IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 30 September 2024, accepted 26 November 2024, date of publication 11 December 2024,
date of current version 27 December 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3514751

RESEARCH ARTICLE

Uncovering Threats in Container Systems: A Study on Misconfigured Container Components in the Wild

DONGMIN CHOI[1], HYUNMIN SEO[2], KWANWOO KIM[2], MYOUNGSUNG YOU[2], SEUNGWON SHIN[2], (Member, IEEE), AND JINWOO KIM[3]

[1]Graduate School of Information Security, KAIST, Yuseong-gu, Daejeon 34141, Republic of Korea
[2]School of Electrical Engineering, KAIST, Yuseong-gu, Daejeon 34141, Republic of Korea
[3]School of Software, Kwangwoon University, Nowon-gu, Seoul 01897, Republic of Korea

Corresponding author: Jinwoo Kim (jinwookim@kw.ac.kr)

This work was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP) under the project "Development of Ransomware Attack Source Identification and Analysis Technology" (No. RS-2022-II220745, 70%), and by the National Research Foundation of Korea (NRF) grant funded by the Korean Government [Ministry of Science and ICT (MSIT)] (No. RS-2024-00457937, 30%).

ABSTRACT The increasing popularity of cloud computing has led to a significant rise in the use of container technology. Docker and Kubernetes have emerged as the de facto standards for container orchestration frameworks due to their reliability, flexibility, and ease of operation, supported by scalable, HTTP-based interfaces. Given the critical nature of infrastructure systems, container components within such orchestration frameworks should adhere to strict security levels. However, administrative misconfigurations can introduce serious vulnerabilities, exposing security-critical container components to external networks and allowing adversaries to discover and exploit them as attack vectors. In this paper, we investigate the security threats posed by misconfigured container components (MCC) that are exposed to the Internet. Through an Internet-scale measurement, we identify a total of 1,003,947 MCCs, with the majority operating under default configurations and outdated software versions. Our analysis reveals that renowned institutes, governments, and enterprises that are operating exposed MCCs, suggesting significant security risks. In addition, we conduct a real-world experiment within multi-branch campus network, scanning 150,235 IP addresses to uncover actual vulnerabilities in MCCs. We identify five distinct vulnerabilities that either leak sensitive information or allow remote code execution, demonstrating the real-world feasibility and potential impact of exploiting these misconfigured container components.

INDEX TERMS Cloud computing security, network security, internet security.

I. INTRODUCTION

Recent advancements in cloud computing have significantly contributed to the widespread adoption of container technologies. Containers enable the execution of applications in isolated environments, optimizing deployment, scalability, and management processes. Notably, Docker [1] and Kubernetes [2] have emerged as industry standards due to their robustness and versatility in orchestrating these environments [3], [4]. Docker encapsulates applications within containers, providing customized environments tailored to specific application requirements. Meanwhile, Kubernetes automates the deployment, scaling, and management of containers, substantially reducing operational overhead. Together, these technologies empower developers and administrators to improve productivity and operational efficiency.

Effective management and coordination within a container environment require the seamless integration of various container components. In Docker, this is facilitated by the Docker daemon and Docker API server, while in Kubernetes, the key components include kube-apiserver, kubelet, and etcd. These elements communicate via HTTP-based interfaces

The associate editor coordinating the review of this manuscript and approving it for publication was Amjad Mehmood.

(e.g., REST APIs), providing the system with flexibility and scalability [5], [6], [7]. Although this architecture streamlines development and management, its inherent openness can introduce critical security vulnerabilities if not properly secured [8], [9]. The trade-off between ease of use and security highlights the need for robust security strategies and meticulous configuration in containerized environments.

Misconfigurations, often resulting from administrative errors or oversight, can lead to severe security breaches within container environments. Core infrastructure components, such as those in Docker and Kubernetes systems, should be strictly protected from external access. However, misconfigurations can inadvertently expose these components, providing adversaries with direct entry points for intrusion [10]. Moreover, critical security controls, including authentication mechanisms and access controls, can be disabled due to configuration errors [11]. Such vulnerabilities enable the exfiltration of sensitive information, such as server versions, credentials, and privileges. Given the simplicity of the HTTP-based interface and its central role in container environments, these breaches can escalate to full compromise of the entire container ecosystem [12], resulting in the complete theft of container data and resources. Empirical evidence indicates that a substantial number of container components are misconfigured, thereby significantly undermining their overall security posture [13].

In this paper, we define *Misconfigured Container Components (MCCs)* as container components that are exposed to the Internet due to configuration errors. These MCCs render other components within the container system susceptible to various attacks, including information leakage and remote code execution. Adversaries can discover these MCCs in target container systems over the Internet and compromise them to gain unauthorized access to sensitive data or deploy malicious applications [14]. Additionally, compromised MCCs can act as stepping stones for further attacks, enabling adversaries to move laterally within the system [15], [16], [17] and compromise other accessible containers.

Therefore, identifying and mitigating existing MCCs within containerized environments is of critical importance for security practitioners. To this end, we aim to address the following research questions:

- **RQ1**: Are MCCs prevalent in the wild? If so, which container components are most common, and where are they typically found?
- **RQ2**: Do MCCs provide adversaries with opportunities to launch attacks? If so, how severe are the associated vulnerabilities?

To answer these questions, we conduct a measurement study consisting of two parts: (i) evaluating the prevalence of MCCs on an Internet scale (**RQ1**) and (ii) identifying security vulnerabilities in these components through real-world case studies (**RQ2**).

First, to demonstrate the prevalence of MCCs in the wild, we conducted a comprehensive survey to assess their global distribution and status. Our findings revealed over one million exposed IP addresses associated with MCCs. We classified these components by analyzing protocol-specific attributes, such as HTTP response headers and content, to accurately identify each MCC. Second, to evaluate the vulnerabilities of MCCs, we conducted a real-world experiment within a campus network to identify active MCCs and assess their security posture. This process involved scanning all 150,235 IP addresses within the network and determining whether these MCCs could be feasibly exploited in potential attacks.

Our findings indicate that a significant number of MCCs operate with default configurations and outdated software versions, making them susceptible to known threats. Furthermore, we identified several well-known organizations deploying MCCs, and some of MCCs are already blocklisted, suggesting prior exploitation. In our campus network study, we uncovered five critical vulnerabilities among 57 MCCs and demonstrated the feasibility of two types of attacks: (i) information leakage and (ii) remote code execution.

A. CONTRIBUTIONS

To the best of our knowledge, this is the first comprehensive investigation into MCCs conducted at a large scale. Additionally, we have successfully quantified the potential risks and malicious exploitation opportunities associated with these components. Our key findings are summarized as follows:

- We conducted an extensive investigation and identified container components exposed due to misconfigurations across the Internet. A total of 1,003,947 MCCs were discovered, each of which is publicly accessible and presents a potential attack vector.
- We revealed that the majority of MCCs operate with default configurations and outdated software versions, creating significant security vulnerabilities and making them particularly susceptible to known threats such as 1-day attacks (***Finding I, II***).
- We identified over 1,700 MCCs deployed within organizations with extensive user bases, posing a substantial risk to sensitive information (***Finding III***). Furthermore, over 1,000 of these MCCs have been blocklisted, indicating prior compromise and malicious exploitation (***Finding IV***).
- We demonstrated the presence of MCC vulnerabilities in a real-world campus network, identifying 57 MCCs out of 150,235 IP addresses and uncovering five distinct vulnerabilities. Our findings show that it is possible to access sensitive information and compromise these vulnerable container components (***Finding V***).

II. BACKGROUND AND MOTIVATION

Containerization and orchestration technologies, such as Docker and Kubernetes, streamline the deployment and management of applications across different environments.

Docker is a platform that leverages containerization technology to encapsulate an application and its dependencies within a Docker container, ensuring consistent operation across different computing environments [1]. Kubernetes, an open-source container orchestration tool, facilitates the management of containerized applications through the initialization of clusters consisting of worker machines, known as *nodes*, which host these applications [2], [18]. The cluster is managed by the *Control Plane*, also referred to as the *master*, which oversees both the worker nodes and the containers within the cluster. Each worker node, under the supervision of the Control Plane, executes the containerized applications [19]. REST API is an architectural style for web-based applications that enables stateless information exchange between client and server using standard HTTP methods. Its simplicity and scalability make it widely used for cloud resource management and service integration.

To protect container components from adversaries, various security mechanisms are employed within cloud environments to mitigate data leakage risks. Initially, container components that oversee cloud operations are configured by administrators to prevent external exposure. Subsequently, authentication among nodes is facilitated through token usage, which verifies the authenticity of messages originating from trusted nodes within the cluster. Moreover, when administrators access a component, robust access control is enforced via Role-Based Access Control (RBAC), predicated on precise administrative configurations.

Nonetheless, if these component settings are inadequately configured due to administrative oversight, unauthorized third parties may gain access to components exposed to the Internet, thereby obtaining unfettered access to the resources of the cloud cluster. In such instances, the inherent simplicity and flexibility of the REST API could paradoxically become detrimental, elevating the risk of significant exposure of resource and system data.

A. MOTIVATION

We define Misconfigured Container Components (MCC) as components with erroneous configurations that, when exposed to the Internet, facilitate unauthorized access and potential data exfiltration from the container system. The exposed container components such as Kube-apiserver, Kubelet, and Docker daemon pose significant security risks, as they provide adversaries with potential attack vectors. Exploiting their HTTP-based interfaces can allow unauthorized access to systems and resources [13], [17]. For instance, inadequate security configurations may enable adversaries to leverage vulnerabilities in the REST API components, resulting in privilege escalation, unauthorized access to sensitive information, or execution of privileged commands within the container environment [20], [21]. Moreover, adversaries can use these attack paths to move laterally and extract data or install ransomware within cloud environments they cannot directly access [15], [16], [17]. Adversaries may also target MCC for Denial-of-Service (DoS) attacks, potentially disrupting services and causing significant damage to organizations. Thus, it is critical to understand the characteristics of MCC and configure container components securely to reduce exposure to the Internet.

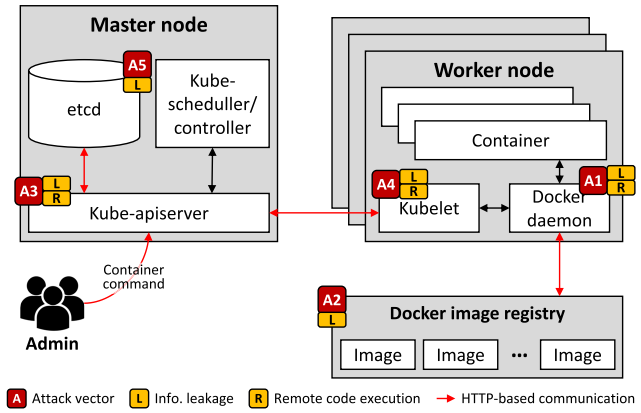


FIGURE 1. Attack vectors in misconfigured container components (MCCs).

B. ATTACK VECTORS OF CONTAINER COMPONENTS

Our research targets container components that are exposed to the Internet and publicly accessible due to misconfigurations within the container environment. Specifically, we focus on components utilizing the REST API within Docker and Kubernetes environments. These components include the Docker daemon, Docker registry API, kube-apiserver, kubelet, and etcd. Each of these components presents unique attack vectors (**A1**-**A5**) to adversaries, as shown in Figure 1.

1) A1. DOCKER DAEMON

Anonymous access might be enabled when initializing the Docker daemon by manually. Adversaries may gain access to the Docker commands as if they were operating their own Docker daemon. For example, they can list images and containers and even run commands on containers. This component may contribute to a remote code execution (RCE) vector.

2) A2. DOCKER REGISTRY API

The Docker Registry API provides interfaces for accessing Docker images. If misused, adversaries can extract private images from unauthenticated registries, allowing them to analyze configurations and identify vulnerabilities in the applications contained within the images.

3) A3. KUBE-APISERVER

Kube-apiserver serves as the central control plane for a Kubernetes cluster and provides a REST API for managing the cluster state. It does not exist on the worker node, only on the control plane. By attacking the Kube-apiserver,

adversaries can enumerate all of the running containers in a cluster.

4) A4. KUBELET

Kubelet, operating as a node agent on each node in the cluster, is primarily responsible for managing containers on their respective nodes. Kubelet servers provide adversaries with RCE vector. System commands can be executed inside the containers that permit anonymous authentication. As Kubelet runs on each node, every misconfigured Kubernetes node has an RCE vector.

5) A5. ETCD

etcd is a key-value store that serves as the fundamental backing repository for all cluster data within the Kubernetes system. It is commonly used to store and distribute passwords and configuration settings. Adversaries may exploit exposed etcd servers to access or disclose Kubernetes configurations, secret values, and authentication data, potentially revealing server credentials.

C. THREAT MODEL

When configuring a containerized system, administrators may inadvertently misconfigure component settings due to oversight. Such misconfigurations, particularly involving authentication mechanisms or access controls, can create significant security vulnerabilities. MCCs can serve as potential attack vectors for adversaries. Since these components often provide a REST API, adversaries can readily access the servers and invoke the API over HTTP once a misconfigured server is identified.

The goal of attacking misconfigured MCCs is to distribute malicious applications by executing code on Kubernetes pods (i.e., a group of containers sharing the same resources). The attack method is straightforward: adversaries can attack servers by calling REST API via HTTP using web browsers or command-line tools such as `curl` [22]. To worsen the situation, Kube-apiserver and Kubelet servers are exposed to the Internet by default. In other words, an adversary only needs to identify a misconfigured and exposed server, gain unauthorized access, and execute malicious code through the REST API.

D. MOTIVATING EXAMPLE

Figure 2 shows a real-world example attack scenario targeting a Kube-api-server (A3). In this scenario, the Kube-api-server has no authentication and is publicly accessible due to misconfiguration. Adversaries begin by gathering information about active containers within the target system, exploiting the Kube-apiserver (1). Using the obtained data, they identify a vulnerable container and proceed to exploit the known vulnerabilities. This is achieved by executing malicious commands through API requests directed at the compromised Kube-apiserver (2). In order to target the exposed servers, adversaries first need to attain the necessary information, including the open ports of the targeted servers. This can be achieved by scanning or searching for open ports on the targeted servers. For example, adversaries may scan all open ports of the targeted servers or the whole network address of targeted organizations using port scanning tools like nmap [23]. Afterward, adversaries can visit the ports via HTTP to gather HTTP information to determine whether the services of open ports are Kubernetes or not.

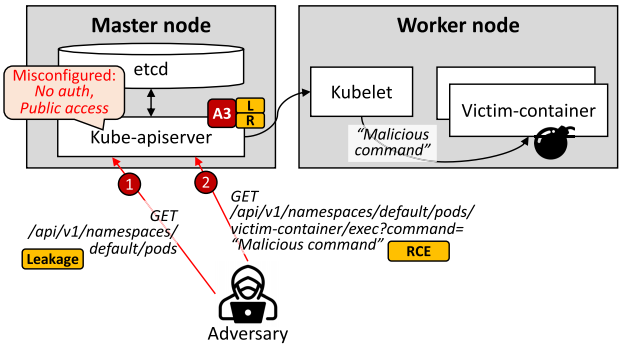


FIGURE 2. An example of REST API attack.

III. MEASUREMENT METHODOLOGY

This section describes the preliminary steps undertaken to measure MCC along with the methodology of data collection, the approach to component identification, and the ethical considerations that must be taken into account.

A. DATA COLLECTION

Shodan [24] is a search engine that enables its users to conduct searches for diverse categories of internet-connected servers. They provide an API which helps access to the comprehensive Shodan dataset. This API enables users to selectively obtain specific information aimed at their requirements from the data stored within the Shodan. Utilizing the Shodan API, we systematically collect data regarding externally exposed components. Specifically, we gather information on open ports, HTTP headers, HTTP content, and various protocols. By processing this data, we do not need to access the servers directly. In our comprehensive dataset, 1,003,947 IP addresses are compiled, and we can classify them through the distinct signatures of each component type. The data we collect reflects the state of the server on January 03, 2023. Table 2 shows the overview of our collected data.

B. COMPONENT SIGNATURES

Our approach is grounded in the observation that each container component exhibits distinct characteristics. For instance, the Kube-apiserver and Kubelet components generate different HTTP messages. We leverage three key features: i) HTTP response headers, ii) HTTP content, and iii) HTTP/HTTPS usage. By extracting unique signatures from these features, we can accurately identify each compo-

```
x-kubernetes-pf-prioritylevel-uid: 4239a9a0-xxxx-xxxx-1781b3bc331d
x-kubernetes-pf-flowschema-uid: bbe7ecf3-xxxx-xxxx-2cfcc0e2fd1f
```

(a) Examples of *x-kubernetes* HTTP response headers

```
Docker-Distribution-Api-Version: registry/2.0
```

(b) An example of *Docker-Distribution-Api-Version* HTTP response header

FIGURE 3. Examples of HTTP response headers of server signatures. Some of *x-kubernetes* headers are redacted.

TABLE 1. Allowed protocols for each server.

Protocol	Components
HTTP	etcd, Docker Daemon, Docker Registry API
HTTPS	Kube-apiserver, Kubelet

nent. However, to adhere to ethical considerations outlined in Section III-C, we avoid directly accessing these components. Instead, we utilize the Shodan API, which provides both HTTP header and content information for each component, allowing us to identify them without direct interaction. The rationale behind each feature is as follows:

1) HTTP RESPONSE HEADERS

The Kubernetes API server, Kube-apiserver, includes two additional HTTP response headers: *x-kubernetes-pf-flowschema-uid* and *x-kubernetes-pf-prioritylevel-uid*. These headers provide details about the flow schema that matched the request and the priority level assigned to it, respectively. In the case of the Docker Registry API, the HTTP response includes the *Docker-Distribution-Api-Version* header, indicating the API version of the Docker Registry API component. Figure 3 illustrates how these HTTP headers appear in practice.

2) HTTP CONTENT

Exposed components exhibit distinct HTTP content depending on the specific component. For example, the Kube-apiserver returns results in JSON format, which include keys such as *kind*, *apiVersion*, and *metadata*, among others. In contrast, when accessing the Kubelet server through HTTP, it responds with the message: "*The client has sent an HTTP request to an HTTPS server*". Similarly, the etcd component responds with a *404 page not found* page, while the Docker daemon returns a JSON format containing only a single key: *message*. Lastly, when accessing the Docker Registry API, it returns a blank page with an HTTP status code of 200.

3) HTTP/HTTPS USAGE

Although each component is exposed to the web, each uses two different protocols unique to it: HTTP and HTTPS. For instance, while the kube-apiserver permits HTTPS protocol to access, it does not provide access through HTTP protocol.

TABLE 2. Distribution of MCCs. The data collected represents the status of the server as of January 3, 2023.

MCC	Default Ports	Open Ports	# of Results
Kube-apiserver	6443	443, 6443, ...	744,612
Kubelet	10250	10250	252,326
etcd	2379	2379	3,465
Docker Daemon	2375	2375, 443,	844
Docker Registry API	5000	5000, 49153, ...	2,680

Table 1 summarizes the protocols that are available for each component.

C. ETHICAL CONSIDERATION

In this paper, our measurement is divided into two parts: (i) measuring MCCs at Internet scale (Section IV) and (ii) measuring vulnerable servers in real-world case (Section V).

For the first part, we aim to identify the number of exposed servers on the Internet. However, conducting direct scans on unauthorized servers can raise ethical concerns, as it may lead to the collection of more information than is necessary. To address this, we rely on the Shodan API to gather only the required information, such as open ports and HTTP headers, without directly interacting with the servers. Importantly, we do not exfiltrate any data or compromise personal privacy during our measurements. This approach allows us to identify server types without direct access. In other words, we avoid accessing or collecting sensitive data and do not engage in any intrusive actions during our Internet-scale measurement.

In the latter case, to gain a deeper understanding of exposed Kubernetes threats, we conducted our experiments within an institutional environment. To evaluate vulnerabilities, we collaborated with the institution's network and information security teams. An official document was issued to these teams to support our study and ensure its safety. We adhered to their guidelines and obtained formal permission before proceeding with our assessments.

Throughout our experiments, we ensured that no privacy was breached and no sensitive data was compromised. Our objective was solely to confirm the presence of vulnerabilities without impacting any services. For example, we used non-intrusive commands, such as the `date` command in Linux, to verify that remote code execution was possible without affecting server operations. At the conclusion of our study, we disclosed our findings to the relevant teams via email, resulting in all identified vulnerabilities being patched. Consequently, no harm was done in any of the cases.

IV. INTERNET SCALE MEASUREMENT

This section discusses our findings regarding worldwide exposed Kubernetes and Docker components. Our measurement uncovers how many they are, what they are, and who they are. We also discover the attributes of the identified servers.

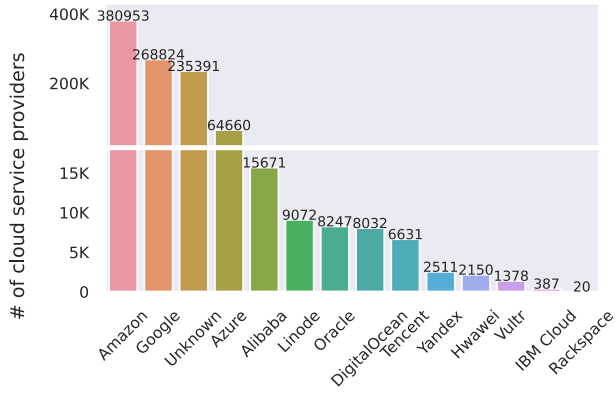


FIGURE 4. The number of cloud service providers running MCCs.

A. EXPOSURE FROM MISCONFIGURATION

The empirical evidence presented in this section quantitatively demonstrates the prevalence of a significant number of components exposed to the attacks discussed in Section II-C.

1) DISTRIBUTION OF MCCS

We identified a total of 1,003,947 MCCs, with their distribution summarized in Table 2. Of these, approximately three-quarters are Kube-apiserver instances, followed by Kubelet instances. Specifically, we found 1,000,403 exposed Kubernetes components, consisting of 744,612 Kube-apiserver, 252,326 Kubelet, and 3,465 etcd components. For Docker, we identified 2,680 exposed Docker Registry HTTP APIs and 844 Docker daemons.

2) MCCS IN PUBLIC CLOUDS

To identify the cloud services that have MCCs, we utilize a dataset of public IP addresses [25], [26], [27]. Consistent with cloud service market share, AWS occupies the highest proportion about 38%, followed by Google and Azure. Approximately 6.5% of the exposed servers cannot be determined to a specific cloud service provider. Figure 4 illustrates the distribution of servers across various cloud service providers. This result highlights that MCCs are pervasive in public clouds, allowing adversaries to target a large amount of exposed hosts.

3) DEFAULT PORT USAGE

Each container component has its own default service ports. For example, Kube-apiserver and Kubelet typically operate on ports 6443 and 10250, respectively. These defaults can be manually altered during Kubernetes initialization, such as by using the `-secure-port` flag when starting the Kube-apiserver. However, we observe that administrators rely on default ports in the wild. The "Open Ports" column in Table 2 lists the actual ports identified in our findings for each server component. For instance, the predominant port used by Kube-apiserver is port 443, accounting for approximately 89.3% of the observed instances, followed by ports 6443 (7.7%) and 10250 (2.7%), with other observed ports including 8443 (0.1%), 31337 (0.05%), and 7443 (0.04%). Similarly, for Docker, the default port 2375 has the highest proportion of usage, observed in 91.1% of instances, followed by ports 4243 (3.5%) and 2376 (1.7%). The remaining 3.8% of Docker instances utilize other ports, such as 5000 (0.95%), 5555 (0.83%), and 80 (0.6%).


```
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "forbidden: User \"system:anonymous\" cannot get path \"/\"",
  "reason": "Forbidden",
  "details": {
  },
  "code": 403
}
```


(a) The HTTP content of a server with anonymous access allowed


```
{
  "kind": "Status",
  "apiVersion": "v1",
  "metadata": {

  },
  "status": "Failure",
  "message": "Unauthorized",
  "reason": "Unauthorized",
  "code": 401
}
```


(b) The HTTP content of a server where anonymous access is *not* allowed

FIGURE 5. Difference in HTTP content between a server with anonymous access allowed and one with anonymous access not allowed.

4) ANONYMOUS ACCESS IN KUBERNETES

Kubernetes handles unauthenticated requests as anonymous, assigning them the username *system:anonymous*. If this user is granted access to $\star$, it can access all resources within the cluster, making it possible for an adversary to leak the entire list of pods and nodes through the REST API. Thus, detecting this misconfiguration can be as simple as checking for the presence of the *system:anonymous* string in the HTTP content, as shown in Figure 5. The HTTP content will differ depending on whether anonymous access is permitted or not.

Unfortunately, in the Kubernetes Role-Based Access Control (RBAC) model, anonymous access is enabled by default [28]. As a result, the number of Kubernetes servers allowing anonymous access is over six times higher than those that do not. Approximately 86% of servers allow anonymous access, while only 14% have this feature disabled. This disparity is due to the default configuration, which can be

TABLE 3. Top 10 exposed versions of Kube-apiservers, Docker daemon, and etcd.

	Kube-apiservers			Docker Daemon			etcd		
Rank	Version	Exposed #	Proportion	Version	Exposed #	Proportion	Version	Exposed #	Proportion
#1	1.21.14	228,882	30.74%	18.09.7	246	29.15%	3.3.12	601	17.34%
#2	1.22.15	203,615	27.35%	18.05.0	175	20.73%	3.3.8	422	12.18%
#3	1.19.16	65,183	8.75%	20.10.21	97	11.49%	3.4.15	230	6.64%
#4	1.22.13	41,216	5.54%	20.10.22	39	4.62%	3.3.11	224	6.46%
#5	1.20.15	38,057	5.11%	20.10.17	33	3.91%	3.4.3	214	6.18%
#6	1.22.16	10,362	1.39%	1.13.1	22	2.61%	3.5.4	160	4.62%
#7	1.22.12	9,626	1.29%	20.10.12	20	2.37%	3.5.0	142	4.10%
#8	1.20.11	2,937	0.39%	20.10.18	19	2.25%	3.5.5	109	3.15%
#9	1.21.5	2,082	0.28%	20.10.7	12	1.42%	3.5.1	92	2.66%
#10	1.18.20	1,877	0.25%	20.10.9	11	1.30%	3.5.2	92	2.66%

TABLE 4. List of CVEs by exposed version.

CVEs	MCC	Impact
CVE-2022-3294	Kube-apiserver	Authentication bypass
CVE-2022-3172	Kube-apiserver	URL Redirect, Information leak
CVE-2019-5736	Docker daemon	Privilege escalation
CVE-2019-16884	Docker daemon	Image modification
CVE-2021-21284	Docker daemon	Privilege escalation
CVE-2018-16886	etcd	Authentication bypass
CVE-2023-32082	etcd	Information leak

mitigated by manually setting the `--anonymous-auth` flag to `false` during Kubernetes server initialization.

Finding I. We found that over 86% of MCCs operate with default configurations (e.g., ports, anonymous access), and many are hosted on major public cloud platforms, providing adversaries with opportunities to exploit these components. Although administrators can modify these settings during deployment, secure configurations are often not applied in practice. The key to reducing the risk of attack is to ensure these components are not exposed to the Internet.

B. DISTRIBUTION OF EXPOSED VERSIONS

Identifying the software version of a target is crucial for adversaries, as it enables them to pinpoint exploitable vulnerabilities. An attack that leverages a known vulnerability in a specific software version is referred to as a "1-day attack."

1) FEASIBILITY OF 1-DAY ATTACKS

To evaluate the feasibility of such attacks in MCCs, we analyzed the distribution of exposed Kubernetes and Docker versions. Table 4 summarizes major CVEs affecting popular container components such as Kube-apiserver, Docker Daemon, and etcd. To determine if software versions are a major factor, we conducted an in-depth analysis of exposed versions for each MCC. Table 3 lists the top 10 most exposed versions for each MCC.

2) EXPOSED VERSIONS IN KUBE-APISERVERS

The *Kube-apiservers* column in Table 3 presents the top 10 most exposed Kubernetes versions. The latest release as of January 2023 is v1.25 [29], which was released in December 2022. However, our dataset reveals that the most commonly operating versions are v1.21.14 and v1.22.15, both of which were released before October 2022 and together account for over 50% of all observed instances. Notably, v1.21.14, the most widely used version, is vulnerable to CVE-2022-3294 and CVE-2022-3172, while v1.22.15, the second most prevalent version, also has known vulnerabilities, including CVE-2022-3294 [9], [30]. By exploiting CVE-2022-3294, adversaries can bypass security restrictions, redirecting requests to private networks [31]. Similarly, CVE-2022-3172 can be leveraged to perform URL redirection attacks, potentially resulting in credential theft or leakage of sensitive information [32].

3) EXPOSED VERSIONS IN DOCKER DAEMON

The *Docker Daemon* column in Table 3 shows the top 10 most exposed Docker daemon versions. The latest version as of January 2023 is v20.10.22 [33], released in December 2022. However, our dataset indicates that the most commonly used versions are v18.09.7 and v18.05.0, both of which were released before October 2022 and are known to have security vulnerabilities. Specifically, these versions are susceptible to several CVEs. For instance, by exploiting CVE-2019-5736, adversaries can achieve privilege escalation by overwriting the host `runc` binary, allowing an adversary with access to the host file system to gain root privileges [34]. Similarly, CVE-2019-16884 allows adversaries to modify Docker images without detection [35]. Additionally, exploiting CVE-2021-21284 enables privilege escalation within Docker instances running with the *–userns-remap* option enabled [36].

4) EXPOSED VERSIONS IN ETCD

The *etcd* column in Table 3 lists the top 10 most exposed etcd versions. The latest release as of January 2023 is v3.5.7, which was released in December 2022 [37]. However, our dataset shows that the most commonly operating versions are v3.3.12 and v3.3.8, both released prior to October 2022. These versions are vulnerable to several CVEs, such as CVE-2018-16886 [38] and CVE-2023-32082 [39]. By exploiting CVE-2018-16886, adversaries can bypass the authentication mechanism [38], while CVE-2023-32082 allows unauthorized access to key names even when read permissions are not granted [39].

TABLE 5. Reverse DNS lookup results. The abbreviation **Gov.** denotes governmental organizations, **Edu.** represents educational institutions, **F500** refers to Fortune 500 companies, and **Orgs.** signifies other organizations.

MCC	Gov. *(.gov, .go.xx)*	Edu. *(.edu, .ac.xx)*	F500 Companies	Orgs. *(.org)*
Kube-apiserver	11	72	7	283
Kubelet	9	284	26	993
etcd	1	4	0	10
Docker Daemon	0	0	0	1
Docker API	0	12	3	46
Total	21	372	36	1,333

Finding II. We found that MCCs are susceptible to known vulnerabilities, often referred to as "1-day attacks." The primary cause is that administrators frequently fail to update MCC software versions, even six months after a new release. This allows adversaries to exploit known CVEs to escalate privileges and access sensitive data.

C. MCCS FROM KNOWN ORGANIZATIONS

We now analyze which entities are operating the MCCs. For this, we aim to reveal their domain names by resolving IP addresses of the identified MCCs.

1) OBTAINING DOMAIN NAMES

The acquisition of special domain names is restricted to a specific group; therefore, identifying a particular domain can serve as a means to determine who they are. For example, the `.gov` domain can only be used by governmental institutes. For this, we examine the methodology of retrieving DNS information based on an IP address. Our proposed approach involves utilizing the PTR record type to query the reverse DNS database of the Internet. By leveraging this reverse DNS database, IP addresses can be mapped to their corresponding domain names. We use `getaddrinfo` and `gethostbyaddr` functions from Python `socket` library [40] to perform this process. We also use ipinfo [27] service to obtain DNS information for an IP address. Through these methods, a total of 748,674 IP addresses are successfully converted out of the collected IP. We categorize the domains that are operating MCCs into three groups: (i) governments, (ii) educational institutions, (iii) enterprises, and (iv) organizations. The results are summarized in Table 5.

2) MCCS IN GOVERNMENTS

Government web pages often use domains ending in `.gov` or `gov.xx`. Our investigation uncovered a total of 21 government-related MCCs exposed on the Internet across 15 institutions. These components span various government servers, including municipal websites, ministry portals, regional government sites, and climatology websites. Since these servers are frequently part of confidential networks, administrators must exercise special care when managing MCCs to mitigate potential security risks.

3) MCCS IN EDUCATIONAL INSTITUTIONS

In the case of educational institutions, such as universities or schools, commonly use domain names that end with either `.edu` or `.ac.xx`. As research institutes or research groups are often affiliated with universities, they frequently have this type of domain names. In a total of 38 educational institutes, 284 MCCs have been identified, each of which is exposed to the Internet. These institutions encompass a broad range of academic institutions, including Ivy League universities, national universities, Institute of Technology, and supercomputer research institutes.

4) MCCS IN ENTERPRISES

Enterprises are also vulnerable to the risks associated with misconfigured servers. A notable example is Tesla's incident [41], where hackers exploited their AWS instances for cryptojacking. To identify companies running MCCs, we compiled a list of Fortune 500 companies and their domain names [42]. The Fortune 500 is an annual ranking published by Fortune magazine, listing the top 500 largest U.S. corporations based on total revenue for their respective financial years [43]. Our analysis revealed that four enterprises are operating a total of 36 misconfigured servers. These companies are large corporations listed on the NYSE and NASDAQ stock exchanges, each with a market capitalization in the multi-billion dollar range. Additionally, we found one company exposing both development and storage containers to the Internet.

5) MCCS IN ORGANIZATIONS

The use of the `.org` domain encompasses a wide range of organizations, including sole proprietors, small businesses, large corporations, and nonprofit organizations, and even for personal purposes such as running a personal blog. Our investigation has revealed that organizations using this domain operate a total of 1,333 MCCs.

Finding III. Even widely recognized web domains, including 21 government-related entities, 284 educational institutions, and 36 leading enterprises, are operating MCCs. As these containers likely store sensitive user information, such as authentication data, attacks on these MCCs could result in significant real-world consequences.

D. BLOCKLISTED MCCS

An IP blocklist is a list of IP addresses that have been flagged as sources of spam, malware, or other malicious activities. These lists are used to facilitate collaboration and knowledge-sharing among security experts. For example, Computer Emergency Response Teams (CERTs) use IP blocklists to enhance their incident response capabilities and strengthen their cybersecurity posture. In this context, we aim to leverage this knowledge to determine whether the identified MCCs are associated with any IP blocklists.

1) ANALYZING BLOCKLISTED IP ADDRESSES

We collected a total of 3,380,903 IP addresses associated with malicious activities from publicly available IP blocklists [44], [45]. We then examined whether any MCC IP addresses matched those on these blocklists. Our analysis revealed that 1,071 MCC IP addresses were listed on the blocklists, indicating potential links to servers that may have been compromised and exploited in cyberattacks, such as serving as command-and-control (C2) botnets or being utilized by cyber threat actors. Notably, Kubelet was the most frequently listed component, appearing in 707 instances (69.4%), followed by Kube-apiserver (24.9%), Docker Registry API (2.5%), and Docker (2.4%). Given that Kubelets offer vectors for remote code execution (RCE), their frequent appearance on the blocklist suggests they may have been leveraged as entry points to compromise additional servers.

> ***Finding IV***. The blocklists include a total of over 1,000 MCCs. These components are of significant concern as they may have already been compromised or exploited in cyberattacks. This observation is supported by the fact that there are a higher number of Kubelet components serving as RCE vectors compared to Kube-apiserver.

V. REAL-WORLD CASE STUDY

Although MCCs are acknowledged as susceptible to attacks, the presence of actual vulnerabilities within MCCs warrants further investigation. Therefore, we engaged with organizations operating container environments and conducted empirical research to assess the real-world vulnerability of MCCs and acquire a comprehensive understanding of the identified MCC threats. In order to evaluate the potential vulnerabilities, we established a working relationship with the institution's campus network and information security teams.

The aforementioned institute is a prestigious national research and higher education institution with an extensive infrastructure including hundreds of laboratories. It serves as a hub for a significant workforce of over ten thousand members, including employees, students, and professors. To ensure the maintenance of a robust security posture, the institute is subject to periodic security audits. These audits play a vital role in evaluating and enhancing the security measures implemented within the operations.

A. ETHICAL CONSIDERATIONS

To guarantee the safety of our research, our works were supported by an official document issued to the aforementioned teams. We followed their guidelines and obtained permission from them. Throughout the duration of our experiments, we ensured that neither privacy was breached nor any sensitive data was compromised, non-invasive manner had been done. As our primary objective was only to ascertain the existence of vulnerabilities, we did not conduct any impact on the servers. For instance, by using commands unrelated to server sensitivity, such as the Linux `date` command, we were able to confirm the possibility of remote code execution.

B. SCANNING METHODOLOGY

The institute's network comprises four branches with a total of 157,082 IP addresses, encompassing dormitories, servers, and personal computers. Our scanning methodology is divided into three high-level stages: (i) identifying open ports for each IP address, (ii) determining the type of server, and (iii) verifying vulnerabilities and potential attack scenarios.

1) STAGE I: PORT SCANNING

To assess the online status and accessibility of endpoints on the Internet, we utilize `zmap` [46], a high-speed, single-packet scanning tool that can efficiently cover large IP address ranges. Our scanning methodology targets the default ports of each component, as well as additional port numbers identified in Section IV-A. To reduce the risk of false negatives, we perform multiple scans across various time zones, including both morning and evening hours, and on different days of the week.

2) STAGE II: SERVER IDENTIFICATION

Not all servers with open ports correspond to container components, so we must ensure that our scanning results are within the defined scope. To achieve this, we use the server signatures described in Section III-B to accurately identify and classify each component. For instance, port 443 may be used for hosting an HTTPS web server, and port 5000 could serve as a network-attached storage (NAS) server rather than a Docker Registry API server. This method helps eliminate false positives from our results.

3) STAGE III: VULNERABILITY ASSESSMENT

After identifying and classifying the servers, we assess their vulnerabilities by making HTTP REST requests to determine if the attack scenarios outlined in Section II-B are feasible. For example, we send specific payloads to exposed Kubelet instances to disclose a list of pods (i.e., sets of containers) or execute remote commands. Each component is tested with tailored payloads to accurately verify potential vulnerabilities.

C. DISCOVERED VULNERABILITIES

We found that 57 MCCs within the institute are exposed, and three of them are vulnerable. Three vulnerable components are the Kubelet, Docker daemon and Docker Registry API server. Table 6 summarizes the MCCs and vulnerabilities discovered from the institute. Overall, we revealed five critical vulnerabilities (**V1**-**V5**):

TABLE 6. Summary of MCCs and vulnerabilities identified in the institute through our analysis.

MCC	Open Ports	# of MCCs	# of Vuln.	Discovered Vulnerabilities
Kube-apiserver	6443	7	1	Information Leakage (V1)
Kubelet	10250, 10255	42	1	Information Leakage (V2)
etcd	2379	6	0	-
Docker Daemon	2375	1	2	Information Leakage (V3), Remote Code Execution (V4)
Docker Registry API	5000	1	1	Information Leakage (V5)
Total	-	57	5	-

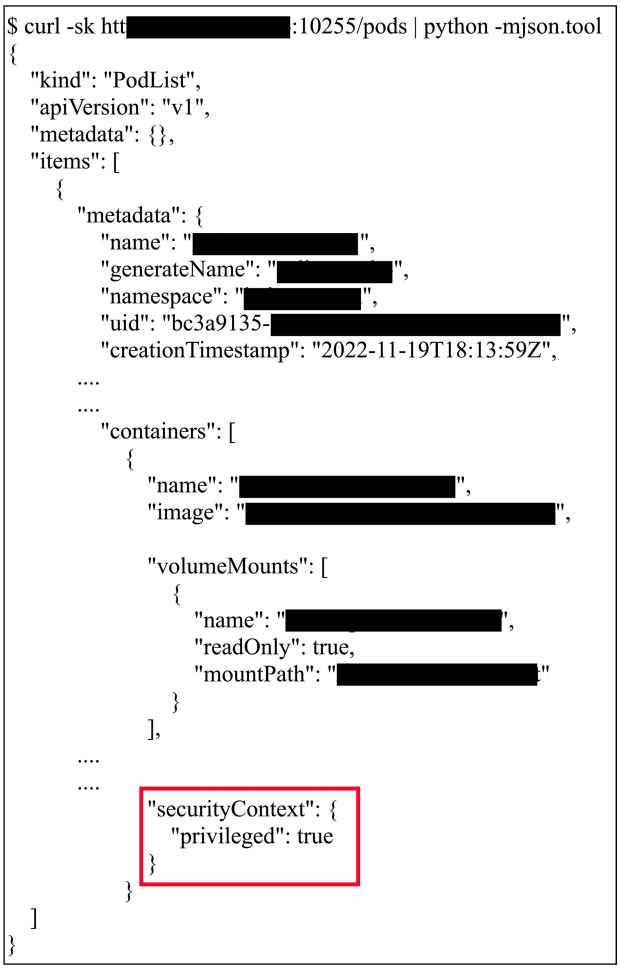
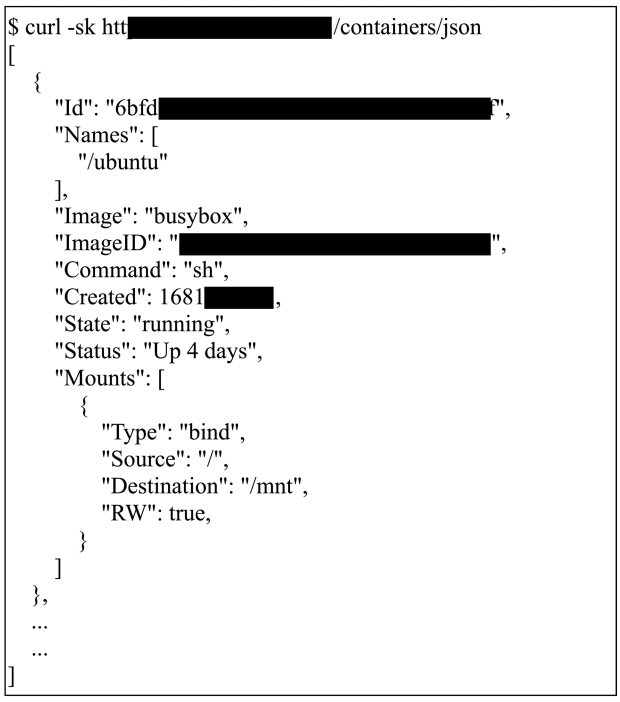
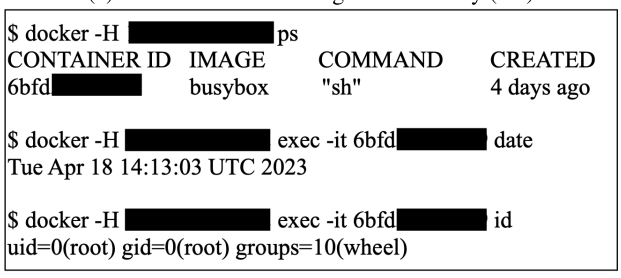


FIGURE 6. The information leakage vulnerability (V2) in the Kubelet MCC. Note that the privileged mode is turned on as highlighted by the red box. Some sensitive information has been redacted.



(a) The information leakage vulnerability (V3)



(b) The remote code execution vulnerability (V4)

FIGURE 7. Vulnerabilities in the Docker daemon MCC, with some sensitive information redacted.

1) V1. INFORMATION LEAKAGE IN KUBE-APISERVER

We identified seven exposed Kube-apiserver components, none of which were found to be directly vulnerable. However, four of these instances permit anonymous access (Section IV-A), enabling us to extract their versions: v1.17.17, v1.25.3, and v1.24.8 (with v1.24.8 appearing twice). Adversaries may utilize this knowledge to exploit vulnerabilities that are associated specific software versions.

2) V2. INFORMATION LEAKAGE IN KUBELET

Among 42 Kubelet servers, we identified one vulnerable server that allows sensitive information to be leaked, including container names, image versions, environment variables, pod internal IP addresses, and currently running applications. Figure 6 shows the outcome of exploiting this vulnerability. Several critical attributes were exposed by sending a request

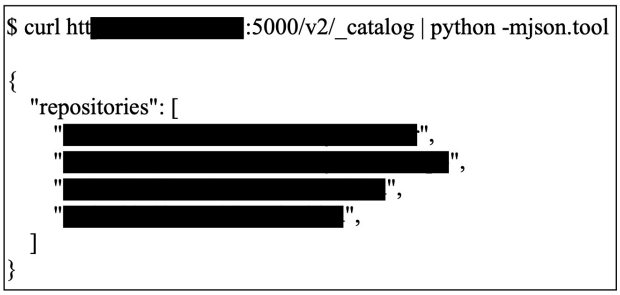


FIGURE 8. The information leakage vulnerability (V5) in the Docker Registry API MCC. Some sensitive information has been redacted.

to `/pods` on the Kubelet server. Additionally, one of the identified containers was found to be running in *privileged mode*, which poses a severe security risk. Adversaries could potentially exploit this container to execute commands with root privileges, compromising the security and integrity of the entire system.

3) V3. INFORMATION LEAKAGE IN DOCKER DAEMON

We identified one vulnerable Docker daemon that allowed both container list disclosure and remote code execution due to misconfigured authentication. Figure 7a illustrates the results of exploiting this Docker daemon via the HTTP REST API. We were able to access sensitive information such as image IDs, image names, container names, and internal IP addresses. Consequently, adversaries could leverage image-specific vulnerabilities to carry out further attacks.

4) V4. REMOTE CODE EXECUTION IN DOCKER DAEMON

Figure 7b shows the attack on the same server using the docker command to extract the various information of the container. The exposed container image is busybox,[1] and the container ID begins with `bfd5`. We verified that remote code execution over the MCC is feasible and the container is running in privileged mode, allowing for the execution of commands with root permissions.

5) V5. INFORMATION LEAKAGE IN DOCKER REGISTRY API

We found one vulnerable, exposed server regarding the Docker Registry API server. The registry instance contains multiple repositories. Figure 8 shows their list can be accessed through the `/v2/_catalog`. Adversaries may exploit the DELETE method to delete an existing repository and then use the PUT method to upload a new repository and image with the same name, thus promoting the deployment of malicious images.

6) RESPONSIBLE DISCLOSURE

Upon completing our investigation, we emailed our findings to the institute's security team. Consequently, all discovered vulnerabilities have been remediated. Notably, in every instance, no harm has been incurred.

[1] https://hub.docker.com/_/busybox

Finding V. Although not all MCCs are vulnerable, we have demonstrated that some exhibit critical security flaws, allowing adversaries to gain access to sensitive information and execute arbitrary code by targeting these components. Thus, if adversaries can identify these servers from external sources, they may exploit these MCCs to infiltrate target networks.

VI. MITIGATION

We now present potential countermeasures for mitigating these threats.

A. LIMITING NETWORK EXPOSURES

Administrators should ensure that no MCCs are exposed to the Internet to protect Kubernetes components, such as Kube-apiserver, Kubelet, or etcd, from unauthorized access. To further reduce the risk of unauthorized access and security breaches, administrators should establish network security measures, such as implementing firewall rules that permit access only from trusted networks. Additionally, deploying container components in an air-gapped environment or within a DMZ can prevent adversaries from infiltrating internal networks, even if MCCs are compromised.

B. AUTHENTICATION ENFORCEMENT

Implementing robust identity management practices is essential to strengthen access control and minimize the risk of unauthorized access. Role in RBAC should be enforced to the principle of least privilege. Every role is granted only the permissions that are necessary for its function. Also, no components should not accept or respond to anonymous or unauthenticated requests. All requests must be authenticated to prevent unauthorized access to sensitive information or critical functions.

C. FREQUENT SOFTWARE UPDATES

Regular updates help address potential security issues and minimize the attack surface. Kubernetes and Docker facilitate update mechanisms, including rolling updates [47]; however, they lack explicit guidelines regarding update frequencies. Administrators should manually ensure that applications and container platforms are maintained to the latest version by updating frequently by their own strategies. Therefore, the provision of official, well-defined security update strategies would significantly alleviate the operational burden on administrators while enhancing the overall security posture of container systems.

D. GUIDING SECURITY RECOMMENDATIONS

Kubernetes' official documentation provides best practices for secure configuration [2]. Adhering to these recommendations can help organizations strengthen the overall security posture of Kubernetes, Docker, and other container orchestration technologies. Additionally, container platforms

should display appropriate warnings—similar to context integrity warnings in the privacy domain—when sensitive configurations are enabled during deployments.

E. CLOUD MOVING TARGET DEFENSES

The static nature of MCCs—such as fixed IP addresses and ports—makes it easier for adversaries to identify and infiltrate internal networks. Implementing moving target defenses that randomize these attributes can mitigate such threats. For instance, port randomization [48] effectively thwarts attempts to detect open ports in MCCs, while IP address randomization [49] and topology obfuscation [50] complicate adversaries' efforts to gather information about the network, disrupting the reconnaissance needed for lateral movement attacks.

VII. DISCUSSION AND LIMITATIONS

This section discusses various aspects of our findings and its limitations that could be enhanced in future work.

A. BIAS IN KUBERNETES MCCS

Our investigation reveals that the number of Kubernetes MCCs is significantly higher than that of Docker instances. This disparity primarily arises due to differences in their default configurations. Kubernetes components automatically expose their web endpoints, whereas Docker requires manual configuration to expose and enable the REST API function during Docker daemon initialization. This fundamental difference in setup contributed to the observed imbalance in the number of instances between the two platforms.

B. IMPLICATIONS IN ORGANIZATION DOMAINS

Additionally, we address misconceptions surrounding `.org` domains. While these domains are typically associated with open-source organizations, non-profits, and other institutions, they are also used by individuals for personal websites or blogs. Therefore, not all `.org` domains represent official organizations, and caution should be exercised when interpreting the implications of these domains within the scope of our research.

C. VULNERABILITIES OF UNEXPOSED COMPONENTS

It is important to recognize that *unexposed components*, which are not directly accessible from the Internet, can still harbor critical vulnerabilities. For instance, adversaries can exploit Kubernetes' container runtime interface (CRI-O) vulnerabilities to escape containers and execute arbitrary code as root (e.g., CVE-2022-0811 [20], CVE-2022-1708 [21]). These vulnerabilities arise not from exposed web endpoints but from inherent flaws within the component itself. Identifying these potential threats is essential to prevent adversaries from stealthily exploiting them for malicious purposes.

D. TRUSTWORTHINESS OF IP BLOCKLISTS

Our study relied on IP blocklists to identify potentially malicious IP addresses. However, we cannot ascertain that not all blocklisted IP addresses can be considered compromised. Most of these addresses might be malicious, as expected, but some could be benign IP addresses that were involved in an attack in the past and have since changed their status. This highlights the limitations of relying on blocklists for security assessments, as they may lead to false positives.

VIII. RELATED WORK

This section reviews existing studies related to our work, which can be categorized into two areas: (i) container security and (ii) vulnerabilities resulting from misconfigurations. Table 7 provides a comparison between these studies and ours. *Note that our study is the first to conduct an Internet-wide measurement to discover MCCs in the wild and identify critical vulnerabilities through a real-world case study.*

A. CONTAINER SECURITY

As containers have become a crucial component of cloud systems, there is a wealth of literature that highlights their vulnerabilities and security threats [51], [52], [54]. For example, Duarte and Antunes [51] systematically categorized the vulnerabilities of Docker-based on their causes and consequences and demonstrated the importance of static code analysis in the Docker codebase. In contrast, Brady et al. [52] explored a continuous integration and deployment system that tests the security of Docker containers, particularly in cloud computing scenarios, using dynamic analysis to complement the limitations of static analysis security assessment. From a broader perspective, Sultan et al. [54] studied vulnerabilities across the Docker ecosystem and not only considered security threats and vulnerabilities of containers but also presented real-world scenarios illustrating how Docker can be compromised. Aside from the literature on container vulnerabilities, other researchers have investigated network-side attacks on container systems [55], [56], [57], [60], [61]. Li et al. [55] conducted a survey of both academic and industrial literature to systemize security challenges related to serverless computing security, identifying a gap between academic and industrial explanations. As a defense method, Nam et al. [56] analyzed the network side of container security and proposed a security-enforcing network stack, BASTION, which can mitigate attacks against container networks and improve container performance simultaneously. HardWhale [57] further enhances the security of container networking by employing a programmable hardware NIC. It offloads the entire container network security stack to the hardware NIC and creates an isolation layer between the NIC and the host operating system. This ensures the effectiveness of the offloaded network security stack even under host-compromised scenarios.

TABLE 7. Comparison of previous studies and our work (CNI: Container Network Interface, MCC: Misconfigured Container Component, ● denotes fulfillment, and ○ denotes non-compliance).

Work	Focus	Targeting Containers	Internet-wide Measurement	Real-world Case Study
Duate et al. [51]	Docker vulnerabilities	●	○	●
Bardy et al. [52]	Docker vulnerabilities	●	○	●
Haq et al. [53]	Docker vulnerabilities	●	○	●
Sultan et al. [54]	Overall container security	●	○	○
Li et al. [55]	Overall serverless security	○	○	○
Nam et al. [56]	CNI vulnerabilities	●	○	●
You et al. [57]	CNI vulnerabilities	●	○	●
Cui et al. [58]	Misconfigured embedded devices	○	●	●
Karl et al. [59]	Misconfigured web endpoints	○	●	●
Rahman et al. [10]	Misconfigured Kubenetes componetns	●	○	●
Our work	MCCs and vulnerabilities	●	●	●

B. VULNERABILITIES FROM MISCONFIGURATION

In addition to container security, our study is based on the measurement of misconfigured or exposed systems in the wild. Cui and Stolfo [58] conducted a pioneering study that quantitatively measured the number of vulnerable embedded devices on the worldwide web. Their work drew attention to the practice of setting embedded devices with default authentication by identifying over 500,000 exposed embedded devices across 144 countries. More recently, Karl et al. [59] investigated misconfigured authentication of web endpoints such as Gitlab, WordPress, and Jupyter. Although Kubernetes was among the 25 objects considered in their work, they lacked a study focused on Kubernetes, including an analysis of exposed organizations and corporations missing authentication. Rahman et al. [10] specifically focused on Kubernetes systems and identified Kubernetes components that could be affected by security misconfigurations. Their work also systematized the misconfigurations in Kubernetes manifests. Thanks to this systematization, we can address the problem of identifying misconfigured Kubernetes administrators and their locations.

IX. CONCLUSION

In this paper, we conducted a thorough investigation into the security risks of misconfigured container components (MCCs) in Docker and Kubernetes, focusing on their prevalence and vulnerabilities. Using the Shodan API, we collected a dataset of over a million publicly accessible IP addresses to measure MCC exposure. Our findings revealed that numerous MCCs used default settings and outdated software, making them prone to 1-day attacks. We also found these MCCs frequently on institutional servers and some of MCCs were already blocklisted which indicates existing compromises. A real-world case study further demonstrated the presence of vulnerabilities in the MCCs, identifying five critical vulnerabilities within the campus network.

ACKNOWLEDGMENT

(Dongmin Choi and Hyunmin Seo contributed equally to this work.)

REFERENCES

[1] docker. (2024). *Docker.com*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.docker.com/

[2] kubernetes. (2024). *Kubernetes.io*. Accessed: Sep. 29, 2024. [Online]. Available: https://kubernetes.io/

[3] M. A. Rodriguez and R. Buyya, "Container-based cluster orchestration systems: A taxonomy and future directions," *Softw., Pract. Exper.*, vol. 49, no. 5, pp. 698–719, May 2019.

[4] C. Pahl, "Containerization and the PaaS cloud," *IEEE Cloud Comput.*, vol. 2, no. 3, pp. 24–31, May 2015.

[5] R. Kumar and M. C. Trivedi, "Networking analysis and performance comparison of kubernetes CNI plugins," in *Advances in Computer, Communication and Computational Sciences: Proceedings of IC4S 2019*. Springer, 2021, pp. 99–109.

[6] A. Malviya and R. K. Dwivedi, "A comparative analysis of container orchestration tools in cloud computing," in *Proc. 9th Int. Conf. Comput. Sustain. Global Develop. (INDIACom)*, Mar. 2022, pp. 698–703.

[7] V. Atlidakis, P. Godefroid, and M. Polishchuk, "Checking security properties of cloud service REST Apis," in *Proc. IEEE 13th Int. Conf. Softw. Test., Validation Verification (ICST)*, Oct. 2020, pp. 387–397.

[8] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem–vulnerability analysis," *Comput. Commun.*, vol. 122, pp. 30–43, Jun. 2018.

[9] nvd.nist.gov. (2022). *CVE-2022-3172*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-3172

[10] 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, Oct. 2023.

[11] Amazon EKS. *Control Network Access To Cluster API Server Endpoint*. Accessed: Sep. 29, 2024. [Online]. Available: https://docs.aws.amazon.com/eks/latest/userguide/cluster-endpoint.html

[12] M. Katchinskiy and A. Morag. (2023). *Kubernetes Exposed: One Yaml Away From Disaster*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.aquasec.com/blog/kubernetes-exposed-one-yaml-away-from-disaster/

[13] Infosecurity Mag. (2022). *Nearly One Million Exposed Misconfigured Kubernetes Instances Could Cause Breaches*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.infosecurity-magazine.com/news/misconfigured-kubernetes-exposed/

[14] Infosecurity Mag. (2018). *The Tesla Hack is a Serious Cryptojacking Warning*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.infosecurity-magazine.com/opinions/tesla-hack-cryptojacking-warning/

[15] Arcserve. (2023). *7 Most Infamous Cloud Security Breaches*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.arcserve.com/blog/7-most-infamous-cloud-security-breaches

[16] Y. Avrahami and S. B. Hai. (2022). *Kubernetes Privilege Escalation: Container Escape == Cluster Admin?*. Accessed: [Online]. Available: https://www.blackhat.com/us-22/briefings/schedule/index.html#kubernetes-privilege-escalation-container-escape–cluster-admin-26344

[17] Spiceworks. (2023). *How Vulnerabilities in Kubernetes Are Potential Attack Vectors*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.spiceworks.com/it-security/vulnerability-management/articles/kubernetes-vulnerabilities-as-attack-vectors/

[18] Kubernetes. (2024). *Nodes*. Accessed: Sep. 29, 2024. [Online]. Available: https://kubernetes.io/docs/concepts/architecture/nodes/

[19] Kubernetes. (2024). *Kubernetes Components*. Accessed: Sep. 29, 2024. [Online]. Available: https://kubernetes.io/docs/concepts/overview/components/

[20] nvd.nist.gov. (2022). *CVE-2022-0811*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2022-0811

[21] nvd.nist.gov. (2023). *CVE-2022-1708*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/cve-2022-1708

[22] man7. (2023). *Curl*. Accessed: May 22, 2023. [Online]. Available: https://man7.org/linux/man-pages/man1/curl.1.html

[23] nmap. (2024). *Nmap.org*. Accessed: Sep. 29, 2024. [Online]. Available: https://nmap.org/

[24] shodan. (2023). *Shodan.io*. Accessed: May 22, 2023. [Online]. Available: https://developer.shodan.io/

[25] github. (2024). *Github.com*. Accessed: May 18, 2023. [Online]. Available: https://github.com/nccgroup/cloud_ip_ranges/

[26] github. (2023). *Cloud IP Ranges*. Accessed: Sep. 29, 2024. [Online]. Available: https://github.com/femueller/cloud-ip-ranges/

[27] (2024). *IPinfo.io: Trusted IP Data Provider, From IPv6 To IPv4*. Accessed: Sep. 29, 2024. [Online]. Available: https://ipinfo.io/

[28] kubernetes.io. (2024). *Authenticating*. Accessed: Sep. 29, 2024. [Online]. Available: https://kubernetes.io/docs/reference/access-authn-authz/authentication/

[29] kubernetes.io. (2023). *Docs.kubernetes.io*. Accessed: Sep. 29, 2024. [Online]. Available: https://v1-25.docs.kubernetes.io/

[30] nvd.nist.gov. (2024). *CVE-2022-3294*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-3294

[31] github.co. (2023). *CVE-2022-329*. Accessed: Sep. 29, 2024. [Online]. Available: https://github.com/kubernetes/kubernetes/issues/113757

[32] paloaltonetworks.com. (2022). *New Vulnerability in Kubernetes CVE-2022-3172*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.paloaltonetworks.com/blog/prisma-cloud/new_vulnerability_in_kubernetes_cve-2022-3172/

[33] docker.com. (2023). *Docs.docker.com*. Accessed: Sep. 29, 2024. [Online]. Available: https://docs.docker.com/engine/release-notes/20.10/#201022

[34] nvd.nist.gov. (2019). *CVE-2019-5736*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-5736

[35] nvd.nist.gov. (2019). *CVE-2019-16884*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-16884

[36] nvd.nist.gov. (2021). *CVE-2021-21284*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-21284

[37] etcd.io. (2024). *Etcd.io*. Accessed: Sep. 29, 2024. [Online]. Available: https://etcd.io/docs/v3.5/

[38] nvd.nist.gov. (2018). *CVE-2018-16886*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-16886

[39] nvd.nist.gov. (2023). *CVE-2023-32082*. Accessed: Sep. 29, 2024. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-32082

[40] P. socket. (2023). *Docs.python.org*. Accessed: Sep. 29, 2024. [Online]. Available: https://docs.python.org/3/library/socket.html

[41] Wired. (2018). *Hack Brief: Hackers Enlisted Tesla's Public Cloud To Mine Cryptocurrency*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.wired.com/story/cryptojacking-tesla-amazon-cloud/

[42] Zyxware. (2024). *List of Fortune 500 Companies and Their Websites*. Accessed: Sep. 29, 2024. [Online]. Available: https://www.zyxware.com/articles/4344/list-of-fortune-500-companies-and-their-websites

[43] S. Lee, T. Hwang, and H. Lee, "Corporate blogging strategies of the fortune 500 companies," *Manage. Decis.*, vol. 44, no. 3, pp. 316–334, Mar. 2006.

[44] (2024). *IP Sum*. Accessed: Sep. 29, 2024. [Online]. Available: https://github.com/stamparm/ipsum

[45] (2024). *BlackIP*. Accessed: Sep. 29, 2024. [Online]. Available: https://github.com/maravento/blackip

[46] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *Proc. 22nd USENIX Secur. Symp. (USENIX Security)*, Aug. 2013, pp. 605–620.

[47] dockerdocs. (2023). *Dockerdocs*. [Online]. Available: https://docs.docker.com/engine/swarm/swarm-tutorial/rolling-update/

[48] S. Shin, Z. Xu, Y. Kim, and G. Gu, "CloudRand: Building heterogeneous and moving-target network interfaces," in *Proc. 27th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2018, pp. 1–11.

[49] J. Kim, E. Marin, M. Conti, and S. Shin, "EqualNet: A secure and practical defense for long-term network topology obfuscation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–18.

[50] J. Kim, J. Nam, S. Lee, V. Yegneswaran, P. Porras, and S. Shin, "BottleNet: Hiding network bottlenecks using SDN-based topology deception," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3138–3153, 2021.

[51] A. Duarte and N. Antunes, "An empirical study of Docker vulnerabilities and of static code analysis applicability," in *Proc. 8th Latin-Amer. Symp. Dependable Comput. (LADC)*, Oct. 2018, pp. 27–36.

[52] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," in *Proc. 10th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2020, pp. 0975–0980.

[53] M. Sadun Haq, T. Duc Nguyen, A. Ş. Tosun, F. Vollmer, T. Korkmaz, and A.-R. Sadeghi, "SoK: A comprehensive analysis and evaluation of Docker container attack and defense mechanisms," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 4573–4590.

[54] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52976–52996, 2019.

[55] X. Li, X. Leng, and Y. Chen, "Securing serverless computing: Challenges, solutions, and opportunities," *IEEE Netw.*, vol. 37, no. 2, pp. 166–173, Apr. 2022.

[56] J. Nam, S. Lee, H. Seo, P. Porras, V. Yegneswaran, and S. Shin, "BASTION: A security enforcement network stack for container networks.," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf. (USENIX ATC)*, Jan. 2020, pp. 81–95.

[57] M. You, J. Nam, H. Seo, M. Seo, J. Kim, D. Choi, and S. Shin, "HardWhale: A hardware-isolated network security enforcement system for cloud environments," in *Proc. IEEE 44th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2024, pp. 496–507.

[58] A. Cui and S. J. Stolfo, "A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan," in *Proc. 26th Annu. Comput. Secur. Appl. Conf. (ACSAC)*, Dec. 2010, pp. 97–106.

[59] M. Karl, M. Musch, G. Ma, M. Johns, and S. Lekies, "No keys to the kingdom required: A comprehensive investigation of missing authentication vulnerabilities in the wild," in *Proc. 22nd ACM Internet Meas. Conf.*, Oct. 2022, pp. 619–632.

[60] M. You, J. Nam, M. Seo, and S. Shin, "HELIOS: Hardware-assisted high-performance security extension for cloud networking," in *Proc. ACM Symp. Cloud Comput. (SoCC)*, Oct. 2023, pp. 486–501.

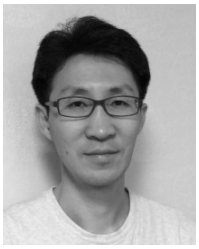
[61] M. You, M. Seo, J. Kim, S. Shin, and J. Nam, "Hyperion: Hardware-based high-performance and secure system for container networks," *IEEE Trans. Cloud Comput.*, vol. 12, no. 3, pp. 844–858, Sep. 2024.

DONGMIN CHOI received the B.S. degree from the Department of Computer Science, Chungbuk National University, and the M.S. degree in information security from KAIST. He is currently an Offensive Security Researcher. His research interests include networks and system security.

HYUNMIN SEO received the B.S. and M.S. degrees in electrical engineering from KAIST, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering. His research interests include programmable network data planes, cyberattack, and cloud security.

KWANWOO KIM received the B.S. degree in electrical engineering and physics and the M.S. degree in electrical engineering from KAIST, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering. His research interests include graph modeling and reasoning for data-driven security and cyber safety in online communities.

MYOUNGSUNG YOU received the B.S. degree in computer science from Chungbuk National University, and the M.S. degree from the Graduate School of Information Security, KAIST, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering. His research interests include programmable network data planes, cloud security, and distributed systems.



SEUNGWON SHIN (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from KAIST, and the Ph.D. degree in computer engineering from the Electrical and Computer Engineering Department, Texas A&M University. He is currently an Associate Professor with the School of Electrical Engineering, KAIST, and the Executive Vice President of Samsung Electronics. His research interests include software-defined networking security, dark web analysis, and cyber threat intelligence.

JINWOO KIM received the B.S. degree in computer science and engineering from Chungnam National University, the M.S. degree from the Graduate School of Information Security, KAIST, and the Ph.D. degree from the School of Electrical Engineering, KAIST. He is currently an Assistant Professor with the School of Software, Kwangwoon University, Seoul, South Korea. His research interests include investigating security issues with software-defined networks and cloud systems.