On Prescription or Off Prescription? An Empirical Study of Community-prescribed Security Configurations for Kubernetes

Shazibul Islam Shamim
Dept of SWEGD
Kennesaw State University
Marietta, GA, USA
mshamim@kennesaw.edu

Hanyang Hu
Team Lead
Company-A
CA, USA
phenom.hu@gmail.com

Akond Rahman
Dept. of Computer Science and Software Engineering
Auburn University
Auburn, AL, USA
akond@auburn.edu

***Abstract*—Despite being beneficial for rapid delivery of software, Kubernetes deployments can be susceptible to security attacks, which can cause serious consequences. A systematic characterization of how community-prescribed security configurations, i.e., security configurations that are recommended by security experts, can aid practitioners to secure their Kubernetes deployments. To that end, we conduct an empirical study with 53 security configurations recommended by the Center for Internet Security (CIS), 20 survey respondents, and 544 configuration files obtained from the open source software (OSS) and proprietary domains.**

From our empirical study, we observe: (i) practitioners can be unaware of prescribed security configurations as 5%∼40% of the survey respondents are unfamiliar with 16 prescribed configurations; and (ii) for Company-A and OSS respectively, 18.0% and 17.9% of the configuration files include at least one violation of prescribed configurations. From our evaluation with 5 static application security testing (SAST) tools we find (i) only Kubescape to support all of the prescribed security configuration categories; (ii) the highest observed precision to be 0.41 and 0.43 respectively, for the Company-A and OSS datasets; and (iii) the highest observed recall to be respectively, 0.53 and 0.65 for the Company-A and OSS datasets. Our findings show a disconnect between what CIS experts recommend for Kubernetes-related configurations and what happens in practice. We conclude the paper by providing recommendations for practitioners and researchers. Dataset used for the paper is publicly available online.

***Index Terms*—configuration, container orchestration, devops, devsecops, empirical study, Kubernetes, security, static analysis**

I. Introduction

Practitioners use containers to rapidly deploy software changes to end-users. In order to efficiently manage provisioned containers, practitioners use the practice of container orchestration, where popular tools, such as Kubernetes are used to manage thousands of containers [11], [23]. Usage of Kubernetes has yielded benefits for organizations. For example, practitioners at OpenAI reported that using Kubernetes, the deployment time reduced from 'couple of months' to 'two or three days' for hundred of servers [12]. As another example, in the case of Spotify, the time to run a new software service in production reduced from 'one hour' to 'seconds' [13].

Despite reported benefits, practitioners face challenges in securing their Kubernetes deployments. According to the '2024 State of Kubernetes Security Report', security of Kubernetes deployments is reported as one of the biggest concerns for adoption by practitioners [47]. The survey also reports 67% of 600 survey respondents, delayed or slowed down software deployments because of Kubernetes-related security concerns. Furthermore, 46% of the respondents in the same survey reported to experience losses in revenue or customers because of security concerns related to container orchestration [47]. Kubernetes-related security weaknesses can cause serious consequences. For example, a Kubernetes-related configuration facilitated a security attack called cryptojacking. Cryptojacking is the attack of using a computing resource to stealthily mine cryptocurrency without the user's awareness [67]. Cryptojacking can be used to incur thousands of dollars in unwanted expenditures, as it happened for the Telnet cyptojacking attack [1].

The above-mentioned discussion showcases the importance of using configurations that can enhance the security of Kubernetes deployments. In order to aid practitioners, both researchers and practitioners have provided guidelines on how to integrity security into Kubernetes deployments in forms of tools [26], [29], [44], [59] and prescribed security configurations, i.e., security configurations that are recommended by a group of cybersecurity experts [18], [51]. For example, the 'Center for Internet Security (CIS)' organization has recommended multiple security configurations that practitioners can follow [18].

Despite their availability, there is a lack of understanding on how frequently prescribed configurations are followed by practitioners. Let us consider the examples presented in Figure 1 in this regard. Figures 1a and 1b presents an example of violating a prescribed configuration, which resides in a repository maintained by Company-A and an open-source

[1]https://thenewstack.io/cryptojacking-free-money-for-attackers-huge-cloud-bill-for-you/#

Pre-print

```
containers:
   - name: efs-plugin
     securityContext:
       privileged: true
     image: amazon/aws-efs-csi-driver:v1.5.1
     imagePullPolicy: IfNotPresent
```

a

```
cni:
  cniBinDir: /var/lib/cni/bin
  cniConfDir: /etc/cni/multus/net.d
  chained: false
  cniConfFileName: "istio-cni.conf"
  excludeNamespaces:
    - istio-system
    - kube-system
  logLevel: info
  privileged: true
```

b

Fig. 1: Examples where CIS-prescribed configurations are violated in Company-A's repository (Figure 1a) and in an OSS repository (Figure 1b).

software (OSS) repository [35]. Both configuration files use `privileged: true`, which violates the recommendation of 'minimize the admission of privileged containers'. One Kubernetes expert colloquially referred to the configuration `privileged: true` as the "*the most dangerous flag in the history of computing*", as this configuration gives the illusion of containerization but in fact disables security features provided by Kubernetes for a container or a group of containers [28].

Examples presented in Figure 1 showcases that Kubernetes configuration files maintained by OSS and proprietary organizations are susceptible to include configurations that violate what is recommended by cybersecurity experts, namely from CIS. We take motivation from the examples presented in Figure 1 and conduct a systematic analysis of how frequently prescribed security configurations are violated. As part of our analysis, we also investigate what tools are capable of detecting violations of CIS-prescribed security configurations. Such an analysis, which currently remains under-explored, can aid practitioners with recommendations on how to secure their Kubernetes deployments.

We answer the following research questions:

- **RQ1 [Support]**: *How frequently do static application security testing tools support community-prescribed security configurations in Kubernetes deployments?*
- **RQ2 [Perception]**: *What are the practitioner perceptions of community-prescribed security configurations for Kubernetes deployments?*
- **RQ3 [Frequency]**: *How frequently are community-prescribed security configurations violated in Kubernetes configuration files?*
- **RQ4 [Accuracy]**: *What is the detection accuracy of static application security testing tools in detecting violations of community-prescribed security configurations in Kubernetes configuration files?*

The goals of our paper are to help (i) practitioners secure their Kubernetes deployments; and (ii) researchers identify new research avenues. We accomplish our goals by conducting a mixed-method empirical study [17] with 5 static application security testing (SAST) tools and 53 CIS-prescribed configurations. First, we compute how frequently does each of the 5 SAST tools support each of the 53 prescribed configurations. Using 188 and 356 configuration files collected respectively, from 33 OSS repositories and 2 repositories used by Company-A, we compute the detection accuracy of five SAST tools. We also conduct an online survey with practitioners from Company-A and the OSS domain to evaluate the perceptions of the 53 prescribed configurations. Datasets and scripts used in our paper is available online as a replication package [52].

Pre-print

Dataset Availability: Dataset used for the paper is publicly available online [52].

Contributions: We list our contributions as follows:

- An evaluation of how frequently prescribed security configurations are violated in Kubernetes configuration files;
- An evaluation of detection accuracy of five SAST tools for configuration files used in the OSS and the proprietary domain; and
- An evaluation of practitioner perceptions of CIS-prescribed security configurations.

II. RQ1: Support for CIS-prescribed Configurations

We first provide necessary background information in Section II-A. Next, we provide the methodology and results respectively, in Sections II-B and II-C.

A. Background

1) Kubernetes Configuration Files: Container orchestration is the practice of automatically provisioning and managing multiple containers [11]. Kubernetes is a tool to implement the practice of container orchestration [11], [23], [44]. A Kubernetes installation is colloquially referred to as a Kubernetes cluster or a Kubernetes deployment [36], [44]. In order to

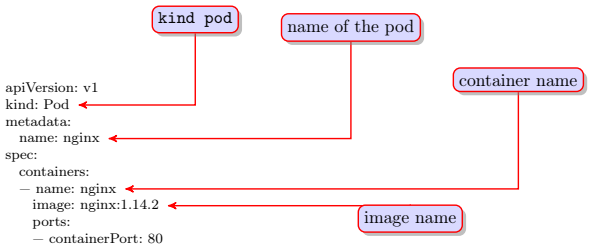

Fig. 2: An example of a Kubernetes configuration file.

manage multiple containers, practitioners develop configuration files similar to that of Figure 2. The example shows the configurations of a Kubernetes entity called 'pod' [28], which is the most fundamental deployment unit. These developed configuration files are later executed by 'kubectl', which is provided by Kubernetes. The configuration files are developed in YAML.

2) CIS-prescribed Configurations for Kubernetes: Our empirical study focuses on prescribed configurations, i.e., security configurations that are recommended by a group of cybersecurity experts. CIS is a U.S-based non-profit organization that provides security guidelines for a wide range of software systems, such as database systems and operating systems [14]. Each guideline is derived using two steps: first, experts from different backgrounds, such as academia, industry, and government, prepare a working draft of recommendations. Second, the experts discuss the draft and reach a consensus. A collection of recommendations is referred to as a guideline, where one guideline includes multiple prescriptions. Each prescription is a recommended activity to enhance software/system security.

In the case of Kubernetes, CIS provides guidelines on which configurations can secure Kubernetes-base deployments. We use CIS-prescribed configurations as our empirical study focuses on security configurations that are prescribed by a community of cybersecurity experts who specialize in Kubernetes. We use a guideline called the 'CIS Amazon EKS benchmark v1.2.0' that prescribes 53 configurations for Kubernetes deployments that use Amazon EKS. Amazon EKS is a service that can be used to run Kubernetes in the AWS cloud and on-premise data centers [50]. Each of the 53 configurations map to a category that is listed in Table I. For example, the prescribed configuration 'enable audit logs' belongs to the category 'logging'. Each of the prescribed configuration includes a name, a description of the reasoning behind the recommendation, and necessary remediation steps when the configuration is violated.

These 53 CIS-prescribed configurations are (i) adopted by the cloud-native community [2]; (ii) demonstrated through webinars [3]; and (iii) recommended by AWS for its customers [4].

[2]https://orca.security/resources/blog/kubernetes-hardening-guide/

B. Methodology for RQ1

We use the following steps:

1) Identify the Source of Community-prescribed Security Configurations: In our empirical study, we use the 'CIS Amazon Elastic Kubernetes Service (EKS) Benchmark' [18] that lists a total 53 security configurations. We use this set of recommendations because of Company-A's Kubernetes deployment pipeline uses AWS EKS. Each recommendation is mapped to a category. For example, the recommendation 'enable audit logs' is mapped to a category called 'logging'. In all, the guidebook includes 53 recommendations that are mapped to 15 categories.

2) Selection of SAST Tools: In order to answer RQ1, we identify a set of SAST tools that is capable of detecting violations of the 53 recommendations listed in the AWS EKS guide book. We start the selection process by conducting a search using the Google search engine with the search string 'security tools for kubernetes'. From the collected top 25 search results we identify 22 program analysis tools for Kubernetes. From the list of 22 tools, we further apply the following criteria to identify the tools that we plan to use.

- *Criterion-1*: The tool must be publicly available online.
- *Criterion-2*: The tool must be able to detect violations of prescribed configurations by applying static analysis with configuration files. The first author of the paper read the documentation of each tool related to Kubernetes to determine if the tool detect violations of security-related configurations.
- *Criterion-3*: The tool must be executable automatically using the command line interface. We exclude tools, such as Snyk as it requires manually uploading of Kubernetes configuration files for security-related analysis.
- *Criterion-4*: The tool must detect violations of at least five security configurations. Our assumption is that by using this criterion we will be able to exclude tools that are specialized to detect a specific type of security weakness in Kubernetes-based deployments. This criterion is consistent with prior research on security tool evaluation that has used the count of five security weakness types to determine how generalizable a tool is [32].

We use the source code and documentation of each tool to apply the filtering criteria. Upon application of the following criteria we identify five tools of which Checkov, Kubescape, and Trivy are internally used at Company-A. Attributes of these five tools are available in Table I.

Checkov is an OSS SAST tool that scans Kubernetes configuration files for security weaknesses [8]. Checkov can be installed as a package and executed from the command line interface. Checkov provides output results in SARIF format.

[3]https://www.youtube.com/watch?v=HNL6Nx48xZI

[4]https://aws.amazon.com/blogs/containers/introducing-cis-amazon-eks-benchmark/

Pre-print

TABLE I: Attributes of Selected SAST Tools

Tool	Size (KLOC)	Source	Output Format
Checkov	786.22	GitHub [8]	SARIF, JSON, XML, CSV
KubeLinter	31.03	GitHub [27]	SARIF, JSON
Kubescape	257.61	GitHub [29]	SARIF, JSON, XML, HTML, PDF
SLI-KUBE	13.86	TOSEM'23 [44]	SARIF, CSV
Trivy	514.05	GitHub [59]	SARIF, JSON, XML, HTML

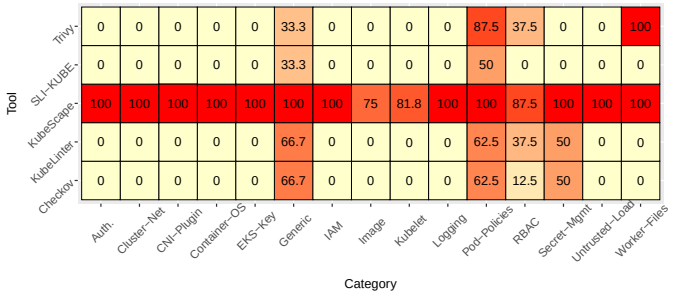

Fig. 3: Answer to RQ1: Support for CIS-prescribed configurations.

Kubescape is an OSS SAST tool developed by ARMO that supports misconfiguration scanning, risk analysis, and security compliance inside a Kubernetes deployment [29]. Kubescape can work as a SAST tool to scan source code in a local directory with a command line interface.

KubeLinter is an OSS SAST tool developed by Stackrox that identifies security misconfigurations and deviations from security best practices in Kubernetes configuration files [27]. According to the Red Hat survey 2023, KubeLinter is the most popular static security analysis tool among Kubernetes practitioners [46]. KubeLinter can analyze Kubernetes configuration files from the command line interface in the local directory.

SLI-KUBE [1] is an OSS SAST tool developed by researchers that identifies 11 categories of security misconfigurations in Kubernetes manifests [1] [44]. SLI-KUBE can be executed from the command line.

Trivy is an OSS SAST tool developed by Aqua Security that can detect security weaknesses in operating systems, language-specific packages, and Kubernetes configuration files [59]. Trivy identifies security weaknesses for known vulnerabilities(CVE), misconfigurations, and runtime security issues.

3) Map Prescribed Configurations to Rules Implemented in the Tools: Answer to RQ1 requires a systematic analysis to identify which of the recommendations are supported by which tool. We use a qualitative technique called closed coding [49] to perform a mapping between each of the 53 recommendations and a rule implemented within the tool. As part of applying the closed coding technique, the first author and the second author of the paper read each of the recommendations in the guidebook and identify if the recommendation security configuration is detected by inspecting the source code or documentation of the tool. Upon completion of the closed coding process we record a Cohen's Kappa of 0.94 between the first and second authors, which indicates 'almost perfect' agreement [30]. The disagreement occurred for rules of the tools for 6 recommendations that are resolved by the last author of the paper. The last author's decision is final in this regard. Upon completion of the closed coding process, we derive a mapping between each prescribed configuration and one or multiple rules for each tool.

4) Metric to Answer RQ1: We use the derived mapping from Section II-B3 to answer RQ1. We report which of the prescribed security configurations are detected by each tool. We also report the support by computing the proportion of supported configurations for each category using Equation 1.

$$\text{Support/Category } (c) = \frac{\text{Total \# of configurations detected by tool} * 100}{\text{Total \# of configurations in the category}} \quad (1)$$

C. Answer to RQ1

We find one recommendation to be supported by at least two of our five selected tools. A complete breakdown is available in Table II. The 'Category' column presents the category name to which each recommendation belongs to. For example, violations of 'enable audit logs' is detected by Kubescape. We also report the support of each tool based on categories in Figure 3. We observe Kubescape to have support of $>= 75.0\%$ for all 11 categories of prescribed configurations.

Pre-print

> ***Answer to RQ1****: Amongst the five SAST tools, Kubescape provides the most support with respect to detecting violations of CIS-prescribed configurations.*

III. RQ2: Practitioner Perceptions

We provide the methodology and results respectively, in Sections III-A and III-B.

A. Methodology for RQ2

We conduct a quantitative survey by recruiting practitioners from Company-A and from the OSS domain. We initially start with a 5-item Likert survey similar to that of prior research [43]. Following Kitchenham and Pfleeger's guidelines [24], we conduct a pilot survey that revealed practitioners may not be familiar with some of the prescribed configurations. Accordingly, we added another item called 'Unfamiliar' Inclusion of Likert items that express unfamiliarity is commonplace and useful in opinion-based studies [61]. A 6-item Likert survey similar to ours is also reported to have higher validity [57].

We deploy the constructed survey to practitioners working in Company-A and contributing to OSS projects. The second author of the paper provided the email addresses of 14 practitioners who are working in Company-A. In the case of OSS,

we emailed 36 practitioners. The email addresses are collected by the first author who contacted the same set of practitioners who they interviewed in prior work. All selected practitioners are confirmed to be knowledgeable about Kubernetes by the first and second author. The survey is conducted from October 2023 to July 2024.

The online survey consists of 53 questions in total. Each of the 53 remaining questions is a question asked using a six-item Likert scale. The six items are 'Unfamiliar', 'Not important', 'Little important', 'Somewhat important', 'Important', and 'Extremely important'. Each of these questions focus on assessing the practitioners' perceptions about the importance of the 53 prescribed security configurations. Following guidelines from Kitchenham and Pfleeger [24], we use a five-item Likert scale to assess importance. We also add one item called 'Unfamiliar' as from our pilot deployment, we found a voluntary participant to not be familiar with 35 of the 53 prescribed configurations. We do not collect any personal information, such as name, email, industry experience as we did not get permission from Company-A. The survey questionnaire is available as part of replication package [52].

B. Answer to RQ2

We receive responses from 20 practitioners. Of the 20 respondents, 10 work in Company-A and 10 contribute to OSS projects. Mean and maximum years of professional experience for participants from the OSS domain is respectively, 6.0 and 13.0 years. Our OSS participants reported their job titles to be 'vice-president', 'senior software engineer', 'CTO', and 'solution architects'.

Our answers to RQ2 is available in Table III. In the 'Survey Response' column, the numbers on the left hand side present the proportion of participants who are not familiar with a prescribed configuration. For example, 5% of the surveyed practitioners are not familiar with 'enabling audit logs'. The data presented on the right hand side, showcases the proportion of participants who find a configuration to be 'extremely important'. For example, 40% of the surveyed practitioners find 'enabling audit logs' as extremely important. Surveyed practitioners are not familiar with all 53 configurations. For example, for 16 configurations we found at least one respondent to be unfamiliar.

> ***Answer to RQ2****: All 20 survey respondents are familiar with 37 out of 53 CIS-prescribed configurations. For the remaining 16 configurations, 5%∼40% survey respondents are unfamiliar.*

IV. RQ3: Frequency of Violated Configurations

We provide the methodology and results respectively, in Sections IV-A and IV-B.

A. Methodology for RQ3

We use the following steps:

1) Construction of Evaluation Dataset: We use two datasets to evaluate the detection accuracy: (i) the 'Company-A' dataset, i.e., the dataset with configuration files that are provided by our partner at Company-A and (ii) OSS configuration files mined from repositories on GitHub. All configuration files provided by Company-A is obtained from two repositories for which we apply no filtering criteria. The two repositories in total includes 364 Kubernetes configuration files.

We use the GHTorrent data dump [19] hosted on Google BigQuery to mine OSS repositories. Initially, we start with 14,747,836 repositories. We apply a filtering criteria to identify repositories with Kubernetes configuration files. The criteria are: (i) at least 10% of the files in the repository must be Kubernetes manifests; (ii) the repository must be available for download; (iii) the repository is not a clone to avoid duplicates; (iv) the repository must have ≥ 2 commits per month. Munaiah et al. [38] previously used the threshold of ≥ 2 commits per month to determine which repositories have enough software development activity; (v) the repository has ≥ 5 contributors; (vi) the repository is not used for a 'toy' project. We consider a project as 'toy' project if description and content of the README file for each projects indicates that the project is used to demonstrate examples, conduct course work, and used as book chapters; and (vii) the repository is deployed using Amazon EKS. We use this criterion as the repositories used by Company-A also uses Amazon EKS. This criterion allows us to identify repositories that are consistent with Company-A's repositories that are deployed using Amazon EKS. We apply this criterion by reading each of the README files and descriptions for the 185 repositories that we identified by applying criteria (i) - (vi).

Pre-print

Upon application of this filtering criteria we identify 33 OSS repositories that contains 4,800 configuration files. We apply a 95% confidence interval to select a random sample from both datasets namely, Company-A and OSS. Using our selected random sample we identify 188 and 356 configuration files in the 'Company-A' and OSS dataset. Attributes of both datasets is available in Table IV.

2) Qualitative Analysis: Each of the configuration files in the Company-A and OSS datasets is inspected for presence of violations by two raters: the first and second authors of the paper. Each rater individually inspects each file in order to determine which file violates what security configuration. The raters apply closed coding [49] while inspecting, where they first identify coding patterns in the file that are indications of violating a prescribed configurations. Next, they map the identified coding pattern to any of the 53 prescribed configurations. If the mapping is found then that coding pattern is identified as a true positive, i.e., an instance of a valid violation. The same process is repeated for all 188 and 356 files respectively, in the Company-A and the OSS dataset. For example, the coding pattern `Audit: Disabled` is a violation of "*ensure audit logs*", whereas the coding pattern `Audit: Enabled` is in compliance for the recommendation.

TABLE II: Answer to RQ1: Support for Detecting Violations of Configurations Prescribed by CIS

Category	ID: Prescribed Configuration	Checkov	SLI-KUBE	KubeLinter	Kubescape	Trivy
Logging	2.1.1: Enable audit Logs	×	×	×	✓	×
	3.1.1: Ensure that the kubeconfig file permissions are set to 644 or more restrictive	×	×	×	✓	✓
Worker Node Configuration Files	3.1.2: Ensure that the kubelet kubeconfig file ownership is set to root:root	×	×	×	✓	✓
	3.1.3: Ensure that the kubelet configuration file has permissions set to 644 or more restrictive	×	×	×	✓	✓
	3.1.4: Ensure that the kubelet configuration file ownership is set to root:root	×	×	×	✓	✓
Kubelet	3.2.1: Ensure that the Anonymous Auth is Not Enabled	×	×	×	✓	✓
	3.2.2: Ensure that the –authorization-mode argument is not set to AlwaysAllow	×	×	×	✓	✓
	3.2.3: Ensure that a Client CA File is Configured	×	×	×	✓	✓
	3.2.4: Ensure that the –read-only-port is disabled	×	×	×	✓	✓
	3.2.5: Ensure that the –streaming-connection-idle-timeout argument is not set to 0	×	×	×	✓	✓
	3.2.6: Ensure that the –protect-kernel-defaults argument is set to true	×	×	×	✓	✓
	3.2.7: Ensure that the –make-iptables-util-chains argument is set to true	×	×	×	✓	✓
	3.2.8: Ensure that the –hostname-override argument is not set	×	×	×	✓	✓
	3.2.9: Ensure that the –eventRecordQPS argument is set to 0 or a level which ensures appropriate event capture	×	×	×	✓	✓
	3.2.10: Ensure that the –rotate-certificates argument is not present or is set to true	×	×	×	✓	✓
	3.2.11: Ensure that the RotateKubeletServerCertificate argument is set to true	×	×	×	✓	✓
Container Optimized OS	3.3.1: Prefer using a container-optimized OS when possible	×	×	×	✓	×
RBAC and Service Accounts	4.1.1: Ensure that the cluster-admin role is only used where required	×	×	×	✓	✓
	4.1.2: Minimize the access to secrets	✓	×	✓	✓	✓
	4.1.3: Minimize the wildcard use in Roles and ClusterRoles	✓	×	✓	✓	✓
	4.1.4: Minimize access to create pods	×	×	✓	✓	×
	4.1.5: Ensure that default service accounts are not actively used	✓	×	✓	✓	×
	4.1.6: Ensure that the Service Account Tokens are only mounted where necessary	✓	×	×	✓	✓
	4.1.7: Avoid use of system:masters group	×	×	×	✓	×
	4.1.8: Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster	✓	×	×	✓	✓
Pod Security Policies	4.2.1: Minimize the admission of privileged containers	✓	✓	✓	✓	✓
	4.2.2: Minimize the admission of containers wishing to share host process ID namespace	✓	✓	✓	✓	✓
	4.2.3: Minimize the admission of containers wishing to share host network namespace	✓	✓	✓	✓	✓
	4.2.4: Minimize the admission of containers wishing to share host IPC namespace	✓	✓	✓	✓	✓
	4.2.5: Minimize the admission of containers with allowPrivilegeEscalation	✓	✓	✓	✓	✓
	4.2.6: Minimize the admission of root containers	✓	×	✓	✓	✓
	4.2.7: Minimize the admission of containers with added capabilities	✓	×	×	✓	✓
	4.2.8: Minimize the admission of containers with capabilities assigned	✓	✓	×	✓	✓
CNI Plugin	4.3.1: Ensure CNI supports network policies	×	×	×	✓	×
	4.3.2: Ensure that all Namespaces have Network Policies	×	✓	✓	✓	✓
Secrets Management	4.4.1: Prefer using secrets as files over secrets as environment variables	✓	×	✓	✓	×
	4.4.2: Consider external secret storage	×	×	×	✓	×
Generic	4.6.1: Create administrative boundaries between reources using namespaces	✓	✓	✓	✓	×
	4.6.2: Apply Security Context to Your Pods and Containers	✓	✓	×	✓	✓
	4.6.3: The default namespace should not be used	✓	✓	✓	✓	✓
Image	5.1.1: Ensure Image Vulnerability Scanning using Amazon ECR image scanning or a third party provider	×	×	×	✓	×
	5.1.2: Minimize user access to Amzon ECR	×	×	×	✓	×
	5.1.3: Minimize cluster access to read-only for Amzon ECR	×	×	×	✓	×
	5.1.4: Minimize Container Registries to only those approved	×	×	×	✓	×
Identity and Access Management (IAM)	5.2.1: Prefer using dedicated EKS Service Accounts	×	×	×	✓	×
AWS EKS Key Management Service	5.3.1: Ensure Kubernetes are encrypted using Custom Master Keys (CMKs) managed in AWS KMS	✓	×	×	✓	×
Cluster Networking	5.4.1: Restrict Access to the Control Plane Endpoint	✓	×	×	✓	×
	5.4.2: Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled	✓	×	×	✓	×
	5.4.3: Ensure clusters are created with Private Nodes	×	×	×	✓	×
	5.4.4: Ensure Network Policy is Enabled and set as appropriate	✓	×	×	✓	×
	5.4.5: Encrypt traffic to HTTPS load balancers with TLS certificates	×	×	×	✓	×
Authentication & Authorization	5.5.1: Manage Kubernetes RBAC users with AWS IAM Authenticator for Kubernetes	×	×	×	✓	×
Untrusted Workload	5.6.1: Consider Fargate for running untrusted workloads	×	×	×	✓	×

Pre-print

TABLE III: Answer to RQ2: the 'Survey Response' column presents perceptions of practitioners related to the importance for each of the 53 recommendations.

ID: Prescribed Configuration	Survey Response						
2.1.1: Enable audit logs	5%						40%
3.1.1: Ensure that the kubeconfig file permissions are set to 644 or more restrictive	0%						50%
3.1.2: Ensure that the kubelet kubeconfig file ownership is set to root:root	0%						35%
3.1.3: Ensure that the kubelet configuration file has permissions set to 644 or more restrictive	0%						40%
3.1.4: Ensure that the kubelet configuration file ownership is set to root:root	0%						40%
3.2.1: Ensure that the Anonymous Auth is Not Enabled	10%						60%
3.2.2: Ensure that the –authorization-mode argument is not set to AlwaysAllow	15%						50%
3.2.3: Ensure that a Client CA File is Configured	10%						40%
3.2.4: Ensure that the –read-only-port is disabled	20%						15%
3.2.5: Ensure that the –streaming-connection-idle-timeout argument is not set to 0	35%						5%
3.2.6: Ensure that the –protect-kernel-defaults argument is set to true	35%						30%
3.2.7: Ensure that the –make-iptables-util-chains argument is set to true	35%						20%
3.2.8: Ensure that the –hostname-override argument is not set	20%						25%
3.2.9: Ensure that the –eventRecordQPS argument is set to 0 or a level which ensures appropriate event capture	40%						10%
3.2.10: Ensure that the –rotate-certificates argument is not present or is set to true	15%						25%
3.2.11: Ensure that the RotateKubeletServerCertificate argument is set to true	15%						15%
3.3.1: Prefer using a container-optimized OS when possible	0%						20%
4.1.1: Ensure that the cluster-admin role is only used where required	0%						60%
4.1.2: Minimize the access to secrets	0%						65%
4.1.3: Minimize the wildcard use in Roles and ClusterRoles	0%						30%
4.1.4: Minimize access to create pods	0%						20%
4.1.5: Ensure that default service accounts are not actively used	0%						10%
4.1.6: Ensure that the Service Account Tokens are only mounted where necessary	5%						25%
4.1.7: Avoid use of system:masters group	5%						25%
4.1.8: Limit use of the Bind, Impersonate and Escalate permissions in the Kubernetes cluster	5%						10%
4.2.1: Minimize the admission of privileged containers	10%						35%
4.2.2: Minimize the admission of containers wishing to share host process ID namespace	20%						35%
4.2.3: Minimize the admission of containers wishing to share host network namespace	20%						35%
4.2.4: Minimize the admission of containers wishing to share host IPC namespace	20%						30%
4.2.5: Minimize the admission of containers with allowPrivilegeEscalation	10%						40%
4.2.6: Minimize the admission of root containers	5%						35%
4.2.7: Minimize the admission of containers with added capabilities	15%						25%
4.2.8: Minimize the admission of containers with capabilities assigned	15%						25%
4.3.1 Ensure CNI supports network policies	15%						30%
4.3.2: Ensure that all Namespaces have Network Policies	15%						25%
4.4.1: Prefer using secrets as files over secrets as environment variables	0%						25%
4.4.2: Consider external secret storage	5%						25%
4.6.1: Create administrative boundaries between resources using namespaces	10%						30%
4.6.2: Apply Security Context to Your Pods and Containers	10%						25%
4.6.3: The default namespace should not be used	5%						20%
5.1.1: Ensure Image Vulnerability Scanning using Amazon ECR image scanning or a third party provider	0%						35%
5.1.2: Minimize user access to Amzon ECR	5%						20%
5.1.3: Minimize cluster access to read-only for Amzon ECR	5%						20%
5.1.4: Minimize Container Registries to only those approved	0%						45%
5.2.1: Prefer using dedicated EKS Service Accounts	5%						35%
5.3.1: Ensure Kubernetes are encrypted using Custom Master Keys (CMKs) managed in AWS KMS	5%						15%
5.4.1: Restrict Access to the Control Plane Endpoint	0%						35%
5.4.2: Ensure clusters are created with Private Endpoint Enabled and Public Access Disabled	5%						35%
5.4.3: Ensure clusters are created with Private Nodes	5%						20%
5.4.4: Ensure Network Policy is Enabled and set as appropriate	0%						20%
5.4.5: Encrypt traffic to HTTPS load balancers with TLS certificates	0%						60%
5.5.1: Manage Kubernetes RBAC users with AWS IAM Authenticator for Kubernetes	10%						25%
5.6.1: Consider Fargate for running untrusted workloads	15%						5%

Unfamiliar Not important Little important Somewhat important Important Extremely Important

Pre-print

Once done, we compute the Cohen's Kappa [16] between the two raters is respectively, 0.96 and 0.89 for the Company-A and the proprietary dataset. In the case of disagreements, the raters discussed on why an identified coding pattern is a valid or invalid violation. The second author's decision is final during disagreement resolution as the they are actively working for Company-A and has more practical perspectives than the first author. Upon completion of this process we obtain two datasets where each file is labeled with a violation.

3) Metrics: We use two metrics to answer RQ3: (i) the count of configuration files that violates a certain prescribed configuration; and (ii) the proportion of configuration files that include at least one violation of a prescribed configuration. We use these three metrics as they can help us contextualize the

TABLE IV: Attributes of Datasets

Dataset	Attribute	Value
Company-A	Configuration files	188
	Total Size (LOC)	12,423
	Total Repositories	2
OSS	Configuration files	356
	Total Size (LOC)	55,699
	Total Repositories	33

frequency of violations from multiple perspectives.

B. Answer to RQ3

We report our findings in Tables V and VI using the 'Answer to RQ3' column. In this column we report the 'Count' and 'Proportion' data for each CIS-prescribed configurations that are mapped to their CIS IDs. Here, 'Count' reports the count of configuration files for which $>= 1$ violations occur, whereas 'Proportion' reports the proportion of configuration files that include $>= 1$ violation of any of the 53 prescribed configurations. We observe the most frequently violated configuration to be "*ensure that the Service Account Tokens are only mounted where necessary*" for the OSS dataset and "*create administrative boundaries between resources using namespaces*" for the Company-A dataset. In all, we observe a total of 178 and 85 configuration files for which violations occur respectively, for the Company-A and OSS dataset.

> ***Answer to RQ3****: For the Company-A dataset, 18.0% of 188 configuration files include at least one violation of CIS-prescribed configurations. For the OSS dataset, 17.9% of 356 configuration files include at least one violation of CIS-prescribed configurations.*

V. RQ4: Detection Accuracy of SAST Tools

We provide the methodology and results respectively, in Sections V-A and V-B.

A. Methodology for RQ4

We use the evaluation dataset that we construct in Section IV-A to answer RQ4. The dataset contains a mapping of which Kubernetes configuration file violates one or multiple of the 53 prescribed configurations for Company-A's repositories and OSS repositories. Similar to prior research on security tool evaluation [32], [45], we use precision and recall to determine the accuracy of the five tools for two datasets. We use the following formulas to calculate precision and recall:

$$Precision = \frac{TruePositive(TP)}{TruePositive(TP) + FalsePositive(FP)}$$

$$Recall = \frac{TruePositive(TP)}{TruePositive(TP) + FalseNegative(FN)}$$

For OSS and Company-A datasets we calculate precision and recall for two scenarios: (i) *individual:* here, we compute the precision and recall for each SAST tool individually. Here, we report the precision and recall for each of the violated prescribed configurations; and (ii) *combined*: here, we compute the precision and recall for multiple combinations of SAST tools. We report the precision and recall across all prescribed configurations for both datasets. We use combinations of two, three, four and five SAST tools in this case.

B. Answer to RQ4

We report the precision and recall for the five SAST tools using Tables V and VI. The highest average precision and recall for the Company-A dataset is observed respectively, for KubeLinter and Trivy. The highest average precision for the OSS dataset is observed both for Checkov and Trivy. The highest recall for the OSS dataset is observed for Checkov. For both datasets, the average precision is < 0.50, which is lower than that of 'acceptable precision' for static analysis tools according to practitioners [48]. One possible explanation for this low precision and recall could be the use of patterns that they use to detect violations. Another possible explanation is related to the differences between the taxonomy of CIS that we used and the taxonomy of the tools. For example, categories used by SLI-KUBE and categories prescribed by CIS are different.

Precision and recall for each combination are available in Table VII. When combinations of tools are used we observe an increase in precision and recall. For example, for the Company-A dataset, the precision of KubeLinter increases from 0.41 to 0.48 when combined with Checkov and Kubescape. The recall of Trivy increases from 0.35 to 0.53 for the same dataset when combined with Checkov, SLI-KUBE, KubeLinter, and Kubescape.

For the OSS dataset, the recall of Checkov is 0.65 for two combinations: (i) Checkov, Trivy, KubeLinter and (ii) Checkov, Trivy, KubeLinter, and Kubescape. The highest precision is 0.42 that is lower than that of the average precision of Checkov and Trivy. Despite improvements in recall, for both datasets the precision is < 0.50, which is still lower than that of 'acceptable precision'. According to prior work [48], practitioners expect a precision of $>= 0.90$ for usage of static analysis tools.

> ***Answer to RQ4****: When SAST tools are evaluated individually, the highest observed precision is 0.41 and 0.43 respectively, for the Company-A and OSS datasets. The highest observed recall is 0.38 and 0.35 respectively, for the Company-A and OSS datasets. When all the five SAST tools are combined, the recall increases to 0.53 and 0.65 respectively, for the Company-A and OSS datasets.*

VI. Discussion

We discuss the implications of our findings and threats to validity respectively, in Sections VI-A and VI-B.

Pre-print

TABLE V: Frequency of Violated Prescriptions and Detection Accuracy of Kubernetes SAST for the Company-A Dataset

CIS ID	Answer to RQ3 (Freq.)		Answer to RQ4 (Detection Accuracy)									
	Count	Proportion	SLI-KUBE		Checkov		Kubescape		KubeLinter		Trivy	
			Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
4.1.2	14	7.87	0.0	0.0	0.0	0.0	0.0	0.0	0.25	0.14	1.0	0.75
4.1.3	2	1.12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0
4.1.4	6	3.37	0.0	0.0	0.0	0.0	0.0	0.0	0.25	0.17	0.0	0.0
4.1.5	54	30.34	0.0	0.0	0.0	0.0	0.0	0.0	0.93	0.74	0.0	0.0
4.1.6	100	56.18	0.0	0.0	0.93	0.56	0.91	0.30	0.0	0.0	0.0	0.0
4.2.1	1	0.56	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
4.2.2	1	0.56	1.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0
4.2.3	3	1.69	1.0	1.0	1.0	0.67	1.0	0.67	1.0	0.67	1.0	0.67
4.2.5	2	1.12	1.0	0.5	0.02	0.5	0.0	0.0	0.5	0.5	0.02	0.5
4.2.6	3	1.69	0.0	0.0	0.0	0.0	0.03	0.67	0.04	0.67	0.03	0.67
4.2.7	1	0.56	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4.2.8	2	1.12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4.4.1	37	20.79	0.0	0.0	1.0	0.30	0.0	0.0	1.0	0.35	0.0	0.0
4.6.1	156	87.64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4.6.2	45	25.28	1.0	0.24	0.70	0.42	0.62	0.93	0.0	0.0	0.23	0.96
4.6.3	112	62.92	0.95	0.48	0.93	0.79	0.58	0.34	0.96	0.59	0.0	0.0
5.4.5	7	3.93	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
All CIS ID	178	18.04	0.35	0.25	0.39	0.31	0.24	0.23	0.41	0.34	0.31	0.38

TABLE VI: Frequency of Violated Prescriptions and Detection Accuracy of Kubernetes SAST for the OSS Dataset

CIS ID	Answer to RQ3 (Freq.)		Answer to RQ4 (Detection Accuracy)									
	Count	Proportion	SLI-KUBE		Checkov		Kubescape		KubeLinter		Trivy	
			Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
4.1.2	4	4.71	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4.1.3	7	8.23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.43
4.1.5	28	32.94	0.0	0.0	0.0	0.0	0.0	0.0	0.65	0.93	0.0	0.0
4.1.6	48	56.47	0.0	0.0	0.83	0.90	0.44	0.08	0.0	0.0	0.0	0.0
4.2.1	8	9.41	0.0	0.0	1.0	0.63	1.0	0.38	1.0	0.5	1.0	0.63
4.2.2	4	4.71	1.0	0.75	1.0	0.75	0.0	0.0	1.0	0.75	1.0	0.75
4.2.3	16	18.82	1.0	0.25	1.0	0.58	1.0	0.42	1.0	0.5	1.0	0.58
4.2.6	10	11.76	0.0	0.0	0.0	0.0	0.04	0.2	0.05	0.2	0.42	0.2
4.2.7	1	1.18	0.0	0.0	0.5	1.0	0.0	0.0	0.0	0.0	0.5	1.0
4.2.8	11	12.94	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.08	0.33
4.4.1	4	4.71	0.0	0.0	0.5	0.25	0.0	0.0	0.5	0.25	0.0	0.0
4.6.1	18	21.18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4.6.2	41	48.24	0.9	0.22	0.94	0.76	0.79	0.90	0.0	0.0	0.74	0.9
4.6.3	27	31.76	0.94	0.41	0.75	0.64	0.54	0.52	0.71	0.52	1.0	0.4
5.4.5	2	2.35	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
All CIS ID	85	17.96	0.26	0.11	0.43	0.37	0.25	0.17	0.33	0.18	0.43	0.35

Pre-Print

A. Implications

1) Implications for SAST Tool Usage in Kubernetes Configuration Management: Reported detection accuracy in Section V-B showcases the limitation of SAST tools' with respect to precision. While we acknowledge this limitation, we advocate for combining SAST tools in order to detect violations of prescribed configurations. According to recent research practitioners have expressed that even if SAST tools have low precision they still provide value in secure software development as "*finding something would be better than nothing*" [4]. Based on data precision and recall data presented in Table VII, we advocate for the following combinations: (i) KubeLinter and Checkov; (ii) KubeLinter and Kubescape; and (iii) the combination of Checkov, KubeLinter, Kubescape, SLI-KUBE, and Trivy.

Findings reported in Section III-B provide a nuanced perspective for the 53 CIS-prescribed configurations. We observe $>= 15\%$ of survey respondents to be unfamiliar with 17 of the 53 configurations, which shows how practitioners experienced in Kubernetes can be unaware of potential security threats due to Kubernetes-related configurations. While alarming, this finding is not surprising as it aligns with prior research that has shown practitioners to be unaware of all configurations in software systems [63]. This lack of awareness is also applicable for a software system, such as Kubernetes, which is complex, relatively novel, and evolving [9], [66]. We advocate for the usage of security testing techniques, such as SAST tools so that practitioners are informed on the configurations that can have security-related consequences.

2) Implications on Improving SAST Tools: Data reported in Table II also shows 5% of the survey respondents to find 32 of the 53 prescribed configurations to be 'not important' or 'little important'. This finding showcases that certain prescriptions can be perceived as irrelevant to practitioners. This finding is counter-intuitive as the 53 prescriptions are derived by industry experts, i.e., CIS contributors. While this finding in counter-intuitive, the finding can enhance the configuration process in the following manner:

- **Localization of identified configurations:** In the case of security tools, practitioners seek information on how a detected alert is used in the software project, which in turn leads to taking an action [55]. Accordingly, for Kubernetes-related SASTs we advocate for accurate reporting of the violations and the locations where the violations occur.

TABLE VII: Detection Accuracy of SAST Tools When Used as Combinations

Combination	Company-A		OSS	
	Precision	Recall	Precision	Recall
SLI-KUBE ∪ KubeLinter	0.38	0.31	0.20	0.31
SLI-KUBE ∪ Checkov	0.35	0.34	0.29	0.48
SLI-KUBE ∪ Trivy	0.21	0.24	0.19	0.34
SLI-KUBE ∪ Kubescape	0.36	0.31	0.19	0.33
KubeLinter ∪ Checkov	0.43	0.40	0.42	0.59
KubeLinter ∪ Trivy	0.29	0.30	0.30	0.44
KubeLinter ∪ Kubescape	0.48	0.34	0.33	0.42
Checkov ∪ Trivy	0.33	0.37	0.41	0.53
Checkov ∪ Kubescape	0.42	0.40	0.39	0.50
Trivy ∪ Kubescape	0.24	0.23	0.27	0.34
SLI-KUBE ∪ KubeLinter ∪ Checkov	0.37	0.43	0.29	0.60
SLI-KUBE ∪ KubeLinter ∪ Trivy	0.28	0.38	0.22	0.47
SLI-KUBE ∪ KubeLinter ∪ Kubescape	0.33	0.42	0.23	0.45
SLI-KUBE ∪ Checkov ∪ Trivy	0.27	0.40	0.25	0.54
SLI-KUBE ∪ Checkov ∪ Kubescape	0.36	0.43	0.26	0.52
SLI-KUBE ∪ Trivy ∪ Kubescape	0.26	0.33	0.19	0.37
KubeLinter ∪ Checkov ∪ Trivy	0.34	0.46	0.37	0.64
KubeLinter ∪ Checkov ∪ Kubescape	0.45	0.48	0.40	0.62
KubeLinter ∪ Trivy ∪ Kubescape	0.32	0.36	0.30	0.46
Checkov ∪ Trivy ∪ Kubescape	0.32	0.42	0.34	0.53
SLI-KUBE ∪ KubeLinter ∪ Checkov ∪ Trivy	0.31	0.49	0.27	0.65
SLI-KUBE ∪ KubeLinter ∪ Checkov ∪ Kubescape	0.39	0.51	0.29	0.63
SLI-KUBE ∪ KubeLinter ∪ Trivy ∪ Kubescape	0.31	0.43	0.22	0.49
SLI-KUBE ∪ Checkov ∪ Trivy ∪ Kubescape	0.29	0.45	0.25	0.54
KubeLinter ∪ Checkov ∪ Trivy ∪ Kubescape	0.35	0.50	0.36	0.64
SLI-KUBE ∪ KubeLinter ∪ Checkov ∪ Trivy ∪ Kubescape	0.32	0.53	0.27	0.65

- **Incorporate attack-related information:** A lack of meaningful alerts is a pain-point in using SAST tools [4], which we postulate can be mitigated through incorporation of attack-related information. While reporting violated configurations tools should also discuss if a single or a combination of configurations can lead to a security attack.
- **Improve Detection Accuracy:** The detection accuracy of the studied tools can be improved by (i) enhancing the rules with examples that are available in our dataset; (ii) incorporating the configurations that are not covered by tools but are present in CIS guidelines; (iii) incorporating practitioner feedback; and (iv) deriving control and data flow analysis techniques unique to Kubernetes entities.

3) On the Value of Dynamic Analysis for Securing Kubernetes Deployments: According to Table II, a SAST tool alone can only detect 20 out 53 prescribed configurations. Furthermore, the precision and recall for SAST tools is underwhelming as discussed in Section V-B. These shortcomings of SAST tools highlight the potential of using dynamic application security testing (DAST) tools, such as KubeBench [26]. Practitioners can use DAST tools to identify violations of prescribed security configurations in Kubernetes deployments.

4) Future Work: Our findings provide the groundwork for further research in the following directions:

- derive human and socio-technical factors that explain why practitioners do not implement prescribed configurations;
- derive methodologies to increase usage of CIS-prescribed configurations;
- conduct evaluation of CIS-prescribed security guidelines for other software systems;
- evaluate studied tools for non-security and security defects; and
- derive defect and vulnerability detection techniques by understanding the unique properties of Kubernetes entities.

B. Threats to Validity

We describe the limitations of our paper as follows:

External Validity: Our survey participants include 10 practitioners from Company-A, which may not reflect the opinion of other practitioners who use Kubernetes. We mitigate this limitation by including 10 additional practitioners in our survey who are contribute to OSS projects. Also, reported findings for RQ3 and RQ4 may not generalize for all repositories that contain Kubernetes configuration files. We mitigate this limitation by using OSS repositories mined from GitHub.

Our static analysis tools evaluation results are limited to two proprietary repositories used by Company-A, which may not generalize to other repositories. We mitigate this challenge by evaluating the static analysis tools on 33 OSS repositories. Answers to RQ3 are dependent on the scripts that we select and their sources. A different dataset may result in different precision and recall values for the studied tools.

Conclusion Validity: The mapping between the rules of security analysis tools and prescribed configurations are susceptible to rater bias. We mitigate this limitation by using two raters, of which one is an industry practitioner with industry experience in cybersecurity. Construction of our evaluation dataset is also susceptible to rater bias. We mitigate this limitation by using two raters. The survey respondent count is 20, which may influence the results. We acknowledge that the survey response rate is on the lower end, but low survey response rare is common in software engineering where one publication reports the survey response rate to be as lows as 6% [54].

Internal Validity: One of the raters who performs the mapping is a practitioner working at Company-A. This affiliation may affect intuitively affect the mapping process. We mitigate this limitation by using another rater who has no affiliation with Company-A. Usage of precision and recall may also bias the evaluation process of the SAST tools that we studied.

VII. Related Work

Our paper is related with prior research that have addressed defects in Dockerfiles, Docker Swarm, Kubernetes security, and security tool evaluation.

Pre-print

a) Prior Research on Dockerfiles: Defects in configuration scripts used to set up Docker containers have garnered a lot of interest amongst researchers. Azuma et al. [5] and Ksontini et al. [25] in separate research studies systematically investigated defects in Dockerfiles. Other researchers, such as Haque et al. [20], Wist et al. [62], Jain et al. [22], Pinnamaneni et al. [42], Liu et al. [34], Shu et al. [53], Zerouali et al. [64], [65], and Lin et al. [33] have focused on characterizing security defects in container images.

b) Prior Research on Docker Swarm: Similar to Kubernetes, Docker Swarm is also a tool used for container orchestration. While it is not as popular as Kubernetes, researchers [15], [39] have conducted multiple research studies on the scheduling mechanism of systems managed by Docker Swarm. Researchers have conducted performance-related evaluation [40]. Also, through systematic comparative evaluations researchers find Kubernetes to be better than that of Docker Swarm with respect to scalability features [56] and security [37].

c) Prior Research on Kubernetes Security: Prior research has used graph and anomaly-based approaches for Kubernetes security. For securing Kubernetes clusters, researchers adopted anomaly-based approaches [3], [58]. Tien et al developed an anomaly detection tool to monitor and detect attacks in the Kubernetes cluster [58]. Cao et al. developed an anomaly detection tool using a state machine model for Kubernetes deployment [10]. Hariri et al. proposed an anomaly detection tool for scientific applications in Kubernetes [21]. Researchers also have adopted graph-based approaches to secure Kubernetes clusters. Blaise et al. used a graph-based approach to extract the attack path for Kubernetes deployment [6].

Pre-print

Empirical analysis is another topic for Kubernetes security research. Bose et al. performed qualitative analysis and constructed a dataset with security related commits [7]. The research that is closest in spirit to our paper is the paper authored by Rahman et al. [44]. They [44] developed a security analysis tool for detection of misconfiguration [44]. Our paper is different from their paper in the following manner: (i) we analyze the practitioner perceptions of CIS-prescribed recommendations for Kubernetes; (ii) we analyze the support and detection accuracy for 5 SAST tools while detecting violations of CIS-prescribed configurations; and (iii) we quantify the frequency of 17 and 15 CIS-prescribed configurations respectively, for the Company-A and OSS dataset.

d) Prior Research on Evaluation of Security Analysis Tools: Our paper is related to prior research on SAST tool evaluation. Li et al. [32] evaluated 8 SAST tools and found the precision of these tools to be $< 10\%$. Valentina et al. performed a comparison using 6 static analysis tools on 47 Java-related projects [31] and found little to no agreement between the tools. Hamda et al. evaluated the effectiveness of non-commercial security analysis tools based on OWASP Top 10 security vulnerabilities, and reported that SAST tools are inadequate to uncover all common weaknesses [2]. Researchers also have investigated the impact of combining SAST tools. Paulo et al. combined 5 security analysis tools to detect web-based vulnerabilities [41]. Fransesc et al. combined static, dynamic, and interactive analysis tools and reported the n-tools combination's average effectiveness in reducing false positives [60]. Ranganath et al. evaluated 14 Android security analysis tools and found a combination of these tools to detect 30 out 42 known vulnerabilities [45].

The above-mentioned discussion showcases a lack of research in SAST tool evaluation for Kubernetes configuration files. We address this gap in our paper. Our paper shows CIS-prescribed configurations are violated in practice and existing SAST tools are inadequate in detecting these violations. In the context of Kubernetes-related studies, these findings are novel as none of the above-mentioned papers have reported these findings. Furthermore, our paper shows a disconnect between what CIS experts recommend and what happens in practice.

VIII. Conclusion

Despite the existence of community-prescribed guidelines for securing Kubernetes deployments, there is a lack of understanding of how practitioners perceive these guidelines and to what extent existing SAST tools detect violations of these guidelines. We have conducted an empirical study with 53 configurations prescribed by CIS. For Company-A and OSS respectively, we find 18.0% and 17.9% of the configuration files to include at least one violation of prescribed configuration. We also observe 5%∼40% of the surveyed 20 practitioners be unaware about the 16 configurations. When SAST tools are evaluated individually, the highest observed precision is 0.41 and 0.43 respectively, for the Company-A and OSS dataset. The highest observed recall is 0.38 and 0.35 respectively, for the Company-A and OSS dataset. When all the five SAST tools are combined, the recall increases to 0.53 and 0.65 respectively, for Company-A and OSS. Based on our findings, we recommend practitioners to use a combination of SAST tools as combinations of SAST tools can identify violations of prescribed configurations with more accuracy.

Acknowledgments

We thank the PASER group at Auburn University for their valuable feedback. This research was partially funded by the U.S. National Science Foundation (NSF) Award # 2247141 and Award # 2312321. This work has benefitted from Dagstuhl Seminar 23082 "Resilient Software Configuration and Infrastructure Code Analysis."

References

[1] akondrahman, "akondrahman/sli-kube," 2022. [Online]. Available: https://hub.docker.com/repository/docker/akondrahman/sli-kube

[2] H. H. AlBreiki and Q. H. Mahmoud, "Evaluation of static analysis tools for software security," in *2014 10th International Conference on Innovations in Information Technology (IIT)*, 2014, pp. 93–98.

[3] J. G. Almaraz-Rivera, "An anomaly-based detection system for monitoring kubernetes infrastructures," *IEEE Latin America Transactions*, vol. 21, no. 3, pp. 457–465, 2023.

[4] A. Ami, K. Moran, D. Poshyvanyk, and A. Nadkarni, "quot;false negative - that one is going to kill you.quot; - understanding industry perspectives of static analysis based security testing," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 23–23. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00019

[5] H. Azuma, S. Matsumoto, Y. Kamei, and S. Kusumoto, "An empirical study on self-admitted technical debt in dockerfiles," *Empirical Software Engineering*, vol. 27, no. 2, pp. 1–26, 2022.

[6] A. Blaise and F. Rebecchi, "Stay at the helm: secure kubernetes deployments via graph generation and attack reconstruction," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. IEEE, 2022, pp. 59–69.

[7] D. B. Bose, A. Rahman, and S. I. Shamim, "'under-reported' security defects in kubernetes manifests," in *2021 IEEE/ACM 2nd International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS)*. IEEE, 2021, pp. 9–12.

[8] bridgecrew, "checkov," https://www.checkov.io/4.Integrations/Kubernetes.html, 2022, [Online; accessed 12-May-2022].

[9] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: up and running*. " O'Reilly Media, Inc.", 2022.

[10] C. Cao, A. Blaise, S. Verwer, and F. Rebecchi, "Learning state machines to monitor and detect anomalies on a kubernetes cluster," in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 2022, pp. 1–9.

[11] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Comput. Surv.*, vol. 55, no. 7, dec 2022. [Online]. Available: https://doi.org/10.1145/3539606

[12] Case Study: OpenAI, "Kubernetes," https://kubernetes.io/case-studies/openai/, 2024, [Online; accessed 28-July-2024].

[13] Case Study: Spotify, "Kubernetes," https://kubernetes.io/case-studies/spotify/, 2024, [Online; accessed 27-July-2024].

[14] Center for Internet Security(CIS), https://www.cisecurity.org/, 2024, [Online; accessed 12-march-2024].

[15] C. Cerin, T. Menouer, W. Saad, and W. B. Abdallah, "A new docker swarm scheduling strategy," in *2017 IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, 2017, pp. 112–117.

[16] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960. [Online]. Available: http://dx.doi.org/10.1177/001316446002000104

[17] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, *Selecting Empirical Methods for Software Engineering Research*. London: Springer London, 2008, pp. 285–311.

[18] C. for Information Security (CIS), "CIS Kubernetes Benchmarks," 2024. [Online]. Available: https://www.cisecurity.org/benchmark/kubernetes

[19] G. Gousios and D. Spinellis, "Ghtorrent: Github's data from a firehose," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2012, pp. 12–21.

[20] M. U. Haque and M. A. Babar, "Well begun is half done: An empirical study of exploitability & impact of base-image vulnerabilities," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 1066–1077.

[21] S. Hariri and M. C. Kind, "Batch and online anomaly detection for scientific applications in a kubernetes environment," in *Proceedings of the 9th Workshop on Scientific Cloud Computing*, ser. ScienceCloud'18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3217880.3217883

[22] V. Jain, B. Singh, M. Khenwar, and M. Sharma, "Static vulnerability analysis of docker images," in *IOP Conference Series: Materials Science and Engineering*, vol. 1131, no. 1. IOP Publishing, 2021, p. 012018.

[23] K. Kamieniarz and W. Mazurczyk, "A comparative study on the security of kubernetes deployments," in *2024 International Wireless Communications and Mobile Computing (IWCMC)*, 2024, pp. 0718–0723.

[24] B. A. Kitchenham and S. L. Pfleeger, *Personal Opinion Surveys*. London: Springer London, 2008, pp. 63–92. [Online]. Available: https://doi.org/10.1007/978-1-84800-044-5_3

[25] E. Ksontini, M. Kessentini, T. d. N. Ferreira, and F. Hassan, "Refactorings and technical debt in docker projects: An empirical study," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 781–791.

[26] Kubebench, "Kubebench," https://aquasecurity.github.io/kube-bench/v0.6.15/, 2024, [Online; accessed 15-March-2024].

[27] kubelinter, "kubelinter," https://docs.kubelinter.io/#/generated/checks, 2022, [Online; accessed 13-May-2022].

[28] Kubernetes, "Production-grade container orchestration," 2021. [Online]. Available: https://kubernetes.io/

[29] Kubescape, "Kubescape," https://hub.armosec.io/docs/controls, 2024, [Online; accessed 15-March-2024].

[30] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977. [Online]. Available: http://www.jstor.org/stable/2529310

[31] V. Lenarduzzi, F. Pecorelli, N. Saarimaki, S. Lujan, and F. Palomba, "A critical comparison on six static analysis tools: Detection, agreement, and precision," *Journal of Systems and Software*, vol. 198, p. 111575, 2023.

[32] K. Li, Y. Xue, S. Chen, H. Liu, K. Sun, M. Hu, H. Wang, Y. Liu, and Y. Chen, "Static application security testing (sast) tools for smart contracts: How far are we?" *arXiv preprint arXiv:2404.18186*, 2024.

[33] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on linux container security: Attacks and countermeasures," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 418–429.

[34] P. Liu, S. Ji, L. Fu, K. Lu, X. Zhang, W. Lee, T. Lu, W. Chen, and R. Beyah, "Understanding the security risks of docker hub," in *Computer Security – ESORICS 2020 - 25th European Symposium on Research in Computer Security, Proceedings*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), L. Chen, S. Schneider, N. Li, and K. Liang, Eds. Germany: Springer Science and Business Media Deutschland GmbH, 2020, pp. 257–276, funding Information: Acknowledgements. This work was partly supported by the Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars under No. LR19F020003, the National Key Research and Development Program of China under No. 2018YFB0804102, NSFC under No. 61772466, U1936215, and U1836202, the Zhe-jiang Provincial Key RD Program under No. 2019C01055, and the Ant Financial Research Funding.; 25th European Symposium on Research in Computer Security, ESORICS 2020 ; Conference date: 14-09-2020 Through 18-09-2020.

[35] ManagedKube, "kubernetes-ops," https://github.com/ManagedKube/kubernetes-ops/tree/main, 2019, [Online; accessed 02-August-2024].

[36] S. Miles, *Kubernetes: A Step-By-Step Guide For Beginners To Build, Manage, Develop, and Intelligently Deploy Applications By Using Kubernetes (2020 Edition)*. Independently Published, 2020. [Online]. Available: https://books.google.com/books?id=M4VmzQEACAAJ

[37] A. Modak, S. D. Chaudhary, P. S. Paygude, and S. R. Ldate, "Techniques to secure data on cloud: Docker swarm or kubernetes?" in *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, 2018, pp. 7–12.

[38] N. Munaiah, S. Kroh, C. Cabrey, and M. Nagappan, "Curating GitHub for engineered software projects," *Empirical Software Engineering*, pp. 1–35, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10664-017-9512-6

Pre-print

[39] N. Naik, "Building a virtual system of systems using docker swarm in multiple clouds," in *2016 IEEE International Symposium on Systems Engineering (ISSE)*, 2016, pp. 1–3.

[40] ——, "Performance evaluation of distributed systems in multiple clouds using docker swarm," in *2021 IEEE International Systems Conference (SysCon)*, 2021, pp. 1–6.

[41] P. Nunes, I. Medeiros, J. C. Fonseca, N. Neves, M. Correia, and M. Vieira, "Benchmarking static analysis tools for web security," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 1159–1175, 2018.

[42] J. Pinnamaneni, S. Nagasundari, and P. Honnavalli, "Identifying vulnerabilities in docker image code using ml techniques," in *2022 2nd Asian Conference on Innovation in Technology (ASIANCON)*. IEEE, 2022, pp. 1–5.

[43] A. Rahman, D. B. Bose, Y. Zhang, and R. Pandita, "An empirical study of task infections in ansible scripts," *Empirical Software Engineering*, vol. 29, no. 1, p. 34, 2024.

[44] 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, may 2023. [Online]. Available: https://doi.org/10.1145/3579639

[45] V.-P. Ranganath and J. Mitra, "Are free android app security analysis tools effective in detecting known vulnerabilities?" *Empirical Software Engineering*, vol. 25, no. 1, pp. 178–219, 2020.

[46] RedHat, "State of Kubernetes Security Report 2023," 2023. [Online]. Available: https://www.redhat.com/en/resources/state-kubernetes-security-report-2023

[47] ——, "The state of Kubernetes security report: 2024 edition," 2024. [Online]. Available: https://www.redhat.com/en/engage/state-kubernetes-security-report-2024

[48] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at google," *Commun. ACM*, vol. 61, no. 4, p. 58–66, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3188720

[49] J. Saldaña, *The coding manual for qualitative researchers*. Sage, 2015.

[50] A. W. Services, "Amazon Elastic Kubernetes Service," https://aws.amazon.com/eks/, 2024, [Online; accessed 01-August-2024].

[51] 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 *2020 IEEE Secure Development (SecDev)*. IEEE, 2020, pp. 58–64.

[52] S. I. Shamim, H. Hu, and A. Rahman, "Replication package for paper," https://figshare.com/s/8c1b7c0cb5e2915d974c, 2024, [Online; accessed 12-Dec-2024].

[53] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 269–280.

[54] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, May 2013, pp. 89–92.

[55] J. Smith, B. Johnson, E. Murphy-Hill, B. Chu, and H. R. Lipford, "Questions developers ask while diagnosing potential security vulnerabilities with static analysis," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 248–259. [Online]. Available: https://doi.org/10.1145/2786805.2786812

[56] P. Stromberg, "Automatically scaling a system across multiple servers: A comparison of docker swarm and kubernetes," B.S. thesis, Tampere University, 2021.

[57] H. Taherdoost, "What is the best response scale for survey and questionnaire design; review of different lengths of rating scale/attitude scale/likert scale," *Hamed Taherdoost*, pp. 1–10, 2019.

[58] C.-W. Tien, T.-Y. Huang, C.-W. Tien, T.-C. Huang, and S.-Y. Kuo, "Kubanomaly: anomaly detection for the docker orchestration platform with neural network approaches," *Engineering reports*, vol. 1, no. 5, p. e12080, 2019.

[59] Trivy, "Trivy," https://aquasecurity.github.io/trivy/, 2024, [Online; accessed 15-March-2024].

[60] F. Tudela, J.-R. Higuera, J. Bermejo, J. A. Montalvo, and M. Argyros, "On combining static, dynamic and interactive analysis security testing tools to improve owasp top ten security vulnerability detection in web applications," *Applied Sciences*, vol. 10, 12 2020.

[61] F. K. Willits, G. L. Theodori, and A. Luloff, "Another look at likert scales," *Journal of Rural Social Sciences*, vol. 31, no. 3, p. 6, 2016.

[62] K. Wist, M. Helsem, and D. Gligoroski, "Vulnerability analysis of 2500 docker hub images," in *Advances in Security, Networks, and Internet of Things*. Springer, 2021, pp. 307–327.

[63] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 307–319. [Online]. Available: https://doi.org/10.1145/2786805.2786852

[64] 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 *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 619–623.

[65] A. Zerouali, T. Mens, and C. De Roover, "On the usage of javascript, python and ruby packages in docker hub images," *Science of Computer Programming*, vol. 207, p. 102653, 2021.

[66] Y. Zhang, R. Meredith, W. Reeves, J. Coriolano, M. Ali Babar, and A. Rahman, "Does generative ai generate smells related to container orchestration?: An exploratory study with kubernetes manifests," in *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*, 2024.

[67] A. Zimba, Z. Wang, M. Mulenga, and N. H. Odongo, "Crypto mining attacks in information systems: An emerging threat to cyber security," *Journal of Computer Information Systems*, 2018.

Pre-print