‘Under-reported’ Security Defects in Kubernetes Manifests

Dibyendu Brinto Bose* Akond Rahman† Shazibul Islam Shamim ‡
*Reve Systems, Dhaka, Bangladesh
†Department of Computer Science, Tennessee Technological University, Cookeville, TN, USA
Email: *brintodibyendu@gmail.com †arahman@tntech.edu ‡ mshamim42@tntech.edu

***Abstract*—With the advent of the fourth industrial revolution, industry practitioners are moving towards container-based infrastructure for managing their digital workloads. Kubernetes, a container orchestration tool, is reported to help industry practitioners in automated management of cloud infrastructure and rapid deployment of software services. Despite reported benefits, Kubernetes installations are susceptible to security defects, as it occurred for Tesla in 2018. Understanding how frequently security defects appear in Kubernetes installations can help cybersecurity researchers to investigate security-related vulnerabilities for Kubernetes and generate security best practices to avoid them. In this position paper, we first quantify how frequently security defects appear in Kubernetes manifests, i.e., configuration files that are use to install and manage Kubernetes. Next, we lay out a list of future research directions that researchers can pursue.**

We apply qualitative analysis on 5,193 commits collected from 38 open source repositories and observe that 0.79% of the 5,193 commits are security-related. Based on our findings, we posit that security-related defects are under-reported and advocate for rigorous research that can systematically identify undiscovered security defects that exist in Kubernetes manifests. We predict that the increasing use of Kubernetes with unresolved security defects can lead to large-scale security breaches.

***Index Terms*—dataset, devops, devsecops, kubernetes, security**

I. Introduction

The fourth industrial revolution advocates for modernization of manufacturing using computing resources. Hermann et al. [1] identifies four design principles integral to the fourth industrial revolution namely, interconnection, information transparency, technical assistance, and decentralized decisions. Integration of software-based services into the industrial manufacturing process is pivotal to achieve these design principles [2].

With the advent of the fourth industrial revolution, companies from multiple sectors are adopting container-based infrastructures to manage their digital workloads. Container is a software that packages up code and all its dependencies so the application runs reliably from one computing environment to another [1]. Because of perceived benefits, such as separation of production and test environments, and rapid deployment of software applications, containers have become a viable option for companies to manage their software workloads [2]. The increasing use of containers necessitates an automated tool, such as Kubernetes for orchestration.

Kubernetes is an open-source software for automating management of computerized services, such as containers [3]. Practitioners use Kubernetes because it reduces repetitive manual processes involved in container deployment and management. Kubernetes is considered one of the most popular open-source container orchestration tools and it is used in organizations, such as Adidas, Booz Allen Hamilton, and Sling TV [4]. Benefits of Kubernetes usage have been documented: for example using Kubernetes, Booz Allen Hamilton was able to reduce costs by 50% for maintaining a government website [3]. For Adidas, the load time for an e-commerce website was reduced by half, and release frequency increased from once every 4∼6 weeks to 3∼4 times a day [4]. Despite reported benefits, Kubernetes installations are susceptible to security defects. For example, in 2018, malicious users gained access to Tesla’s Amazon Web Services (AWS) resources using an insecure Kubernetes console that was not password protected [5].

The presence of security defects in Kubernetes installations motivate us to gain an understanding of how frequently security defects appear in Kubernetes manifests. Such understanding can inform researchers about the state of security in Kubernetes and outline a list of future action items in which cybersecurity researchers can participate.

One approach to study defects in Kubernetes installations is to investigate security defects that appear in Kubernetes manifests [3], i.e., configuration files used to maintain Kubernetes installations. Our hypothesis is that by quantifying security defects in Kubernetes manifests, we can gain an understanding of how frequently security defects appear in Kubernetes installations. We evaluate our hypothesis by answering the following research question: ***How frequently do security defects occur in Kubernetes manifests?***

Our contribution is listed as follows:

- A curated dataset of security defects in Kubernetes manifests; and
- An empirical analysis of how frequently security defects occur in Kubernetes manifests.

[1] https://www.docker.com/resources/what-container
[2] https://www.arcweb.com/blog/end-industrial-automation-we-know-it
[3] https://kubernetes.io/case-studies/booz-allen/

II. Background and Related Work

We provide background and related work related to Kubernetes in this section.

A. Background

Kubernetes is an open-source software for automating management of computerized services such as containers [3]. A Kubernetes installation is colloquially referred to as a Kubernetes cluster [3]. Each Kubernetes cluster contains a set of worker machines defined as nodes. As shown in Figure 1, two types of nodes exist for Kubernetes: master nodes and worker nodes.

Each master node includes the following components: 'API server', 'scheduler', 'controller', and 'etcd' [3]. The 'API server' is responsible for orchestrating all the operations within the cluster. Kubernetes serves its functionality through an application program interface from the 'API server'. The 'controller' is a component on the master that watches the state of the cluster through the 'API server' and changes the current state towards the desired state. The 'scheduler' is the component in the control plane responsible for scheduling pods across multiple nodes. The 'etcd' is a key-value based database that stores all configuration information for the Kubernetes cluster. Users use a command-line tool 'Kubectl' to communicate with the 'API server' in the master node.

While provisioning Kubernetes, practitioners can provide configuration-related information in forms of configurations files, such as YAML files and JSON files. These files hold crucial information on what kind of network, CPU, and memory settings will be used by the Kubernetes installation. For example, using the `cpu:1` tag in a YAML file, a practitioner can specify that the Kubernetes installation will use 1 CPU. Listing 1 shows an example Kubernetes manifest.

```
kind: Pod
metadata:
  name: cpu-demo
  namespace: cpu-example
spec:
  containers:
  - name: cpu-demo-ctr
    image: vish/stress
    resources:
      limits:
        cpu:1
```

Listing 1: An example Kubernetes manifest

B. Related Work

Our paper is related to prior research that has investigated usage and maintenance of Kubernetes. Burns et al. [6] described the evolution of container management systems at Google, and described how two initial internal systems called Borg and Omega was evolved into Kubernetes. Brewer [7] conducted a case study on Kubernetes and discussed how key concepts of Kubernetes can be used to simplify scaling of containers. Medel et al. [8] used real data collected from Kubernetes and applied formal modeling to characterize performance and resource management in Kubernetes. Chang et al. [9] constructed a monitoring platform to dynamically provision cloud resources using Kubernetes. Vayghan et al. [10] investigated availability of Kubernetes using a set of experiments, and reported that service outages can occur frequently. Shah and Dubaria [11] compared orchestration management features of Docker Swarm, Kubernetes, and Google Cloud Platform, and observed Kubernetes to provide features, such as deployment, monitoring, and easy scalability. Shamim et al. [12] identified 11 practices to secure Kubernetes installations.

The above-mentioned discussion highlights Kubernetes research in two areas: (i) use of Kubernetes in creating systems, such as monitoring systems and (ii) case studies on Kubernetes related to performance and resource management. We observe a lack of research related to datasets that can be used to conduct security-related research for Kubernetes. We address this research gap by systematically constructing a security defect dataset for Kubernetes manifests.

III. Methodology

Answering our research question involves two steps: repository curation and qualitative analysis of commits.

A. Repository curation

We use open source software (OSS) repositories hosted on GitLab [4] to construct our dataset. As advocated by prior research [13], OSS repositories need to be curated. Software developers often use OSS repositories to store personal projects that are not reflective of the professional software development. We apply the following criteria to curate our collected repositories: **Criterion-1**: The repository is labeled as 'kubernetes' on GitLab. **Criterion-2**: The repository must be available for download. **Criterion-3**: The repository is not a clone. **Criterion-4**: The repository must have at least two commits per month. Munaiah et al. [13] used the threshold of at least two commits per month to determine which repositories have enough software development activity. We use this threshold to filter repositories with little activity. **Criterion-5**: The repository has at least 5 contributors. Our assumption is that the criteria of at least 5 contributors may help us to filter out irrelevant repositories. Previously, researchers have used the cutoff of at least nine contributors [14]. **Criterion-6**: The repository includes manifest files needed to provision Kubernetes. A repository can be mislabeled of using Kubernetes due to human error. We mitigate this limitation by manually inspecting if the repository includes Kubernetes manifest files.

B. Qualitative analysis of commits

We use commits from the collected OSS repositories obtained from Section III-A. We use commits because commits summarize changes that are made to a source code file and could identify the types of changes that are being performed on a source code file. We apply a qualitative analysis technique called closed coding [15] on the collected commits to determine which commit is related to a security defect.

[4] https://about.gitlab.com/

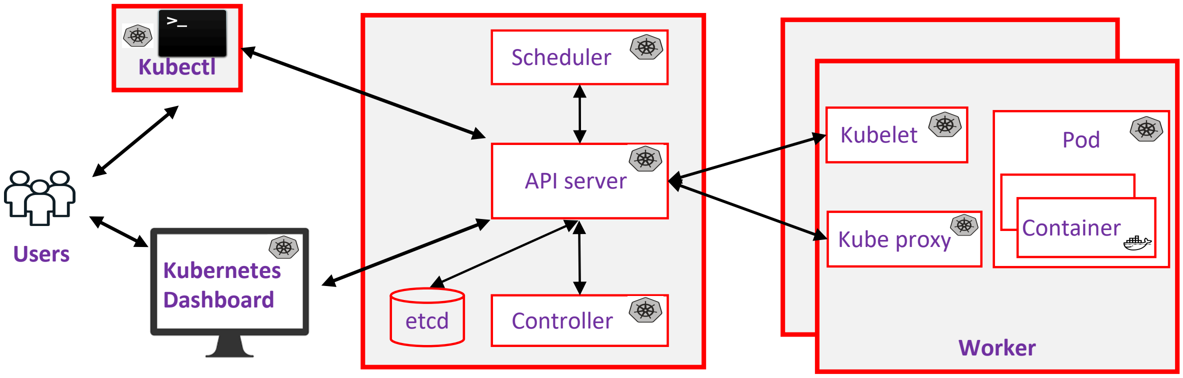

Fig. 1: A brief overview of Kubernetes. Kubernetes users interact with the installation using the Kubernetes dashboard and 'kubectl'.

We use two raters who are well-versed on software security to conduct the closed coding process. The first rater is the first author of the paper with two years of academic experience in cybersecurity. The second rater is not an author of the paper and participated voluntarily. The second rater is a graduate student in the department with one year of professional experience in cybersecurity. Both raters individually determine if each of the collected commits to be security-related by performing the following activities: *Activity-1*: The rater observes if any of the following keywords appear in each of the collected commit messages: 'race', 'racy', 'buffer', 'overflow', 'stack', 'integer', 'signedness', 'widthness', 'underflow', 'improper', 'unauthenticated', 'gain access', 'permission', 'cross site', 'css', 'xss', 'htmlspecialchar', 'denial service', 'dos', 'crash', 'deadlock', 'sql', 'sqli', 'injection', 'format', 'string', 'printf', 'scanf', 'request forgery', 'csrf', 'xsrf', 'forged', 'security', 'vulnerability', 'vulnerable', 'hole', 'exploit', 'attack', 'bypass', 'backdoor', 'threat', 'expose', 'breach', 'violate', 'fatal', 'blacklist', 'overrun', and 'insecure'. We collect these keywords from prior work [16]. *Activity-2*: The rater determines a commit to be a security-related defect if the message indicates that an action was taken to address a security concern for the software of interest. The rater determines a commit message to be related to security concern if any of the following security objects are violated: confidentially, integrity, or availability. We apply this step because only relying on keyword search could generate false positives. *Activity-3*: We calculate rater agreement using Cohen's Kappa [17].

Upon completion of the above-mentioned activities we obtain a dataset where each commit is labeled as a security defect or not. If the commit is related to a security defect then the label is 'INSECURE'. Otherwise the commit is labeled as 'NEUTRAL'. We answer our research question by reporting the count and proportion of commits that are labeled as 'INSECURE'.

C. Limitations of Our Methodology

The derived dataset is subject to rater bias, which we mitigate by using two raters. Open source repositories are susceptible to contain noisy data, which we mitigate using a systematic filtering criteria. Our dataset is collected from the open source GitLab repositories, which could be limiting. Our identification process of security defects use a keyword-based approach that can generate false positives. We mitigate this limitation through manual inspection.

TABLE I: OSS Repositories Satisfying Criteria (Sect. III-A)

Initial Repo Count	3,405,303
Criteria-1 (Kubernetes Label)	15,779
Criteria-2 (Available)	13,095
Criteria-3 (Not a clone)	3,173
Criteria-4 (Commits/Month ≥ 2)	173
Criteria-5 (Contributors ≥ 5)	38
Criteria-6 (Manifest Usage)	38
Final Repo Count	38

TABLE II: Repository Attributes

Attribute	**Count**
Repositories	38
Manifests	1,796
Manifest-related Commits	5,193
Duration	10/2015-07/2020

IV. Results and Conclusion

We provide empirical findings with discussion below:

A. Results

Using our filtering criteria mentioned in Section III-A we obtain 38 repositories. A complete breakdown of how many repositories are satisfied using each criterion is listed in Table I. We download these repositories on July 11, 2020. Attributes of the dataset is available in Table II. As shown in Table II, we collect 1,796 Kubernetes manifests that are modified in 5,193 commits.

Both raters, the first author of the paper and the volunteer performed the qualitative analysis individually as described in Section III-B to determine what commits are related to security defects. The process took 117 and 210 hours respectively, for the first author and the volunteer. The Cohen's Kappa is 0.7, which is 'substantial', according to Landis and Koch [18]. The labeled dataset is available online as a CSV file [19], which can be imported using existing APIs, such as Python Pandas [20].

Upon application of qualitative analysis on 5,193 commits we identify 41 commits to be labeled as a security dataset. The proportion of security defects is 0.79%. In our dataset we observe 39 Kubernetes manifests to be modified while addressing a security defect. Attributes of our security defect dataset is available in Table III.

TABLE III: Attributes of the Security Defect Dataset

Attribute	**Count**
Repositories with >= 1 Security Defect	9
Manifests Modified in a Security Defect	39
Security Defects	41

B. 'Under-reported' Security Defects in Kubernetes Manifests

Based on our findings we posit that security defects are under-reported in Kubernetes manifests. Our hypothesis is that multiple categories of security defects are latent in Kubernetes manifests. Existence of such latent security defects can facilitate attacks from malicious user, which could cause serious consequences. We advocate researchers to pursue future research directions:

- Identify violations of security best practices in Kubernetes;
- Construct static and dynamic analysis tools to identify security violations in Kubernetes; and
- Identify root causes of security defects in Kubernetes.

C. Conclusion

The use of Kubernetes is becoming more and more popular in maintaining critical systems, and could be of relevance to organizations aiming to join the fourth industrial revolution. The prevalence in usage of Kubernetes necessitates secure Kubernetes manifests. A research study that systematically quantifies security defects can help cybersecurity researchers in conducting future research related secure Kubernetes management. We construct a dataset of security defects that occur in Kubernetes manifests to help cybersecurity researchers. We have applied a qualitative analysis technique called closed coding to determine security defects that occur in Kubernetes manifests. We observe security defects to be rare: 0.79% of the 5,193 commits in our dataset to be security-related. Based on our findings we posit that (i) reported security defects in Kubernetes manifests are rare, and (ii) cybersecurity researchers should investigate how frequently latent security defects appear in Kubernetes manifests.

Acknowledgement

We thank the PASER group at Tennessee Technological University (TTU) for their valuable feedback. This research was partially funded by the National Science Foundation (NSF) award # 2026869.

References

[1] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3928–3937.

[2] G. Erboz, "How to define industry 4.0: The main pillars of industry 4.0. 2017," *URL: https://www. researchgate. net/publication/326557388*.

[3] 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

[4] Kubernetes User Case Studies, May 2020. [Online]. Available: https://kubernetes.io/case-studies/

[5] Tesla cloud resources are hacked to run cryptocurrency-mining malware, February 2018. [Online]. Available: https://arstechnica.com/information-technology/2018/02/tesla-cloud-resources-are-hacked-to-run-cryptocurrency-mining-malware/

[6] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, omega, and kubernetes," *Queue*, vol. 14, no. 1, p. 70–93, Jan. 2016. [Online]. Available: https://doi.org/10.1145/2898442.2898444

[7] E. A. Brewer, "Kubernetes and the path to cloud native," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, ser. SoCC '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 167. [Online]. Available: https://doi.org/10.1145/2806777.2809955

[8] V. Medel, O. Rana, J. a. Banares, and U. Arronategui, "Modelling performance & resource management in kubernetes," in *Proceedings of the 9th International Conference on Utility and Cloud Computing*, ser. UCC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 257–262. [Online]. Available: https://doi.org/10.1145/2996890.3007869

[9] C. Chang, S. Yang, E. Yeh, P. Lin, and J. Jeng, "A kubernetes-based monitoring platform for dynamic cloud resource provisioning," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, 2017, pp. 1–6.

[10] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Deploying microservice based applications with kubernetes: Experiments and lessons learned," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 970–973.

[11] J. Shah and D. Dubaria, "Building modern clouds: Using docker, kubernetes google cloud platform," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, 2019, pp. 0184–0189.

[12] M. 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)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2020, pp. 58–64. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SecDev45635.2020.00025

[13] 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

[14] A. Rahman, A. Agrawal, R. Krishna, and A. Sobran, "Characterizing the influence of continuous integration: Empirical results from 250+ open source and proprietary projects," in *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics*, ser. SWAN 2018. New York, NY, USA: ACM, 2018, pp. 8–14. [Online]. Available: http://doi.acm.org/10.1145/3278142.3278149

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

[16] A. Bosu, J. C. Carver, M. Hafiz, P. Hilley, and D. Janni, "Identifying the characteristics of vulnerable code changes: An empirical study," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 257–268. [Online]. Available: https://doi.org/10.1145/2635868.2635880

[17] 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

[18] 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

[19] Anonymous, "Dataset for Paper," 1 2021. [Online]. Available: https://figshare.com/s/6faa4db25ee9481c8a81

[20] W. McKinney *et al.*, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, vol. 14, no. 9, 2011.