



PIACERE

Deliverable D4.4

IaC Code security and components security inspection –
v1

Editor:	Matija Cankar
Responsible Partner:	XLAB d.o.o.
Status-Version:	1.0
Date:	30.11.2021
Distribution level (CO, PU):	Public

Project Number:	101000162
Project Title:	PIACERE

Title of Deliverable:	IaC Code security and components security inspection
Due Date of Delivery to the EC	30.11.2021

Workpackage responsible for the Deliverable:	WP4 – Verify the trustworthiness of Infrastructure as a code
Editor(s):	Matija Cankar (XLAB)
Contributor(s):	Anže Luzar (XLAB), Giuseppe Celozzi (ERICSSON), Matija Cankar (XLAB)
Reviewer(s):	Leire Orue-Echevarria Arrieta (TECNALIA)
Approved by:	All Partners
Recommended/mandatory readers:	WP2, WP3

Abstract:	This deliverable will present the outcome of Task T4.2 and Task T4.3. The deliverable comprises both a software prototype [KR6-KR7] and a Technical Specification Report. The document will include the Security Inspector technical design and implementation aspects. The document will also include the Security Inspector technical design and implementation aspects
Keyword List:	IaC, SAST, IaC Security, DevOps, DevSecOps
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and neither Agency nor the Commission are responsible for any use that may be made of the information contained therein

Document Description

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	17.06.2021	First draft version	XLAB
v0.2	15.09.2021	Comments and suggestions received by consortium partners	ALL
V0.3	22.11.2021	Finalised document, sent to review.	Anže Luzar, Matija Cankar, Giuseppe Celozzi
V0.9	26.11.2021	Addressed review comments.	Anže Luzar, Matija Cankar
V1.0	29.11.2021	Ready for submission	Leire Orue-Echevarria

Table of contents

Terms and abbreviations.....	6
Executive Summary.....	7
1 Introduction.....	8
1.1 About this deliverable.....	8
1.2 Document structure.....	8
2 Implementation.....	9
2.1 Purpose.....	9
2.1.1 State of the art (SotA).....	9
2.2 Approach.....	11
2.3 Functional description.....	14
2.3.1 Check selection.....	16
2.3.2 Validation of the component.....	18
2.3.3 Fitting into overall PIACERE Architecture.....	18
2.4 Technical description.....	18
2.4.1 Prototype architecture.....	20
2.4.2 Component description.....	21
2.4.3 Technical specifications.....	21
3 Delivery and usage.....	23
3.1 Package information.....	23
3.2 Installation instructions.....	24
3.3 User Manual.....	26
3.3.1 REST API.....	26
3.3.2 CLI.....	29
3.4 Licensing information.....	29
3.5 Download.....	29
4 Future plans.....	30
5 Conclusions.....	31
6 References.....	32

List of tables

TABLE 1. FUNCTIONAL REQUIREMENTS [1].....	15
TABLE 2. LIST OF POTENTIAL CHECKS TO BE INCLUDED.....	17
TABLE 3. IAC SCAN RUNNER REST API ENDPOINTS.....	26
TABLE 4. CURRENTLY SUPPORTED IAC CHECKS WITHIN IAC SCAN RUNNER.....	28
TABLE 5. IAC SCAN RUNNER CLI COMMANDS.....	29

List of figures

FIGURE 1 MAIN ACTIONS FROM PIACERE SECURITY INSPECTOR 14

FIGURE 2. ACTIVITY DIAGRAM WITH IAC AND COMPONENT SECURITY INSPECTORS 19

FIGURE 3. IAC SCAN RUNNER 20

FIGURE 4. BANDIT TOOL IAC CHECK OUTPUT 21

FIGURE 5. DOCKER HUB REPOSITORY FOR IAC SCAN RUNNER 23

FIGURE 6. IAC SCANNER DOCUMENTATION PAGE HOSTED ON GITHUB PAGES 24

FIGURE 7. SWAGGER UI PAGE FOR IAC SCAN RUNNER REST API 25

FIGURE 8. REDOC PAGE FOR IAC SCAN RUNNER REST API 25

FIGURE 9. FILTERING OUT TERRAFORM CHECKS IN THE IAC SCAN RUNNER REST API..... 27

FIGURE 10. AN EXAMPLE OF IAC SCAN RUNNER REST API SCANNING OUTPUT..... 28

FIGURE 11 RUNNING IAC SCAN RUNNER API USING THE CLI 29

DRAFT

Terms and abbreviations

ASAP	As soon as possible
CI	Continuous integration
CVE	Common Vulnerabilities and Exposures
CSAR	Cloud Service Archive
CSP	Cloud Service Provider
DevOps	Development and Operations
DevSecOps	Development, Security and Operations
DoA	Description of Action
EC	European Commission
GA	Grant Agreement to the project
HCL	HashiCorp Configuration Language (Terraform HCL)
IaC	Infrastructure as Code
IEP	IaC execution platform
IOP	IaC Optimization
KPI	Key Performance Indicator
SW	Software
SAST	Static Application Security Testing
SCA	Software Composition Analysis
SotA	State of the Art
TOSCA	Topology and Orchestration Specification for Cloud Applications

Executive Summary

This is the first technical deliverable from a series of three, describing the progress and plans of the IaC Security Inspector and Component security inspector (KR6 and KR7). The mentioned results are output of tasks T4.2 and T4.3 from the WP4.

In this deliverable we are describing the IaC Security Inspector and the IaC Component Inspector which will mostly rely on a Static Application Security Testing (SAST) approach to inspect the IaC code and dependant components for security vulnerabilities. As PIACERE is focusing on multiple IaC languages and designed applications can rely on multiple components, the first innovation developed by this work package is an umbrella engine designed as a service, which can perform a set of checks with the ability to enable or disable specific security checks. The included checks can be open source, proprietary or third-party services outside PIACERE.

The second innovation will focus on checking performance on real data provided by PIACERE use case providers and DevOps experts inside the consortium. On a first level we will review the existing checks, include them in the PIACERE SAST tools and make them available to the partners. From the check tests and PIACERE partners, we will get the feedback of the combined check performance. Wherever the quality of the checks will not be satisfied, we will focus on resolving the gaps with creating new check engines, check services or updating existing solutions with open-source contributions.

With the last important innovation, we will target enterprise users, which could benefit with the SAST check management. The management of checks would allow organising a SAST project for each deployment with its own automated check rules and users would be able also to share that project between others. These features would also add an ability to notify the user when the part of the application becomes vulnerable at runtime because a new exploit was discovered and published.

The presented innovations are the mission of this work package, and this deliverable describes the current state and progress – overall, we mostly covered the first innovation part and we set the foundation for other two. The document first describes the purpose and functional description of the proposed solutions, followed by the explanation of the solution and ending with the current results. Currently, the core framework of the solution, called IaC Scan Runner, is already created as a Proof of Concept (PoC) and accessible to other partners for testing. In the future we will expand it with more functionalities in case of content (more security/component checks) and usability (e.g., user management). The new additions will allow better integration and easier exploitation of the results.

The next version of this deliverable will be submitted in month 24, which will be in November 2022.

1 Introduction

This is the first technical deliverable from a series of three, describing the progress and plans of the IaC Security Inspector and Component security inspector (KR6 and KR7). The mentioned results are output of tasks T4.2 and T4.3 from the WP4.

The task T4.2 focuses on statical code analysis (SAST) of the IaC which will be generated from the PIACERE DOML (DevOps Modelling Language). The focus of this task is to find and create a set of useful code checks for different IaC options (e.g., Terraform HCL, TOSCA Simple Profile in YAML, Ansible...) and issues such as typos, syntactical problems, and secret management. An example for this is a check if the IaC include hard coded passwords.

The task T4.3 focuses on detecting the components used by the IaC. This means that IaC is inspected for the dependencies and their versions, which might have vulnerabilities like zero-day exploits. An example of this check would be if the OpenSSL library referred in IaC code is mature enough to not have a heart-bleed¹ vulnerability.

The approach of both tasks is different, but if we consider them as a black box, both have a similar behaviour – accepting IaC as an input and providing the list of vulnerabilities in the form of configuration errors, warnings, and suggestions in the output. Based on this observation we will develop and describe them jointly to ease the integration instructions and provide detailed instructions of how the checks can define the functionality of one or another.

1.1 About this deliverable

The deliverable will explain the scenarios behind the PIACERE static security inspection services and their goal, which is to find the vulnerabilities in the deployable applications and therefore provide the better IaC without any security issues. The document will present the initial architecture plan and requirements defined in Deliverable D2.1 [1] for the PIACERE IaC Security Inspector and Component Security Inspector. It will also describe how the first versions of the tools can be used to provide IaC security and current possible integrations.

1.2 Document structure

The rest of this document is structured as follows:

- Section 1 presents an overall description of the delivery and its main goal is provided.
- Section 2 focuses on state of the art, purpose, the implementation details, functional requirements along with validation and technical description of the product.
- Section 3 describes the delivery and usage of the developed tools.
- Section 4 lists the ideas for future improvements.
- Section 5 summarizes the achievements of this deliverable and draws the conclusions.
- Section 0 lists references and citations used through the document.

¹ <https://en.wikipedia.org/wiki/Heartbleed>

2 Implementation

2.1 Purpose

The rise of usage of cloud resources lead to the use of DevOps and DevSecOps paradigms for faster and better deployment of the applications. The idea bases on describing the deployment and configuration specifications of an application in a code. At first practitioners created their custom deployment scripts, which were usually not systematically tester or reviewed. The use of common tools for deployment and orchestration lead to the infrastructure as code (IaC) principle. The use of IaC provides a scalable and automatable environment defined as a code, which enables code versioning, tracking of changes, code reviews and easy rollbacks. Additionally, users can benefit with less intervention in the time of deployment and reusability of the IaC parts published in catalogues [2] [3]. These benefits allow DevOps engineers to quickly develop a new IaC and empower them with ability to reconfigure networks, change software components and scale the infrastructure with small changes in the code.

The benefits of applying IaC are important, as they accelerate business processes, control costs, reduce some involved risks and tighten the security. However, resolving some threats and issues, create new ones – the ease of configuration changes brings potentially dangerous mistakes, miss-configurations, ability to include an outdated IaC component. The ease of change can result in many hidden security vulnerabilities in applications or underlying infrastructure. The danger of IaC misuse is so high, that some enterprises do not use it yet and devaluates the trust in the power of IaC, for example according to the TechRepublic Divvy research found that data breaches due to the cloud misconfiguration cost \$5 trillion in 2018-2019 [4].

To gain the trust in the use of IaC we need to create appropriate tools, to help DevOps Engineers minimising the potential issues with using their code, which should be done in two folds. First is by using static application security testing (SAST) tools on the IaC code which performs analysis before the deployment and the second is by creating live tests and specialised monitoring that finds issues during the application lifecycle. Secondly by providing tools where SAST tool management can be applied on the IaC cases with ease.

2.1.1 State of the art (SotA)

The static application security testing (SAST) of IaC and application code is performed on a source code (IaC and app) during the design phase before the code is executed. SAST tools are a special variant of static analysis that identify security weaknesses of the code. The research behind that focuses on detecting common issues in IaC by code analysis like characterization of defective IaC scripts using text mining techniques [5], automatic testing of convergence and idempotence issues [6] [7], and automatic modifications of IaC code with respect to manual configuration changes in the infrastructure [8]. These issues are frequently addressed as *code smells* and analysed, described, and integrated inside SAST tools in the form of linters or similar check services. A comprehensive list of errors and that lead to the code smells is available in a well-documented and organised Common Weakness Enumeration database (CWE) [9].

The existing SAST tools for IaC are Terrafirma [10], which focuses on security analysis of Terraform code and reports violations of predefined policies. Similarly, Cookstyle [11] provides checks for Chef, TFLint [12] for Terraform Hashi Corp Configuration Language (HCL) and puppet-lint [13] for Puppet code.

Another SAST-like approach is to check the important dependencies used by the application and notify if they are outdated and potentially have known vulnerabilities. Example of such tools are RetireJS [14] (JavaScript), and Hakiri [15] (Ruby, Ruby on Rails) and OWASP's Dependency-Check

[16] (checks the dependencies of Java and .NET applications) and Wycheproof [17] for detecting crypto weaknesses.

Despite the fact of general usefulness of SAST tools to check the IaC, there are also critiques of that approach [18]. The critiques mostly point out the issues of slowing or stopping the business process. One kind of SAST issues are exhausting reports with large number of issues that are unbearable to handle. The second SAST issue that slows the business process is *pointing out false positive* issues.

Minimizing the human factor with automated IaC security scanning is currently the most common way to protect the created IaC. To better categorize existing scanning tools and find out what they can provide and what we can contribute inside PIACERE, we categorized the following IaC vulnerabilities, which seem to be recognized as the most common also by others [18] [19]:

- **Weak secret handling:**
 - **Exposed secrets** (hard-coded cloud credentials, private SSH keys in plain text, IP addresses, etc.) that include sensitive data, which is not properly secured can cause breaches.
 - **Suspicious code comments** that state too much about the security can give the attackers some clues how to breach into the systems.
 - **Permissions** for files are often too open as deployment servers has more than just write-only permission when configuring the clients.
 - **Weak or common misconfigurations** such as using HTTP without TLS, insecure address binding (i.e., 0.0.0.0), use of admin user by default and others can also result in decreasing the level of security.
- **Cryptography misuse** can be a weakness when weak algorithms such as MD5 and SHA-1 are being used as they do not provide collision-resistant hashing and can be broken by the attackers.
- **Hard-coded paths** can provide the attacker with additional knowledge and can also be problematic when the developers need to find and correct them on multiple places.
- **Dependencies** - IaC tools and dependencies often target packages that already include source code bugs with security issues.

The categorisation will help us to understand, which tools can contribute to each category and on the other side, uses will be able to explain which categories are not covered yet properly and require better attention. This categorisation will take place after the intensive test of the available checks that will be integrated or developed in the tools described in this deliverable.

Apart from being aware of the most common IaC vulnerabilities, the DevOps engineers should follow the following best practices for minimizing the risk of security alerts [20]:

- **IaC code scanning** with SAST and Software Composition Analysis (SCA) tools is the priority number one and should be done before pushing the source code with version control systems and definitely before the deployment.
- **Automated security scans** with CI/CD are important to continuously provide the IaC security within Git repositories and other environments where the code is being stored. In the case of PIACERE, this should be already covered in design phase.
- **Preventing hard-coded secrets** should include taking care when publishing the code and also checking the version control history for any possible exposure of secrets.
- **Reducing the impacts of code leaks** is related to establishing a contingency plan that contains necessary steps that need to be done in case of security issues.

- **Strengthening authentication and policies** within the DevOps tools includes picking strong passwords, rotating credentials through time, using two-factor authentication and so on.
- **Preventing IaC tampering** includes mechanisms that compare IaC packages between different phases of Software Development Life Cycle and notify the developers in case of any strange behaviour.

We will abide by the best practices presented above when picking the SAST tools and when implementing the IaC components to minimize the security leaks.

2.2 Approach

Based on the purpose, SotA and project documentation we created the plan of a solution. In first stage (a) we prepared a functional description based on the stakeholders' needs inside PIACERE. In parallel (b) we searched for known checkers that could be beneficial for our solution then (c) we developed the PoC of SAST tools including the IaC security Inspector and Component Security inspector that can be quickly integrated in the PIACERE environment and perform some of the existing checks. Checks will focus on all targeted IaC languages supported by PIACERE. After this (d) we plan to evaluate the performance and usability of the solution and continue with improving it iteratively from the feedback gained from the users.

The benefits and innovations which will be achieved through the development of IaC Security Inspector and IaC Component inspector were already described in executive summary and are the following:

- a) **IaC SAST service engine:** develop an umbrella engine designed as a service which can perform a set of checks with the ability to enable or disable specific checks. The included checks can be open source, proprietary or services outside PIACERE.
- b) **Review and develop checks:** review the existing checks, include them in the PIACERE SAST tools and make them available to the partners. From the check tests and PIACERE partners we will get the feedback of the combined check performance. Wherever the quality of the checks will not be satisfied, we will focus on resolving the gaps with creating and developing new check engines, new check services or updating existing solutions with (open-)source contributions.
- c) **Enterprise version – IaC SAST SaaS:** with this innovation, we will target enterprise users, which could benefit with SAST check management. The management of checks would allow organising SAST project for each deployment with its own automated check rules and users would be able also to share that project between others. These features would also add an ability to notify the user when the part of the application becomes vulnerable in runtime because a new exploit was discovered and published.

The results of the innovations presented above will be available as:

IaC Security Inspector (PIACERE KR6): the IaC Code Security Inspector will check cybersecurity issues for early identification of threats. Following the *shift left* approach, i.e., application security at the earliest stages in the development lifecycle, that allow reduction of costs due to early vulnerability detection in the DevSecOps lifecycle. The tool will integrate various tools from the open-source communities, proprietary (i.e., third-party) services and those developed inside PIACERE. The extensible design will allow adding new rules for detection of potentially dangerous (unforeseen) IaC code patterns. The expectation is to provide a report to the IaC developers, before the IaC is deployed, containing warnings about potential security issues of the IaC for the most used programming languages and tools (e.g., Ansible, Terraform HCL, Bash, TOSCA Simple Profile, etc.). A secondary objective will be to also provide suggestions on how existing code can be sanitized.

IaC Component Inspector (PIACERE KR7): The Component Security Inspector is checking for known security vulnerabilities in software components of the target application, this will consider the application referenced by the IaC code, as well as dependencies and libraries against one of more databases of known vulnerabilities. Special attention will be given to imported cryptographic libraries. The most popular cryptographic libraries will be selected on a use-case driven approach and where required, additional security tests for cryptographic issues will be designed. If the use-cases will not provide enough feedback, the selection process will feature exploring the usage statistic of crypto libraries from different sources, e.g., PyPI Stats [21].

The expectation regarding the Component Security Inspector is to report the list of components, that is based on security vulnerabilities from publicly available databases of vulnerable software (CVE) and are not suited for deployment since they have to guarantee the expected security levels promised by the PIACERE environment. It is also PIACERE's expectation to deliver details on potential attacks with suggestions for their fixes whenever possible. The cryptography libraries verification tool will check the commonly used cryptographic libraries used in IaC and find if they are vulnerable (outdated, etc.) or use obvious misconfiguration, e.g., generating weak keys. The tools shall be implemented so that tests can be periodically run (for instance with the CI/CD pipelines or cron-like jobs) to keep vulnerabilities under control and implementation shall be easily extensible so that unforeseen tests for new attacks can be added to the tool coverage. Inside PIACERE, we will create our own tool for checking the IaC components and also specific component checks where available ones will not perform good enough.

However, the idea of creating similar linter engines and SAST tools for IaC as we are describing here is not brand new and some tools are already available. The services that combine a set of checks and will be on our watch list are:

- **AnyLint**² is a linter created to lint anything with custom rules. You can easily write your own linters in any languages.
- **Super-linter**³: is a GitHub Action to run a Super-Linter. It is a simple combination of various linters, written in bash, to help validate your source code.
- **Mega-Linter**⁴: is an open-Source tool for CI/CD workflows that analyzes consistency and quality of 47 languages, 22 formats, 18 tooling formats, abusive copy-pastes and spelling mistakes in your repository sources. The tool generates various reports, and can even apply formatting and auto-fixes, to ensure all the projects sources are clean, whatever IDE/toolbox are used by their developers.
- **Snyk**⁵: Is a service that continuously find & fix vulnerabilities in dependencies pulled from npm, Maven, RubyGems, PyPI and more.
 - Users can use Snyk SAST, API, CLI, Docker container, etc. to test the code (all modes require authentication to Snyk account)
 - Has a lot of integrations (through repositories such as GitHub, GitLab and BitBucket, CI/CD tools, IDEs, cloud providers, containers, etc.)
 - Free version has limited code runs and does not contain an API, which is included in enterprise plans
- **Checkov**⁶: Checkov is a static code analysis tool for infrastructure-as-code supporting Terraform, AWS CloudFormation, Kubernetes, Serverless or ARM templates. Bridgecrew

² <https://github.com/fand/anylint>

³ <https://github.com/github/super-linter>

⁴ <https://nvuillam.github.io/mega-linter/>

⁵ <https://snyk.io/>

⁶ <https://github.com/bridgecrewio/checkov>

identifies, fixes, and prevents misconfigurations in cloud resources and infrastructure-as-code files. It supports AWS, Azure and Google Cloud best practices and compliance checks

- **CloudSploit**⁷ or Cloud Security Scan is a tool for compliance checks of cloud resources. It uses read-only permissions for the cloud account and supports AWS, Azure, GCP and Oracle Cloud infrastructure

To position the PIACERE Verification tools properly inside the presented field of code inspection, we are aiming to create a service that is capable to execute combined checks over the IaC or even apply checks provided by other user's (paid) services. In other words, if the user has bought the professional version of some already available vulnerability scanner (e.g., Snyk [22]), the PIACERE Inspectors will be able to add this service among the list of checks to be performed. Beside the improved usability of the PIACERE tools we will focus also on developing particular checks of the IaC that are not yet available in other tools – planned for the next period.

The overall goal of PIACERE IaC Security and Component Inspector is to establish a component that will offer significant advantages over other available IaC scanning tools (such as Snyk [22]). The core component able to perform IaC Security Inspector and IaC Component Inspector tasks will be developed as a service, which will integrate multiple SAST tools. The first advantage is that this core service will be open-source and the users outside of PIACERE will be able to contribute. A lot of enterprise SAST platforms require creating an account before any tool can be used and our difference will be that the core component could be used offline, so that everyone will be able to install it and run it locally. Later, we will develop multiple modes of interaction (API, SaaS API, SaaS CLI, IDE plugins, etc.) over the core component to ensure better user-experience and integration.

Another novelty is also related to supporting two kinds of checks considering their type of access – local checks that every user can install separately and third-party remote service checks (Snyk [22], SonarCloud [23], etc.) that require online accounts and a special configuration. The ability to configure each IaC check according to user's preferences will be an important feature that is usually not offered by the other SaaS-based SAST services. This means that the users will be able to override the default settings and supply their own configuration files, which can be especially useful in cases when SAST tools are detecting false positive vulnerabilities. One notable disadvantage of (enterprise) SAST tools is targeting or emphasizing only specific types of IaC (for instance only Terraform). In PIACERE we will need to integrate multiple IaC checks as we require to have a multipurpose solution that can handle scanning all types of IaC packages with the ability that users can choose their own set of checks and apply them to their IaC.

The goal of IaC Security and Component Security Inspector

The overall goal of PIACERE IaC Security and Component Inspector is to establish a component that will offer the following advantages over other available IaC scanning tools presented in this document:

- An open-source component for running scans that can be used offline.
- Wider range of interaction modes (API, SaaS API, SaaS CLI, plugins, etc.).
- Support for multiple integrated and remote (also third-party) service checks.
- The ability to configure each IaC check according to user's preferences.
- Possible exploitation and integration with other tools like xOpera SaaS [24] or Template Library [3] components.
- Competitive pricing plan for Enterprises that would require a SaaS edition.

⁷ <https://github.com/aquasecurity/cloudsploit>

This service is intended to facilitate the security checks within PIACERE and will allow us to facilitate the use of multiple local and remote SAST tools. Those often require setting up a special environment and installing (mutual-exclusive) requirements. Our component performing the IaC inspection will take care of the installation and environment preparation and will also explain the configuration of every IaC check within the documentation.

As mentioned in the executive summary, we will follow a lean and incremental implementation approach of the tools where we will try to incorporate the ability to (a) add new checks into the final tool, (b) review them and (c) improve them. The overall first impression of how the conceptually the tools will perform in action is presented in Figure 1, where the commonalities of both tools are presented. Both tools will receive IaC on the input, perform a set of checks for vulnerabilities of different IaC content and returned combined results to the user.

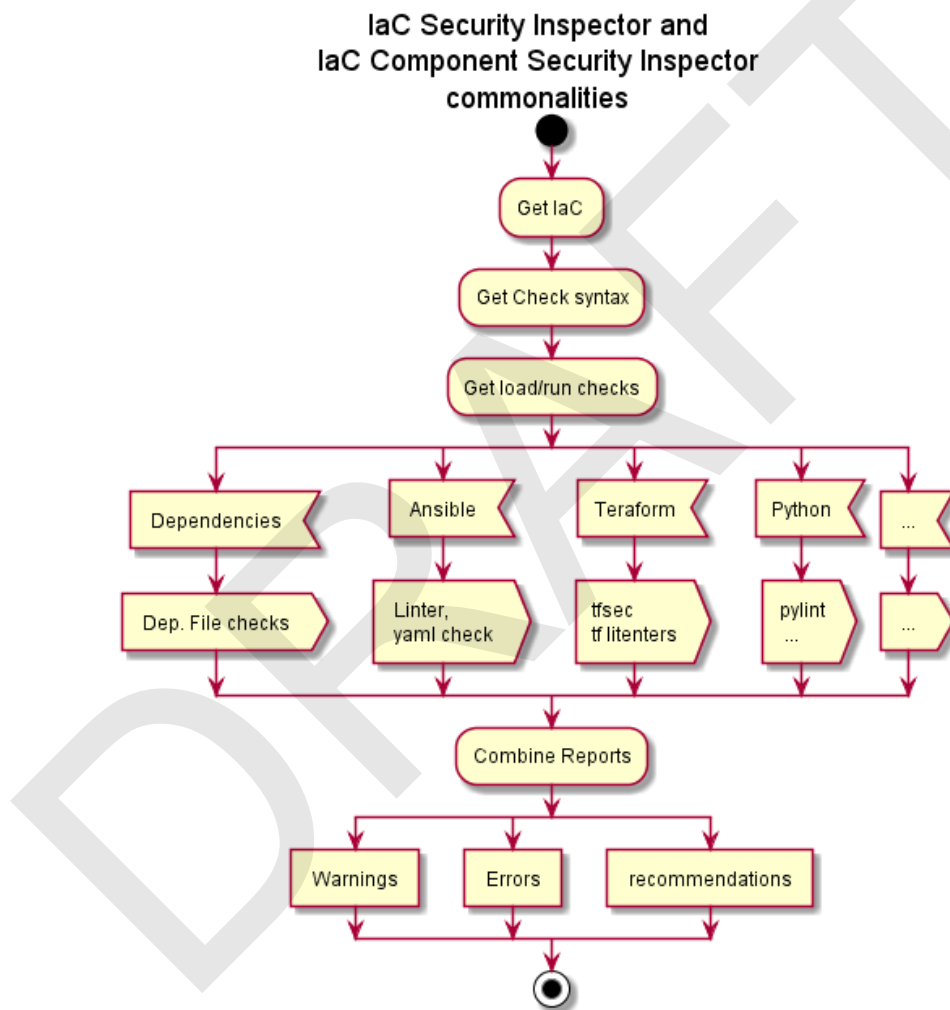


Figure 1 Main actions from PIACERE Security Inspector

2.3 Functional description

While the SotA presents the available tools and our opportunities to improve them, the PIACERE consortium partners provide another set of necessary requirements for static code inspection. These requirements are available in the PIACERE architectural specification [1] and includes integrational requirements and use-case requirements.

Table 1. Functional requirements [1]

REQ #	Description	Priority/Task	Current status
REQ23	IaC Code Security Inspector must analyse IaC code w.r.t. security issues of the modules used in the IaC.	Medium T4.2	ACCEPTED MUST HAVE Partially done Started, PoC examples.
REQ24	Security Components Inspector must analyse and rank components and their dependencies used in the IaC.	Medium T4.3	ACCEPTED MUST HAVE Started , no result yet.
REQ65	IaC Security Inspector and Component Security Inspector should hide specificities and technicalities of the current solutions in an integrated IDE.	Low T4.2, T4.3, WP3	ACCEPTED MUST HAVE Partially done , some check configurations available
REQ66	IaC Code security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows.	Medium T4.2, T2.2	ACCEPTED MUST HAVE Partially done REST API available, in evaluation.
REQ67	IaC Component security inspector must provide an interface (CLI or REST API) to integrate with other tools or CI/CD workflows.	Medium T4.2 T2.2	ACCEPTED MUST HAVE Partially done REST API available, in evaluation.
REQ80	SAST tools to check Docker configurations shall be included in the Canary environment.	Medium WP4	PROPOSED MUST HAVE Not started yet.

Table 1 presents the list of functional requirements for IaC Security Inspector and IaC Component Inspector that were agreed among the PIACERE partners. To summarize the content of the table, the final tools should provide:

- A service with a clear API to be available to the PIACERE components hiding as much complexity as possible fulfil the requirement REQ65. Similarly, both inspectors should provide API in such a way that the results will be applicable to the CI/CD workflows.
- A set of security checks that inspect IaC has to be performed to inspect for (REQ23, REQ24, REQ80):
 - Security check (covered by IaC Security Inspector)
 - Configuration check (mistypes, misconfigurations - covered by IaC Security Inspector)
 - Inspect the components that might have vulnerabilities (covered by IaC Component Inspector) and rank components by the threat level.
- As the DOML can be translated by the Infrastructural Code Generator (ICG) into different target IaC, services must be able to run checks for different IaC dialects. where among the use-cases is clear that we will use:
 - Terraform HCL
 - Ansible (can be in combination with Terraform HCL or TOSCA YAML)
 - Docker
 - TOSCA YAML

The requirement list is a cornerstone of the tools specification that we will develop. During preparing our SotA (described in Section 2.1.1) and discussions with the DevOps practitioners, we found out that there are also interesting fields of checks which we will also review and consider adding in the PIACERE IaC Inspector or IaC Component inspector. These focus on the:

- license check
- coverage check
- documentation check
- good practices

For the latter, it is evident that they can be harder to tackle, as they can rely on a large background knowledge set. Most of those has not been yet addressed, therefore, we will investigate more time to find out how these could be technically addressed, create them, and include in our final solutions.

As it can be seen in Table 1 we already started working on fulfilling the requirements. The REQ23 is currently partially satisfied within the PIACERE IaC Security Inspector as IaC can be analysed for some vulnerabilities that we already covered (see Section 3.3). Scanning the other IaC languages will be supported as described within the Section 4. The REQ65 is already started and currently provides a way of expressing the check configurations inside the proposed service. The REQ66 and REQ67 are also partially completed as the PIACERE IaC Inspector can be integrated with other tools and CI/CD workflows through its REST API or CLI. We will also strengthen the integration options by adding new modes of interaction (e.g., SaaS API, SaaS CLI, GUI, IDE plugins, etc.) in the future.

2.3.1 Check selection

Corresponding to the current chosen IaC approaches covered by PIACERE (see requirements of D2.1 [1]), we decided that we need to cover the following syntaxes and IaC languages:

- Terraform HCL (HashiCorp Configuration Language)
- Ansible
- Docker
- TOSCA YAML

Additionally, we require to support the syntaxes and languages that are used to describe those services or are frequently used, like:

- Python
- Java
- YAML
- Bash/shell scripts
- HTML, CSS, JS, TS

From the presented input we generated the initial list of available solutions to be integrated in our solution. The list was populated together with the help of stakeholders inside PIACERE where we find the checks, they would like to integrate in their DevOps workflow.

The list of available checks, potentially applicable for PIACERE based on the requests from the use cases is available in Table 2. The service list is used in development sprints to for reporting the status of the integrated checks. The list of currently available checks is presented in the section called User Manual.

Table 2. List of potential checks to be included

	Name	Short description
Ansible	Ansible Lint	Checks playbooks for practices and behaviour that could potentially be improved [25].
Terraform HCL	TFLint	A Pluggable Terraform HCL Linter [26].
	Tfsec	Uses static analysis of user’s Terraform HCL templates to spot potential security issues [27].
	Terrascan	A static code analyser for Infrastructure as Code (defaults to Terraform HCL) [28].
TOSCA	Terraforma	A static analysis tool for Terraform HCL plans [29].
	tosca-parser	TOSCA IaC validation with CLI command [30].
TOSCA	xopera-opera	Can validate IaC with <i>opera validate</i> CLI command [31].
	Hadolint	A smarter Dockerfile linter that helps you build best practice Docker images [32].
Python	PyLint	A Python static code analysis tool which looks for programming errors, helps enforcing a coding standard, sniffs for code smells and offers simple refactoring suggestions [33].
	Bandit	Designed to find common security issues in Python code [34].
	PyUp	Python Dependency Security (tracks known security vulnerabilities in Python packages) [35].
Java	Checkstyle	A tool for checking Java source code for adherence to a Code Standard or set of validation rules (best practices) [36].
HTML, CSS, JS, TS	HTMLHint	The static code analysis tool you need for your HTML [37].
	ESLint	A tool for identifying and reporting on patterns found in ECMAScript/JavaScript code [38].
	TypeScript ESLint	Enables ESLint to support TypeScript [39].
Shell	ShellCheck	A tool that gives warnings and suggestions for bash/sh shell script [40].
Other local checks	GitLeaks	Detects hard-coded secrets like passwords, API keys, and tokens in Git repositories [41].
	git-secrets	Prevents you from committing secrets and credentials into Git repositories [42].
	Markdown lint	A tool to check markdown files and flag style issues [43].
	Gixy	A tool to analyse Nginx configuration to prevent security misconfiguration and automate flaw detection [44].
	Yamllint	A linter for YAML files (syntax check + key repetition and cosmetic problems such as lines length, trailing spaces, indentation, etc.) [45].
	Dependency-Check	A Software Composition Analysis (SCA) tool that attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies [46].

2.3.2 Validation of the component

The validation process will be two-fold. First set of tests will confirm the integration of the checks and confirm their functionality with simple examples. For this case we will provide the IaC code examples to be used for testing. The final validation will be performed by the use case providers that will use our tools for the development of their use case.

The content for testing the developed tools will be available on the TecNALIA's GitLab or public available GitHub pages and will include a set of examples in the form of:

- Dockerfiles
- Terraform HCL scripts
- Ansible scripts
- TOSCA YAML templates and TOSCA CSARs
- Python scripts
- Bash scripts

For the final validation we will use the use case IaC content which will be more in depth and stretch over multiple targeted IaC options (e.g., Terraform HCL + Ansible + Docker).

2.3.3 Fitting into overall PIACERE Architecture

The *IaC Security Inspector* and *IaC Component Security Inspector* are a part of the *PIACERE Vulnerability tools*. The Vulnerability tools are used to check DOML, which is achieved with *Model checker* and check the IaC generated from the DOML, which is done by IaC Security Inspector and IaC Component Security Inspector.

In the overall architecture, the IaC Security Inspector and IaC Component Security Inspector fit into a set of design tools. The services will be initiated by IDE after IaC will be generated from the DOML language. The **inputs** of the IaC Security Inspector and Component inspector will be simplified to allow scanning IaC packages (such as zip or tar files). The **outputs** will be formatted as JSON and will be sorted by tools.

The main integration point will be a RESTful API that will allow scanning IaC for issues and vulnerabilities. Another possible integration that is still under consideration could be established through the CLI, which will offer the integration in console environments. This could facilitate running the API within shell or interacting with it using different CLI commands. The API will be also encapsulated in a public Docker image, which will make it possible to run it across all platforms. We will use different tools and services for IaC scanning and will document how each of these tools can be used and configured to fit the user's expectations. Future implementations may also introduce Software as a Service component that will allow users to organize their scans in a multi-workspace environment. Here we could provide the SaaS API, CLI, GUI and a possible Eclipse plugin for a smoother integration.

2.4 Technical description

In this section we will present the IaC security inspector and Component security inspector in detail. The sequence diagram in Figure 2 presents activities of both tools. The standard IaC Security Inspector workflow starts with the user that desires to inspect his IaC with triggering the service directly from the PIACERE IDE. Within the IaC code inspection process, the IaC inspector initiates and runs the necessary checks (linters, configuration checks) using its internal worker. After that the inspector obtains the check results and returns them back to the user.

The Component Security Inspector has different task, which focuses on finding vulnerabilities in IaC dependencies (e.g., Python packages). As shown on the second block in Figure 2 the

Component Security Inspector initiate component checks and seeks for component issues and misconfigurations. After getting the job done, the component creates an output and sends it back to IDE.

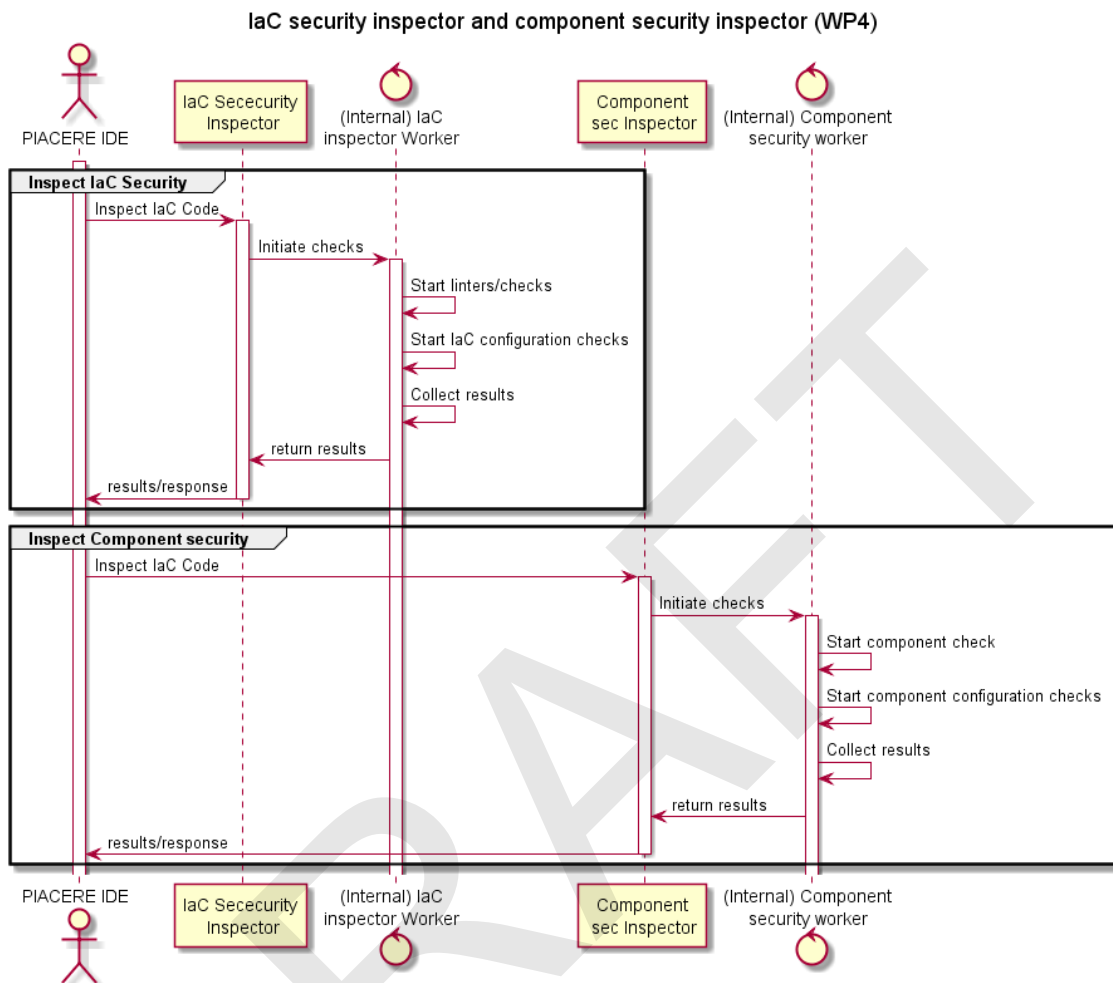


Figure 2. Activity diagram with IaC and Component Security Inspectors

As it is evident from the sequence diagram in Figure 2, both tools have the similar interface and similar basic calls are required for managing the checks, starting checks, and stopping the checks. From the integrational perspective, the tools can be developed within one universal interface that can cover requirements of both tools. During the security check and security scan service review presented in SotA section of this document, we realised that some checks can also perform both types of checking - the IaC and component one. That means that a particular check could act as an IaC Security inspector or IaC component inspector or in case of more comprehensive checks – it can belong to both component types. This led us to the unification of the development of core element for both components, which we called as *IaC Scan Runner*.

IaC Scan Runner is an individual component that can run IaC Security Inspector or IaC Component Inspector checks included in scans. In other words, users interacting with the component can use it as any combination of both tools – only IaC scans, only component scans, and combined. The choice of how this component acts is defined by the list of enabled checks performed over the IaC.

2.4.1 Prototype architecture

The previously introduced IaC Scan Runner is a *technical* component that can run *security scans*, while the *type of the scan* defines which the component type (IaC Security Inspector or IaC Component inspector). The IaC Scan Runner component diagram in Figure 3 presents the idea in the current prototype development. The IaC Scan Runner is developed as a service inside the docker container. Basically, the service provides an API for configuring scans, managing scans, and retrieving outputs. The configuration manager keeps the configuration of each check, the Scan worker sets up the scan workflows according to the documentation and executes the scans. The processes inside a container are performing scans one by one. When the scans are finished, the service combines the output and sends it back to the IDE. The Figure 4 shows one example output from one of the IaC checks. In this case, the Bandit tool is used to locate the insecure segments of Python code.

The idea of hiding complexity of the IaC Security Inspector and Component Security Inspector inside the IaC Scan Runner is done intentionally to ease the tool integration inside the PIACERE solution. Beside relying on the integrated checks, we envision also to run third-party remote scan services that are available to the user through free or paid remote services - (e.g., Snyk [22]). These comprehensive services can include any possible checks from linters, QA, security or component check provided by a third party.

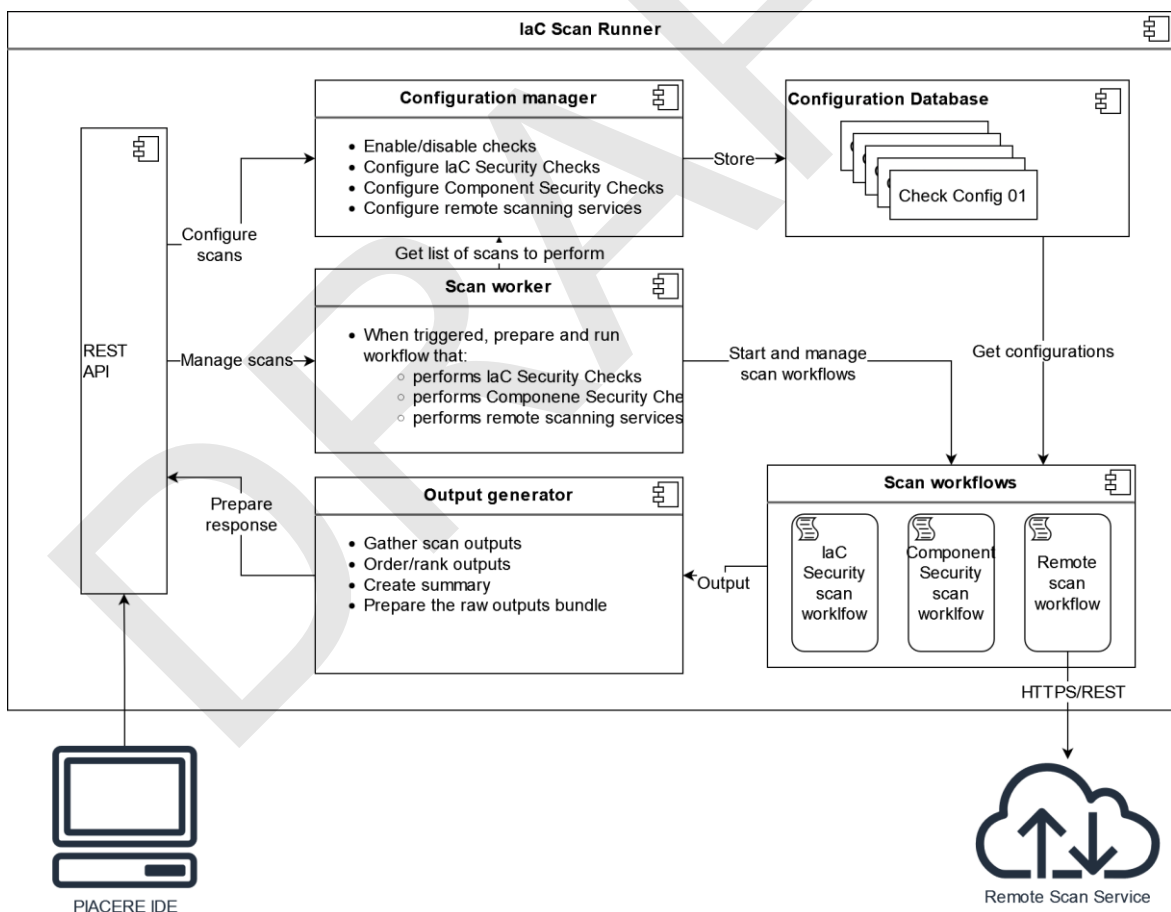


Figure 3. IaC Scan Runner

```
Test results:
>> Issue: [B108:hardcoded_tmp_directory] Probable insecure usage of temp file/directory.
Severity: Medium Confidence: Medium
Location: /repo/cloud/aws/thumbnail-generator/modules/lambda/playbooks/function/image_resize.py:31
More Info: https://bandit.readthedocs.io/en/latest/plugins/b108_hardcoded_tmp_directory.html
30     key = record['s3']['object']['key']
31     original_image_path = '/tmp/{}'.format(key)
32     aws_s3_client.download_file(bucket, key, original_image_path)

-----
>> Issue: [B108:hardcoded_tmp_directory] Probable insecure usage of temp file/directory.
Severity: Medium Confidence: Medium
Location: /repo/cloud/aws/thumbnail-generator/modules/lambda/playbooks/function/image_resize.py:35
More Info: https://bandit.readthedocs.io/en/latest/plugins/b108_hardcoded_tmp_directory.html
34     for size_px in THUMBNAIL_SIZES_PX:
35         resized_image_path = "/tmp/" + str(key) + "_" + str(size_px) + ".jpg"
36         resize_image(original_image_path, resized_image_path, size_px)
```

Figure 4. Bandit tool IaC check output

2.4.2 Component description

The IaC Scan Runner (Figure 3) is comprised of the following components:

- REST API is the main interface that will be called by the PIACERE IDE. In particular cases, if needed it would be also possible to create a corresponding CLI application that could make the call to the API, which would integration of the tool in the continuous integration (CI) scripts.
- The configuration manager takes care of the IaC Scan Runner configuration. One part of the configuration includes the set of available checks, list of enabled/disabled checks for scan and the configuration of each integrated or remote check. For example, remote scan services will need the URL and credentials to perform the scans.
- Configuration database stores all settings of the installed checks and provide them when some scans are performed.
- The Scan Worker takes care of scan execution. This means that it takes all checks and configurations of the same type (IaC, Component or Remote) and prepares workflows to be executed. The output of the scans is collected by the output generator.
- Scan workflows presents a set of processes that perform scans.
- The output generator gathers the outputs of scans and forms the output for IDE. This includes filtering outputs, creating the summary, and ordering and ranking the outputs.

All components are designed to run together inside a docker container, which can be set up locally on a developer's machine or be available as a service.

2.4.3 Technical specifications

The IaC Scan Runner that is being developed within PIACERE is written in Python programming language. The REST API uses OpenAPI Specification [47], whereas Swagger UI [48] and ReDoc [49] are used to document it. The general documentation (in Figure 6) for IaC Scan Runner uses Sphinx [50] documentation tool (with Read the Docs [51] theme), where the docs can be easily rendered from RST files. The IaC Scan Runner CLI, which is used to run the REST API from the console is also written in Python and is regularly published on Python Package Index (PyPI [52]) as the *iac-scan-runner* [53] pip package (the development version of the package is available on Test PyPI [54]). The API is designed using FastAPI [55] - a modern and high-performance web framework and the CLI is build using Click-based Typer [56] Python library. Apart from a local installation, both REST API and docs can be also distributed as a Docker image, where the images are stored within the *xscanner* [57] Docker Hub community organization. The Docker image for

the IaC Scan Runner tries to be as general as possible (currently it is based on Python slim-buster Debian release). This allows installing almost all possible checks, because the check linters and tools are usually written in all different languages and therefore require different installation procedures. The download and installation of checks is initiated by a simple bash script, which ensures that the checks are available to be used by the REST API.

DRAFT

3 Delivery and usage

This section will first describe the package info and the installation of IaC Scan Runner and then its usage through the REST API and CLI.

3.1 Package information

The IaC Scan Runner module is delivered as a Docker application including a service accessible through an API. The `xscanner/runner` [58] Docker image (Figure 5) is updated and published regularly on Docker Hub. The CLI that is currently able to run the API is available as `iac-scan-runner` Python package and is published on PyPI [52]. Both, API and CLI use semantic versioning for new releases and the latest available version is 0.0.6.

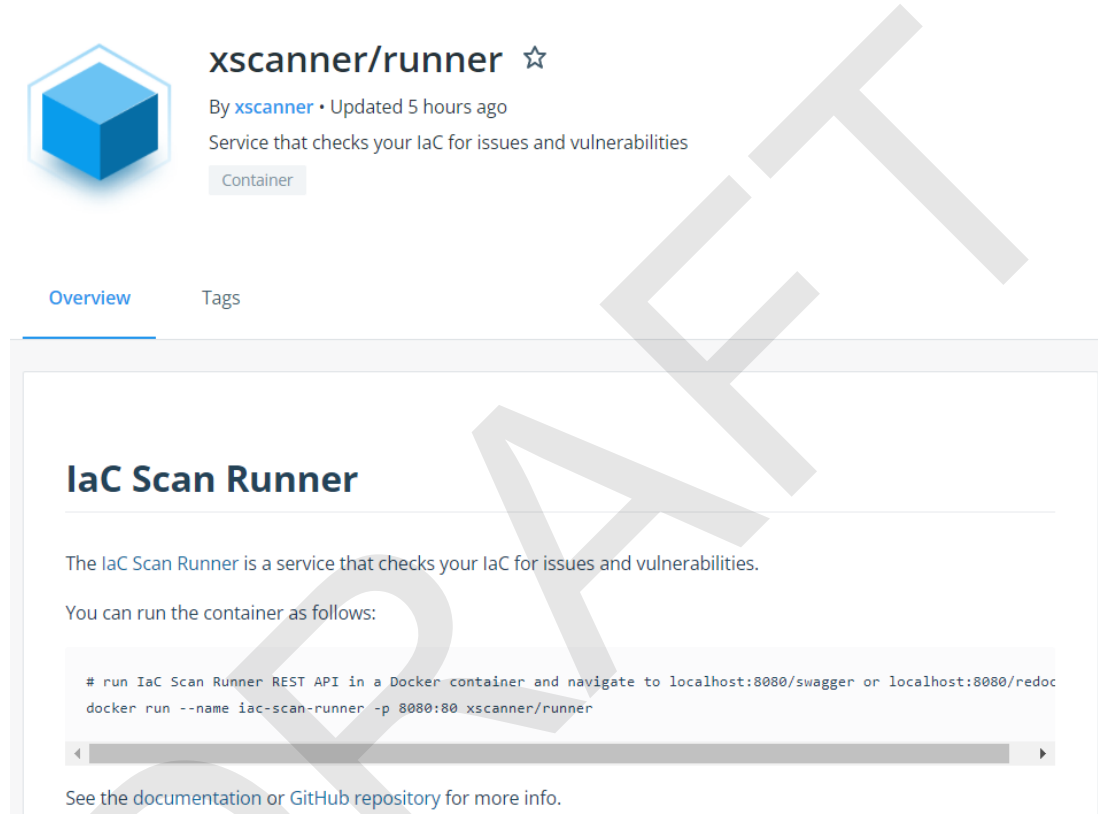


Figure 5. Docker Hub repository for IaC Scan Runner

Some of the IaC Scan Runner services are already available to the PIACERE consortium partners on public links:

- REST API: <https://scanner.xopera.piacere.esilab.org/iac-scan-runner/>
- Swagger UI: <https://scanner.xopera.piacere.esilab.org/iac-scan-runner/swagger/>
- ReDoc: <https://scanner.xopera.piacere.esilab.org/iac-scan-runner/redoc/>
- Documentation: <https://scanner.xopera.piacere.esilab.org/docs/>

The IaC Scanner REST API is protected by HTTP Basic Auth, to avoid the use from bots and robots.

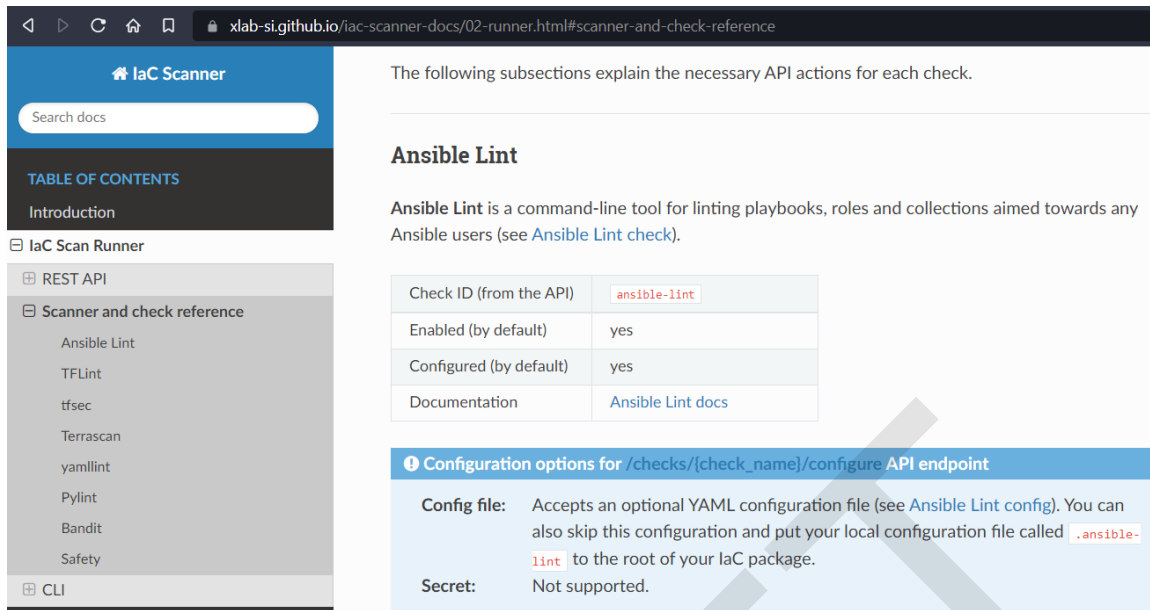


Figure 6. IaC Scanner documentation page hosted on GitHub Pages

3.2 Installation instructions

The user can run the IaC Scan Runner REST API by pulling a public `xscanner/runner` [58] Docker image as follows.

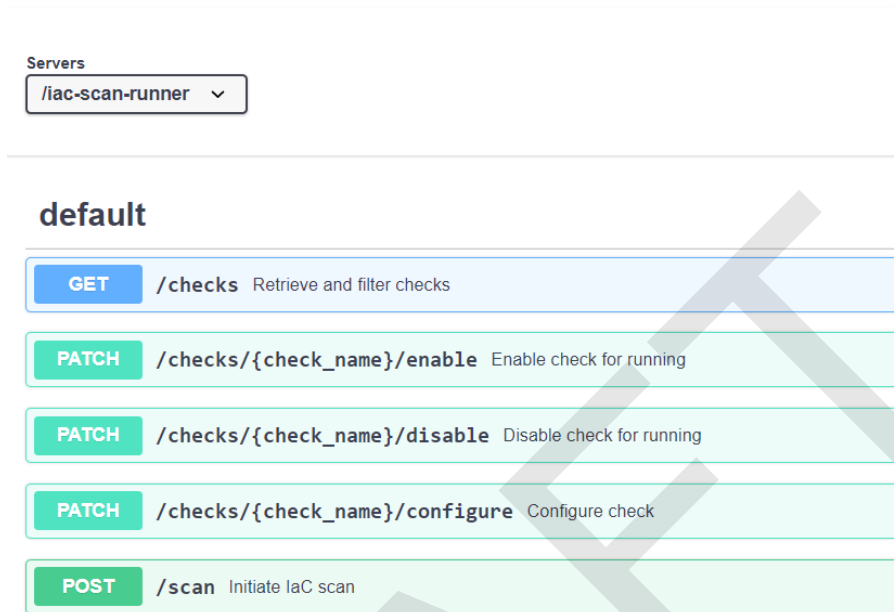
```
$ docker run --name iac-scan-runner -p 8080:80 xscanner/runner
```

After the setup the user will see that the OpenAPI Specification [47] and interactive Swagger UI [48] API documentation are available on `/swagger` (see Figure 7), whereas ReDoc [49] generated API reference documentation is accessible on `/redoc` (see Figure 8). The user can also retrieve an OpenAPI document that conforms to the OpenAPI Specification as JSON file on `/openapi.json` or as YAML file on `/openapi.yaml` (or `/openapi.yml`).

IaC Scan Runner REST API 0.0.6 OAS3

[/iac-scan-runner/openapi.json](#)

Service that checks your IaC for issues and vulnerabilities



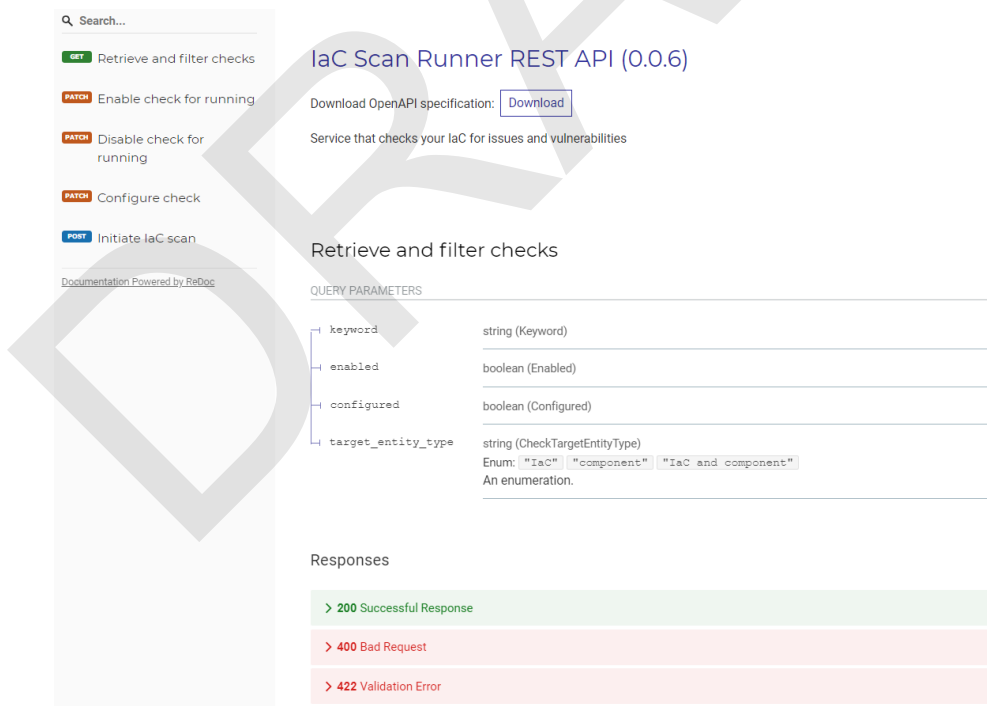
Servers

/iac-scan-runner

default

- GET** /checks Retrieve and filter checks
- PATCH** /checks/{check_name}/enable Enable check for running
- PATCH** /checks/{check_name}/disable Disable check for running
- PATCH** /checks/{check_name}/configure Configure check
- POST** /scan Initiate IaC scan

Figure 7. Swagger UI page for IaC Scan Runner REST API



Search...

IaC Scan Runner REST API (0.0.6)

Download OpenAPI specification: [Download](#)

Service that checks your IaC for issues and vulnerabilities

Retrieve and filter checks

QUERY PARAMETERS

- keyword: string (Keyword)
- enabled: boolean (Enabled)
- configured: boolean (Configured)
- target_entity_type: string (CheckTargetEntityType)
Enum: "IaC" | "component" | "IaC and component"
An enumeration.

Responses

- > 200 Successful Response
- > 400 Bad Request
- > 422 Validation Error

Figure 8. ReDoc page for IaC Scan Runner REST API

The IaC Scan Runner CLI enables easier setup of the IaC Scan Runner API in console environments. The installation requires Python 3, and the best way is to install the *iac-scan-runner* package in a clean virtual environment.

```
$ mkdir ~/iac-scan-runner && cd ~/iac-scan-runner
$ python3 -m venv .venv && . .venv/bin/activate(.venv)
$ pip install --upgrade pip(.venv) $ pip install iac-scan-runner
```

3.3 User Manual

3.3.1 REST API

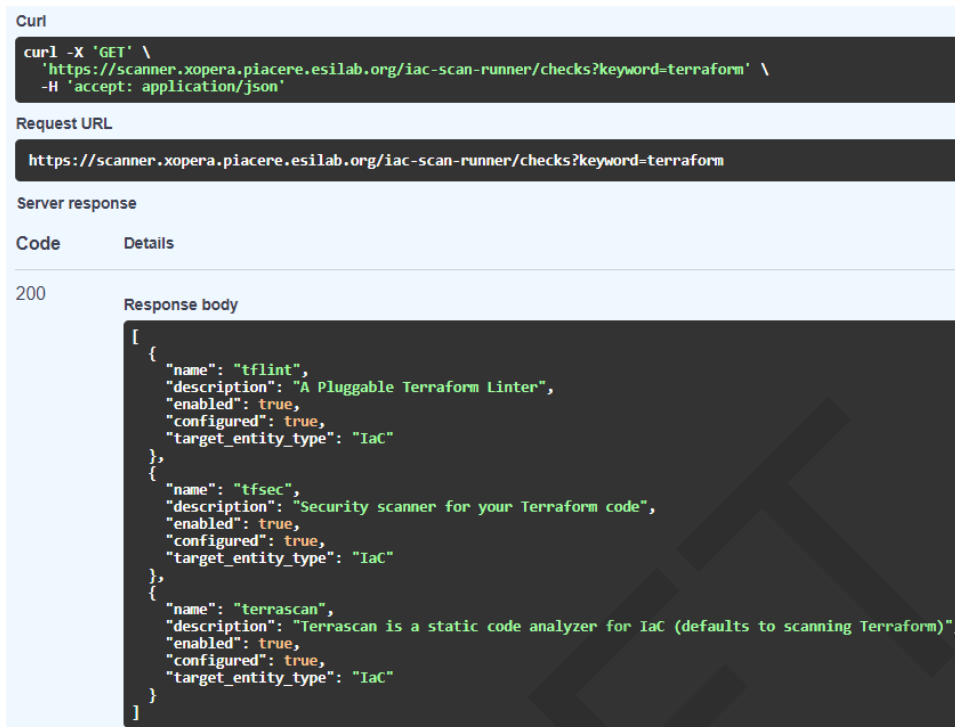
The IaC Scan Runner API can be used to interact with the main IaC inspection component and initialize IaC scans. The API includes various IaC checks that can be filtered and configured. The user can choose to execute all or just the selected checks as a part of one IaC scan. After the scanning process, the API will return all the check results.

Currently available endpoints are shown in the Table 3 and after that each endpoint is explained in detail.

Table 3. IaC Scan Runner REST API endpoints

REST API endpoint	Description
<code>/checks</code>	Retrieve and filter checks
<code>/checks/{check_name}/enable</code>	Enable check for running
<code>/checks/{check_name}/disable</code>	Disable check for running
<code>/checks/{check_name}/configure</code>	Configure check
<code>/scan</code>	Initiate IaC scan

The `/checks` GET endpoint lets user retrieve and filter the supported IaC checks. Checks can be filtered by their keynames (by using the *keyword* request parameter), and users can find out whether they are already enabled (by setting the *enabled* parameter) or configured (by setting the *configured* parameter). Checks can also be filtered by their target entity (by setting the *target_entity_type* parameter). There are three types of checks - IaC (they only check the code), component (they check IaC requirements and dependencies in order to find vulnerabilities) and checks that cannot be placed into one of the two groups and are both IaC and component. Each IaC check in the API has its unique name so that it can be distinguished from other checks. When no filter is specified, the endpoint lists all IaC checks. The Figure 9 shows how the user can only filter the IaC check for Terraform.



```
Curl
curl -X 'GET' \
  'https://scanner.xopera.piacere.esilab.org/iac-scan-runner/checks?keyword=terraform' \
  -H 'accept: application/json'

Request URL
https://scanner.xopera.piacere.esilab.org/iac-scan-runner/checks?keyword=terraform

Server response
Code    Details
200
Response body
[
  {
    "name": "tfLint",
    "description": "A Pluggable Terraform Linter",
    "enabled": true,
    "configured": true,
    "target_entity_type": "IaC"
  },
  {
    "name": "tfsec",
    "description": "Security scanner for your Terraform code",
    "enabled": true,
    "configured": true,
    "target_entity_type": "IaC"
  },
  {
    "name": "terrascan",
    "description": "Terrascan is a static code analyzer for IaC (defaults to scanning Terraform)",
    "enabled": true,
    "configured": true,
    "target_entity_type": "IaC"
  }
]
```

Figure 9. Filtering out Terraform checks in the IaC Scan Runner REST API

IaC checks can be *enabled* – will be used during the scan – or *disabled* – will not be used during the scan – using `/checks/{check_name}/enable` PATCH endpoint. Most of the local checks are enabled by default. The more advanced checks that take longer time or require additional configuration are disabled and have to be enabled before the scanning. This endpoint can be used to enable a specific IaC check (selected by the `check_name` parameter), which means that it will become available for running within IaC scans. On the contrary, users can use `/checks/{check_name}/disable` PATCH endpoint can be used to disable a specific IaC check (selected by the `check_name` parameter), which means that it will become unavailable for running within IaC scans.

The `/checks/{check_name}/configure` PATCH endpoint is used to configure a specific IaC check (selected by the `check_name` parameter). Most IaC checks do not need configuration as they already use their default settings. However, some of them, especially the remote third-party service checks (such as Snyk [22]), require to be configured before using them within IaC scans. Some checks will have to be enabled before they can be configured. The configuration of IaC check takes two optional multipart request body parameters - `config_file` and `secret`. The former (`config_file`) can be used to pass a check configuration file (which is supported by almost every check) that is specific to every check and will override the default check settings. The latter (`secret`) is meant for passing sensitive data such as passwords, API keys, tokens, etc. These secrets are often used to configure the remote service checks - usually to authenticate the user via some token that has been generated in the remote service user profile settings. Some IaC checks support both the aforementioned request body parameters and some support one of them or none. The API will warn the user in case of any configuration problems.

By sending a POST request on `/scan`, the user calls the main endpoint for scanning the IaC and gather the results from the executed IaC checks. The request body is treated as multipart (`multipart/form-data` type) and has two parameters. The first one is named `iac` and is required. Here, the user passes his (compressed) IaC package (currently limited to zip or tar). The second parameter is called `checks` and is an optional array of checks, which the user wants to execute

as a part of his IaC scan. The IaC checks are selected by their unique names. If the user does not specify that field, all the enabled checks are executed. The API will notify the user if there are any non-existent, disabled or unconfigured checks that she wanted to use. After the scanning process, the API will return the results from all checks along with their outputs and return codes. One example of IaC scanning through the IaC Scan Runner API is visible in Figure 10.

```

Curl
curl -X 'POST' \
  'https://scanner.xopera.piacere.esilab.org/iac-scan-runner/scan' \
  -H 'accept: application/json' \
  -H 'Content-Type: multipart/form-data' \
  -F 'iac=@AWS_thumbnail_gen_with_EC2_VM_0.4.0.csar' \
  -F 'checks='

Request URL
https://scanner.xopera.piacere.esilab.org/iac-scan-runner/scan

Server response
Code    Details
200
Response body
{
  "rc": 1,
  "pylint": {
    "output": "***** Module .\\n_init__.py:1:0: F0010: error while code parsing: Unable to load file __init__.py:\\n[Er",
    "rc": 1
  },
  "bandit": {
    "output": "[main]\\tINFO\\tprofile include tests: None\\n[main]\\tINFO\\tprofile exclude tests: None\\n[main]\\tINFO\\tcli include",
    "rc": 1
  }
}

```

Figure 10. An example of IaC Scan Runner REST API scanning output

The scanner (scan worker component) itself is the main component of the IaC Scan Runner and it initiates the scanning process, which makes the supplied IaC go through multiple checks. IaC Scan Runner currently supports IaC checks that are listed in the Table 4.

Table 4. Currently supported IaC checks within IaC Scan Runner

Check name	Target IaC entity	Enabled (by default)	Needs configuration
Ansible Lint	Ansible	yes	no
TFLint	Terraform HCL	yes	no
tfsec	Terraform HCL	yes	no
Terrascan	Terraform HCL	yes	no
yamllint	YAML	yes	no
Pylint	Python	yes	no
Bandit	Python	yes	no
Safety	Python packages	yes	no

It is evident that in the first phase of adding the checks, we focused more on scanning Terraform, Ansible and Python since these are the most crucial to support within PIACERE. We picked TFLint [12], tfsec [59], and Terrascan [60] for Terraform HCL scanning because these three were the most used tools and can find the most IaC vulnerabilities. For Ansible, there is currently not many tools that would support scanning and therefore only Ansible Lint [61] was applicable. On the other hand, there are many linters available for Python. We have selected Pylint [62] and Bandit [63] for the IaC scanning and a promising PyUp Safety [64] tool that performs component check of the Python requirements. To support linting TOSCA YAML templates and CSARs we started off with yamllint [65]. In the future we will be adding new IaC checks – available and

those developed inside PIACARE – that will target other programming languages and IaC tools and according to that we will also be updating our IaC Scan Runner documentation [66] .

3.3.2 CLI

The use of IaC Scan Runner CLI is straightforward. After the installation, the *iac-scan-runner* command allows users to execute the shell commands from Table 5 (also displayed in Figure 11).

Table 5. IaC Scan Runner CLI commands

CLI command	Purpose and description
<i>iac-scan-runner openapi</i>	print OpenAPI Specification (YAML or JSON)
<i>iac-scan-runner install</i>	install the IaC Scan Runner prerequisites
<i>iac-scan-runner run</i>	run the IaC Scan Runner REST API

All the CLI commands are equipped with *-h/--help* option to display help.

```
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
```

Figure 11 Running IaC Scan Runner API using the CLI

3.4 Licensing information

IaC Scan Runner is licensed under open-source **Apache License 2.0**.

3.5 Download

The source code for IaC Scan Runner is available within [xlab-si/iac-scan-runner](https://github.com/xlab-si/iac-scan-runner) GitHub repository and the documentation is visible in [xlab-si/iac-scanner-docs](https://github.com/xlab-si/iac-scanner-docs) or can be explored publicly on [GitHub Pages](https://github.com/xlab-si/iac-scanner-docs). GitHub Actions are being used for the CI/CD tests and for building Docker images and packages.

4 Future plans

For the future of IaC Scan Runner we plan to a) expand the list of integrated checks requested by the use cases, b) collect the feedback of the quality of the checks and usability of the implementation of IaC Security Inspector and IaC Component Inspector, c) develop new checks where no useful check exists or improve existing ones, where possible and d) improve usability in the sense of improving the interface or security. In the following paragraphs we will describe each of the points above.

We plan to add multiple new IaC checks (a) according to the agreed check list, which will be able to find the remaining code and component vulnerabilities. First, we will explore and implement the addition of checks and linters for TOSCA templates to ensure the proper scanning of TOSCA CSARs. For this part we could use OpenStack TOSCA parser [30], xOpera orchestrator TOSCA parser [31] or any other applicable TOSCA tools. After that we will also focus on scanning Git repositories. Some checks that we will try to supply here are git-secrets [42], GitLeaks [41] and Markdown lint [43]. Apart from that we also need to add checks for other IaC components such as Docker and console scripts. Here we plan to use hadolint [32], Gixy [44] and ShellCheck [40] tools. Our target is also to expand the scanning on more programming languages with Checkstyle [36], HTMLHint [37], ESLint [38], TypeScript ESLint [39], Stylelint [67]. Lastly, we will take a closer look at adding remote service checks (for instance Snyk [22], SonarScanner [68], XLAB's Quality Scanner for Ansible playbooks [69], Mega-Linter [70] and others that are enough powerful to scan multiple IaC entities). By adding these remote checks, we will also need to ensure a secure configuration through the API. Apart from updating the API, one possible action could also be updating the IaC Scan Runner CLI in order not to just run the API but to communicate with it directly by exposing multiple CLI commands.

The feedback collection (b) will be done after the first PIACERE integration round. At this point the issues regarding the API will arise and users will be able to test our service which PoC is already available.

We will focus on creation of new checks (c). During our presented IaC check research, we have realized that especially Ansible automation tool and TOSCA orchestration standards are lacking tools and services that would support scanning their security issues. Therefore, we will specifically focus on integrating these kinds of checks. Initial steps can be done by validation of TOSCA templates with TOSCA parsers, but still there is no security checks. The same way some initial steps we have done with collaboration of the team working on Ansible Playbook Quality Scanner [69], but more research and more important user feedback is required to continue the work.

The user experience and usability (d) are crucial if we would like to create an exploitable asset. The future plan for IaC Scanner also includes developing the idea of IaC Scanner SaaS, which is meant to be the Software as a Service edition. It could support IaC scanning within IaC projects and workspaces along with secrets management, multi-tenancy, and multi-user experience. SaaS could be developed around the IaC Scan Runner, having its own REST API and GUI and may also be equipped with SaaS CLI that could facilitate the usage of the SaaS API from the CLI.

5 Conclusions

In the first deliverable *IaC Code security and component inspection* we present the work of first year on the PIACERE IaC Security Inspector and Component Security Inspector will represent a security component that will catch vulnerabilities in IaC packages from other PIACERE designed services. This inspection component will introduce an open-source service and will bring together multiple tools that can be used for IaC scanning such as linters, dependency checkers and even remote cloud services. The additional improvements of this component will lead in the better content - creation of new and better checks - and applying more enterprise features.

The main developed part is currently the IaC Scan Runner, which exposes a RESTful API that enables listing IaC checks, configuring them and scanning IaC packages. The JSON outputs from the API can be then used to fix the vulnerabilities in user's IaC. The API is shipped as a Docker container or can be distributed using the CLI package. Checks and tools are described thoroughly in the IaC Scanner documentation on GitHub Pages.

Future work is described in detail in separated section and will focus on new IaC checks and on developing new types of software (SaaS component or Eclipse plugins for example) that will bring easier integration and better user experience.

6 References

- [1] E. Morganti, A. Motta, L. Blasi, C. Nava and C. Bonferini, “D2.1 PIACERE DevSecOps Framework Requirements specification, architecture and integration strategy - v1,” 2021.
- [2] Ansible, “Ansible Galaxy,” [Online]. Available: <https://galaxy.ansible.com/>.
- [3] “Template Library documentation,” [Online]. Available: <https://xlab-si.github.io/xopera-docs/tps.html>.
- [4] J. Greig, “Cloud misconfigurations cost companies nearly \$5 trillion,” 2020. [Online]. Available: techrepublic.com/article/cloud-misconfigurations-cost-companies-nearly-5-trillion/.
- [5] A. Rahman and L. Williams, “Characterizing Defective Configuration Scripts Used for Continuous Deployment,” *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, pp. 34-45, 2018.
- [6] O. Hanappi, W. Hummer and S. Dustdar, “Asserting reliable convergence for configuration management scripts,” *OOPSLA 2016: Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, p. 328–343, 2016.
- [7] W. Hummer, F. Rosenberg, F. Oliveira and T. Eilam, “Automated Testing of Chef Automation Scripts,” *MiddlewareDPT '13: Proceedings Demo & Poster Track of ACM/IFIP/USENIX International Middleware Conference*, pp. 1-2, 2013.
- [8] A. Weiss, A. Guha and Y. Brun, “Tortoise: interactive system configuration repair,” *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, p. 625–636, 2017.
- [9] OWASP, “Dependency-Check,” [Online]. Available: <https://owasp.org/www-project-dependency-check/>.
- [10] W. Tech, “Terra Firma,” [Online]. Available: <https://github.com/wayfair/terrafirma>.
- [11] Progress, “Cookstyle,” [Online]. Available: <https://docs.chef.io/workstation/cookstyle/>.
- [12] “TFLint check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#tflint>.
- [13] Puppet, “puppet-lint,” [Online]. Available: <http://puppet-lint.com/>.
- [14] Retire.js, “RetireJS,” [Online]. Available: <https://retirejs.github.io/retire.js/>.
- [15] Hakiri, “Hakiri,” [Online]. Available: <https://hakiri.io/>.
- [16] OWASP, “Dependency-Check,” [Online]. Available: <https://owasp.org/www-project-dependency-check/>.

- [17] Google, “Wycheproof,” [Online]. Available: <https://github.com/google/wycheproof>.
- [18] A. Rahman, C. Parnin and L. Williams, “The Seven Sins: Security Smells in Infrastructure as Code Scripts,” *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 164-175, 2019.
- [19] A. Rahman, M. R. Rahman, C. Parnin and L. Williams, “Security Smells in Ansible and Chef Scripts: A Replication Study,” *ACM Transactions on Software Engineering and Methodology*, no. 1, pp. 1-31, 2021.
- [20] T. Loehr, “8 Best Practices for Securing Infrastructure as Code,” [Online]. Available: <https://cycode.com/blog/8-best-practices-for-securing-infrastructure-as-code/>.
- [21] P. Stats, “PyPI Stats,” [Online]. Available: <https://pypistats.org/>.
- [22] “Snyk,” [Online]. Available: <https://snyk.io/>.
- [23] SonarSource, “SonarCloud,” [Online]. Available: <https://sonarcloud.io/>.
- [24] “xOpera SaaS documentation,” [Online]. Available: <https://xlab-si.github.io/xopera-docs/saas.html>.
- [25] Ansible, “Ansible Lint documentation,” [Online]. Available: <https://ansible-lint.readthedocs.io/en/latest/>.
- [26] TFLint, “TFLint documentation,” [Online]. Available: <https://github.com/terraform-linters/tflint/tree/master/docs/user-guide>.
- [27] Tfsec, «Tfsec documentation,» [En línea]. Available: <https://tfsec.dev/docs/installation/>.
- [28] Terrascan, “Terrascan documentation,” [Online]. Available: <https://docs.accurics.com/projects/accurics-terrascan/en/latest/>.
- [29] Terraforma, “Terraforma documentation,” [Online]. Available: https://github.com/wayfair/terraforma/blob/master/docs/basic_usage.md.
- [30] OpenStack, “OpenStack TOSCA Parser documentation,” [Online]. Available: <https://github.com/openstack/tosca-parser/blob/master/doc/source/usage.rst>.
- [31] XLAB, “xOpera documentation,” [Online]. Available: <https://xlab-si.github.io/xopera-docs/>.
- [32] Hadolint, “Hadolint documentation,” [Online]. Available: <https://github.com/hadolint/hadolint/>.
- [33] Pylint, “Pylint documentation,” [Online]. Available: <http://pylint.pycqa.org/en/latest/>.
- [34] Bandit, “Bandit documentation,” [Online]. Available: <https://bandit.readthedocs.io/en/latest/>.
- [35] PyUp, “PyUp Safety documentation,” [Online]. Available: <https://pyup.io/docs/>.

- [36] Checkstyle, “Checkstyle documentation,” [Online]. Available: <https://checkstyle.org/> .
- [37] HTMLHint, “HTMLHint documentation,” [Online]. Available: <https://htmlhint.com/> .
- [38] ESLint, “ESLint documentation,” [Online]. Available: <https://eslint.org/>.
- [39] T. ESLint, “TypeScript ESLint documentation,” [Online]. Available: <https://typescript-eslint.io> .
- [40] ShellCheck, “ShellCheck documentation,” [Online]. Available: <https://www.shellcheck.net>.
- [41] GitLeaks, “GitLeaks documentation,” [Online]. Available: <https://github.com/zricethezav/gitleaks>.
- [42] git-secrets, “git-secrets documentation,” [Online]. Available: <https://github.com/awslabs/git-secrets/> .
- [43] M. lint, “Markdown lint documentation,” [Online]. Available: <https://github.com/markdownlint/markdownlint/blob/master/docs/configuration.md>.
- [44] Gixy, “Gixy documentation,” [Online]. Available: <https://github.com/yandex/gixy> .
- [45] yamllint, “yamllint documentation,” [Online]. Available: <https://yamllint.readthedocs.io/en/stable/> .
- [46] Dependency-Check, “Dependency-Check documentation,” [Online]. Available: https://www.owasp.org/index.php/OWASP_Dependency_Check.
- [47] “OpenAPI Specification,” [Online]. Available: <https://swagger.io/specification/>.
- [48] “Swagger UI,” [Online]. Available: <https://swagger.io/tools/swagger-ui/>.
- [49] “ReDoc,” [Online]. Available: <https://redoc.ly/redoc/>.
- [50] “Sphinx,” [Online]. Available: <https://www.sphinx-doc.org/en/master/>.
- [51] “Read the Docs,” [Online]. Available: <https://readthedocs.org/>.
- [52] “Python Package Index (PyPI),” [Online]. Available: <https://pypi.org/project/iac-scan-runner/>.
- [53] “IaC Scan Runner CLI package,” [Online]. Available: <https://pypi.org/project/iac-scan-runner/>.
- [54] “IaC Scan Runner package from Test PyPI,” [Online]. Available: <https://test.pypi.org/project/iac-scan-runner/>.
- [55] “FastAPI,” [Online]. Available: <https://fastapi.tiangolo.com/>.
- [56] “Typer,” [Online]. Available: <https://typer.tiangolo.com/>.

- [57] “xScanner Docker Hub organization,” [Online]. Available: <https://hub.docker.com/u/xscanner>.
- [58] “IaC Scan Runner Docker image,” [Online]. Available: <https://hub.docker.com/r/xscanner/runner>.
- [59] “tfsec check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#tfsec>.
- [60] “Terrscan check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#terrascan>.
- [61] “Ansible Lint check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#ansible-lint>.
- [62] “Pylint check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#pylint>.
- [63] “Bandit check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#bandit>.
- [64] “PyUp Safety check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#safety>.
- [65] “yamllint check,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/02-runner.html#yamllint>.
- [66] “IaC Scan Runner documentation,” [Online]. Available: <https://xlab-si.github.io/iac-scanner-docs/>.
- [67] Stylelint, “Stylelint documentation,” [Online]. Available: <https://stylelint.io/>.
- [68] SonarSource, “SonarScanner,” [Online]. Available: <https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/>.
- [69] X. Steampunk, “Ansible Playbook Quality Scanner,” [Online]. Available: <https://scanner.steampunk.si/>.
- [70] Mega-Linter, „Mega-Linter documentation,” [Elektronski]. Available: <https://megalinter.github.io/>.
- [71] S. Almuairfi and A. Mamdouh, “Security controls in infrastructure as code,” *Computer Fraud & Security*, no. 10, pp. 13-19, 2020.
- [72] I. Shkedy, “Does SAST Deliver? The Challenges of Code Scanning,” 2021. [Online]. Available: <https://www.traceable.ai/blog-post/does-sast-deliver-the-challenges-of-code-scanning>.