Automated Security Repair for Helm Charts

Minna, Francesco; Blaise, Agathe; Massacci, Fabio; Tuma, Katja

published in
ICSE-Companion '24
2024

DOI (link to publisher)
10.1145/3639478.3643534

document version
Publisher's PDF, also known as Version of record

Automated Security Repair for Helm Charts

Francesco Minna
f.minna@vu.nl
Vrije Universiteit Amsterdam
Netherlands

Agathe Blaise
agathe.blaise@thalesgroup.com
Thales SIX GTS France
France

Fabio Massacci
fabio.massacci@ieee.org
Vrije Universiteit Amsterdam, University of Trento
Netherlands, Italy

Katja Tuma
k.tuma@vu.nl
Vrije Universiteit Amsterdam
Netherlands

ABSTRACT

We aim to evaluate and compare open-source static analyzers for Helm Charts, a package manager to deploy applications on Kubernetes (K8s). Specifically, we developed a pipeline to measure what misconfigurations are found by each tool, to provide automatic misconfiguration repair, and whether this latter breaks application functionalities. To evaluate our approach, we analyzed the 60 most common Helm Charts available on Artifact Hub, seven open-source Helm Charts analyzers, and generated functionality profiles for each chart application. We found several bugs and inconsistency issues with the tools, which we reported on respective tool repositories, and concluded that such tools that should provide automatic security repair still require significant manual intervention.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; *Reliability*; • **Software and its engineering** → *Software configuration management and version control systems*; **Command and control languages**; **Software testing and debugging**; • **Security and privacy** → **Distributed systems security**; *Software security engineering*.

KEYWORDS

Helm Charts, Automated Security Repair, Kubernetes, Misconfigurations

ACM Reference Format:
Francesco Minna, Agathe Blaise, Fabio Massacci, and Katja Tuma. 2024. Automated Security Repair for Helm Charts. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643534

1 RESEARCH PROBLEM AND MOTIVATION

The Helm project [3] is the most popular tool for packaging, sharing, and deploying applications on Kubernetes (K8s) [1], using sets of YAML configuration files called charts, widely shared through Artifact Hub. A recent report showed that over two-thirds of the chart's repositories had security misconfigurations [4, 5], and because containers only live for a few minutes, the time window for effective run-time detection is very limited. Therefore, static security analysis of Helm Charts becomes important, and several industry-level tools (e.g., Checkov, Datree, and Terrascan) have been developed to analyze Helm Charts, based on industry security best practices (e.g., K8s CIS Benchmarks [2]). We are interested in evaluating whether different analyzer tools provide consistent results on the same chart and whether intuitive mitigation (e.g. remove root access) might break the application. Also, after analyzing the policies used by the existing tools, we have not found any guidelines for remediation or possible side effects on the application functionalities. Therefore, we are also interested in providing an automated security repair methodology for Helm charts, based on existing tool findings. For example, we are interested in evaluating whether a 0 container memory requests and limits and an empty network policy (i.e., without ingress or egress rules) will satisfy a tool's policies.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3643534

2 BACKGROUND

Kubernetes (K8s) is a container orchestration engine, that allows to efficiently manage, deploy, and scale applications with hundreds of containers using YAML configuration files (i.e., IaC files). Helm is a package manager for K8s that allows to define, manage, share, version, and deploy applications as a set of directories and YAML files, called charts. These charts can contain misconfigurations that can pose security risks once deployed; for example, over-privileged containers, containers running as root, or containers without memory and CPU limits and requests defined. Therefore, several open-source and proprietary tools have been developed to detect such misconfigurations in Helm charts based on best practices, such as the Kubernetes CIS Benchmark and the NSA Kubernetes Hardening Guide. For example, the Checkov tool has a policy to check that each K8s object (e.g., Deployment, Service, ServiceAccount, etc.) does not use the default K8s namespace. Automated program repair (APR) techniques aim to automatically identify patches for a given error or bug, minimizing human intervention. The goal of this paper is to implement APR for Helm charts, based on existing open-source analyzer tools results.

3 APPROACH

Figure 1 shows the workflow of the proposed pipeline that takes as input a Helm Chart and an analyzer tool, performs automated repair based on the tool output, generates a functionality profile, and finally, generates an updated chart with all issues fixed and the added needed functionalities. Each step of the pipeline is further explained as follows.

Run Tool Analyzer (Step 1, and Step 3 — Debug). At step 1, and step 3 (debug) the pipeline runs a static-time tool analyzer on

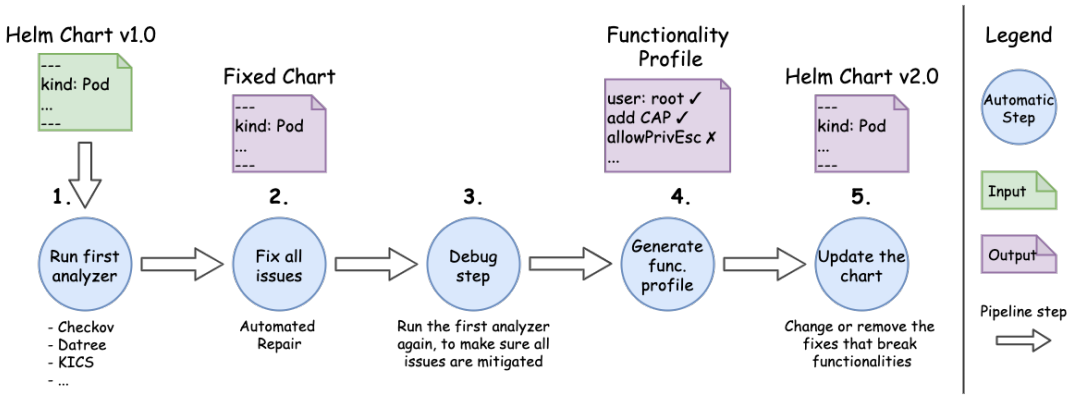

Figure 1: Pipeline workflow to find a functional configuration adhering to the principle of least privilege for a Helm Chart.

the Helm Chart template. Specifically, at step 1, the pipeline runs the tool analyzer on the original chart template; after fixing all the misconfigurations at step 2, and before generating and adding the functionalities, the pipeline runs the same tool again as a control (debug step), to make sure all the misconfigurations were correctly mitigated at the previous step, i.e., the number of misconfigurations equals to 0.

Automated Misconfiguration Fixing (Step 2). At step 2, the pipeline parses the output of the tool previously run and automatically fixes all the detected misconfigurations. To this aim, for each tool's policy, we developed corresponding mitigations that satisfy the policy. YAML files of a Helm Chart can be interpreted as hash maps, therefore mitigation corresponds to either adding or removing a key or changing the value of a key. Whenever values can not be determined statically (e.g., memory and CPU limits) we use random values that still satisfy the tool's policies.

Generate Functionalities (Step 4). In this step, we build a functionality profile for a Helm chart application; this profile should list all functionalities and permissions that are required by one or more containers in the chart to properly function. A *functionality profile* is a profile bound to a specific container that lists only the privileges required by the application to function properly. To generate such a profile, we first deploy the Helm Chart in its initial configuration as the ground truth. Then, we iteratively remove one permission at a time, deploy the container again, and check whether it can start, K8s probes are ready, and whether the logs are similar to the logs of the ground truth. If all of these tests are passed, we assume the given permission is unnecessary and can be removed, otherwise, it is required.

Update the Chart (Step 5). At step 5, the pipeline parses the functionality profile previously generated and correspondingly updates the chart. This step assumes that a functionality profile for the chart has already been computed and is available in the pipeline. Similarly to step 2, this step consists of updating the hash map resources in the Helm Chart, by adding or removing keys, or updating values, based on each required functionality. The output of this step will be an updated Helm chart template, that can be further analyzed or used to deploy the application on a K8s cluster.

4 PRELIMINARY RESULTS AND CONCLUSIONS

For a preliminary validation, we retrieved the 60 most common Helm Charts from Artifact Hub [6] and compared the results of seven open-source chart analyzer tools, namely, Checkov, Datree, KICS, Kube-linter, Kubeaudit, Kubescape, and Terrascan. We found that ClusterRoles, memory limits, and default namespaces are the most common misconfigurations. Instead, regarding functionalities, we found out that using high user IDs, read-only filesystems, and the root user is the most likely to break the functionality of the Helm Chart. Finally, regarding tool performances, we found that Datree detected the highest number of misconfigurations per chart, whereas Kubelinter broke the least number of functionalities. Interestingly, we also found that a combination of tools (i.e., running Datree on the original template and KICS after adding functionalities) returns the best overall results.

In future work, we plan to evaluate more Helm charts and tool analyzers and investigate the performance of AI models in detecting misconfigurations and suggesting mitigations.

ACKNOWLEDGMENTS

This work has been partially supported by the European Union under the H2020 grant AssureMOSS (952647), HE grant 101120393 (Sec41AI4Sec), and the NWO under the grant NWA 121518006 (Theseus).

REFERENCES

[1] CNCF. 2022. CNCF 2022 Annual Survey. https://www.cncf.io/reports/cncf-annual-survey-2022/. Jan. 2024.
[2] Center for Internet Security. 2024. Kubernetes CIS Benchmark. https://www.cisecurity.org/cis-benchmarks. Jan. 2024.
[3] Helm. 2024. The Helm Project. https://helm.sh. Jan. 2024.
[4] Matt Johnson. 2021. *Top trends from analyzing the security posture of open-source Helm charts.* Technical Report. Bridgecrew. Available on the web at https://bridgecrew.io/blog/open-source-helm-security-research/.
[5] Akond Rahman, Shazibul Islam Shamim, Dibyendu Brinto Bose, and Rahul Pandita. 2023. Security Misconfigurations in Open Source Kubernetes Manifests: An Empirical Study. *ACM TOSEM* 33, 1 (jan 2023), 37 pages. https://doi.org/10.1145/3579639
[6] The Linux Foundation. 2022. Artifact HUB. https://artifacthub.io. Jan. 2024.