KubeSec: Violation of Security Practices in Kubernetes Open Source Projects

Md Shazibul Islam Shamim
Tennessee Technological University, Cookeville, Tennessee, USA
Email:mshamim42@tntech.edu

***Abstract*—Kubernetes is an open-source software system for automatically deploying, scaling and managing containerized applications. Kubernetes provides robust feature sets enabling organizations to manage their containerized applications. However, Kubernetes deployment can be vulnerable to security attacks due to misconfiguration and not adhering to best security practices. For secure container deployment and management of software applications over time, identification and following secure practices is pivotal. *The goal of this report is to help practitioners identify the violation of security practices in their Kubernetes configuration files by conducting an empirical study of Kubernetes open source projects.* I apply qualitative analysis on 1,086 Kubernetes configuration files used in 45 Kubernetes open-source repositories to identify the violations of security practices. Next, I build a static analysis tool called KubeSec that automatically identifies the violation of security practices in Kubernetes open source repositories. I apply KubeSec in 12,699 kubernetes configuration files collected from 875 Kubernetes open source repositories.**

I find 8 categories of violation of security practices that include violation of least privilege and insecure HTTP. I find total number of 6,548 instances of violation of security practices from 13785 Kubernetes configuration files used in 920 repositories from Github and Gitlab, among which 2711 instances are exposed secrets. I find 95.6% Gitlab repositories among 45 repositories and 99.43% Github repositories among 875 repositories has at least one instance of security practice violation.

I. Introduction

Kubernetes is an open-source software system for automatically deploying, scaling and managing containerized applications [1]. Kubernetes is considered one of the fastest growing open-source software in the history of open-source software [2]. Kubernetes provides automated solutions and resolves many of the manual processes related container deployment and management over time. Organizations find Kubernetes as a great addition to their system as it improves release frequency. Kubernetes official page reports that organizations such as the Department of Defense have been able to reduce their release effort from eight months to one week [3], [4].

Despite reported benefits, security is one of the prime concerns for Kubernetes practitioners. Anecdotal evidence from recent incidents and survey reports suggests the legitimacy of the practitioners' concern. For example, in 2018, hackers get access to the electric car manufacturer company Tesla's AWS server using Kubernetes console that was not password protected [5]. The Cloud Native Computing Foundation conducted a survey among 1324 practitioners in May June in 2020 and reported 32% practitioners think security is the major challenge in Kubernetes deployment [6]. The State of Kubernetes and Container Security Report 2020 published by Stackrox suggests that 44% organizations delayed their containerized application deployment due to security concerns. The report also states that 94% of organizations face security incidents in the last 12 months and 69% of them are misconfiguration related security issues [7].

One of the major challenges for the practitioners face while working with Kubernetes is that they think Kubernetes is a complex software. A survey among 152 participants in Kubcon+CloudNativeCon 2018 conference reported that 53% find lack of expertise as one of the major challenges in managing Kubernetes [8]. Hence, following secure practices and identifying insecure practices in existing Kubernetes deployments is crucial. However, Kubernetes security is a less explored research area. In the earlier works researchers conducted study on internet artifacts to derive categories [9] but there is no prior research works on how developers actually follow the security practices in their projects. In this work we try to address the research gap.

The goal of the report is to help practitioners identify the violation of security practices in Kubernetes configuration scripts by conducting an empirical study of Kubernetes open source projects.

We answer the following research questions:

- **RQ$_1$ (Categorization)**: *What security practices are violated by developers in Kubernetes open source projects?*
- **RQ$_2$ (Frequency)**: *How frequently the developers violate security practices in their Kubernetes open source projects?*

I did a qualitative study named open coding on 1086 Kubernetes configuration files used in 45 OSS Kubernetes repositories from Gitlab to derive what security practices are violated by developers in Kubernetes OSS projects. Next I build a static analysis tool called KubeSec that automatically identifies the violation of security practices in Kubernetes OSS projects. I apply KubeSec to see violation of security practices on 875 repositories collected from Github.

Contributions: I list my contributions as following:

- A filtered list of categories from previously reported security practices [9] in Kubernetes open source projects that developers do not follow;
- An evaluation of violation of security practices in open source Kubernetes projects. (Section IV-B);

- An empirically-validated static analysis tool called kubesec that can identify the violation of security practices in Kubernetes projects. (Section III).

I organize the rest of the report as follows: I describe the methodology and answer for RQ_1 in section II. In section III, I described methodology and construction of the KubeSec tool. In section IV, I answer RQ2 and discuss my findings. I discuss the implication of my project work for practitoners and researchers in section V. I describe the limitations of my study and KubeSec tool in section VI. I conclude the discussion of the report in section VII.

II. RQ_1: Violation of Security Practices

In this section, we answer **RQ_1: *What security practices are violated by developers in Kubernetes open source projects?***

A. Methodology for RQ_1

Qualitative analysis: In previous study, researchers derived a list of security practices after qualitative analysis of 104 internet artifacts [9]. They derived 11 security practices that practitioners recommend to follow. In this study, I take those 11 security practices and apply a qualitative analysis technique called open coding [10]. I hypothesize that we can identify the violation security practices in Kubernetes projects with qualitative analysis. I first look at the codes for Kubernetes declarative configuration files and extract code snippets to separate out from the raw scripts. I perform open coding on 1086 Kubernetes configuration files on 44 open source Kubernetes repositories collected from Github where each has at least 3 stars. I look at the Kubernetes configurations file which primarily consists of YAML files with .yaml or .yml extensions. As I have already collected the predefined category of security practices from earlier research work [9]. I check the configuration files which security practices developers can apply in their Kubernetes repositories. I find 4 of the 11 security practices are not detectable at compile time with static analysis tools. I synthesize patterns from the code bases for each of security practices implementation. I take reference from Kubernetes official documentation on how the secure practices should be applied in declarative configuration [1].

B. Answer to RQ_1

After applying open coding on 1086 files from 45 Gitlab Kubernetes repositories, I find 7 security practices that are often violated by the developers. I also find one additional category of violation for secure practices namely exposed secrets for not using encrypted secrets. I provide definitions, descriptions, each of the violation of security practices below:

I. Violation of least privilege: Practitioners recommend using RBAC in Kubernetes cluster so that any user can not perform unauthorized activities inside Kubernetes cluster. If any developer does not follow this practice then it also violates principle of least privilege principle 'CWE-250 Execution without Unnecessary Privileges' [11]. Kubernetes provides a built-in feature to implement RBAC to restrict users to perform activities inside the Kuberentes cluster.

II. Use of default Namespace: In Kubernetes, a namespace is a logically isolated cluster inside a physical cluster. Creating a namespace logically isolates resources among multiple teams. If any developer does not use the namespace for a resource then the resource falls into default namespace. If any malicious user gets access to default namespace and there is no other namespace other than default namespace then the entire Kubernetes cluster can be vulnerable to attack.

III. Insecure HTTP: Practitioners recommend enabling SSL and TLS certificates in Kubernetes components and using http with TLS support. The use of HTTP without TLS makes the communication between two Kubernetes components less secure. Using HTTP with TLS ensures that the exchanged data is encrypted. Earlier research shows that using HTTP without TLS can cause man in the middle attack. [12]. The violation of this security practice is related to 'CWE319: Cleartext Transmission of Sensitive Information' [13].

IV. Exposed Secret: This violation of security practices includes exposing the following three credentials: hard-coded or base64 encrypted (i) usernames, (ii) passwords, and (iii) security tokens, such as API tokens in Kubernetes declarative configuration files. Exposure of these credentials in configurations files can make the system vulnerable to attack. The category is related to 'CWE-798: Use of Hard- coded Credentials' [14] and 'CWE-259: Use of Hard-coded Password' [15].

V. Unbounded resource limit: Practitioners recommend using limits to CPU and memory to a pod or a container or a namespace. If any user does not define resource limit in the declarative kubernetes configuration then the resource will start with unbounded memory requests/limits and unbounded CPU access [9]. The violation of this security practice can lead to denial of service attack if any hacker compromises any resource such as container, pod or namespace.

VI. Insecure Pod configuration: Practitioners recommend using proper pod security policy to secure a pod and the containers within itself [9]. Without proper pod security configuration pods can run with root privilege. If any pod runs as a root then it can make the entire cluster vulnerable to attack.

VII. Improper update policy: Practitioners suggest updating containers with multiple replica sets on a rolling basis [9]. If the update strategy is not defined then by default all containers will get updated instantaneously. In that scenario, if any malicious user compromises a container, or any developer misconfigured a container then the Kubernetes will be susceptible to attack.

VIII. Undefined Network Policy: The practitioners suggest applying network policy to protect the Kubernetes cluster components from any undesirable communication [9]. If developers do not define Kubernetes Ingress and Egress policy, then the entire cluster can become vulnerable [9]. Without a restrictive Egress policy secure metadata can be compromised with users and without ingress policy any pod can communicate with all other pods in the cluster. Any malicious user can capitalize on the vulnerability of the undefined or poorly defined network policy.

Answer to RQ$_1$: We identify 8 categories of security practice violation in Kubernetes repositories: violation of least privilege, use of default namespace, insecure HTTP, exposed secret, unbounded resource limit, insecure pod configuration, improper update policy, and Undefined network policy.

III. Security Static Analyzer for Kubernetes (KubeSec)

KubeSec is a static analysis tool that can identify the 8 identified violations of security practices in Kubernetes configuration files. The user of the tool will provide the input directory of the Kubernetes projects in the 'main.py' file. After running 'main.py' the KubeSec tool will run analysis based on the rules and patterns on the Kubernetes configuration files and write all the detected violations of security practices in a CSV file and also print the aggregated results in the terminal console.

A. Kubesec's Security Practice Violation Detection Process

Kubesec detects violation in two steps:

Parsing: The parser is a plain yaml file parser that parses the yaml file with PyYaml [16]. The parser receives a Kubernete configuration file path, parses it and returns the parsed file as a dictionary object in python. The Parser finds a single value or all values for a specific key. Additionally, the parser returns all the keys or values of a given nested dictionary representation of the yaml file.

Rule Execution: kubesec executes a set of rules and patterns to identify security practice violation instances. To generate rules for each security practice violation category, I abstract coding patterns associated with each category. The rules are listed in Table I. To execute the rules provided in Table I kubesec uses pattern matching, similar to prior work [17], [18]. The patterns needed to execute the rules are listed in Table II. For example, to execute the rule for the 'use of default namespace', Kubesec tool first checks $isNamespace(x)$ if the key is '$namespace$' then it checks the value of x whether it is 'default' by using the pattern '$isDefault(x.value)$'. After successful match, KubeSec tool enlists a new occurance of 'use of default namespace' in the confuguration file and the repository. The pattern of strings used in rules are described in Table II.

IV. RQ$_2$: Security Practices Violation Frequency

In this section, we answer **RQ$_2$: *How frequently the developers violate security practices in their Kubernetes open-source projects?***

A. Dataset Collection

We answer RQ$_2$ by mining open source repositories. I along with the 45 repositories from Gitlab, collect repositories from Github as well. Firstly, I get the list of 1000 search results from Github for the repositories that have Kubernetes and Microservice keywords. After that I write an automatic github repository downloader and download the repositories from github. I download all the repositories in September 2020. During the time of the dataset collection, I eliminate 125 repositories as they have name conflicts. Finally I end up with 875 repositories. I collect 51 random repositories from these 875 repositories for sanity check.

Attributes of the collected repositories are available in Table III. Altogether I download 875 repositories by cloning the master branches on September 2020.

B. Answer to RQ$_2$

Altogether, I identify 6,548 instances of violation of security practice. The percentage of violation security practices in Gitlab, Github sample dataset and Github dataset varies between 95.6%$\sim$100% and percentage of insecure Kubernetes configuration files compared to total Kubernetes configuration files vary between 12%$\sim$17% as shown in the 'Combined' row in Table V. The most frequently violated security practice category is 'Exposed Secret'. A complete breakdown of our findings is provided in Table IV.

Answer to RQ$_2$: We identify 6,548 instances of violation of security practices in 875 Kubernetes repositories that include 2711 exposed secrets.

V. Discussion

In this section, I discuss the implications of our findings for practioners and resarchers below:

Implications for practitioners: I advise practitioners to apply the following measures to resolve and mitigate the violation of security practices to avoid potential security breach:

Violation of least privilege: I recommend using 'Role Based Access Control(RBAC)' authorization after authentication for all the users who interact with the Kubernetes system.

Use of default namespace: I recommend following good security practice such as using separate namespace for different teams and workspaces and avoiding '$default$' namespace.

Insecure HTTP: I suggest following best security practice to establish secure communication channel between Kubernetes components by setting up HTTP with SSL/TLS for all Kubernetes components. I also suggest that the practitioners should periodically check for the existence of insecure HTTP endpoints, as a URL that uses HTTP with TLS now can later be changed to HTTP without SSL/TLS.

Exposded secret: Practitioners can apply static analysis tools, such as CredScan [19] and Kubesec to detect exposed secrets in Kubernetes configuration files. I also suggest using security best practice such as using a vault for additional security.

Unbounded resource limit: I recommend following standard security practice for pod or any resource configuration such as specifying memory requests, limits and CPU usage before depoloyment of applications.

Insecure pod configuration: Practitioners strongly suggest to use secure context for a pod as good security practice. Practitioners also suggest running containers inside a pod as a

TABLE I: Rules for detecting violation of security practices in Kubernetes OSS repositories

Violation of Security Practices	Rule
Violation of least privilege	$isKind(x) \wedge isRBACobject(x.value)$
Use of default namespace	$isNamespace(x) \wedge isDefault(x.value)$
Insecure HTTP	$isKey(x) \wedge isHTTP(x.value)$
Exposed secret	$isKey(x) \wedge (isUser(x) \vee isPassword(x) \vee isToken(x))$
Unbounded resource limit	$(isKind(x) \wedge isPod(x) \wedge (isKey(x) \wedge (isSpec(x) \vee isContainer(x)) \wedge isLimitResources \wedge (isLimitMemory \wedge isLimitRequests)))$
Insecure pod configuration	$isPod(x) \vee isPodSecurityPolicy(x) \wedge ((isSecurityContext(x) \wedge (isPrivilegeEscalation(x) \wedge isTrue(x.value)) \vee (isRootPrivilegedValue(x) \wedge isTrue(x.value))$
Improper Update Policy	$isReplica(x) \wedge isReplica(x.value) > 0 \wedge ((isStrategy(x) \wedge isRollingUpdate(x)) \vee (isStrategyType(x) \wedge isRollingUpdate(x)))$
Undefined Network Policy	$(isKind(x) \wedge (isNetworkPolicy(x) \vee isIngressKind(x) \vee isEgressKind(x))) \vee (isKey(x) \wedge (isIngress(x) \vee isEgress(x)))$

TABLE II: String patterns used for functions in rules

Function	String Pattern
$isKind()$	'kind'
$isRBACobject()$	'Role','RoleBinding','ClusterRole', 'ClusterRoleBinding'
$isNamespace()$	'namespace'
$isDefaultValue()$	'default'
$isHTTP()$	'http:'
$isPassword()$	'password'
$isToken()$	'key'
$isUser()$	'user'
$isPod()$	'Pod'
$isSpec()$	'spec'
$isContainer()$	'container'
$isLimitResources()$	'resources'
$isLimitMemory()$	'limits'
$isLimitRequests()$	'requests'
$isPodSecurityPolicy()$	'PodSecurityPolicy'
$isSecurityContext()$	'securityContext'
$isPrivilegeEscalation()$	'allowPrivilegeEscalation'
$isPriviledged()$	'privileged'
$isReplica()$	'replicas'
$isStrategy()$	'strategy'
$isStrategyType()$	'type'
$isRollingUpdate()$	'rollingUpdate'
$isNetworkPolicy()$	'NetworkPolicy'
$isIngressKind()$	'Ingress'
$isIngress()$	'ingress'
$isEgressKind()$	'Egress'
$isEngress()$	'egress'

TABLE III: Dataset attributes

Attribute	Gitlab	GitHub sample	Github
Repositories	45	51	875
Kubernetes files	1086	1980	12699

non-root user with read-only permission and enabling Linux security modules to ensure container security.

Improper update policy: It is a standard practice to use rolling update strategy for container deployment and updating Kubernetes as well. I suggest using rolling update strategy for updating containers inside a pod or upgrading the Kubernetes cluster.

Undefined Network Policy: I recommend defining a strict Network Policy by following best security practice to restrict any undesirable communication inside a cluster. I also suggest using '$Ingress$' and '$Egress$' policy for Kubernetes cluster as a good security practice.

From Table V, I find the violations of security practices to exist in both Gitlab and GitHub repository. The percentage suggest that most practitioners does not follow the security practices. One possible reason can be that they does not know the evil consequences of the using bad practices or misconfigurations in their Kubernetes configuration files. Prevalence of existing bad practices in Kubernetes configuration files could potentially trigger the propagation of these malpractices onwards, as new practitioners may perceive bad practices in OSS projects as an acceptable practice. The most frequent type of security practice violation is exposed secret: credentials that applications, containers use to run the applications in the Kubernetes cluster. I find that 99.43% repositories from Github has atleast one security practice violation. This alarming percentage indicates that practitioners lack adequate security education and security awareness. This hypothesis is also consistent with earlier survey result [8], [6] that practitioners may not understand Kubernetes tool and they just try to make it work anyways possible by running commands and using sample configurations from OSS projects. Based on the findings, I advocate for detection and mitigation of the best security practice violation in the early stages of development. As Kubernetes itself performs the role of an orchestrator, any security compromise in Kuberentes deployment can make the entire application susceptible to attacks [20]. My tool Kubesec can be helpful to locate where the violation of security practices appear.

Implications for researchers: This project report can provide initial groundwork to conduct empirical studies that will investigate the identification and mitigation of security practice violation. Researchers [21]–[23] have reported multiple reasons on why practitioners introduce malpractices, e.g., lack of education, tool complexity, and unreliable knowledge sources. Future research can investigate to what extent these reasons are applicable for Kubernetes configuration fikes. Furthermore, researchers can investigate if security practices are neglected in most open source repositories in Github and Gitlab then this malpractices can propagate as other practitioner might find these practices as a reliable practice. Based on the findings of results of Table IV evidence reported in Section IV-B, I advocate static and dynamic analysis of Kubernetes environment and also suggest cyber security awareness of Kubernetes

TABLE IV: Answer to RQ_2: Frequency of Violation of security practices in Kubernetes OSS repositories.

ICP Name	Gitlab Sample(45)	GitHub Sample(51)	Github(875)
Violation of least privilege	36	50	831
Use of default namespace	15	11	262
Insecure HTTP	256	79	1013
Exposed secret	408	368	2303
Unbounded resource limit	19	15	345
Insecure Pod configuration	43	17	386
Improper update Policy	62	6	402
Undefined Network Policy	51	11	154
Combined	890	557	5696

TABLE V: Answer to RQ_2:Violation of Security Practices percentage in Kubernetes OSS repositories and files

	Gitlab Sample		Github Sample		Github	
Violation of Security Practices	**Insecure**	**Total**	**Insecure**	**Total**	**Insecure**	**Total**
Number of repositories	43	45	51	51	870	875
Percentage of insecure repositories		**95.6%**		**100%**		**99.43%**
Number of configuration files	131	1,086	238	1980	2166	12699
Percentage of insecure configuration files		**12.06%**		**12.02%**		**17.06%**

practitioners.

VI. Threats to Validity

I discuss the limitations of the CSC-6903 KubeSec project in this term paper as following:

Conclusion Validity: The derivation of violation of security practice categories, corresponding file types in which each category is located, and the rules and patterns for each security practices are limited to the YAML file types only with .yaml or .yml extension. I constructed the security practice implementation patterns based on Kubernetes documentation guideline. It may happen that there are some other way of defining the same security practices which I did not consider. It is also possible that my rules are too specific and in most of the cases it causes a mismatch and eventually results in a high percentage of violation of security practices with KubeSec as I report in Section IV-B. It is possible that my tool generates a lot of false positive because of my strict rules as I ruled out the possibilities the other way of implementation. I also ruled out any possibility to configure or apply the security practices imperatively during runtime. Overall, the derived 8 categories are prone to my bias we could be mitigated by allocating another rater. The possibility of higher false positive result can be my inexperience and limitation of knowledge in Kubernetes configuration and how the entire Kubernetes ecosystem works. I hope to overcome this limitations in subsequent version of this project work.

I acknowledge that kubesec may generate false positives when applied on other datasets as I did not perform any sanity check or evalution process other than running on the entire 875 repositories. Although I do randomly separated 51 repository as a sample of entire repository, it's purpose is not for sanity check rather saving time by running KubeSec on a relatively small dataset.

External Validity: My datasets are constructed by mining OSS repositories. My findings may not generalize for proprietary datasets. Also, I did not curate the datasets based on filtering criteria such as number of commits, number of developers, age of the project etc. Hence, the results may not depict original violatin of security practices as stated. In future, we will use microservices constructed using other tools, such as the Typescript-based Loopback [24] tool.

Internal Validity: I did not construct any oracle or sanity dataset so the outcomes can be biased. I try to use random 51 repositories from github repositories for sanity check. My construction of the sanity dataset does not represent accurate sample of the orginal evaluation datasets from github.

VII. Conclusion

Violation of security practices in kubernetes can provide malicious users an opportunity to conduct attacks that can create serious consequences. I conduct an empirical study to find whether practitioners use security practices in their Kubernetes installations. I find 8 categories security practice violation: violation of least privilege, use of default namespace, insecure HTTP, exposed secret, unbounded resource limit, insecure pod configuration, improper update policy, and Undefined network policy. I construct and validate a static analysis tool called Kubesec to automatically identify security practice violation in Kubernetes repositories. Using Kubesec, I analyze 13,785 Kubernetes configuration files used 920 repositories overall and identify 6,548 security practice violations. I observe number of insecure files in Kubernetes configuration files on average to vary between 12%$\sim$17%.

I advocate for rigorous inspection efforts and application of static analysis tools, such as Kubesec so that violation of security practices can be detected before they are being deployed for production. I hope this work will advance the domain of Kubernetes security research.

References

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

[2] C. N. C. Foundation, "Cncf kubernetes project journey report." [Online]. Available: https://www.cncf.io/cncf-kubernetes-project-journey/

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

[4] With Kubernetes, the U.S. Department of Defense Is Enabling DevSecOps on F-16s and Battleships, May 2020. [Online]. Available: https://www.cncf.io/case-study/dod/

[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] CNCF SURVEY 2020, November 2020. [Online]. Available: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf

[7] Stackrox Kubernetes Security Report 2020, February 2020. [Online]. Available: https://www.stackrox.com/post/2020/02/5-surprising-findings-from-stackroxs-latest-kubernetes-security-report/

[8] Kubecon 2018, March 2019. [Online]. Available: https://containerjournal.com/topics/container-ecosystems/survey-finds-lack-of-kubernetes-expertise-hindering-adoption/

[9] 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.

[10] J. Saldana, *The coding manual for qualitative researchers*. Sage, 2015.

[11] CWE, "Cwe-250: Execution without unnecessary privileges," https://cwe.mitre.org/data/definitions/250.html, 2020, [Online; accessed 01-December-2020].

[12] E. Rescorla, "Http over tls," 2000.

[13] CWE, "Cwe-319:cleartext transmission of sensitive information," https://cwe.mitre.org/data/definitions/319.html, 2020, [Online; accessed 1-December-2020].

[14] ——, "Cwe-798: Use of hard-coded credentials," https://cwe.mitre.org/data/definitions/798.html, 2020, [Online; accessed 1-December-2020].

[15] ——, "2020 cwe top 25 most dangerous software errors," https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html, 2020, [Online; accessed 2-December-2020].

[16] Yaml, "PyYAML 5.3.1," https://pypi.org/project/PyYAML/, 2020, [Online; accessed 18-August-2020].

[17] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 164–175. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00033

[18] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams, "Security smells in ansible and chef scripts: A replication study," *ACM Trans. Softw. Eng. Methodol.*, 2020, to appear. pre-print: https://arxiv.org/pdf/1907.07159.pdf.

[19] Microsoft Corporation, "Microsoft/credscan," https://secdevtools.azurewebsites.net/helpcredscan.html, 2017, [Online; accessed 23-August-2020].

[20] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, *Microservices: Yesterday, Today, and Tomorrow*. Cham: Springer International Publishing, 2017, pp. 195–216. [Online]. Available: https://doi.org/10.1007/978-3-319-67425-4_12

[21] N. Meng, S. Nagy, D. D. Yao, W. Zhuang, and G. A. Argoty, "Secure coding practices in java: Challenges and vulnerabilities," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: ACM, 2018, pp. 372–383. [Online]. Available: http://doi.acm.org/10.1145/3180155.3180201

[22] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You get where you're looking for: The impact of information sources on code security," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 289–305.

[23] S. Nadi, S. Kruger, M. Mezini, and E. Bodden, "Jumping through hoops: Why do java developers struggle with cryptography apis?" in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 935–946. [Online]. Available: https://doi.org/10.1145/2884781.2884790

[24] StrongLoop, "Loopback," https://loopback.io/, 2020, [Online; accessed 24-August-2020].