*	Sandbox
meshery	CVE-2024-36535	**9.8**	CWE-295	✓	*V1 of R1*	Sandbox
fabedge	CVE-2024-36536	**9.8**	CWE-749	✓	*V2, V3, V4, V5 of R2; V5 of R5*	Sandbox
cert-manager	CVE-2024-36537	7.2	CWE-732	✓	*V2, V3, V4, V5 of R2*	Incubating
chaos-mesh	CVE-2024-36538	8.8	CWE-312	✓	*V2, V3, V4, V5 of R1*	Incubating
contour	CVE-2024-36539	**9.8**	CWE-276	✓	*V2, V3, V4 of R2*	Incubating
external-secrets	CVE-2024-36540	**9.8**	CWE-926	✓	*V2, V3, V4, V6 of R2*	Sandbox
logging-operator	CVE-2024-36541	8.8	CWE-275	✓	*V2, V3, V4 of R2; V2, V3, V4, V5 of R3, R4*	Sandbox
kuma	CVE-2024-36542	8.8	CWE-926	✓	*V3, V4 of R2*	Sandbox
kyverno	CVE-2024-32968	confirmed	CWE-1153	✗	*V2, V3 of R1*	Incubating
kubeslice	CVE-2024-53350	confirmed	CWE-732	✗	*V1, V3, V4 of R2*	Sandbox
loxilb	CVE-2024-53348	confirmed	CWE-312	✗	*V2, V3, V4 of R2*	Sandbox
kuadrant	CVE-2024-53349	confirmed	CWE-295	✗	*V1, V2 of R2*	Sandbox
pipecd	CVE-2024-53351	confirmed	CWE-926	✗	*V1, V2, V3, V4 of R2*	Sandbox

malicious accounts, so as to complete the authorization and take over the entire cluster.

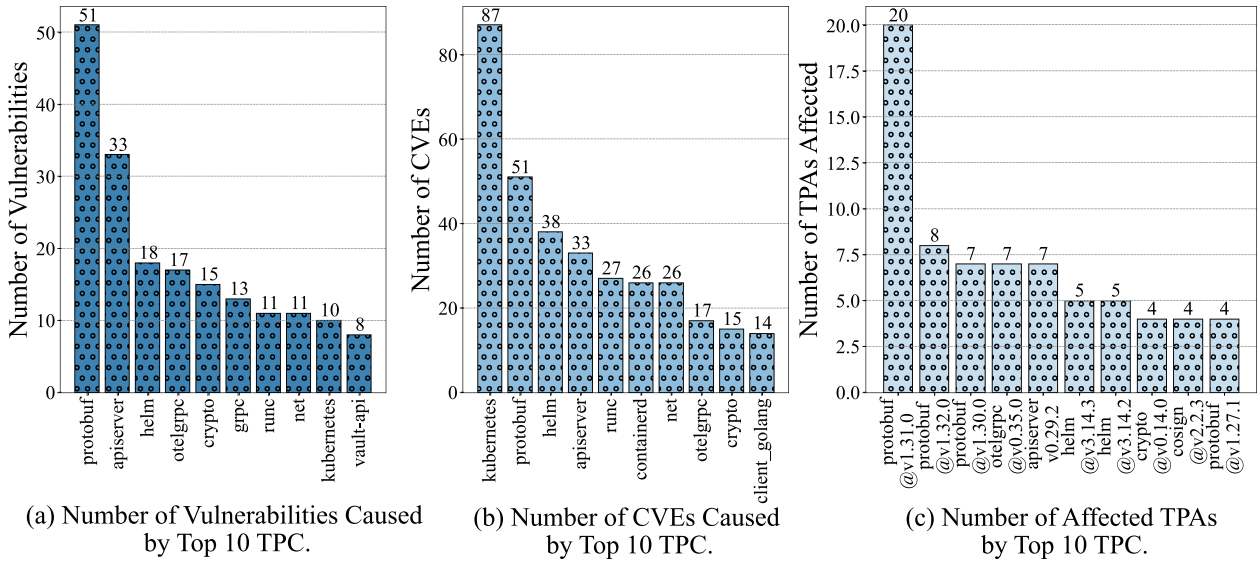
We found that there is a significant difference in the frequency of risk authorization in different types of projects in CNCF, in which (1) the frequency of taking over risk authority in sandbox projects is 210, of which 10 projects are in the high level of risk, and they can manipulate any resources in Kubernetes, 14 projects are in the middle level of risk, and 44 projects are in the low level of risk. (2) The frequency of takeover risk privileges in incubation projects is 120, and the number of projects affected by each level of risk privileges is 6, 10, and 21, respectively. (3) The frequency of risk privileges in graduation projects is 40, and the number of projects affected by each level of risk privileges is 6, 5, and 11, respectively. (4) In addition, we find that *list, get*, and *watch* operations on cluster secrets resources are widely used, which implies that a large number of TPAs have *ServiceAccounts* that directly list cluster secrets, which can be utilized by an attacker to take over the cluster.

As shown in Table IV, we further also analyze the authorization of pre-installed apps from mainstream cloud vendors, including Google GKE, Amazon EKS, Azure AKS, and Alibaba ACK, which have a higher probability of being used from the startup UI compared to other TPAs. As shown in Table V, we found privilege vulnerabilities with different risk levels in real service providers (1, 2, and 3 indicate high, medium, and low level of risk, respectively). In view of these findings, we responsibly submitted risk reports to the project maintainers and the cloud service provider security teams, and a portion of the maintainers believed that certain risky permissions are important for their TPAs and it is a great challenge to implement the functionality of their TPAs without such coarse-grained authorization. Another subset of maintainers adjusted their TPA's authorization strategy to effectively mitigate these risks. We do not publicize information about TPAs that have not yet been fixed due to security risks, and we currently have 17 CVEs from the community that have been officially recognized as fatal vulnerabilities, and are in close communication with the cloud provider security teams about vulnerability fixes.

TABLE V
PRIVILEGE VULNERABILITY SCENARIOS FOR DIFFERENT RISK LEVELS IN CLOUD SERVICE PROVIDERS

Kubernetes Vendors	Risk number / Total number	Risk type
Alibaba-ACK	6 / 13	1+2+3
AWS-EKS	3 / 13	1+2+3
Azure-AKS	2 / 7	1+3
Google-GKE	5 / 17	1+3

Answer to RQ3: We detected multiple types of vulnerabilities in 177 CNCF projects and 50 pre-installed TPAs in vendors. We analyzed the current state of different projects where there is a takeover of risky privileges and responsibly reported all the vulnerabilities. We have now obtained 21 CVEs, and 7 CVEs are officially certified as critical (CVSS score greater than 9.5).



(a) Number of Vulnerabilities Caused by Top 10 TPC.

(b) Number of CVEs Caused by Top 10 TPC.

(c) Number of Affected TPAs by Top 10 TPC.

Fig. 10. Number of affected vulnerabilities, CVEs and TPAs caused by the Top 10 TPCs.

TABLE VI
NUMBER OF TPC VULNERABILITIES AND CVES IN THE TPA

Type	TPC vulnerabilities	CVE numbers
TPA-Sandbox	168	71
TPA-Incubating	106	22
TPA-Graduated	83	31
TPA-Vendors	18	10
Count	375	134

TABLE VII
TOP 5 CWE SOFTWARE WEAKNESSES

CWE ID	Weakness	CVE numbers
264	Access Control Error	43
399	Resource Management Error	37
289	Authentication Bypass-Name	20
628	Function Error	13
94	Code Injection	11

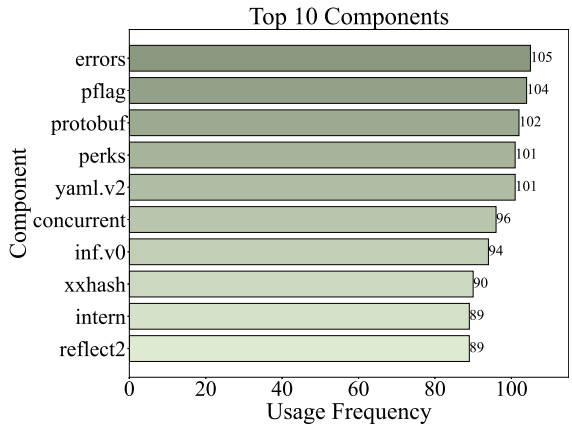


Fig. 11. Top 10 TPCs used in CNCF Project.

D. RQ4: Status of TPC Vulnerabilities

1) TPC Vulnerability Types and Distribution: After obtaining TPCs using KubeSec, we search for corresponding vulnerabilities in the vulnerability database. As shown in Table VI, KubeSec detects a total of 375 potential vulnerabilities in 227 TPAs involving 134 CVEs. The results show that the vulnerabilities introduced by TPCs involve a wide range of security issues. Specifically, we find that the TPAs in the Kubernetes ecosystem have huge security risks, among which, the TPAs in the three category are in a very insecure situation, with an average distribution of TPC vulnerabilities and CVEs amounting to 168 and 71 (Sandbox), 106 and 22 (Incubation), and 83 and 31 (Graduated). The number of TPC vulnerabilities and CVEs in vendors are 18 and 10, but these products have been heavily deployed in real production environments, for example, KubeVirt, which already has **5.2 k** stars on GitHub, and the famous *protobuf* vulnerability is still widely existed in it, which tends to cause even more serious harm.

Table VII lists the top 5 CWE software weaknesses by the number of CVEs, accounting for 92.5% of all CVEs KubeSec detected. The results further indicate that permission level risk and code level risk are the main risks faced by TPAs. In addition, we give the results of the top 10 TPCs used in TPAs, as shown in Fig. 11. The collected TPAs are categorized according to the Landscape graph of the CNCF project, and based on these results, we have the following findings: (1) The number of TPCs included in the TPAs of the Application Definition and Development category reaches 31.4 on average. The use of a large number of TPCs in this type of TPA is justified due to the fact that they need to perform a large number of cross-node tasks and need to handle functional interactions with other types of TPAs. (2) The provisioning type of TPAs contains the least number of TPCs, only 2.7 on average. We found that most TPAs in this category are used to accomplish a single task in a specific scenario, e.g., Notary's CLI tool for signing and validating

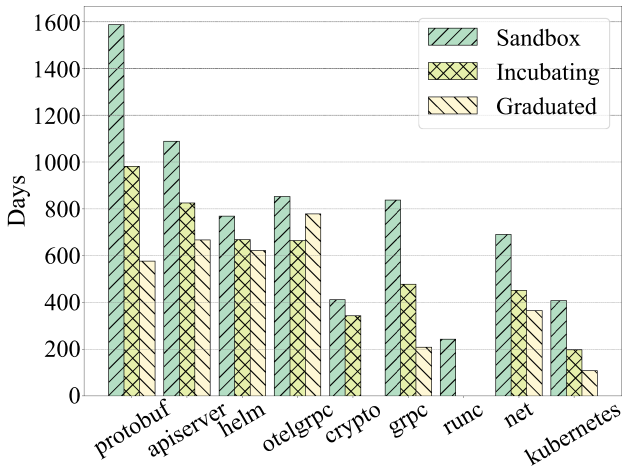


Fig. 12. Delay time of TPCs.

artifacts, which usually employs its own implementation instead of TPCs. (3) Different types of TPAs share commonalities in terms of employing TPCs, e.g., *protobuf* are in the top 10 of each type of TPA.

We list the top 10 vulnerable TPCs in terms of the number of affected TPAs in Fig. 10, the number of resulting CVEs, and the number of resulting vulnerabilities. We find that although KubeSec detects a large number of TPCs, most vulnerabilities are concentrated in a few TPCs. For example, *pflag* contains 78 CVEs affecting 102 TPAs, resulting in a total of 280 vulnerabilities.

2) Vulnerability Remediation Status: Most of the TPC-induced vulnerabilities are due to the fact that the TPA is still using an outdated TPC; however, developers may not use the latest TPC and ignore these vulnerabilities for performance and stability reasons. Therefore, we explore the fixing of TPC vulnerabilities in this section.

We use the delay time (the number of days between the TPC version at the time of TPA development and the TPC version at the time of TPA release) for the statistics. Specifically, we define the Remediation Delay (D) as follows:

$$D = T_{TPA\_release} - T_{TPC\_version}$$

where $T_{TPA\_release}$ is the release date of the TPA version, and $T_{TPC\_version}$ is the official release date of the specific TPC version declared in the TPA's dependency manifest (e.g., go.mod). We calculate D by following the procedure below: (1) parsing the TPA's source code or manifests to identify the exact TPC version used; (2) querying official registries (e.g., GitHub Releases, PyPI, or Maven Central) to obtain the ground-truth release timestamps; and (3) addressing complexities like vendoring by prioritizing the version specified in the repository's lock-files.

We counted the top 10 TPAs with the highest number of GitHub stars from the three CNCF project types, and Fig. 12 shows the average latency of the TPCs used for each TPA. Overall, the average time to fix vulnerabilities for TPCs was more than 10 months. Among them, *protobuf* uses the longest average delay of 1200 days, i.e., it is still using TPCs released 3 years ago. *runc* has the shortest delay of less than 250 days, which may be due to the fact that this type of security-related apps pay more attention to their own software security.

Dependency-Track aggregates vulnerability intelligence from multiple public sources (NVD, GitHub Advisories, OSS Index) that may occasionally over-approximate affected component versions. To quantify this effect we manually audited a randomly selected 10% subset of the 375 TPC vulnerabilities reported by KubeSec (38 cases). Each entry was compared against upstream vendor advisories, commit logs, and any available VEX statements; three CVEs were found to be incorrectly flagged for the exact versions present in our TPAs, yielding an empirical false-positive rate of 7.9% within the sample. These three cases were retained in the table but are explicitly tagged with a `VEX-false-positive` field in the public dataset so that future analyses can filter them if desired. We further note that only vulnerabilities with CVSS $\geq$ 5.0 and CWE classes related to privilege escalation or remote code execution were included in the risk discussion, which additionally mitigates the impact of potential misclassifications.

Answer to RQ4: We detected a total of 375 potential vulnerabilities from 227 TPAs across 134 CVEs, with the sandbox-type TPAs having the worst security profile. We also found that most of the vulnerabilities are concentrated in a few TPCs. Meanwhile, the TPC vulnerability fixing situation is not optimistic, and its average fixing time is more than 10 months.

VII. Conclusion

In this paper, we use KubeSec to conduct a large-scale empirical study of the takeover risk caused by TPAs. We identified 562 insecure permission binding methods and detected 375 TPC vulnerabilities associated with 134 CVEs. Our analysis reveals the security status of TPAs widely used in the Kubernetes ecosystem. We believe that our work will provide insights for further research on the security of TPAs in the Kubernetes ecosystem.

References

[1] "CNCF," 2023. Accessed: May 23, 2024. [Online]. Available: https://www.cncf.io/reports/cncf-annual-survey-2023/
[2] "Sysdig," 2023. Accessed: May 23, 2024. [Online]. Available: https://sysdig.com/2023-cloud-native-security-and-usage-report/
[3] "Arstechnica," 2023. Accessed: May 23, 2024. [Online]. Available: https://arstechnica.com/information-technology/2018/02/tesla-cloud-resources-are-hacked-to-run-cryptocurrency-mining-malware/
[4] N. Yang, W. Shen, J. Li, X. Liu, X. Guo, and J. Ma, "Take over the whole cluster: Attacking Kubernetes via excessive permissions of third-party applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 3048–3062.
[5] Y. He et al., "Cross container attacks: The bewildered {eBPF} on clouds," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 5971–5988.
[6] Z. Li et al., "Lost along the way: Understanding and mitigating path-misresolution threats to container isolation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2023, pp. 3063–3077.
[7] Q. Zeng, M. Kavousi, Y. Luo, L. Jin, and Y. Chen, "Full-stack vulnerability analysis of the cloud-native platform," *Comput. Secur.*, vol. 129, 2023, Art. no. 103173.
[8] A. Rahman, S. I. Shamim, D. B. Bose, and R. Pandita, "Security misconfigurations in open source Kubernetes manifests: An empirical study," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 4, pp. 1–36, 2023.

[9] M. U. Haque, M. M. Kholoosi, and M. A. Babar, "KGSecConfig: A knowledge graph based approach for secured container orchestrator configuration," in *Proc. IEEE Int. Conf. Softw. Anal., Evol. Reeng.*, 2022, pp. 420–431.
[10] J. Xiao et al., "Attacks are forwarded: Breaking the isolation of MicroVM-based containers through operation forwarding," in *Proc. 32nd USENIX Secur. Symp.*, 2023, pp. 7517–7534.
[11] "Audit," 2024, Accessed: May 23, 2024. [Online]. Available: https://github.com/cyberark/kubernetes-rbac-audit
[12] "RBACK," 2024. Accessed: May 23, 2024. [Online]. Available: https://github.com/team-soteria/rback?tab=readme-ov-file
[13] "RBAC police," 2024. Accessed: May 23, 2024. [Online]. Available: https://github.com/PaloAltoNetworks/rbac-police
[14] V. Jain, B. Singh, M. Khenwar, and M. Sharma, "Static vulnerability analysis of docker images," in*IOP Conf. Ser., Mater. Sci. Eng.*, 2021, vol. 1131, no. 1, Art. no. 012018.
[15] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," in *Proc. 10th Annu. Comput. Commun. Workshop Conf.*, 2020, pp. 975–980.
[16] S. Kwon and J.-H. Lee, "DIVDS: Docker image vulnerability diagnostic system," *IEEE Access*, vol. 8, pp. 42666–42673, 2020.
[17] Y. Zheng, W. Dong, and J. Zhao, "ZeroDVS: Trace-ability and security detection of container image based on inheritance graph," in *Proc. IEEE 5th Int. Conf. Cryptogr., Secur. Privacy*, 2021, pp. 186–192.
[18] N. Zhao et al., "Large-scale analysis of the docker hub dataset," in *Proc. 2019 IEEE Int. Conf. Cluster Comput.*, 2019, pp. 1–10.
[19] M. Zhan, Y. Li, H. Yang, G. Yu, B. Li, and W. Wang, "Coda: Runtime detection of application-layer CPU-exhaustion dos attacks in containers," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 1686–1697, May/Jun. 2023.
[20] Z. Li, H. Jin, D. Zou, and B. Yuan, "Exploring new opportunities to defeat low-rate DDoS attack in container-based cloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 695–706, Mar. 2020.
[21] Y. C. Jadhav, A. Sable, M. Suresh, and M. K. Hanawal, "Securing containers: Honeypots for analysing container attacks," in *Proc. 15th Int. Conf. Commun. Syst. Netw.*, 2023, pp. 225–227.
[22] A. Y. Wong, E. G. Chekole, M. Ochoa, and J. Zhou, "On the security of containers: Threat modeling, attack analysis, and mitigation strategies," *Comput. Secur.*, vol. 128, 2023, Art. no. 103140.
[23] I. Mavridis and H. Karatza, "Orchestrated sandboxed containers, unikernels, and virtual machines for isolation-enhanced multitenant workloads and serverless computing in cloud," *Concurrency Computation, Pract. Experience*, vol. 35, no. 11, 2023, Art. no.e6365.
[24] B. Tak, H. Kim, S. Suneja, C. Isci, and P. Kudva, "Security analysis of container images using cloud analytics framework," in *Proc. 25th Int. Conf. Web Serv., Held Part Serv. Conf. Federation*, Seattle, WA, USA, Springer, 2018, pp. 116–133.
[25] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the relation between outdated docker containers, severity vulnerabilities, and bugs," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reeng.*, 2019, pp. 491–501.
[26] A. Zerouali, V. Cosentino, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the impact of outdated and vulnerable Javascript packages in docker images," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reeng.*, 2019, pp. 619–623.
[27] K. Wist, M. Helsem, and D. Gligoroski, "Vulnerability analysis of 2500 docker hub images," in *Proc. Adv. Secur., Netw., Internet Things*, 2021, pp. 307–327.
[28] P. Liu et al., "Understanding the security risks of docker hub," in *Proc. 25th Eur. Symp. Res. Comput. Secur.*, Guildford, U.K., Springer, 2020, pp. 257–276.
[29] A. Tomar, D. Jeena, P. Mishra, and R. Bisht, "Docker security: A threat model, attack taxonomy and real-time attack scenario of DoS," in *Proc. 10th Int. Conf. Cloud Comput., Data Sci. Eng.*, 2020, pp. 150–155.
[30] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52976–52996, 2019.
[31] T.-P. Doan and S. Jung, "DAVS: Dockerfile analysis for container image vulnerability scanning," *Comput., Mater. Continua*, vol. 72, no. 1, pp. 1699–1711, 2022.
[32] X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang, "Houdini's escape: Breaking the resource rein of Linux control groups," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1073–1086.
[33] A. Agache et al., "Firecracker: Lightweight virtualization for serverless applications," in *Proc. 17th USENIX Symp. Networked Syst. Des. Implementation*, 2020, pp. 419–434.
[34] X. Gao, B. Steenkamer, Z. Gu, M. Kayaalp, D. Pendarakis, and H. Wang, "A study on the security implications of information leakages in container clouds," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 1, pp. 174–191, Jan./Feb. 2021.
[35] Y. Wang et al., "ContainerGuard: A real-time attack detection system in container-based Big Data platform," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 3327–3336, May 2022.
[36] R. Vaño, I. Lacalle, P. Sowiński, R. S-Julián, and C. E. Palau, "Cloud-native workload orchestration at the edge: A deployment review and future directions," *Sensors*, vol. 23, no. 4, 2023, Art. no. 2215.
[37] C.-C. Chang, S.-R. Yang, E.-H. Yeh, P. Lin, and J.-Y. Jeng, "A kubernetes-based monitoring platform for dynamic cloud resource provisioning," in *Proc. 2017 IEEE Glob. Commun. Conf.*, 2017, pp. 1–6.
[38] K. Senjab, S. Abbas, N. Ahmed, and A. u. R. Khan, "A survey of kubernetes scheduling algorithms," *J. Cloud Comput.*, vol. 12, no. 1, 2023, Art. no. 87.
[39] G. Turin, A. Borgarelli, S. Donetti, F. Damiani, E. B. Johnsen, and S. L. T. Tarifa, "Predicting resource consumption of Kubernetes container systems using resource models," *J. Syst. Softw.*, vol. 203, 2023, Art. no. 111750.
[40] Z. Zhong, M. Xu, M. A. Rodriguez, C. Xu, and R. Buyya, "Machine learning-based orchestration of containers: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 1–35, 2022.
[41] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, "XI commandments of Kubernetes security: A systematization of knowledge related to Kubernetes security practices," in *Proc. IEEE Secure Develop.*, 2020, pp. 58–64.
[42] M. Sayagh and A. E. Hassan, "ConfigMiner: Identifying the appropriate configuration options for config-related user questions by mining online forums," *IEEE Trans. Softw. Eng.*, vol. 47, no. 12, pp. 2907–2918, Dec. 2021.
[43] H. Kermabon-Bobinnec et al., "ProSPEC: Proactive security policy enforcement for containers," in *Proc. 12th ACM Conf. Data Appl. Secur. Privacy*, 2022, pp. 155–166.
[44] A. Blaise and F. Rebecchi, "Stay at the Helm: Secure Kubernetes deployments via graph generation and attack reconstruction," in *Proc. IEEE 15th Int. Conf. Cloud Comput.*, 2022, pp. 59–69.
[45] R. R. Karn, P. Kudva, H. Huang, S. Suneja, and I. M. Elfadel, "Cryptomining detection in container clouds using system calls and explainable machine learning," in*IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 674–691, Mar. 2021.
[46] S. I. Shamim, "Mitigating security attacks in kubernetes manifests for security best practices violation," in *Proc. 29th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1689–1690.
[47] "CycloneDX," 2023. Accessed: May 23, 2024. [Online]. Available: https://cyclonedx.org
[48] "DependencyTrack," 2023. Accessed: May 23, 2024. [Online]. Available: https://dependencytrack.org
[49] J. Wu et al., "OSSFP: Precise and scalable C/C++ third-party library detection using fingerprinting functions," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng.*, 2023, pp. 270–282.
[50] X. Zhan et al., "Automated third-party library detection for android applications: Are we there yet?," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2020, pp. 919–930.
[51] L. Jiang, H. Yuan, Q. Tang, S. Nie, S. Wu, and Y. Zhang, "Third-party library dependency for large-scale SCA in the C/C++ ecosystem: How far are we?," in *Proc. 32nd ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2023, pp. 1383–1395.
[52] W. Tang et al., "Towards understanding third-party library dependency in C/C++ ecosystem," in *Proc. 37th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2022, pp. 1–12.

Tao Zheng received the PhD from Sichuan University in 2024. He is currently a cloud security expert with Huawei Cloud Computing Technology Company, Ltd. His research findings have been published in renowned journals and conferences, such as *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Intelligent Transportation Systems*, *IEEE Internet of Things Journal*, and IEEE GLOBECOM 2022. His research interests include cloud security, software analytics, and LLM safety and its security applications.

Qiyu Hou is currenly working toward the master's degree with the School of Cyber Science and Engineering, Sichuan University. His research interests include cloud computing security and software analysis.

Hao Ren (Member, IEEE) received the PhD degree from the University of Electronic Science and Technology of China in 2020. From 2018 to 2019, he was a visiting PhD student with the University of Waterloo. From 2021 to 2022, he was a research fellow with The Hong Kong Polytechnic University and Nanyang Technological University from 2022 to 2024. He is currently an Associate Professor with the School of Cyber Science and Engineering, Sichuan University. His research outcomes appeared in major conferences and journals, such as WWW, ICML, AAAI, ACM ASIACCS, ACSAC, *IEEE Transactions on Information Forensics and Security*, and *IEEE Transactions on Dependable and Secure Computing*. His research interests include data security and privacy, AI security and privacy, APT hunting, and tracing. He was the recipient of the Best Paper Award from IEEE BigDataSecurity 2023. He is an Associate Editor for *IEEE Transactions on Dependable and Secure Computing*.

Xingshu Chen received the PhD degree from Sichuan University in 2004. She is currently a professor with the School of Cyber Science and Engineering, Cyber-Security Research Institute, Sichuan University. She is also a member of the China Information Security Standardization Technical Committee. Her research interests include cloud computing, cloud security, distributed file system, Big Data processing, network protocol analysis, and new media supervision.

Gelei Deng received the bachelor's degree from the Singapore University of Technology and Design in 2018 and the PhD with Nanyang Technological University in 2024. He is currently a research assistant professor with Nanyang Technological University. He has authored or coauthored papers in reputable conferences, such as ACM CCS, USENIX Security, and NDSS. His research interests include AI system security and security testing.

Tianwei Zhang (Member, IEEE) received the bachelor's degree with Peking University in 2011 and the PhD degree from Princeton University in 2017. He is currently an associate professor with the School of Computer Science and Engineering, Nanyang Technological University. His research interests include computer system security, security threats and defenses in machine learning systems, autonomous systems, computer architecture, and distributed systems.

Guowen Xu (Senior Member, IEEE) received the PhD degree from the University of Electronic Science and Technology of China in 2020. He is currently a professor with the School of Computer Science and Engineering (School of Cyber Security), University of Electronic Science and Technology of China. His research interests include applied cryptography and privacy-preserving issues in deep learning. He was the recipient of the Best Paper Award of the 26th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2020), Best Student Paper Award of the Sichuan Province Computer Federation (SCF 2019), Student Conference Award of IEEE International Conference on Computer Communications (INFOCOM 2020), and Distinguished Reviewer of ACM Transactions on the Web. He is serving as an associate editors for *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Circuits and Systems for Video Technology*, *IEEE Transactions on Network and Service Management*, and *Pattern Recognition*.

Hongwei Li (Fellow, IEEE) is currently the vice dean and professor with the School of Computer Science and Engineering (School of Cyber Security), University of Electronic Science and Technology of China. His research interests include network security and applied cryptography. He serves/served as the technical symposium co-chair of IEEE ICC 2022, ACM TUR-C 2019, IEEE ICCC 2016, and many technical program committees for international conferences, such as IEEE INFOCOM, IEEE WCNC, IEEE SmartGridComm, BO-DYNETS, and IEEE DASC. He serves as an associate editor for *IEEE Internet of Things Journal*, and the lead guest editor of *IEEE Network*, *IEEE Transactions on Vehicular Technology*, and *IEEE Internet of Things Journal*. He is the distinguished lecturer of IEEE Vehicular Technology Society.