Automated Approach to IaC Code Inspection Using Python-Based DevSecOps Tool

1st Nenad Petrović
XLAB d.o.o.
Ljubljana, Slovenia
nenad.petrovic@xlab.si
Faculty of Electronic Engineering
University of Niš
Niš, Serbia
nenad.petrovic@elfak.ni.ac.rs

2nd Matija Cankar
XLAB d.o.o
Ljubljana, Slovenia
matija.cankar@xlab.si

3rd Anže Luzar
XLAB d.o.o.
Ljubljana, Slovenia
anze.luzar@xlab.si

***Abstract*— One of main benefits enabled by DevOps ideology is to automatize activities and operations related to development, testing, integration and deployment of software, to fulfill the needs of relevant organization's goals. On the other side, quality of code, security, together with compliance according to given standards represent highly relevant considerations. In this paper, we present an open-source Python-based tool with web-based graphical interface which enables automation of static code analysis and checks when it comes to Infrastructure as Code (IaC) scripts. The proposed tool is evaluated in several scenarios when it comes to terraform scripts.**

Keywords— DevOps, DevSecOps, IaC, Python

I. Introduction

DevOps methodology is the trending area in software engineering which strives to automatize the activities and operations related to development, deployment, continuous integration, testing and delivery of fixes, new features and enhancements to end-users aligned with business and other goals of the target organization [1]. For that purpose, numerous novel concepts and workflow automation techniques have emerged. In that context, various approaches and auxiliary tools are adopted as means, covering various relevant aspects, while Infrastructure as Code (IaC) [1, 2] is one of them.

IaC represents the concept of computing resources management (either virtual machines or network elements) leveraging descriptive models defining the structure and topology of the considered system. In that context, machine-readable scripts are adopted in order to enable the automation of related activities, instead of relying on manual configuration and direct interaction with physical hardware or even auxiliary interactive tools. Widely adopted industry-level standards and tools in this area are Ansible [3], TOSCA [4] and Terraform [5].

However, considering the fact that IaC aims describing physical deployments, it also implies many concerns related to security and other potential problems within IaC scripts, as they can be the cause of potentially damage, even with fatal consequences, especially in sensitive usage domains and scenarios. For that reason, IaC script quality and security inspection are identified among the critical steps of workflows adopting DevOps principles. Therefore, integration of security-oriented aspects in DevOps workflows has emerged into related field, referred to as DevSecOps [6, 7].

In this paper, we introduce a Python-based tool which aims to automatize IaC archive security inspection by aggregating various tools in a coherent way. Additionally, in order to speed-up the process even further, user-defined scan configurations are adopted, while the tool is accessible via REST API with basic graphical interface thanks to Swagger UI. Our tool is open-source and publicly available on Git Hub [8].

II. Background and Related Work

In our prototype, the following IaC languages are supported: Terraform, Ansible, Docker and TOSCA YAML. Additionally, auxiliary syntaxes and languages which are used to describe those services are also included, such as YAML, Java, Python, Bash/shell scripts, HTML and JavaScript. Table I gives an overview of similar tools integrating various IaC checks.

TABLE I
Related Works Summary

Tool	Description
Checkov [9]	Static code analysis tool for infrastructure-as-code targeting Terraform, AWS CloudFormation, Kubernetes, Serverless and ARM templates.
Super-linter [10]	Part of GitHub workflow which represents a simple combination of various linters
Mega-Linter [11]	Open-Source tool that analyzes consistency and quality of 47 languages, 22 formats, 18 tooling formats, abusive copy-pastes and spelling mistakes. Generates various reports and offers auto-fixes
Snyk [12]	Service aiming to continuously find and fix vulnerabilities in dependencies pulled from npm, Maven, RubyGems, PyPI and others.

Despite the fact that IaC check integration is not entirely new idea, our tool aims to provide the possibility to execute combined checks over the IaC and even include checks provided by other paid services (if the user has bought the full/premium version of some already available vulnerability scanner), while providing future-proof extendibility with new check tools. Apart from the so-called linters that check if code complies with certain standards with respect to formatting (like indentations, spaces and similar aspects), our tool also includes more sophisticated security checks that detect vulnerabilities within IaC, such as terrascan and tfsec for Terraform and Steampunk Spotter [13, 14] for Ansible.

III. Implementation

A. Working Principle Overview

High-level working principle overview of IaC Scan Runner is depicted in Fig. 1. First, user provides the desired IaC archive in compressed format which is about to be checked for vulnerabilities. After that, the archive is pre-processed in order to check the available file types which are present inside. Once the IaC content is analyzed, a compatibility matrix is used in order to filter only the relevant checks for given file types (avoid execution of irrelevant checks). Optionally, user-specific information about the preferred checks that are enabled are individual tool configurations (for paid services) is retrieved from persistence layer. This way, re-use of custom settings is enabled, so it is not required to spend time on configuration for each archive when multiple scans are performed. The final list of checks that will be actually executed is determined as intersection of several factors, as given in formula (1) below. As it can be seen, intersection of enabled (and properly configured) checks, compatible ones for the given set of file types detected within IaC archive is considered.

$$final\_checks = enabled\_checks \cap compatible\_checks(types) \quad (1)$$

Finally, once the scan workflow is finished, the logs of individual checks are summarized in order to generate output that is easily understandable by user. There are two possibilities: 1) JSON file – which is the raw output of scan tool, suitable for both machines and readable by humans 2) HTML page – tabular scan result summary visualization. Additionally, if used in persistent mode, scan results are stored into database, so they can be browsed and visualized later as well.

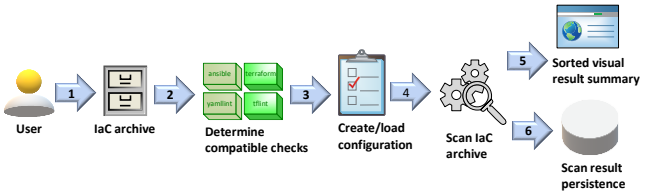

Figure 1. IaC Scan Runner general flow overview: 1 – file upload; 2 – scan archive to detect file types 3 -compatible check list 4 -final check list 5 – tool logs 6 – storing results into database

Finally, once the scan workflow is finished, the logs of individual checks are summarized in order to generate output that is easily understandable by user. There are two possibilities: 1) JSON file – which is the raw output of scan tool, suitable for both machines and readable by humans 2) HTML page – tabular scan result summary visualization. Additionally, if used in persistent mode, scan results are stored into database, so they can be browsed and visualized later as well.

Once the results are persisted, user is able to browse and filter them. In order to avoid repeating scan preference by particular user, it is possible to select the checks that will be enabled and store project configuration which can be later used. Additionally, scan results older than given number of days can be periodically deleted. Use case diagram of the tool is shown in Fig. 2.

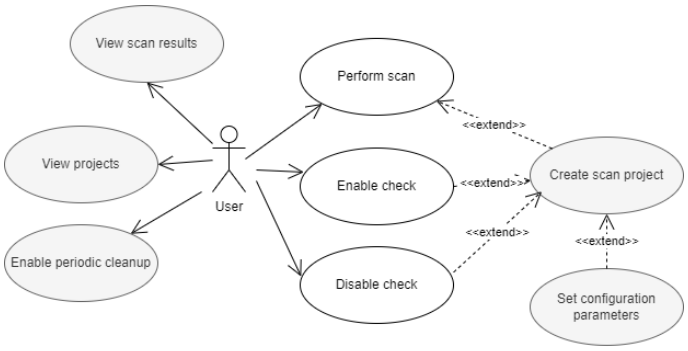

Figure 2. Use case diagram of IaC Scan Runner

B. Scan Tool Compatibility Matrix

Scan tool compatibility matrix is leveraged in order to avoid overall IaC archive scan time by avoiding non-compatible checks. It is a 2D array referred to as M_{TxC}, where rows represent various file types (denoted as *t*), while columns are relevant scan checks for that file type (denoted as *c*). In Table II, the current state of compatibility matrix used within our tool is shown. In scenario where certain file does not belong to any of the categories from the left, than no scans will be performed. On the other side, there is also a set of common checks executed for every IaC archive, such as git-leaks and git-secrets that discover disclosure of sensitive information inside Git repositories and cloc, which as an informational tool that counts files and distinguishes the detected file types.

TABLE II
Matrix of currently supported file types and checks

File type (t)	Check (c)
Terraform	tflint, terrascan, tfsec
Ansible	ansible-lint, steampunk-scanner
TOSCA/ YAML	yamllint
Python	pylint, bandit pyup-safety
HTML	htmlhint
Java	checkstyle
JavaScript	ts-lint, es-lint
Docker	hadolint
Shellscript	shellcheck
Common	git-leaks, git-secrets, cloc
Other	-

C. Data Model

When it comes to data persistence layer, the relevant data about scan results and projects configurations is stored to a MongoDB [15] document store. The underlying data model is depicted in Fig. 3.

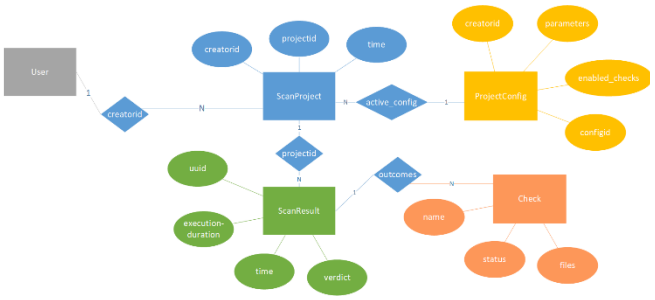

Figure 3. Simplified representation of IaC Scan Runner data model

As it can be seen, the scan results are organized by projects with common configuration. For each of the project, the relevant info consists of the following: 1) creatorid – identifier of a user who created the scan project 2) time – the moment when the project was created 3) active_config – a reference to identifier of a configuration, as each project can have a configuration assigned. At one point in time, project can have up to one project configuration active. On the other side, project configuration info consists of the following fields: 1) parameters – a dictionary of tool-specific parameters, such as secrets (usernames, passwords, access tokens) for scan checks that require additional configuration 2) enabled_checks – the list of checks that are enabled within a scan project.

D. *Scan Workflow*

Table II shows the sequence of steps representing pseudo-code of the underlying scan workflow behind IaC Scan Runner. IaC archive and response type (HTML or JSON) are obligatory user-provided input parameters, while the list of preferred checks and identifier of scan project are optional.

TABLE II
SCAN WORKFLOW PSEUDO-CODE

Input: iac_archive, selected_checks, projected, response_type
Output: HTML or JSON result summary
Steps:
1. Get archive file types list;
   file_types:= get_types(iac_archive)
2. For each ft in file_types
3. Find relevant checks for ft in compatibility matrix;
   compatible_checks.append(compatibility_matrix(ft))
4. End for each
5. Get enabled checks for given scan project;
6. enabled_checks:=get_enabled(projectid)
7. Configure paid services for project;
8. additional_checks:=configure(projecti_id)
9. Disable non-configured additional checks from enabled list
10. enabled_checks:=filter(additional_checks)
11. Determine final check list
    final_checks:= selected_checks ⋂ compatible_checks
12. For each check in final_checks
13. Run scan of IaC archive;
14. outcomes[scan][log]:=scan(iac);
15. Summarize outcome logs;
16. JSON_result:=summarize_results(outcomes);
17. Create HTML page;
18. HTML:=generate_html(JSON_result);
19. If(response_type is JSON)
20. Return JSON_result;
21. Else
22. Return HTML_result;
23. persist_result(JSON_result);
24. End

A screenshot of Swagger UI visual interface to REST API for IaC archive scanning is shown in Fig. 4.

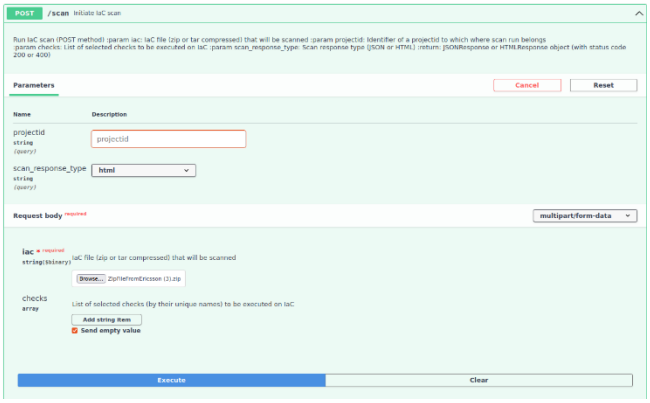

Figure 4. Swagger UI scan interface

E. *Result Visualization*

The most convenient way of result visualization fromm user perspective is HTML page. It is generated based on the JSON file and includes the mentioned fields, represented in tabular form, as shown in Fig. 5.

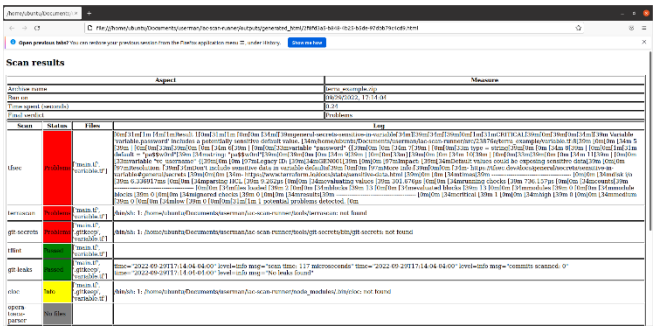

Figure 5. HTML scan result summary

As it can be seen, the page consists of the two main parts: auxiliary information and scan outcomes. The first one consists of the following fields: 1) archive – name of IaC archive file 2) run on – timestamp when scan process was initiated 3) time spent (seconds) – overall duration of IaC archive scanning 4) final verdict – the overall outcome of the scan process, returning “passed” if no problems were reported by any of the scanning tools, while it is “problems” otherwise.

On the other side, the second part deals with scan result summarization and additionally, it includes the aspects of scan outcome prioritization, taking into account the actual status of the performed scans and leveraging them in context of visualization. Therefore, the scan results will be grouped according to the outcome in given order and colored the following way: 1) problems – red color, shown firs 2) passed – green, second 3) info – yellow 4) no files – grayed out, fourth.

IV. EVALUATION

This section provides an overview of automated IaC archive scan experiments and the achieved results. For evaluation purposes, the execution was performed on a Lenovo laptop, model ThinkPad P14s Gen 2, equipped with

2.30GHz 6-core AMD Ryzen 5 PRO 5650U Processor and 16GB DDR4 RAM, running on Ubuntu 20 Linux-based operating system.

The proposed approach was evaluated on three realistic case study IaC archives with various types of files, two of them provided by PIACERE project partners and the third one from another project (not publicly available): 1) public safety applications (Ericsson) [16]; 2) National Interoperability Framework Portal (Slovenian Ministry of Public Administration) [17]; 3) deep learning cloud infrastructure tests. Table III gives the summary of experiments and obtained experimental results, considering the following aspects: file types covered, number of scanned files within IaC archive, scanning duration leveraging compatibility matrix and without it (in seconds, as average on 10 runs). The estimated time required for manual IaC archive checks (in case of expert typing tool CLI commands for individual checks) without using the proposed approach was put for comparison on the other side. The time spent for IaC archive check without relying on automated tool was order of magnitude of minute, while the automated method does not exceed 10 seconds.

TABLE III
EVALUATION RESULTS

IaC archive	Types	File Count	Scan comp. [s]	Scan [s]	Manual scan [s]
Public safety [16]	Terraform Docker JavaScript Shell script	15	1.37	1.64	120
Government [17]	Ansible JSON Terraform TOSCA YAML Shell script	130	5.13	6.69	210
Deep learning test	Ansible JSON Python YAML	56	3.91	4.93	180

V. CONCLUSION

In this paper, an automated approach to IaC archive quality and security check using Python-based tool with REST API is presented. According to the obtained results, it seems quite effective, as time required for scanning does not exceed 10 seconds, which is much faster than manual approach. On the other side, in synergy with existing IaC Scan Runner tool, the proposed methodology is significantly more efficient compared to manual usage of distinct tools for IaC archive security checks. Furthermore, the adoption of compatibility matrix reduces the time necessary for scanning more than 20% in our cases by avoiding the non-compatible checks. Moreover, it is more beneficial for IaC archives with larger number of files, leading to greater scan speed-up. In future, it is planned to extend the presented framework with additional check relying on machine learning techniques as well and integrate within PIACERE IDE.

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 101000162 (PIACERE).

REFERENCES

[1] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero and D. A. Tamburri, "DevOps: Introducing Infrastructure-as-Code," 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 497-498, 2017. https://doi.org/10.1109/ICSE-C.2017.162

[2] J. Alonso, C. Joubert, L. Orue-Echevarria, M. Pradella, D. Vladusic, "Programming trustworthy Infrastructure As Code in a Secure Framework", First SWForum workshop on trustworthy software and open source (SWForum.eu). CEUR-WS.org, vol. 2878., pp. 16-23, 2021. https://doi.org/10.5281/zenodo.6881894

[3] A. Luzar, S. Stanovnik, M. Cankar, "Examination and Comparison of TOSCA Orchestration Tools", ECSA 2020, Communications in Computer and Information Science, vol 1269. Springer, Cham, pp. 247-259, 2020. https://doi.org/10.1007/978-3-030-59155-7_19

[4] Red Hat Ansible [online], available on: https://www.ansible.com/ , last accessed: 25/10/2022.

[5] Terraform [online], available on: https://www.terraform.io/ , last accessed: 25/10/2022.

[6] R. Rajapakse et al., "Challenges and solutions when adopting DevSecOps: A systematic review", Information and Software Technology Volume 141, January 2022, 106700. https://doi.org/10.1016/j.infsof.2021.106700

[7] J. Alonso, R. Piliszek and M. Cankar, "Embracing IaC through the DevSecOps philosophy: Concepts, challenges, and a reference framework", IEEE Software, 2022. https://doi.org/10.1109/MS.2022.3212194

[8] IaC Scan Runner [online], available on: https://github.com/xlab-si/iac-scan-runner , last accessed: 25/10/2022.

[9] Checkov [online], available on: https://github.com/bridgecrewio/checkov, last accessed: 25/10/2022.

[10] SuperLinter [online], available on: https://github.com/github/super-linter , last accessed: 25/10/2022.

[11] MegaLinter [online], available on: https://nvuillam.github.io/mega-linter/ , last accessed: 25/10/2022.

[12] Snyk [online], available on: https://snyk.io/ , last accessed: 25/10/2022.

[13] Steampunk Spotter [online], available on: https://steampunk.si/ , last accessed: 25/10/2022.

[14] XLAB, The importance of high-quality Ansible Collections for your product [online], available on: https://steampunk.si/pdf/Importance_of_High_quality_Ansible_Collections_XLAB_Steampunk_ebook.pdf , last accessed: 25/10/2022.

[15] MongoDB [online], available on: https://www.mongodb.com/ , last accessed: 25/10/2022.

[16] Public Safety on IoT in 5G [online], available on: https://www.piacere-project.eu/public-safety-iot-5g, last accessed: 26/10/2022.

[17] The Slovenian Ministry of Public Administration (SI-MPA) [online], available on: https://www.piacere-project.eu/slovenian-ministry-public-administration-si-mpa, last accessed: 26/10/2022.