

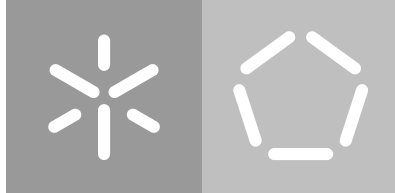
University of Minho

School of Engineering

Rafaela Maria Soares da Silva

**Infrastructure as Code:
Analysis of Misconfiguration and
Non-Compliance Problems**

June 2022



University of Minho

School of Engineering

Rafaela Maria Soares da Silva

**Infrastructure as Code:
Analysis of Misconfiguration and
Non-Compliance Problems**

Master's Dissertation

Integrated Master's in Informatics Engineering

Dissertation supervised by

Vitor Francisco Mendes Freitas Gomes Fonte (supervisor)

João Marco Cardoso Silva (co-supervisor)

Daniela da Cruz (supervisor in workplace)

June 2022

COPYRIGHT AND TERMS OF USE FOR THIRD PARTY WORK

This dissertation reports on academic work that can be used by third parties as long as the internationally accepted standards and good practices are respected concerning copyright and related rights.

This work can thereafter be used under the terms established in the license below.

Readers needing authorization conditions not provided for in the indicated licensing should contact the author through the RepositóriUM of the University of Minho.

LICENSE GRANTED TO USERS OF THIS WORK:



CC BY

<https://creativecommons.org/licenses/by/4.0/>

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Checkmarx, the company that allowed me to embrace this research. Especially to Daniela da Cruz, who regularly motivated, guided, and helped me throughout the dissertation progress. Her assistance and knowledge sharing were crucial. And also to everyone involved with KICS, who shared their knowledge with me and motivated me.

My warmest appreciation also goes to my supervisors, Vitor Francisco Mendes Freitas Gomes Fonte and João Marco Cardoso Silva, for their insightful observations, support and encouragement.

I would also like to extend my thanks to my "poconico" and my family for being patient with me and for their support since day one. Last but not least, thanks also should go to my friends, who supported my journey.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Infrastructure as Code: Analysis of Misconfiguration and Non-Compliance Problems

Infrastructure as Code (IaC) is an innovative DevOps approach to infrastructure configuration and management. Instead of using traditional interactive tools – such as command line – or cloud provider web interfaces, it automates several tasks through extensive use of scripting languages and tools.

Being a relatively new field, with a fast-paced developing set of tools, it is of crucial importance to assist its users and its developers to tackle security concerns that might affect the environments these tools are meant to manage. Some of those security concerns must always be handled within an actual live, running environment. This is the case, for example, of checking for service availability. Issues like this are already being addressed by existing dynamic analysis tools. Others should be handled using a static analysis approach, which, in turn, should prevent those security concerns from ever becoming a live security issue.

In this dissertation, we focus on trying to bridge the gap between the set of security checks currently being addressed by tools that follow these approaches. We identify 150 security checks currently being performed only by dynamic analysis tools, and we implement 23% of them in KICS, a Checkmarx-backed, open source, static code analysis tool for IaC solutions.

The new checks we contribute to KICS address misconfiguration and non-compliance problems that can be prevented using static analysis, mainly focusing on access control, but also on network security. Overall, this dissertation addresses 34 security checks, effectively bridging the gap between static and dynamic analysis for IaC in the KICS context.

Although not always possible, we strive to make available each security check to Ansible, CloudFormation, and Terraform. These new security checks and the necessary changes to KICS were submitted to the GitHub project's repository, were approved by the KICS team, and are now into its master branch. This means that new KICS releases will make available these security checks to its current users and to a broader audience, and, hopefully, will foster the development of community-based extensions and enhancements, such as support for other IaC platforms and security domains that we were unable to tackle due to time constraints.

Keywords: DevOps, Dynamic Analysis, Infrastructure as Code, KICS, Static Analysis.

RESUMO

Infrastructure as Code: Análise de Problemas de Misconfiguration e Non-Compliance

Infrastructure as Code (IaC) é uma prática inovadora de *DevOps* para configuração e gestão de infraestrutura. Em alternativa ao uso tradicional de ferramentas interativas — como a linha de comandos — ou interfaces *web* de *cloud providers*, IaC automatiza várias tarefas, através do uso de linguagens e ferramentas de *script*.

Por ser um campo novo, com um conjunto de ferramentas em desenvolvimento acelerado, é fulcral ajudar os seus utilizadores e *developers* a lidar com problemas de segurança, que possam afetar os ambientes que essas ferramentas devem gerir. Algumas dessas preocupações devem sempre ser tratadas num ambiente em execução. É o caso, p. ex., da verificação de *service availability*. Questões como esta já são abordadas por ferramentas de análise dinâmica. Outras devem ser tratadas através de uma abordagem de análise estática, que deve impedir que essas preocupações se tornem um problema de segurança ativo.

Nesta dissertação, focámo-nos em tentar preencher a lacuna entre o conjunto de *security checks* das ferramentas que seguem estas abordagens, atualmente. Identificámos 150 *security checks* atualmente realizadas apenas por ferramentas de análise dinâmica e implementámos 23% delas no KICS, uma ferramenta *open source* de análise estática de código, apoiada pela Checkmarx, para soluções de IaC.

As novas *security checks*, que contribuímos para o KICS, abordam problemas de *misconfiguration* e *non-compliance*, que podem ser evitados através de análise estática, com foco principal em *access control*, mas também em *network security*. No geral, esta dissertação aborda 34 *security checks*, preenchendo efetivamente a lacuna entre a análise estática e dinâmica para IaC, no contexto do KICS.

Embora nem sempre seja possível, esforçámo-nos para disponibilizar cada *security check* para Ansible, CloudFormation e Terraform. As novas *security checks* e alterações no KICS foram submetidas no repositório GitHub do KICS, foram aprovadas pela equipa do KICS e estão no *master branch*. Tal significa que as novas versões do KICS terão essas *security checks* para os seus utilizadores atuais e para um público mais amplo e, esperançosamente, promoverão contribuições da comunidade, como o suporte para outras plataformas de IaC e domínios de segurança que não conseguimos resolver devido a limitações de tempo.

Palavras-chave: *DevOps, Dynamic Analysis, Infrastructure as Code, KICS, Static Analysis.*

LIST OF CONTENTS

1	INTRODUCTION	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives	2
1.4	Methodology Approach	3
1.5	Document Structure	3
2	INFRASTRUCTURE AS CODE	5
2.1	Main Concepts	5
2.1.1	Infrastructure as Code	5
2.1.2	Code and Security Smells	7
2.1.3	IaC Misconfiguration and Non-Compliance Problems	8
2.1.4	Security Query	10
2.1.5	Static Analysis vs. Dynamic Analysis	11
2.2	Related Work	12
2.2.1	Prior Work on IaC Scripts	12
2.2.2	Scanning IaC Scripts using Static Code Analysis Tools	14
2.2.3	Dynamic Analysis Tools for Cloud-Based Infrastructure	16
2.2.3.1	Commercial Tools	16
2.2.3.2	Open Source Tools	18
2.3	Summary	18
3	KICS: A CASE STUDY	19
3.1	Static Analysis Tools vs. Dynamic Analysis Tools	19
3.1.1	Methodology	19
3.1.2	Results	21
3.2	Why KICS	33
3.3	KICS Overview	33

3.3.1	Architecture	33
3.3.2	Command Line Interface	34
3.3.3	Core	34
3.3.4	IaC Solutions	35
3.3.5	Queries Execution Engine	36
3.3.6	Results	38
3.4	The Problem	40
3.5	Challenges	40
3.5.1	Filtration of the Collected Security Queries	40
3.5.2	Implementation of the Selected Security Queries	41
3.6	Summary	41
4	FITTING NEW SECURITY QUERIES INTO KICS	43
4.1	Introduction	43
4.2	Implementing the Security Queries	43
4.2.1	[AWS_AC_08] API Gateway without WAF	44
4.2.2	[AWS_AC_10] API Gateway Without Configured Authorizer	44
4.2.3	[AWS_AC_11] Certificate Has Expired & [AWS_AC_66] Certificate RSA Key Bytes Lower Than 256	46
4.2.4	[AWS_AC_18] Elasticsearch Without IAM Authentication	47
4.2.5	[AWS_AC_24] Neptune Cluster With IAM Database Authentication Disabled	47
4.2.6	[AWS_AC_32] SES Policy With Allowed IAM Actions	48
4.2.7	[AWS_AC_35] IAM Access Analyzer Undefined	48
4.2.8	[AWS_AC_41] IAM Group Without Users	49
4.2.9	[AWS_AC_53] Cross-Account IAM Assume Role Policy Without ExternalId or MFA	49
4.2.10	[AWS_NS_05] Default EC2 security group are in use & [AWS_NS_07] Default VPC in use for EC2 instance	51
4.2.11	[AWS_NS_49] Elastic MapReduce Without VPC & [AWS_NS_50] ElastiCache Without VPC	52
4.2.12	[AWS_NS_51] ElastiCache Using Default Port & [AWS_NS_58] Relational Database Service (RDS) Using Default Port & [AWS_NS_61] Redshift Using Default Port	53

4.2.13	[AWS_NS_56] VPC Without Network Firewall	54
4.2.14	[AWS_NS_57] RDS Associated with Public Subnet	54
4.2.15	[AWS_NS_74] Shield Advanced Not In Use	55
4.2.16	[AZURE_AC_21] Role Assignment Not Limit Guest User Permissions	56
4.2.17	[AZURE_AC_22] Role Definition Allows Custom Role Creation	56
4.2.18	[AZURE_AC_29] Storage Share File Allows All ACL Permissions	57
4.2.19	[AZURE_AC_39] Storage Table Allows All ACL Permissions	57
4.2.20	[AZURE_NS_29] Virtual Network with DDoS Protection Plan Disabled	57
4.2.21	[GCP_AC_13] Service Account With Improper Privileges	58
4.2.22	[GCP_AC_16] IAM Role Assigned to User	58
4.2.23	[GCP_AC_17] User with KMS Admin and CryptoKey Roles	58
4.2.24	[GCP_AC_21] KMS Crypto Key is Publicly Accessible	59
4.2.25	[GCP_AC_23] Container Cluster Using Default Service Account	59
4.2.26	[GCP_NS_06] Google Compute Network Using Default Firewall Rule	59
4.2.27	[GCP_NS_07] Google Compute Network Using Firewall Rule that Allows All Ports & [GCP_NS_08] Google Compute Network Using Firewall Rule that Allows Port Range	60
4.2.28	[GCP_NS_43] Compute Subnetwork with Private Google Access Disabled	60
4.3	Discarded Queries	61
4.4	Summary	64
5	EXTENDING KICS	65
5.1	Setup of the KICS Development Environment	65
5.2	Development of the Security Queries	66
5.2.1	Metadata File	66
5.2.2	Query File	67
5.2.3	Security Query Development Example	68
5.3	Tests	72
5.4	Creation of the Pull Request	74
5.5	Contribution Overview	75
6	CONCLUSIONS AND FUTURE WORK	77
6.1	Conclusions	77

6.2 Future Work	78
References	79

LIST OF FIGURES

Figure 1	Item 3.9 of the CIS Amazon Web Services Foundations Benchmark v1.4.0 (Source: CIS)	10
Figure 2	Example of incidents that can be detected with Bridgecrew	17
Figure 3	KICS Architecture (Source: KICS Website)	34
Figure 4	KICS results in Command Line Interface (CLI)	38
Figure 5	KICS GitHub repository fork	65
Figure 6	KICS GitHub repository fork clone link	66
Figure 7	REGO Playground	67
Figure 8	REGO Playground	70
Figure 9	Pull request	74

LIST OF TABLES

Table 1	Catalog of 24 code smells for Puppet by Sharma et al. (2016)	13
Table 2	Catalog of 17 code smells for Chef by Schwarz et al. (2018)	13
Table 3	Static code analysis tools for IaC	15
Table 4	Commercial dynamic analysis tools for cloud-based infrastructure	17
Table 5	Open source dynamic analysis tools for cloud-based infrastructure	18
Table 6	[AWS] Access Control context - Part I	23
Table 7	[AWS] Access Control Part II	24
Table 8	[AZURE] Access Control context	25
Table 9	[GCP] Access Control context	26
Table 10	[AWS] Network Security context - Part I	27
Table 11	[AWS] Network Security - Part II	28
Table 12	[AZURE] Network Security	29
Table 13	[GCP] Network Security context	30
Table 14	Collected security queries according to IaC problem	32
Table 15	IaC solutions configuration file extension	35
Table 16	[AWS_AC_08] API Gateway without WAF	44
Table 17	[AWS_AC_10] API Gateway Without Configured Authorizer	46
Table 18	[AWS_AC_11] Certificate Has Expired & [AWS_AC_66] Certificate RSA Key Bytes Lower Than 256	47
Table 19	[AWS_AC_18] Elasticsearch Without IAM Authentication	47
Table 20	[AWS_AC_24] Neptune Cluster With IAM Database Authentication Disabled	47
Table 21	[AWS_AC_32] SES Policy With Allowed IAM Actions	48
Table 22	[AWS_AC_35] IAM Access Analyzer Undefined	49
Table 23	[AWS_AC_41] IAM Group Without Users	49
Table 24	[AWS_AC_53] Cross-Account IAM Assume Role Policy Without ExternalId or MFA	51

Table 25	[AWS_NS_05] Default EC2 security group are in use	51
Table 26	[AWS_NS_07] Default VPC in use for EC2 instance	52
Table 27	[AWS_NS_49] Elastic MapReduce Without VPC	52
Table 28	[AWS_NS_50] ElastiCache Without VPC	52
Table 29	Default ports	53
Table 30	[AWS_NS_51] ElastiCache Using Default Port	53
Table 31	[AWS_NS_58] Relational Database Service (RDS) Using Default Port	53
Table 32	[AWS_NS_61] Redshift Using Default Port	54
Table 33	[AWS_NS_56] VPC Without Network Firewall	54
Table 34	[AWS_NS_57] RDS Associated with Public Subnet	55
Table 35	[AWS_NS_74] Shield Advanced Not In Use	56
Table 36	[AZURE_AC_21] Role Assignment Not Limit Guest User Permissions	56
Table 37	[AZURE_AC_22] Role Definition Allows Custom Role Creation	56
Table 38	[AZURE_AC_29] Storage Share File Allows All ACL Permissions	57
Table 39	[AZURE_AC_39] Storage Table Allows All ACL Permissions	57
Table 40	[AZURE_NS_29] Virtual Network with DDoS Protection Plan Disabled	57
Table 41	[GCP_AC_13] Service Account With Improper Privileges	58
Table 42	[GCP_AC_16] IAM Role Assigned to User	58
Table 43	[GCP_AC_17] User with KMS Admin and CryptoKey Rules	58
Table 44	[GCP_AC_21] KMS Crypto Key is Publicly Accessible	59
Table 45	[GCP_AC_23] Container Cluster Using Default Service Account	59
Table 46	[GCP_NS_06] Google Compute Network Using Default Firewall Rule	60
Table 47	[GCP_NS_07] Google Compute Network Using Firewall Rule that Allows All Ports & [GCP_NS_08] Google Compute Network Using Firewall Rule that Allows Port Range	60
Table 48	[GCP_NS_43] Compute Subnetwork with Private Google Access Disabled	61
Table 49	List of the discarded queries according to "dynamic", "known", "inapplicable", and "unknown" scopes	63
Table 50	List of security queries contributed to KICS	76

LIST OF LISTINGS

1	AWS CLI command example	6
2	Terraform script	6
3	AWS Access Key hard-coded	8
4	IaC Misconfiguration example	8
5	Payload example	35
6	Terraform file example	36
7	Query file tree	36
8	Metadata example	36
9	Query example	37
10	KICS results in a JSON report	38
11	All actions to all principals	48
12	MFA configuration	49
13	External ID configuration	50
14	Payload example	68
15	metadata.json of the query "Neptune Cluster With IAM Database Authentication Disabled"	70
16	query.rego of the query "Neptune Cluster With IAM Database Authentication Disabled"	71
17	Test folder tree	72
18	Positive sample example ('positive1.tf')	72
19	Positive sample example ('positive2.tf')	72
20	Negative sample example ('negative.tf')	73
21	Positive expected result sample example ('positive_expected_result.json')	73

INTRODUCTION

1.1 Context

Cloud Computing Services assert themselves to the detriment of other data management and processing forms. Instead of using our hardware to satisfy requirements from storage to management and processing of data, these services offer the possibility to call upon remote servers on the Internet. "Software is remotely deployed in a virtualized runtime environment using shared hardware/software resources, and hosted in a third-party infrastructure", as maintained by [Bai et al. \(2011\)](#).

That is possible through approaches like Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) that involve the management of some parts of the stack instead of the full traditional stack. That is the most likely reason why more than 90% of companies use at least one Cloud Computing Service¹. Examples of well-known providers are Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure.

Apart from these services, Infrastructure as Code (IaC) appears as a modern way of infrastructure configuration and management. According to [Guerriero et al. \(2019\)](#), this technology can be seen as a DevOps practice that provides configuration and management of infrastructure through scripts instead of traditional interactive tools — such as command line — or cloud provider web interfaces. Examples of IaC technologies are Ansible², Chef³, CloudFormation⁴, Docker⁵, Kubernetes⁶, Puppet⁷, and Terraform⁸, among others.

IaC technologies present substantial benefits over configuration management and server provisioning compared to other alternatives. In addition to being a faster process, they also allow the reuse of scripts and implementation of software engineering practices.

¹ <https://techjury.net/blog/how-many-companies-use-cloud-computing/>

² <https://docs.ansible.com/ansible/latest/index.html>

³ <https://docs.chef.io/>

⁴ <https://docs.aws.amazon.com/cloudformation/>

⁵ <https://www.docker.com/>

⁶ <https://kubernetes.io/>

⁷ <https://puppet.com/>

⁸ <https://www.terraform.io/>

However, during the development of these scripts, inadvertent infrastructure misconfigurations, non-compliance problems, or security vulnerabilities can occur, either because of a lack of awareness of the best security practices in IaC, or even due to bugs in the code. Additionally, various challenges can arise from these technologies, the main one being testability, as stated by Guerriero et al. (2019).

1.2 Motivation

According to Google Trends⁹, Infrastructure as Code started gaining traction in 2015, which indicates that this practice is a recent field. As with any recent field of study, there are several significant subjects to investigate and explore. In this dissertation, however, the focus will be on the analysis of misconfiguration and non-compliance problems in IaC scripts.

Before IaC code is executed, the best way to prevent potential vulnerabilities in cloud infrastructure (as a result of misconfiguration and non-compliance problems from IaC scripts) is to focus on a static approach. However, this type of approach has disadvantages compared to a dynamic one, which results in a gap between the static analysis tools for IaC and dynamic tools for cloud-based infrastructure. The exploration of this gap can benefit the IaC community since it will be new material to consider and contribute to a more extensive testability of IaC scripts.

For that matter, the present dissertation explores how information usually collected by dynamic tools for cloud-based infrastructure can be fully or partially implemented or addressed in an IaC static approach in a specific context.

1.3 Objectives

As a starting point, it is necessary to understand the concept of IaC to better grasp how these technologies work. That is essential to perceive the best security practices desired in this type of technology.

Secondly, it is essential to study dynamic analysis tools for cloud-based infrastructure that already exist and identify what problems and information they can recognize in a specific context. Subsequently, the present study intends to verify which of those problems and information recognized by the dynamic analysis tools for cloud-based infrastructure are not identified by IaC static analysis tools.

As a final goal, the study aims to pinpoint which of the potential problems identified by the dynamic approach are feasible to be transposed to a static one.

To this end, the main objectives are:

- Understand the IaC concept and explore its technologies.

⁹ <https://trends.google.com/trends/explore?date=all&geo=US&q=infrastructure%20as%20code>

- Study the gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure.
- Study and attempt to define solutions bridging the gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure in a static approach.
- Contribute solutions to KICS (Keeping Infrastructure as Code Secure), an open source static analysis tool for IaC provided by Checkmarx.

1.4 Methodology Approach

The methodology followed in this dissertation consists of the following eight steps:

1. Literature review about all aspects considered relevant, supported by scientific articles, publications, and documentation.
2. Identify and explore static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure.
3. Choose a security context, between the several existing domains of security problems, to compare both kinds of tools.
4. Collect the information that both kinds of tools recognize in a given context.
5. Study the gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure from the information collected; Present the results of the study.
6. Study and attempt to define solutions bridging the gap in a static approach; Contribute the solutions to KICS.
7. Simultaneously to the previous task, tests the effectiveness of the proposed solutions.
8. Focus on the writing of the dissertation and, later, revision.

1.5 Document Structure

The present document is composed of four main chapters, preceded by this Introduction (Chapter 1) and succeeded by Conclusions and Future Work (Chapter 6).

Chapter 1, the current one, is an introduction to the project, which provides an explanation of the dissertation context, including the motivation, objectives, and methodology approach.

Chapter 2, Infrastructure as Code, is constituted by two main sections, Infrastructure as Code (2.1) and Related Work (2.2). Section 2.1 includes all relevant background necessary to understand the context of this dissertation. For its part, Section 2.2 highlights an overview of prior work on IaC scripts.

Chapter 3, KICS: A Case Study, details the case study of the present dissertation, i.e., a comparison between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure in the defined context. Furthermore, it reports its results.

Chapter 4, Fitting New Security Queries Into KICS, highlights how the dissertation study results can or cannot be implemented in KICS. Moreover, it presents a set of reasons regarding the queries discard process.

Chapter 5, Extending KICS, addresses the application of the solutions presented in Chapter 4 to KICS. Therefore, it lists all the solutions implemented in the KICS GitHub repository, including the GitHub pull request related to each one.

Chapter 6, Conclusions and Future Work, presents the conclusions of the present dissertation, focusing on contributions to KICS and the prospect for future work.

INFRASTRUCTURE AS CODE

This chapter details the essential concepts needed to understand Infrastructure as Code. Moreover, it provides an overview of the related work in this field.

2.1 Main Concepts

2.1.1 *Infrastructure as Code*

DevOps is the collaboration result between Dev (Development) and Ops (Operations). This concept comes up with two different needs in mind. On the one hand, the demand for speed by the developers: since they need to deliver software products as soon as possible, they need a fast deployment and release process, as stated by [Guerriero et al. \(2019\)](#). On the other hand, the demand for system stability by the system operations. Any change in the system could present a risk of instability, which hampers its maintenance.

In prior years, both teams used to work separately. Ops teams had to manage a great deal of hardware manually to deploy and run software products done by the Dev team. According to [Brikman \(2019\)](#), this might lead to mistakes since the number of servers increases with time, resulting in slow and uncertain releases. Consequently, the system becomes unstable, which delays its delivery.

However, with the emergence of the Cloud Computing concept, Ops teams no longer have to manage the full traditional stack. Cloud Computing Services¹ offer the possibility to manage just some parts of the stack. This management is done by software, which brings both teams together.

With that being said, it seems that the division of work and teams has become obsolete. There are several DevOps practices such as Continuous Integration, Continuous Delivery, Continuous Deployment, Build Automation, Configuration Management, Orchestration, Monitoring, Microservices, and Infrastructure as Code (the main subject of this dissertation). All these practices allow both teams to cooperate and achieve speed and stability.

¹ Infrastructure as a Service, Platform as a Service, Software as a Service, and Function as a Service

Infrastructure as Code (IaC) is a DevOps practice that provides configuration and management of infrastructure through scripts instead of traditional interactive tools – such as command line – or cloud provider web interfaces. [Guerriero et al. \(2019\)](#) These files are known as IaC scripts, configuration scripts, or configuration as code scripts. [Rahman, Parnin and Williams \(2019\)](#)

Concerning configuration management (software installation and management on existing servers), the most popular tools are Chef², Puppet³, Ansible⁴, and SaltStack⁵. On the other hand, for server provisioning, well-known tools are Terraform⁶, CloudFormation⁷, and OpenStack Heat⁸. [Brikman \(2019\)](#)

As said before, according to Google Trends⁹, the interest in this modern practice started to increase in 2015. Major IT companies such as GitHub, Mozilla, Netflix, Google, and Facebook have adopted this approach. [Rahman, Mahdavi-Hezaveh and Williams \(2019\)](#)

Practitioners use IaC scripts because this technology is faster than the manual execution of shell commands and interactive web pages. Moreover, they also use this approach due to re-usability and reliability. [Sharma et al. \(2016\)](#) Beyond that, IaC allows the implementation of software engineering practices like self-service, consistency, documentation, version control, and validation. [Brikman \(2019\)](#)

Imagine a scenario with an Amazon EC2 instance launch. Through an interactive web page, more precisely the AWS Management Console, the necessary steps are presented in the official AWS documentation¹⁰.

Through manual execution of shell commands (in this context, AWS CLI) after configuring security credentials, a command similar to Listing 1 is required:

```
aws ec2 run-instances --image-id ami-40d28157 --count 1
--instance-type t2.micro
```

Listing 1: AWS CLI command example.

The previous example does not provide a practical way of documentation, version control, validation, consistency, or re-usability. Instead of using them, one option is to use an IaC technology, which can offer these features, as shown in Listing 2.

```
provider "aws" {
  region = "us-west-2"
```

² <https://www.chef.io/>

³ <https://puppet.com/>

⁴ <https://docs.ansible.com/ansible/latest/index.html>

⁵ <https://saltproject.io/>

⁶ <https://www.terraform.io/>

⁷ <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>

⁸ <https://docs.openstack.org/heat/latest/>

⁹ <https://trends.google.com/trends/explore?date=all&geo=US&q=infrastructure%20as%20code>

¹⁰ <https://docs.aws.amazon.com/quickstarts/latest/vmlaunch/step-1-launch-instance.html>

```
}  
  
resource "aws_instance" "example" {  
    ami = ami-40d28157  
    instance_type = "t2.micro"  
}
```

Listing 2: Terraform script.

This kind of script can be easily changed or reused, which provides a more intuitive way of configuring and managing infrastructure. However, during the development of the IaC scripts, inadvertent infrastructure misconfigurations, non-compliance problems, or security vulnerabilities can occur. It can arise from a lack of the best security practices in IaC, misconfigurations, code smells, security smells, or even due to bugs in the code.

The possibility of lack of the best security practices in IaC, misconfigurations, code smells, security smells, and bugs in the code seem to raise concerns for developers. As stated by [Guerriero et al. \(2019\)](#), the main one being testability. Since this technology is recent, the field is under construction when it comes to the security of these scripts.

The IaC platforms use unique code structures, which require specific code implementation for each one. Finding a range of tool options that incorporates the same target platform or all the platforms is quite hard. In addition to that, need to be updated often and push forward to support more and more cloud solutions. Consequently, the tools that scan the platform's scripts also need to be constantly updated.

Finally, the best approach to analyze IaC scripts is static analysis - a static analysis tool for IaC that runs security queries. Nevertheless, there is a gap between the static analysis tools for IaC and dynamic tools for the cloud-based infrastructure. Chapter 3 addresses this issue.

2.1.2 Code and Security Smells

Code smells are "flaws in code which may lead to problems". They do not "lead to a run-time error but usually indicate that the code needs to be improved". [Schwarz et al. \(2018\)](#) For example, incomplete tasks can be considered as a code smell, which can lead to misconfigurations since the environment is not completely configured.

On the other hand, as defined by [Rahman, Parnin and Williams \(2019\)](#), "security smells are recurring coding patterns that are indicative of security weakness, and requires further inspection". As an example of this, consider a Terraform file with AWS as the cloud provider. As a best practice, AWS Access Key should not be hard-coded in the script, as depicted below in Listing 3:

```
provider "aws" {  
  region = "us-west-2"  
  access_key = "my_access_key"  
}
```

Listing 3: AWS Access Key hard-coded.

If the AWS Access Key is hard-coded, anyone who has access to the file will have access to the credentials, which is something to avoid.

2.1.3 IaC Misconfiguration and Non-Compliance Problems

Misconfiguration problems can be seen as an incorrect or inadequate configuration of the infrastructure. The usage of default settings or deprecated protocols and unsafe configurations (e.g., not using the latest Transport Layer Security (TLS) version) are examples of this type of problem.

An example of a misconfiguration problem can be the definition of an AWS Redshift cluster in Terraform with default settings for the optional parameters. When the field 'publicly_accessible' (optional parameter) is not defined in the resource 'aws_redshift_cluster', the default value will be used. Since the default value is true, the Redshift cluster will be accessible from a public network¹¹. It is important to read the documentation and be aware of this type of problem. See Listing 4 for a suggestion of remediation for this problem.

```
resource "aws_redshift_cluster" "example" {  
  cluster_identifier = "tf-redshift-cluster"  
  database_name      = "mydb"  
  master_username    = "exampleuser"  
  master_password    = "Mustbe8characters"  
  node_type          = "dc1.large"  
  cluster_type       = "single-node"  
  publicly_accessible = false  
}
```

Listing 4: IaC Misconfiguration example.

¹¹ https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/redshift_cluster#publicly_accessible

Non-compliance problems occur when standards are not followed. The standards help the developers to establish a guide of best practices to prevent unsafe configurations. CIS (Center for Internet Security) benchmarks are examples of well-known sets of standards.

There are several CIS benchmarks available¹², such as CIS Benchmarks for Amazon Web Services Foundations, CIS Benchmarks for Google Cloud Platform Foundation, and CIS Benchmarks for Microsoft Azure Foundations, which are examples of relevant ones in the cloud-based infrastructure context. Each item of the benchmark presents the following topics: (i) a description of the best practice, (ii) a rationale statement that explains the importance of the rule, (iii) an impact statement that presents the possible consequences of not following the best practice, (iv) an audit procedure that demonstrates how to verify if the recommendation is already applied, and (v) remediation that details how to follow the best practice, among others. See Figure 1 as an example of a best practice from the CIS Amazon Web Services Foundations Benchmark v1.4.0.

¹² <https://www.cisecurity.org/cis-benchmarks/>

3.9 Ensure VPC flow logging is enabled in all VPCs

↑ PARENT : CIS Amazon Web Services Foundations Benchmark 1.2.0 | 2.9-Ensure VPC flow logging is enabled in all

◀ ▶ Edit Propose Change Duplicate Export Profiles

Automated

Applicable Profiles

Level 2

Description

VPC Flow Logs is a feature that enables you to capture information about the IP traffic going to and from network interfaces in your VPC. After you've created a flow log, you can view and retrieve its data in Amazon CloudWatch Logs. It is recommended that VPC Flow Logs be enabled for packet "Rejects" for VPCs.

Rationale Statement

VPC Flow Logs provide visibility into network traffic that traverses the VPC and can be used to detect anomalous traffic or insight during security workflows.

Impact Statement

By default, CloudWatch Logs will store Logs indefinitely unless a specific retention period is defined for the log group. When choosing the number of days to retain, keep in mind the average days it takes an organization to realize they have been breached is 210 days (at the time of this writing). Since additional time is required to research a breach, a minimum 365 day retention policy allows time for detection and research. You may also wish to archive the logs to a cheaper storage service rather than simply deleting them. See the following AWS resource to manage CloudWatch Logs retention periods:

1. <https://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/SettingLogRetention.html>

Audit Procedure

Perform the following to determine if VPC Flow logs is enabled:

From Console:

1. Sign into the management console
2. Select **Services** then **VPC**
3. In the left navigation pane, select **Your VPCs**
4. Select a VPC
5. In the right pane, select the **Flow Logs** tab.
6. Ensure a Log Flow exists that has **Active** in the **Status** column.

Remediation Procedure

WARNING: The contents of this section may not render correctly in the Word Export

Perform the following to determine if VPC Flow logs is enabled:

From Console:

1. Sign into the management console
2. Select **Services** then **VPC**
3. In the left navigation pane, select **Your VPCs**
4. Select a VPC
5. In the right pane, select the **Flow Logs** tab.
6. If no Flow Log exists, click **Create Flow Log**

Figure 1: Item 3.9 of the CIS Amazon Web Services Foundations Benchmark v1.4.0 (Source: CIS).

2.1.4 Security Query

Also known as a security check or guide, a security query can detect possible security vulnerabilities, code and security smells, misconfigurations, non-compliance problems, or the lack of best practices. It can cover several categories, such as:

- **Access Control:** This technique offers preventive mechanisms to avoid untrusted individuals (authentication) and restrict access permissions to data (authorization). If these two types of processes are not implemented or are misconfigured, it can result in a vulnerability or lack of best practice. For example, if the authentication is disabled in some environment, anyone can gain access to it.
- **Backup:** Backup is a (periodic) copy of the environment used to keep its most up-to-date data. It provides recovery from lost or corrupted data. Therefore, this feature should be enabled and properly configured.
- **Encryption:** Ensuring secure communication in an open channel is essential to prevent information leaks. The same applies even if it is a private channel. Enabling encryption, ensuring encryption key rotation, and forcing the usage of HTTPS are examples of measures to verify in this category.
- **Network Security:** This category protects the connection of an environment. When talking about network security, aspects like restricting access to the network and implementing a VPC (Virtual Private Cloud) are essential.
- **Observability:** This category refers to the diagnosis of the operational environment. Keeping logs is an example of a way to investigate possible errors or uncommon behaviors.

2.1.5 *Static Analysis vs. Dynamic Analysis*

Static analysis does not imply code execution since it is applied to the source code or the binaries of the system. [Silva and Campos \(2013\)](#) Therefore, this approach can be implemented in the early phase of the software development life cycle, which can lighten the costs of the validation software phase, save time, and prevent potential vulnerabilities.

There are several types of tools using static code analysis. Their main goals range between code best practices enforcement, code improvement, bug detection, and vulnerabilities detection. Examples of them are:

- **Linting tools:** They focus mainly on syntax errors, best practices enforcement, code improvement, and bug detection. These tools are available for almost all programming languages. As an example of an open source one, there is Terraform Linter¹³.
- **Security tools:** They detect and report vulnerabilities, bugs, and lack of best practices. These tools are essential to prevent potential software defects that can compromise the system during the early phase of the software development life cycle.

¹³ <https://github.com/terraform-linters/tflint>

As most people involved with software feel the need to detect software defects as soon as possible, these tools are in great demand. To meet this need, several companies rely on these tools, as is the case of Checkmarx which provides the CxSAST product¹⁴. Apart from the commercial ones, companies and the IT community provide open source tools. KICS (Keeping Infrastructure as Code Secure)¹⁵, provided by Checkmarx, is an example of it.

Nevertheless, this approach cannot identify vulnerabilities introduced at run-time, leading to the need for dynamic analysis approaches. Different from static analysis, dynamic analysis implies code execution, i.e., observation of the run-time system behavior. [Silva and Campos \(2013\)](#)

Through the examination, dynamic tools can provide **vulnerability management** that identifies the vulnerabilities in the run-time environment, and **network visibility**, which is important to monitor the components of the network, for example. Prisma Cloud¹⁶ is an example of a dynamic analysis tool that offers these benefits.

2.2 Related Work

This section presents prior work on IaC scripts. Furthermore, it introduces static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure.

2.2.1 Prior Work on IaC Scripts

In the scope of smells, through the analysis of Puppet repositories, [Sharma et al. \(2016\)](#) present a catalog of 24 code smells (Table 1) distinguished into two categories: **Implementation Configuration Smells** (13) and **Design Configuration Smells** (11).

¹⁴ <https://checkmarx.com/product/cxsast-source-code-scanning/>

¹⁵ <https://github.com/Checkmarx/kics>

¹⁶ <https://docs.paloaltonetworks.com/prisma/prisma-cloud.html>

Table 1: Catalog of 24 code smells for Puppet by Sharma et al. (2016).

Implementation Code Smells	Design Configuration Smells
Missing Default Case	Multifaceted Abstraction
Inconsistent Naming Convention	Unnecessary Abstraction
Complex Expression	Imperative Abstraction
Duplicate Entity	Missing Abstraction
Misplaced Attribute	Insufficient Modularization
Improper Alignment	Duplicate Block
Invalid Property Value	Broken Hierarchy
Incomplete Tasks	Unstructured Module
Deprecated Statement Usage	Dense Structure
Improper Quote Usage	Deficient Encapsulation
Long Statement	Weakened Modularity
Incomplete Conditional	
Unguarded Variable	

From this study, Schwarz et al. (2018) investigated the feasibility of applying this catalog into Chef scripts. To this end, they identified the five most frequent smells in each category and implemented them into a static code analysis tool named Foodcritic. In this investigation, they identified three types of IaC smells: (i) **Technology Agnostic Smells** that can be adapted from Puppet smells, without changing the detection method, (ii) **Technology Dependent Smells**, which cannot be directly adopted due to the differences between Puppet and Chef, so it is necessary to change the detection method and (iii) **Technology Specific Smells** that are only applicable to a specific IaC technology. From this analysis, they defined a catalog of 17 Chef IaC smells (Table 2).

Table 2: Catalog of 17 code smells for Chef by Schwarz et al. (2018).

Agnostic Smells	Dependent Smells	Specific Smells
Improper Alignment	Improper Quote Usage	Hyphens
Long Statement	Insufficient Modularization	Empty Default
Unguarded Variable	Weakened Modularity	
Misplaced Attribute	Unstructured Module	
Multifaceted Abstraction	Law of Demeter	
Duplicate Block	Include Consistency	
Long Resource		
Too many Attributes		
Avoid Comments		

Later, Rahman, Parnin and Williams (2019), through a quantitative analysis of 1,726 IaC scripts, identified seven signs of security smells in IaC scripts: (1) **admin by default**, (2) **empty password**, (3) **hard-coded secret**, (4) **invalid IP address binding**, (5) **suspicious comment**, (6) **use of HTTP without**

TLS, and (7) **use of weak cryptography algorithms**. To identify the occurrence of these smells, they implemented and validated a static analysis tool called SLIC (Security Linter for Infrastructure as Code scripts).

However, [Almuairfi and Alenezi \(2020\)](#) recognize **permissions** and **configuration path** as security smells to consider, in addition to the smells presented by the studies of [Schwarz et al. \(2018\)](#) and [Rahman, Parnin and Williams \(2019\)](#). Also, [Almuairfi and Alenezi \(2020\)](#) suggest what they considered to be the best security practices in IaC: (i) **manual security assessment**, which implies inspection of the live infrastructure before and after deployments (ii) **codify everything** related to the infrastructure specifications in IaC scripts, discarding the manually changes (iii) **IaC documentation** should be reduced to the minimum since the infrastructure documentation will have the status automatically registered by the IaC scripts, (iv) **version everything** is important to version control any changes in the IaC scripts, (v) **continuously test system, integrate and deploy** to ensure a test environment before deployment, (vi) **modular code** in order to prefer small changes to big ones, (vii) **immutable infrastructure** that suggests replacement of the infrastructure elements rather than changing them, and (viii) **continuous security and service availability** to also ensure security in the continuous delivery toolchain and test environment.

2.2.2 Scanning IaC Scripts using Static Code Analysis Tools

Apart from the studies mentioned in the previous section, several static analysis tools for IaC are currently available. They detect possible flaws in IaC technologies according to best practices from cloud providers and IaC technologies.

Referring to the Terraform script presented in Section 2.1.2 with an AWS Access Key hard-coded. As a best practice in IaC scripts, it is not recommended to use hard-coded credentials in Terraform¹⁷. That is considered a security smell by [Rahman, Parnin and Williams \(2019\)](#), as recognizable in Section 2.2.1.

Considering another scenario with the same provider (AWS), IaC technology (Terraform), and a Virtual Private Cloud (VPC) configuration. As an AWS best practice¹⁸ it is recommended to enable VPC Flow Logs to "capture information about IP traffic going to and from network interfaces in our VPC".

Both cases can be detected by static code analysis tools that scan Terraform scripts (or other technologies depending on their restrictions), which covers these checks. These static code analysis tools are fundamental to ensure best practices and prevent security flaws in the cloud before IaC scripts execution.

¹⁷ <https://registry.terraform.io/providers/hashicorp/aws/latest/docs#static-credentials>

¹⁸ <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security-best-practices.html>

Of the open source static analysis tools available, Ansible Linter¹⁹, CFN_NAG²⁰, CFRipper²¹, Checkov²², Haskell Dockerfile Linter²³, KICS²⁴, Kube-Score²⁵, Puppet linter²⁶, Terrafirma²⁷, Terrascan²⁸, TFLint²⁹, and TFSEC³⁰ stand out. Companies like Aqua³¹ and Snyk³² also offer static analysis products for IaC scripts.

Table 3 presents the IaC technologies supported by the IaC tools mentioned above. Note that this information is based on the tools documentation understanding from around the end of 2020. Also, it is important to mention that almost none of the above tools supports more than three IaC technologies, except KICS.

Table 3: Static code analysis tools for IaC.

	Ansible	Chef	CloudFormation	Dockerfile	Kubernetes	Puppet	Terraform
Ansible Linter	✓						
Aqua Wave IaC			✓				✓
CFN_NAG			✓				
CFRipper			✓				
Checkov			✓		✓		✓
Haskell Dockerfile Linter				✓			
KICS	✓		✓	✓	✓		✓
Kube-Score					✓		
Prisma Cloud IaC			✓		✓		✓
Puppet Linter						✓	
Snyk IaC					✓		✓
Terrafirma							✓
Terrascan					✓		✓
TFLint							✓
TFSEC							✓

To clearly understand how these tools work, Section 3.3 presents an overview of KICS (Keeping Infrastructure as Code Secure).

19 <https://github.com/ansible-community/ansible-lint>

20 https://github.com/stelligent/cfn_nag

21 <https://github.com/Skyscanner/cfripper>

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

23 <https://github.com/hadolint/hadolint>

24 <https://github.com/Checkmarx/kics>

25 <https://github.com/zegl/kube-score>

26 <https://github.com/rodjek/puppet-lint>

27 <https://github.com/wayfair/terrafirma>

28 <https://github.com/accurics/terrascan>

29 <https://github.com/terraform-linters/tflint>

30 <https://github.com/tfsec/tfsec>

31 <https://wave-support.aquasec.com/support/home>

32 <https://snyk.io/product/infrastructure-as-code-security/>

2.2.3 *Dynamic Analysis Tools for Cloud-Based Infrastructure*

Currently, on the market, there are plenty of dynamic analysis tools capable of inferring data about the infrastructure of the system and finding security issues. These tools can have different features, such as compliance checks, incident response, malware detection, misconfigurations identification, network management, run-time defense, threat detection, and vulnerability management, among others. However, their main goal is to ensure best practices to prevent security flaws.

2.2.3.1 *Commercial Tools*

Currently, several commercial dynamic analysis tools for cloud-based infrastructure are available. All of them implement security checks on the cloud environments that they support, providing several features. See Table 4 (note that the provider's coverage in the table only considers AWS, AZURE, or GCP).

For instance, Bridgecrew³³ offers a platform that permits finding and fixing issues in the cloud. It is possible to infer AWS, Kubernetes, GCP, and Azure misconfigurations in run-time through policies. These policies cover various categories, such as logging, networking, secrets, identity and access management, and others.

Consider a scenario where this platform should check compliance and the security of an AWS account. As a requirement, the user only needs to launch an instance (associated with Bridgecrew) in their AWS account. Figure 2 represents an example of which incidents can be possibly detected.

³³ <https://bridgecrew.io/>






Ensure access keys are rotated every 90 days or less Policy ID: BC_AWS_IAM_4	Last Tue at 05:28 am 
Ensure roles unused in the last 90 days are removed Policy ID: BC_AWS_IAM_34	Last Mon at 02:39 pm 
Ensure users unused in the last 90 days with ADMIN privileges are removed Policy ID: BC_AWS_IAM_37	Last Mon at 02:39 pm 
Ensure credentials unused for 90 days or greater are disabled Policy ID: BC_AWS_IAM_3	Last Mon at 06:08 am 
Ensure a log metric filter and alarm exist for IAM policy changes Policy ID: BC_AWS_MONITORING_4	Last Sat at 01:39 pm 

Figure 2: Example of incidents that can be detected with Bridgecrew.

Prisma Cloud³⁴ provides vulnerability management, compliance checks, runtime-defense, network visibility, incident response, and forensics in AWS, AZURE, GCP, and Alibaba Cloud.

Aqua Enterprise³⁵ scans container images supported on vulnerability data sources in AWS, AZURE, GCP, and Oracle Cloud Infrastructure. Furthermore, this tool finds malware and misconfigurations.

Cloud One - Conformity³⁶ contributes to real-time monitoring and auto-remediation in AWS and Microsoft Azure environments.

Table 4: Commercial dynamic analysis tools for cloud-based infrastructure.

Tool name	Cloud provider's coverage
Bridgecrew	AWS, AZURE, and GCP
Prisma Cloud	AWS, AZURE, and GCP
Aqua Enterprise	AWS, AZURE, and GCP
Cloud One - Conformity	AWS, AZURE

³⁴ <https://docs.paloaltonetworks.com/prisma/prisma-cloud.html>

³⁵ <https://www.aquasec.com/demo/>

³⁶ <https://www.cloudconformity.com/>

2.2.3.2 Open Source Tools

Apart from the closed source tools, several open source tools also scan cloud environments, such as Cloud Reports³⁷, Prowler³⁸, and Scout Suite³⁹. All the tools in Table 5 implement security checks on the cloud environments that they support and provide reporting. Note that the provider's coverage in the table below only considers AWS, AZURE, or GCP.

Table 5: Open source dynamic analysis tools for cloud-based infrastructure.

Tool name	Cloud provider's coverage
Cloud Reports	AWS
Prowler	AWS
Scout Suite	AWS, AZURE, and GCP

2.3 Summary

Infrastructure as Code appears in the cloud community as an innovative and easiest way of infrastructure configuration and management. As suggested by its name, this technology allows the configuration and management of the infrastructure through the development and execution of scripts instead of traditional interactive tools — such as command line — or cloud provider web interfaces.

This new method of dealing with infrastructure is very promising since it seems to be faster compared to other alternatives due to its re-usability. Moreover, it provides the implementation of software engineering practices, which seems to be the main advantage of its use.

The community that uses IaC faces challenges related to the quality and security of the code. As the platforms that provide this technology are recent and unique, there is a lot to explore in their code structures. The best answer to code improvement seems to be the static analysis tools for IaC.

However, these types of tools have their limitations when compared to dynamic analysis tools for the cloud, resulting in a gap that needs to be bridged between them. The next chapter explores how to bridge the gap in two contexts: Access Control and Network Security.

³⁷ <https://github.com/tensult/cloud-reports>

³⁸ <https://github.com/prowler-cloud/prowler>

³⁹ <https://github.com/nccgroup/ScoutSuite>

KICS: A CASE STUDY

As discussed in Section 2.1.5, static analysis has its limitations, which results in a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure. For that reason, this chapter details the case study of the present dissertation, i.e., a comparison between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure. The main focus of this analysis is the collection of security queries (see Section 2.1.4) only covered by dynamic analysis tools for cloud-based infrastructure.

This study is also a contribution to KICS, an open source tool for static code analysis of IaC solutions, provided by Checkmarx. See Section 3.2 and Section 3.3 for information about the tool.

3.1 Static Analysis Tools vs. Dynamic Analysis Tools

Firstly, this section presents the methodology followed in this dissertation, describing the selection criteria regarding the security context, coverage of IaC technologies, and coverage of cloud infrastructure providers. Lastly, it represents the security queries in table format and highlights the results of the study.

3.1.1 *Methodology*

First of all, the triage of which security contexts are covered takes place (see Section 2.1.4). The main focus is on Access Control, which includes all security queries that prevent untrusted individuals and restrict access permissions to data in an environment (Access Control queries).

Secondly, this study also covers a secondary security context, Network Security. This category includes all security queries that protect the connection of an environment (Network Security queries). However, as presented in Section 2.1.4, there are other security query categories that can cover cases like encryption best practices in a network (Encryption category). In these cases, these kinds of security queries are excluded.

The next step consists of identifying what possible Access Control related services are made available by cloud infrastructure providers. It should be noted that the same applies to the Network Security context. This step considers two types of services:

- **Services directly related to each category:** It includes vital services for the implementation of the security queries category.

Regarding Access Control, examples of them are Identity and Access Management (IAM) and API Gateway services. However, this study can include security queries that embrace other security queries categories since it is relevant to protect both services.

Concerning Network Security, services related to Virtual Private Network and Security Group, stand out, for example.

- **Services indirectly related to each category:** Services where it is crucial to apply the category best practices.

Concerning Access Control, for example, if someone launches an Amazon S3 bucket on an AWS cloud infrastructure environment, one must keep in mind its permissions to prevent leaking private information. Simple practices such as restricting write permissions are sometimes disregarded, but they should not be.

Regarding Network Security, for example, if someone launches a Relational Database Service that uses the default port, an attacker can trivially guess it, thus endangering the network security of the environment.

After the identification of the services and before selecting the tools to explore, it is necessary to consider the common bond between the static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure: cloud providers. The chosen cloud providers are AWS, AZURE, and GCP since they are the leading cloud providers on the market.

In addition, it is also important to choose the IaC technologies. The IaC technologies chosen are Ansible, CloudFormation, and Terraform since all of them have detailed documentation and their configuration is very intuitive.

The selection criteria applied to the static analysis tools for IaC presented in Table 3 is the following:

- **Coverage of IaC technologies:** It is immediately discarded tools that do not cover at least one of the IaC technologies chosen, such as Haskell Dockerfile Linter, Kube-Score, and Puppet Linter.
- **Coverage of cloud infrastructure providers:** It is not immediately discarded any tool through this parameter since all the target tools cover at least one of the chosen providers (AWS, AZURE, and/or GCP).

On the other hand, the selection criteria applied to dynamic analysis tools for cloud-based infrastructure mentioned in Section 2.2.3 is the following:

- **Coverage of cloud infrastructure providers:** It is not immediately discarded any tool through this parameter since all the target tools cover at least one of the chosen providers (AWS, AZURE, and/or GCP).

After the application of the two selection criteria and considering the documentation available, the tools chosen for this study are:

- **Static analysis tools for IaC:** CFN_NAG, Checkov, KICS, Terrascan, and TFSec.
- **Dynamic Analysis Tools for Cloud-Based Infrastructure:** Aqua, Cloud Reports, Cloud Conformity, Prowler, and Scout Suite.

As the final step, it is time to choose how to study these tools. One option is the observation of the tools in action. Alternatively, another option is reading documents.

The first alternative would be, in practice, unsatisfactory, since access to the selected commercial tools will be limited to free demos.

The process of obtaining them is not always easy, and their usage period is also limited. Moreover, the free demos do not have all the features compared to the paid solution. Additionally, dynamic analysis tools for cloud-based infrastructure require cloud accounts, which also involves monetary costs.

Although the second option is dependent on the interpretation of the documentation and how up-to-date it is, which can result in (hopefully) minor inaccuracies, it seems to be the best way to better analyze all tools equally. Therefore, this study is based on reading documents from around the end of 2020. See the results in Section 3.1.2.

3.1.2 Results

As mentioned in the previous section, the case study is based on reading the available documentation for the selected tools and extracting information regarding the set of security queries provided by each tool.

The procedure to collect the security queries only covered by dynamic analysis tools for cloud-based infrastructure involves the following steps:

- **Data collection:** Consists of studying the reading documents and listing the security queries according to the methodology defined in the previous section.
- **Comparison between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure:** Identification of what tools cover each security query.
- **Identification of security queries only covered by dynamic analysis tools for cloud-based infrastructure:** Identification of what security queries are not covered by any static analysis tool for IaC.

Based on our understanding, this case study reports 150 security queries as a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure through Tables 6, 7, 8, 9, 10, 11, 12, and 13. From this set, 78 of them are related to Access Control and 72 to Network Security.

The results are grouped in the following tables accordingly to the cloud provider and context. In each table, the collected security queries (as a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure) are shown in gray.

Table 8: [AZURE] Access Control context.

This table can embrace other security queries categories than Access Control that are considered relevant for the study context.

	Static Analysis			Dynamic Analysis		
	Ansible KICS	Terraform		Aqua	Cloud Conformity	ScoutSuite
		Checkov	KICS			
Active Directory	[AZURE_AC_01] Adding Gallery Apps to Access Panel not restricted				✓	
	[AZURE_AC_02] Application Registration for non-privileged users not restricted				✓	
	[AZURE_AC_03] Authentication reconfirmation disabled				✓	
	[AZURE_AC_04] Dual Identification for password reset disabled				✓	
	[AZURE_AC_05] Guest users in use	✓		✓	✓	✓
	[AZURE_AC_06] Guest User invitations not restricted				✓	
	[AZURE_AC_07] Invitations not only to administrators				✓	
	[AZURE_AC_08] Multi-Factor Authentication for privileged users disabled				✓	
	[AZURE_AC_09] Multi-Factor Authentication for non-privileged users disabled				✓	
	[AZURE_AC_10] Non-Admin Access to Administration Portal not restricted				✓	
	[AZURE_AC_11] Not allow only administrators to manage Office 365 groups				✓	
	[AZURE_AC_12] Not allow only administrators to create security groups				✓	
	[AZURE_AC_13] Office 365 Group Creation not restricted only to administrators				✓	
	[AZURE_AC_14] Password without uppercase letter			✓		
	[AZURE_AC_15] Password without minimum length			✓		
	[AZURE_AC_16] Password without numbers			✓		
	[AZURE_AC_17] Password without symbol			✓		
	[AZURE_AC_18] Password without uppercase letter			✓		
	[AZURE_AC_19] Require MFA to Join Devices disabled				✓	
	[AZURE_AC_20] Remembering Multi-Factor Authentication enabled				✓	
	[AZURE_AC_21] Role Assignment not limit Guest User permissions				✓	
	[AZURE_AC_22] Role Definition allows custom role creation				✓	
	[AZURE_AC_23] Self-Service group management enabled				✓	
App Service		✓		✓	✓	✓
[AZURE_AC_25] App Service web applications without client certificates		✓		✓	✓	✓
[AZURE_AC_26] App Service without managed service identity		✓		✓	✓	✓
[AZURE_AC_27] Blob container allows public access		✓		✓	✓	✓
[AZURE_AC_28] Container registry with admin enabled			✓	✓		
[AZURE_AC_29] File service allows all ACL permissions				✓		
[AZURE_AC_30] Kubernetes cluster without RBAC				✓		
[AZURE_AC_31] PostgreSQL Authentication without Azure Active Directory admin				✓		
[AZURE_AC_32] Queue Service allows all ACL permissions				✓		
[AZURE_AC_33] SQL Server has email account admins disabled		✓				
[AZURE_AC_34] SQL Server allows public access		✓				
[AZURE_AC_35] SQL Server without Azure Active Directory admin				✓		✓
[AZURE_AC_36] Activity Log Container have public read access		✓		✓		✓
[AZURE_AC_37] Identity-based Directory Service for Azure File Authentication is enabled for all Azure Files				✓		✓
[AZURE_AC_38] Trusted Microsoft Services access disabled on storage account				✓		✓
[AZURE_AC_39] Table Service allows all ACL permissions				✓		✓

Table 12: [AZURE] Network Security.

	Static Analysis				Dynamic Analysis		
	Ansible		Terrascan		Aqua	Cloud Conformity	ScoutSuite
	KICS	Checkov	KICS	Terrascan			
Cosmos DB		✓				✓	
Network Security Groups		✓			✓	✓	✓
	[AZURE_NS_01] Cosmos DB does not deny public access						
	[AZURE_NS_02] Network Security Group expose all ports to the public				✓	✓	✓
	[AZURE_NS_03] Network Watcher disabled		✓		✓	✓	✓
	[AZURE_NS_04] Open TCP and UDP port 53 (DNS)	✓			✓	✓	
	[AZURE_NS_05] Open TCP port 135 (RPC)		✓		✓	✓	
	[AZURE_NS_06] Open TCP port 1433 or UDP port 1434 (Microsoft SQL Server)	✓			✓	✓	
	[AZURE_NS_07] Open TCP port 1521 (Oracle Database)	✓			✓	✓	
	[AZURE_NS_08] Open TCP port 1522 (Oracle Auto Data Warehouse)				✓	✓	
	[AZURE_NS_09] Open TCP port 20 and 21 (FTP)				✓	✓	
	[AZURE_NS_10] Open TCP port 22 (SSH)	✓			✓	✓	✓
	[AZURE_NS_11] Open TCP port 23 (Telnet)	✓			✓	✓	
	[AZURE_NS_12] Open TCP port 2375 or 2376 (Docker)				✓	✓	
	[AZURE_NS_13] Open TCP port 25 (SMTP)	✓			✓	✓	
	[AZURE_NS_14] Open TCP port 3306 (MySQL Database)	✓			✓	✓	✓
	[AZURE_NS_15] Open TCP port 3389 (RDP)	✓			✓	✓	✓
	[AZURE_NS_16] Open TCP port 445 (Windows SMB over TCP)	✓			✓	✓	
	[AZURE_NS_17] Open TCP port 50070 and 50470 (Hadoop/HDFS NameNode WebUI service)				✓	✓	
	[AZURE_NS_18] Open TCP port 5432 (PostgreSQL Database Server)	✓			✓	✓	✓
	[AZURE_NS_19] Open TCP port 5500 (VNC Client)	✓			✓	✓	
	[AZURE_NS_20] Open TCP port 5601 (Kibana)				✓	✓	
	[AZURE_NS_21] Open TCP port 5900 (VNC Server)	✓			✓	✓	
	[AZURE_NS_22] Open TCP port 8020 (HDFS NameNode metadata service)				✓	✓	
	[AZURE_NS_23] Open TCP ports 4505 or 4506 (Salt master)	✓			✓	✓	
	[AZURE_NS_24] Open UDP port 137 or 138 (NetBIOS)	✓			✓	✓	
	[AZURE_NS_25] Open UDP port 445 (CIFS)				✓	✓	
SQL	[AZURE_NS_26] SQL Server allows ingress from 0.0.0.0/0	✓			✓	✓	✓
	[AZURE_NS_27] SQL Server is publicly accessible	✓			✓	✓	✓
Storage Accounts	[AZURE_NS_28] Storage Account is not restricted to known networks	✓			✓	✓	✓
Virtual Networks	[AZURE_NS_29] Virtual Network with DDoS standard protection disabled	✓			✓	✓	✓

Table 13: [GCP] Network Security context.

	Description	Static Analysis					Dynamic Analysis		
		Ansible KICS	Terraform			Aqua	Cloud Conformity	ScoutSuite	
			Checkov	KICS	Terrascan				
Cloud SQL	[GCP_NS_01] Cloud SQL database instance has public IP addresses	✓							
	[GCP_NS_02] Cloud SQL database instance is wide open to the Internet	✓	✓	✓		✓	✓	✓	
	[GCP_NS_03] VM instance has "Block Project-wide SSH keys", disabled	✓	✓	✓		✓	✓	✓	
	[GCP_NS_04] VM instance has "Enable connecting to serial ports", enabled	✓	✓	✓		✓	✓	✓	
	[GCP_NS_05] VM instance has public IP addresses	✓	✓	✓		✓	✓	✓	
Compute Engine	[GCP_NS_06] VM instance is using a default firewall rule							✓	
	[GCP_NS_07] VM instance is using a firewall rule that allows all ports							✓	
	[GCP_NS_08] VM instance is using a firewall rule that uses port ranges							✓	
	[GCP_NS_09] VM instance launched with "Shielded VM", disabled	✓	✓	✓	✓	✓	✓	✓	
	[GCP_NS_10] VM instance with IP Forwarding enabled	✓	✓	✓		✓	✓	✓	
DNS	[GCP_NS_11] DNS Security is disabled	✓	✓	✓		✓	✓		
Kubernetes Engine	[GCP_NS_12] Kubernetes Engine cluster has alias IP ranges disabled	✓	✓	✓		✓	✓	✓	
	[GCP_NS_13] Kubernetes Engine cluster with private cluster disabled	✓	✓	✓		✓	✓	✓	
Network	[GCP_NS_14] Kubernetes Engine cluster without network policy enabled	✓	✓	✓		✓	✓	✓	
	[GCP_NS_15] Kubernetes Engine cluster without private endpoint	✓	✓	✓		✓	✓	✓	
	[GCP_NS_16] Default network in use	✓	✓	✓		✓	✓	✓	
	[GCP_NS_17] Open TCP and UDP port 53 (DNS)						✓	✓	
	[GCP_NS_18] Open TCP port 135 (RPC)						✓	✓	
	[GCP_NS_19] Open TCP port 1433 or UDP port 1434 (Microsoft SQL Server)						✓	✓	
	[GCP_NS_20] Open TCP port 1521 (Oracle Database)						✓	✓	
	[GCP_NS_21] Open TCP port 1522 (Oracle Auto Data Warehouse)						✓	✓	
	[GCP_NS_22] Open TCP port 20 and 21 (FTP)						✓	✓	
	[GCP_NS_23] Open TCP port 22 (SSH)	✓				✓	✓	✓	
	[GCP_NS_24] Open TCP port 23 (Telnet)						✓	✓	
	[GCP_NS_25] Open TCP port 2375 or 2376 (Docker)						✓	✓	
	[GCP_NS_26] Open TCP port 25 (SMTP)						✓	✓	
	[GCP_NS_27] Open TCP port 27017 (MongoDB)						✓	✓	
	[GCP_NS_28] Open TCP port 3306 (MySQL Database)						✓	✓	
	[GCP_NS_29] Open TCP port 3389 (RDP)	✓				✓	✓	✓	
	[GCP_NS_30] Open TCP port 445 (Windows SMB over TCP)						✓	✓	
	[GCP_NS_31] Open TCP port 50070 and 50470 (Hadoop/HDFS NameNode WebUI service)						✓	✓	
	[GCP_NS_32] Open TCP port 5432 (PostgreSQL Database Server)						✓	✓	
	[GCP_NS_33] Open TCP port 5500 (VNC Client)						✓	✓	
[GCP_NS_34] Open TCP port 5601 (Kibana)						✓	✓		
[GCP_NS_35] Open TCP port 5900 (VNC Server)						✓	✓		
[GCP_NS_36] Open TCP port 6379 (Redis)						✓	✓		
[GCP_NS_37] Open TCP port 7001 (Cassandra)						✓	✓		
[GCP_NS_38] Open TCP port 8020 (HDFS NameNode metadata service)						✓	✓		
[GCP_NS_39] Open TCP ports 4505 or 4506 (Salt master)						✓	✓		
[GCP_NS_40] Open UDP port 137 or 138 (NetBIOS)						✓	✓		
[GCP_NS_41] Open UDP port 445 (CIFS)						✓	✓		
[GCP_NS_42] Range of ports opened to allow incoming traffic						✓	✓		
[GCP_NS_43] Subnet has Private Google Access disabled						✓	✓		
[GCP_NS_44] Unrestricted inbound using Internet Control Message Protocol (ICMP)						✓	✓		
[GCP_NS_45] Unrestricted ingress access to uncommon TCP/UDP ports						✓	✓		
[GCP_NS_46] Unrestricted outbound/egress access						✓	✓		

Table 14 presents the 150 security queries collected as a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure (see Tables 6, 7, 8, 9, 10, 11, 12, and 13). All of them are categorized according to the IaC problem, based on the understanding of misconfiguration and non-compliance problems definition (see Section 2.1.3).

The non-compliance problems are identified through the mapping of the collected security queries with the CIS Amazon Web Services Foundations Benchmark v1.4.0, CIS Microsoft Azure Foundations Benchmark v1.4.0, and CIS Google Cloud Platform Foundation Benchmark v1.4.0. They are labeled as "non-compliance" in the column "IaC Problem", pointing to the number item of the CIS benchmark (e.g., non-compliance (<item-number>)).

On the other hand, the misconfigurations problems are labeled as "misconfiguration". The identification of them is done by excluding the security queries as a non-compliance problem since this type of problem can cover any unsafe configuration.

Table 14: Collected security queries according to IaC problem.

Security Query ID	IaC Problem	Security Query ID	IaC Problem	Security Query ID	IaC Problem
[AWS_AC_01]	misconfiguration	[AWS_NS_30]	misconfiguration	[AZURE_AC_37]	misconfiguration
[AWS_AC_02]	misconfiguration	[AWS_NS_35]	misconfiguration	[AZURE_AC_39]	misconfiguration
[AWS_AC_08]	misconfiguration	[AWS_NS_43]	misconfiguration	[AZURE_NS_08]	misconfiguration
[AWS_AC_10]	misconfiguration	[AWS_NS_45]	misconfiguration	[AZURE_NS_09]	misconfiguration
[AWS_AC_11]	non-compliance (1.19)	[AWS_NS_49]	misconfiguration	[AZURE_NS_12]	misconfiguration
[AWS_AC_12]	misconfiguration	[AWS_NS_50]	misconfiguration	[AZURE_NS_17]	misconfiguration
[AWS_AC_14]	misconfiguration	[AWS_NS_51]	misconfiguration	[AZURE_NS_20]	misconfiguration
[AWS_AC_15]	misconfiguration	[AWS_NS_53]	misconfiguration	[AZURE_NS_22]	misconfiguration
[AWS_AC_17]	misconfiguration	[AWS_NS_55]	misconfiguration	[AZURE_NS_29]	misconfiguration
[AWS_AC_18]	misconfiguration	[AWS_NS_56]	misconfiguration	[GCP_AC_05]	non-compliance (6.1.1)
[AWS_AC_19]	misconfiguration	[AWS_NS_57]	misconfiguration	[GCP_AC_11]	non-compliance (1.3)
[AWS_AC_24]	misconfiguration	[AWS_NS_58]	misconfiguration	[GCP_AC_13]	misconfiguration
[AWS_AC_25]	misconfiguration	[AWS_NS_61]	misconfiguration	[GCP_AC_14]	non-compliance (1.4)
[AWS_AC_27]	misconfiguration	[AWS_NS_63]	misconfiguration	[GCP_AC_15]	misconfiguration
[AWS_AC_30]	misconfiguration	[AWS_NS_64]	misconfiguration	[GCP_AC_16]	misconfiguration
[AWS_AC_31]	misconfiguration	[AWS_NS_65]	misconfiguration	[GCP_AC_17]	misconfiguration
[AWS_AC_32]	misconfiguration	[AWS_NS_66]	misconfiguration	[GCP_AC_20]	misconfiguration
[AWS_AC_33]	misconfiguration	[AWS_NS_67]	misconfiguration	[GCP_AC_21]	non-compliance (1.9)
[AWS_AC_35]	non-compliance (1.20)	[AWS_NS_68]	misconfiguration	[GCP_AC_23]	misconfiguration
[AWS_AC_36]	non-compliance (1.14)	[AWS_NS_69]	misconfiguration	[GCP_NS_06]	misconfiguration
[AWS_AC_37]	non-compliance (1.12)	[AWS_NS_70]	misconfiguration	[GCP_NS_07]	misconfiguration
[AWS_AC_41]	misconfiguration	[AWS_NS_71]	misconfiguration	[GCP_NS_08]	misconfiguration
[AWS_AC_53]	misconfiguration	[AWS_NS_74]	misconfiguration	[GCP_NS_17]	misconfiguration
[AWS_AC_54]	misconfiguration	[AWS_NS_81]	misconfiguration	[GCP_NS_18]	misconfiguration
[AWS_AC_56]	misconfiguration	[AZURE_AC_01]	misconfiguration	[GCP_NS_19]	misconfiguration
[AWS_AC_58]	non-compliance (1.17)	[AZURE_AC_02]	misconfiguration	[GCP_NS_20]	misconfiguration
[AWS_AC_59]	misconfiguration	[AZURE_AC_03]	misconfiguration	[GCP_NS_21]	misconfiguration
[AWS_AC_61]	misconfiguration	[AZURE_AC_04]	misconfiguration	[GCP_NS_22]	misconfiguration
[AWS_AC_62]	misconfiguration	[AZURE_AC_06]	non-compliance (1.12)	[GCP_NS_24]	misconfiguration
[AWS_AC_65]	misconfiguration	[AZURE_AC_07]	misconfiguration	[GCP_NS_25]	misconfiguration
[AWS_AC_66]	misconfiguration	[AZURE_AC_08]	non-compliance (1.1)	[GCP_NS_26]	misconfiguration
[AWS_AC_67]	misconfiguration	[AZURE_AC_09]	non-compliance (1.2)	[GCP_NS_27]	misconfiguration
[AWS_AC_68]	misconfiguration	[AZURE_AC_10]	misconfiguration	[GCP_NS_28]	misconfiguration
[AWS_AC_72]	non-compliance (1.11)	[AZURE_AC_11]	misconfiguration	[GCP_NS_30]	misconfiguration
[AWS_AC_75]	misconfiguration	[AZURE_AC_12]	misconfiguration	[GCP_NS_31]	misconfiguration
[AWS_AC_78]	misconfiguration	[AZURE_AC_13]	non-compliance (1.18)	[GCP_NS_32]	misconfiguration
[AWS_AC_79]	misconfiguration	[AZURE_AC_14]	misconfiguration	[GCP_NS_33]	misconfiguration
[AWS_AC_80]	misconfiguration	[AZURE_AC_15]	misconfiguration	[GCP_NS_34]	misconfiguration
[AWS_AC_81]	misconfiguration	[AZURE_AC_16]	misconfiguration	[GCP_NS_35]	misconfiguration
[AWS_AC_82]	misconfiguration	[AZURE_AC_17]	misconfiguration	[GCP_NS_36]	misconfiguration
[AWS_NS_01]	misconfiguration	[AZURE_AC_18]	misconfiguration	[GCP_NS_37]	misconfiguration
[AWS_NS_04]	misconfiguration	[AZURE_AC_19]	non-compliance (1.19)	[GCP_NS_38]	misconfiguration
[AWS_NS_05]	misconfiguration	[AZURE_AC_20]	misconfiguration	[GCP_NS_39]	misconfiguration
[AWS_NS_07]	misconfiguration	[AZURE_AC_21]	non-compliance (1.12)	[GCP_NS_40]	misconfiguration
[AWS_NS_08]	misconfiguration	[AZURE_AC_22]	misconfiguration	[GCP_NS_41]	misconfiguration
[AWS_NS_16]	misconfiguration	[AZURE_AC_23]	misconfiguration	[GCP_NS_42]	misconfiguration
[AWS_NS_17]	misconfiguration	[AZURE_AC_29]	misconfiguration	[GCP_NS_43]	misconfiguration
[AWS_NS_20]	misconfiguration	[AZURE_AC_31]	non-compliance (4.5)	[GCP_NS_44]	misconfiguration
[AWS_NS_25]	misconfiguration	[AZURE_AC_32]	misconfiguration	[GCP_NS_45]	misconfiguration
[AWS_NS_27]	misconfiguration	[AZURE_AC_35]	non-compliance (4.5)	[GCP_NS_46]	misconfiguration

3.2 Why KICS

The present dissertation is developed in a collaboration with Checkmarx. As stated on the Checkmarx website, this company is a global leader in software security solutions for DevOps. Of the several products that this well-known company provides, KICS (Keeping Infrastructure as Code Secure) is its first open source one.

This Checkmarx solution was created to provide an automatic way of identifying IaC issues. In this way, the development teams can fix the IaC issues quickly and easily rather than relying on manual code reviews.

Since Checkmarx advocates open source projects, KICS was launched as an open source tool to allow the community to keep Infrastructure as Code more secure together with the company, for free, by allowing the community to interact and contribute to KICS. The KICS source code is available in the KICS GitHub repository¹.

Available since 2020, KICS is attracting more and more users as time goes by for being such an attractive open source tool: as stated in Section 2.2.2, KICS is one of the few tools that cover several IaC solutions (Ansible, CloudFormation, Docker, Kubernetes, and Terraform, among others), which makes it so appealing.

3.3 KICS Overview

Provided by Checkmarx, KICS (Keeping Infrastructure as Code Secure) is an open source tool for static code analysis of IaC solutions (Ansible, CloudFormation, Docker, Kubernetes, and Terraform, among others).

Its main goal consists of preventing security vulnerabilities, misconfiguration, and non-compliance problems in various categories² through over 1000 security queries. Each security query detects the presence of the security issue in question. See Section 2.1.4 to obtain further information.

3.3.1 Architecture

This section describes the five main components that make up the KICS architecture, as shown in Figure 3. All of them are written in Golang, using Open Policy Agent (OPA)/REGO for security queries.

Open Policy Agent³ is an open source engine that provides the development of policies as code, using a declarative policy language (REGO⁴) to write them. The policies are used in decision-making processes for structured documents.

1 <https://github.com/Checkmarx/kics>

2 Access Control, Backup, Best Practices, Build Process, Encryption, Insecure Configurations, Insecure Defaults, Networking and Firewall, Observability, Resource Management, Secret Management, Structure and Semantics, and Supply-Chain

3 <https://www.openpolicyagent.org/>

4 <https://www.openpolicyagent.org/docs/latest/policy-language/#what-is-rego>

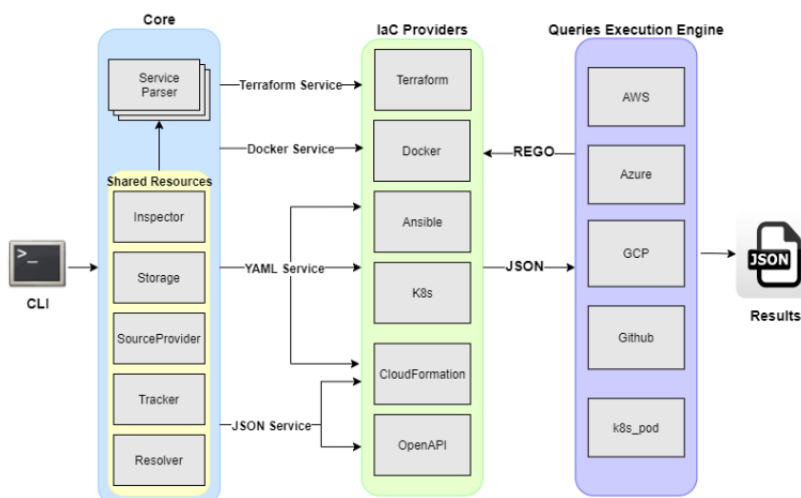


Figure 3: KICS Architecture (Source: KICS Website).

3.3.2 Command Line Interface

The Command Line Interface (CLI) controls KICS scan behavior. There are several features available, such as:

- **Cloud provider(s) restriction:** The scan can be restricted to queries related to the chosen cloud provider(s). Valid values are `aws`, `azure`, and `gcp`.
- **Category exclusion:** The scan can exclude a set of categories.
- **Queries exclusion:** The scan can exclude a set of queries.
- **Queries inclusion:** The scan can include a set of queries.
- **Report format:** The results can be reported in many formats.

3.3.3 Core

Acting as the KICS engine, the core incorporates several agents that are dependent on the Service Parser. Together, these agents are responsible for parsing the files into a payload and storing it.

The Service Parser is responsible for converting the IaC files into JSON format (payload). Since the REGO queries require a structured data as input, which is the case of JSON, the IaC samples need to be normalized into structured data according to each IaC solution (see Section 3.3.4).

3.3.4 IaC Solutions

KICS covers several IaC solutions, such as Ansible, CloudFormation, Docker, Kubernetes, and Terraform, among others. These solutions do not have a common configuration file extension, as can be seen in Table 15:

Table 15: IaC solutions configuration file extension.

IaC solution	Configuration file extension
Ansible	.yaml
CloudFormation	.yaml & .json
Dockerfile	.dockerfile
Helm	.yaml
Kubernetes	.yaml
OpenAPI	.yaml & .json
Terraform	.tf

Although some of them have the same file extension, all have different specifications that require a specific conversion for each IaC provider into a normalized JSON format (payload). The responsible for that task is the Service Parser.

As an example of a payload, see Listing 5 which represents the payload of Listing 6.

```
{
  "document": [
    {
      "file": "positive1.tf",
      "id": "5835e708-66dc-4bb9-b219-6772d9d2d77a",
      "resource": {
        "aws_instance": {
          "positive1": {
            "ami": "ami-003634241a8fcdec0",
            "instance_type": "t2.micro"
          }
        }
      }
    }
  ]
}
```

Listing 5: Payload example.

```
resource "aws_instance" "positive1" {
  ami = "ami-003634241a8fcdec0"

  instance_type = "t2.micro"
}
```

Listing 6: Terraform file example.

3.3.5 Queries Execution Engine

The Queries Execution Engine is the component responsible for applying the REGO queries against the normalized JSON. All the KICS queries are developed and organized according to the IaC technology and cloud provider (AWS, AZURE, and GCP), as can be seen in Listing 7.

```
- <technology>
  |- <provider>
    |- <queryfolder>
      | | |- test
      | | | |- positive<.ext>
      | | | |- negative<.ext>
      | | | |- positive_expected_result.json
      | | | |- metadata.json
      | | | |- query.rego
```

Listing 7: Query file tree.

Each query is composed of:

- **metadata.json:** Describes the relevant aspects of the query: id, query name, severity, category, description text, description URL, and platform. For an example, see Listing 8.

```
{
  "id": "a31a5a29-718a-4ff4-8001-a69e5e4d029e",
```

```

"queryName": "Instance With No VPC",
"severity": "MEDIUM",
"category": "Insecure Configurations",
"descriptionText": "Instance should be configured in VPC (Virtual Private Cloud)",
"descriptionUrl": "https://registry.terraform.io/providers/hashicorp/aws/latest/docs/
    resources/instance",
"platform": "Terraform"
}

```

Listing 8: Metadata example.

- **query.rego:** Establishes the policy to verify the vulnerability and delineates the result. The result includes the document id, search key (specific data used to verify the vulnerability presence), issue type (indicates if the attribute is missing, redundant, or incorrect), expected value, and actual value. For example, Listing 9 presents a query that reports a vulnerability if the attribute 'vpc_security_group_ids' of the resource 'aws_instance' is undefined.

```

package Cx

CxPolicy[result] {
    resource := input.document[i].resource.aws_instance[name]

    not resource.vpc_security_group_ids

    result := {
        "documentId": input.document[i].id,
        "searchKey": sprintf("aws_instance[%s]", [name]),
        "issueType": "MissingAttribute",
        "keyExpectedValue": "Attribute 'vpc_security_group_ids' is set",
        "keyActualValue": "Attribute 'vpc_security_group_ids' is undefined",
    }
}

```

Listing 9: Query example.

- **test folder:** Includes positive and negative IaC samples as test cases (further information will be provided in Section 5.3).

3.3.6 Results

The results are available in two forms: presented in the Command Line Interface (CLI) and saved in a few different file formats. In the CLI, the number of files scanned, parsed files, queries loaded, queries failed to execute, results summary (results total categorized by severity), scan duration (in seconds), and information about the results of each query failed is reported. As an example, see Figure 4.

```
Files scanned: 1
Parsed files: 1
Queries loaded: 1
Queries failed to execute: 0

-----

Instance With No VPC, Severity: MEDIUM, Results: 1
Description: Instance should be configured in VPC (Virtual Private Cloud)
Platform: Terraform

  [1]: assets\queries\terraform\aws\instance_with_no_vpc\test\positive1.tf:1

      001: resource "aws_instance" "positive1" {
      002:   ami = "ami-003634241a8fcdec0"
      003:

Results Summary:
HIGH: 0
MEDIUM: 1
LOW: 0
INFO: 0
TOTAL: 1

Scan duration: 131.3443ms
```

Figure 4: KICS results in Command Line Interface (CLI).

On the other hand, apart from the features reported in CLI, the results can be written in different file formats, such as JSON, and provide more detailed information regarding the execution of each query. As an example, see the Listing 10.

```
{
  "files_scanned": 4,
  "files_parsed": 4,
  "files_failed_to_scan": 0,
  "queries_total": 1,
  "queries_failed_to_execute": 0,
  "queries_failed_to_compute_similarity_id": 0,
  "queries": [
    {
      "query_name": "Instance With No VPC",
      "query_id": "a31a5a29-718a-4ff4-8001-a69e5e4d029e",
```

```

"query_url": "https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources
  /instance",
"severity": "MEDIUM",
"platform": "Terraform",
"files": [
  {
    "file_name": "C:/Users/rafaelas/Desktop/kics2/kics/assets/queries/terraform/aws/
      instance_with_no_vpc/test/positive.tf",
    "similarity_id": "f19ca5c67792801cd7295f65fcc54172c625ee1b5c4f2ce3730b1dc3bc6b8139",
    "line": 1,
    "issue_type": "MissingAttribute",
    "search_key": "aws_instance[positive1]",
    "search_value": "",
    "expected_value": "Attribute 'vpc_security_group_ids' is set",
    "actual_value": "Attribute 'vpc_security_group_ids' is undefined",
    "value": null
  }
],
"category": "Insecure Configurations",
"description": "Instance should be configured in VPC (Virtual Private Cloud)"
}
],
"scan_id": "console",
"severity_counters": {
  "HIGH": 0,
  "INFO": 0,
  "LOW": 0,
  "MEDIUM": 1
},
"total_counter": 1
}

```

Listing 10: KICS results in a JSON report.

3.4 The Problem

As presented in Chapter 2, the community that uses IaC technologies faces challenges related to the IaC scripts. During the development of IaC scripts, inadvertent infrastructure misconfigurations, non-compliance problems, or security vulnerabilities can occur. It can arise from a lack of the best security practices in IaC, misconfigurations, code smells, security smells, or even due to bugs in the code. This concerns the developers, especially the testability of the IaC scripts.

The best way of preventing all the issues mentioned above, before IaC scripts are executed, is to use a static analysis tool for IaC. However, as concluded in Chapter 3, there is a gap between the static analysis tools for IaC and dynamic tools for cloud-based infrastructure.

Unfortunately, the definition of solutions to bridge the gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure in a static approach has its limitations. The main one is the fact that this approach is not capable of identifying possible vulnerabilities, misconfiguration, and non-compliance problems that require a run-time environment. For example, a static approach cannot identify the period of user inactivity through IaC technology. This approach does not have access to all the account information of the cloud infrastructure provider. If it does, it is no longer a static approach since it is just available in a run-time environment (cloud-based infrastructure provider account).

Another limitation is the coverage of the resources. IaC technologies do not provide all the resources made available by the cloud providers. This limits the static approach implementation since it does not have the same resources as a run-time environment. Furthermore, sometimes, it does not provide all the resource information made available by the cloud providers.

Although challenging, this dissertation study explores how some dynamic security queries can be adapted to a static approach.

3.5 Challenges

3.5.1 *Filtration of the Collected Security Queries*

The first challenge is the filtration of the collected security queries presented in Section 3.1.2. This step requires a crucial understanding of whether it is possible to verify each query in a static approach and discard it otherwise. This may sound easy, but there are a few obstacles that need to be tackled:

- **Lack of IaC documentation:** As a very recent technological field, information and documentation are lacking, which impacts the study process.

- **Static analysis limitations:** As mentioned above, a static approach cannot access information available only in a run-time environment. Because of that, some security queries are simply impossible to verify.
- **Differences between the chosen IaC technologies:** Every technology has its specifications and services available, requiring a specific implementation. In some cases, it is impossible to implement the same security query in all technologies. In the worst-case scenarios, it is not possible to implement under no technology.

3.5.2 *Implementation of the Selected Security Queries*

The lack of knowledge of the involved technologies can be challenging since this requires a time-consuming documentation review about all of them.

Concerning the selected IaC technologies (Ansible, CloudFormation, and Terraform), all of them have their specifications which can demand different implementation methods for the same check. For instance, for the same role, it can be necessary to analyze one resource in Terraform and Ansible, and two in CloudFormation. Therefore, different implementation methods entail more time, creativity, and effort.

Regarding the contribution of new security queries for KICS, it is necessary to understand how this tool works and understand the languages involved in KICS queries development, i.e. JSON and OPA (REGO).

Furthermore, if the study implies adding more information to the JSON payload used in KICS queries, understanding the KICS engine is also fundamental. It requires studying its parser to figure out where to add the necessary information, as well as the programming language it uses (Go).

3.6 Summary

This chapter details the core study of the present dissertation, i.e., it discusses the gap between the chosen static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure, from the methodological approach to the results. The main focus is on Access Control, but it also includes Network Security.

It identifies 150 missing security checks between the chosen static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure. A total of 78 of them are related to Access Control and 72 to Network Security.

Moreover, it explains why KICS is the chosen tool for the practical contribution of this dissertation and how it works. It also highlights the challenges that must be addressed during the implementation of security queries. Among them, the lack of documentation, static analysis limitations, differences between the IaC technologies chosen, and the lack of knowledge of the involved technologies stand out.

The next chapter discusses possible solutions for the collected queries, describing how they can be or cannot be implemented in KICS.

FITTING NEW SECURITY QUERIES INTO KICS

The present chapter presents the core developments of this dissertation. In other words, the discussion about the solutions for how the collected security queries can be applied to KICS.

4.1 Introduction

Section 3.1 reports 150 security queries as a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure.

The implementation of these security queries in KICS requires contribution mainly in KICS Queries. However, in some cases, it also entails contribution to KICS Parser, as discussed in Section 4.2.2 and Section 4.2.3.

Unfortunately, not all the security queries have viable solutions to KICS. For more detailed information, see Section 4.3.

4.2 Implementing the Security Queries

This section describes and discusses the development of the security queries reported as a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure (see Section 3.1).

Each security query development highlights the description and the approach attached to itself. The query description is a brief description of the query, which points out why it is relevant. In contrast, the query approach indicates the implementation steps and the correspondent IaC technology target.

Each query also refers to a table, identifying the resource(s) and attribute(s) used for its implementation in each platform.

As mentioned in Section 3.1.1, the selected IaC technologies are Ansible, CloudFormation, and Terraform. Besides, the selected providers are AWS, AZURE, and GCP. Note that Ansible and Terraform cover all the providers, but CloudFormation only covers AWS.

There are cases where it is impossible to implement the query on a specific platform. Sometimes because the specific platform does not support a specific resource necessary for the solution. Others because the specific platform supports the specific resource necessary for the solution, but the specific resource does not provide enough settings to perform the solution. In these scenarios, the platform is not referenced in the table.

4.2.1 [AWS_AC_08] API Gateway without WAF

Query description: AWS Web Application Firewall (WAF) provides protection assistance for web applications or APIs against known web exploits, like Cross-Site Request Forgery (CSRF) or SQL injections. This kind of attack can call into question the CIA (Confidentiality, Integrity, and Availability) Triad. That said, when WAF is integrated with AWS API Gateway, it will provide extra protection against it.

Query approach: To verify if WAF is implemented for API Gateway, it is necessary to check if the resource related to API Gateway Stage is associated with a resource related to WAF Web ACL.

Table 16: [AWS_AC_08] API Gateway without WAF.

Platform	Resource(s)	Attributes
Ansible	community.aws.aws_api_gateway (1), aws_api_gateway (1), community.aws.wafv2_resources (2), and wafv2_resources (2)	stage (1) and arn (2)
CloudFormation	AWS::ApiGateway::Stage (1) and AWS::WAFv2::WebACLAssociation (2)	Properties.StageName (1) and Properties.ResourceArn (2)
Terraform	aws_api_gateway_stage (1) and aws_wafregional_web_acl_association (2)	resource_arn (2)

4.2.2 [AWS_AC_10] API Gateway Without Configured Authorizer

Query description: AWS provides two types of authorizers to restrict an API: Lambda authorizers¹ (also known as "custom authorizers") and Amazon Cognito user pool².

Both are essential to confer a more robust authorization mechanism. If none of these services is defined, the API Gateway could have security issues since IAM roles and policies might not be sufficient. Therefore, it is relevant to verify if an API Gateway does not have a configured Authorizer.

Query approach: Unlike Terraform and CloudFormation, Ansible does not provide a resource related to AWS API Gateway Authorizer. For that reason, this query for Ansible requires a complex implementation.

¹ <https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html>

² <https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools.html>

Since this platform provides two resources related to AWS API Gateway (`community.aws.aws_api_gateway`³ and `aws_api_gateway`⁴) that allow configuration of swagger definitions for API, the verification can be done through a `swagger` attribute (`swagger_file`, `swagger_dict` or `swagger_text`). If the `swagger` attribute does not have the extension `'x-amazon-apigateway-authorizer'`⁵ set in the security definitions, the AWS API Gateway does not have an authorizer configured. However, this is not enough.

This attribute only refers to the file name that contains the swagger definitions. To access the information about the presence of the extension `'x-amazon-apigateway-authorizer'`, the KICS parser needs to be modified. In the `swagger` attribute of the AWS API Gateway resource, the parser has to add the content of the swagger file.

The algorithm to be followed in order to implement this modification is:

1. Get the file path

- a) Verify if the content of `'swagger_file'` is a fully valid path through function `'Stat'` of package `'os'`⁶.
- b) If so, the file path is ready. If not, the `'swagger_file'` could be an incomplete path. In that case, it is necessary to get the directory of the Ansible template and join it to the content of `'swagger_file'` through functions `'Dir'` and `'Join'` of package `'filepath'`⁷.

2. Read and add the file content

- a) Read the file through function `'Unmarshal'` of packages `'json'`⁸ or `'yaml'`⁹ in case of being a JSON or a YAML file, respectively.
- b) Add the content read to the `'swagger_file'` attribute.

For Terraform and CloudFormation, it is only necessary to verify if the AWS API Gateway resource has an AWS API Gateway Authorizer associated.

³ https://docs.ansible.com/ansible/latest/collections/community/aws/aws_api_gateway_module.html

⁴ https://docs.ansible.com/ansible/2.4/aws_api_gateway_module.html

⁵ <https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-swagger-extensions-authorizer.html>

⁶ <https://golang.org/pkg/os/>

⁷ <https://golang.org/pkg/path/filepath/>

⁸ <https://golang.org/pkg/encoding/json/>

⁹ <https://pkg.go.dev/gopkg.in/yaml.v3>

Table 17: [AWS_AC_10] API Gateway Without Configured Authorizer.

Platform	Resource(s)	Attribute(s)
Ansible	community.aws.aws_api_gateway and aws_api_gateway	swagger_dict, swagger_file, and swagger_text
CloudFormation	AWS::ApiGateway::RestApi, AWS::ApiGateway::Authorizer (1), AWS::ApiGatewayV2::Api, and AWS::ApiGatewayV2::Authorizer (2)	Properties.RestApild (1) and Properties.Apild (2)
Terraform	aws_api_gateway_authorizer and aws_api_gateway_rest_api (1)	rest_api_id (1)

4.2.3 [AWS_AC_11] Certificate Has Expired & [AWS_AC_66] Certificate RSA Key Bytes Lower Than 256

Query description: RSA Key length is commonly set to 1024, 2048, or 4096 bits. However, according to the National Institute of Standards and Technology (NIST)¹⁰, the length of an RSA key should be at least 2048-bit.

With the technological innovation related to hardware (computing power), the 1024-bit RSA key is no longer considered to be safe. For that reason, it is a best practice to check if any certificate has a 1024-bit RSA key.

In addition to that, expired certificates could lead to security issues (man-in-the-middle attacks), credibility reduction, and errors in the run-time environment. All of these inconveniences should be avoided.

Query approach: These queries seem to be only possible to implement in Ansible and Terraform since no resource was found to import an existing certificate in CloudFormation.

As the attribute related to the certificate file only mentions the file name, the implementation approach requires a modification to the KICS Parser. The steps to be followed are as follows:

- 1. Get the file path**

- a) Similar to 4.2.2.

- 2. Read the certificate**

- a) Through packages 'os', 'pem'¹¹ and 'x509'¹².

- 3. Add the information about the expiration date and the RSA Key bytes**

- a) Through functions 'NotAfter' and 'PublicKey' of package 'x509'.

¹⁰ <https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-131ar1.pdf>

¹¹ <https://golang.org/pkg/encoding/pem/>

¹² <https://golang.org/pkg/crypto/x509/>

With this modification, it is possible to verify if the certificate uses an RSA key length lower than 256 bytes. Furthermore, it is possible to verify if the certificate has expired.

Table 18: [AWS_AC_11] Certificate Has Expired & [AWS_AC_66] Certificate RSA Key Bytes Lower Than 256.

Platform	Resource(s)	Attribute
Ansible	community.aws.aws_acm	certificate
Terraform	aws_api_gateway_domain_name, aws_iam_server_certificate, and aws_acm_certificate	certificate_body

4.2.4 [AWS_AC_18] Elasticsearch Without IAM Authentication

Query description: Elasticsearch should ensure IAM Authentication to prevent public access.

Query approach: It is necessary to check if the policy related to Elasticsearch allows any Principal. In the case of CloudFormation, the verification is done directly in the resource 'AWS::Elasticsearch::Domain' through 'Properties.AccessPolicies'.

On the other hand, in Terraform, it is necessary to check the 'aws_elasticsearch_domain_policy' attached to the 'aws_elasticsearch_domain'.

Table 19: [AWS_AC_18] Elasticsearch Without IAM Authentication.

Platform	Resource	Attribute(s)
CloudFormation	AWS::Elasticsearch::Domain	Properties.AccessPolicies
Terraform	aws_elasticsearch_domain and aws_elasticsearch_domain_policy (1)	domain_name (1) and access_policies (1)

4.2.5 [AWS_AC_24] Neptune Cluster With IAM Database Authentication Disabled

Query description: Neptune Cluster should have IAM Database Authentication enabled to ensure the use of IAM in database access.

Query approach: It is only necessary to check if the resource related to Neptune Cluster has IAM Database Authentication enabled.

Table 20: [AWS_AC_24] Neptune Cluster With IAM Database Authentication Disabled.

Platform	Resource	Attribute
CloudFormation	AWS::Neptune::DBCluster	Properties.IamAuthEnabled
Terraform	aws_neptune_cluster	iam_database_authentication_enabled

4.2.6 [AWS_AC_32] SES Policy With Allowed IAM Actions

Query description: Simple Email Service (SES) should not allow all actions to all principals in order to grant the least possible privileges.

Query approach: In the policy of an SES, this query searches if all principals have permission to perform all actions. In such a scenario, the statement effect is set to 'allow', the statement action is set to '*', and the statement principal contains '*'. See Listing 11 as an example.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "*",
      "Principal": {
        "AWS": "*"
      },
      "Effect": "Allow",
      "Resource": "*",
      "Sid": ""
    }
  ]
}
```

Listing 11: All actions to all principals.

Table 21: [AWS_AC_32] SES Policy With Allowed IAM Actions.

Platform	Resource(s)	Attributes
Ansible	community.aws.aws_ses_identity_policy and aws_ses_identity_policy	policy
Terraform	aws_ses_identity_policy	policy

4.2.7 [AWS_AC_35] IAM Access Analyzer Undefined

Query description: AWS IAM Access Analyzer is part of Identity and Access Management (IAM) features that attempts to identify unintentional access to the resources and data in an AWS cloud environment.

Query approach: It checks if the resource related to IAM Access Analyzer is undefined in the entire project.

Table 22: [AWS_AC_35] IAM Access Analyzer Undefined.

Platform	Resource
CloudFormation	AWS::AccessAnalyzer::Analyzer
Terraform	aws_accessanalyzer_analyzer

4.2.8 [AWS_AC_41] IAM Group Without Users

Query description: Having an IAM Group without at least one user attached is pointless since every policy defined to it will not be used. It does not represent an immediate risk but could be in the future if the group fades to oblivion and an unauthorized user is able to attach herself to it.

Query approach: This query implies one implementation method for each platform. In an Ansible approach, it is necessary to check if the attribute 'users' exists and is not empty in the AWS IAM Group.

However, different from Ansible, the resource AWS IAM Group provides by CloudFormation and Terraform does not have an attribute directly related to users. So, for Terraform, it is necessary to check if an AWS IAM Group is not associated with an AWS IAM Group Membership (that has at least one user set).

On the other hand, for CloudFormation, it is necessary to check if an AWS IAM Group is in the groups of an AWS IAM User.

Table 23: [AWS_AC_41] IAM Group Without Users.

Platform	Resource	Attribute
Ansible	community.aws.iam_group and iam_group	users
CloudFormation	AWS::IAM::Group and AWS::IAM::User (1)	Properties.Groups (1)
Terraform	aws_iam_group and aws_iam_group_membership (1)	group (1)

4.2.9 [AWS_AC_53] Cross-Account IAM Assume Role Policy Without ExternalId or MFA

Query description: Cross-account access allows resource sharing from a specific account to a third-party entity (external account users). That said, it is crucial to ensure the protection of cross-account access.

For that, there are two mechanisms: (1) MFA (Multi-Factor Authentication) which combine two factors, the credentials, and the MFA device; (2) External ID, a third-party entity identifier that the IAM role uses to ensure the cross-account identity.

Query approach: The best way of giving cross-account access is through an IAM Assume Role policy since it supplies temporary credentials to the cross-account. So, this query verifies if MFA or External ID are not defined correctly in this kind of policy. Listing 12 and Listing 13 represent an example of a correct MFA and External ID configuration, respectively:


```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::987654321145:root"
      },
      "Effect": "Allow",
      "Resource": "*",
      "Sid": "",
      "Condition": {
        "Bool": {
          "aws:MultiFactorAuthPresent": "true"
        }
      }
    }
  ]
}

```

Listing 12: MFA configuration.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "AWS": "arn:aws:iam::987654321145:root"
      },
      "Effect": "Allow",
      "Resource": "*",
      "Sid": "",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "98765"
        }
      }
    }
  ]
}

```

```

    }
  }
}
]
}

```

Listing 13: External ID configuration.

Table 24: [AWS_AC_53] Cross-Account IAM Assume Role Policy Without ExternalId or MFA.

Platform	Resource(s)	Attribute
Ansible	community.aws.iam_role and iam_role	assume_role_policy_document
CloudFormation	AWS::IAM::Role	Properties.AssumeRolePolicyDocument
Terraform	aws_iam_role	assume_role_policy

4.2.10 [AWS_NS_05] Default EC2 security group are in use & [AWS_NS_07] Default VPC in use for EC2 instance

Query description: Using default security groups in an EC2 instance can result in unrestricted access since this type of security group is commonly too permissive. The same applies to the use of default VPC in an EC2 instance.

To avoid malicious attacks that take advantage of unrestricted access, an EC2 should use a specific security group/VPC for its context.

Query description: These queries verify if an EC2 instance is associated with a security group/VPC named as default, respectively.

Table 25: [AWS_NS_05] Default EC2 security group are in use.

Platform	Resources	Attributes
Ansible	amazon.aws.ec2 and ec2	group and group_id
CloudFormation	AWS::EC2::Instance	Properties.SecurityGroups and Properties.SecurityGroupsIds
Terraform	aws_instance	security_groups and vpc_security_group_ids

Table 26: [AWS_NS_07] Default VPC in use for EC2 instance.

Platform	Resources	Attributes
Ansible	amazon.aws.ec2 (1), ec2 (1), amazon.aws.ec2_vpc_subnet (2), and ec2_vpc_subnet (2)	vpc_subnet_id (1) and vpc_id (2)
CloudFormation	AWS::EC2::Instance (1) and AWS::EC2::Subnet (2)	Properties.SubnetId (1) and Properties.VpcId (2)
Terraform	aws_instance (1) and aws_subnet (2)	subnet_id (1) and vpc_id (2)

4.2.11 [AWS_NS_49] Elastic MapReduce Without VPC & [AWS_NS_50] ElastiCache Without VPC

Query description: Virtual Private Cloud (VPC) provides several benefits for a network environment. One of the most relevant ones is most likely the prevention of Internet exposure. That said, both ElastiCache and Elastic MapReduce (EMR) should use VPC to endow security.

Query approach: Regarding ElastiCache, the existence of a subnet group informs if a VPC is attached to it. In contrast, in EMR, the presence of the subnet identifier indicates the use of VPC. Therefore, the first query verifies if ElastiCache does not define the attribute related to the subnet group. The same strategy applies to the EMR query but searches for subnet identifier definition.

Table 27: [AWS_NS_49] Elastic MapReduce Without VPC.

Platform	Resource	Attribute
CloudFormation	AWS::EMR::Cluster	Properties.Instances.Ec2SubnetId(s)
Terraform	aws_emr_cluster	subnet_id(s)

Table 28: [AWS_NS_50] ElastiCache Without VPC.

Platform	Resource(s)	Attribute
Ansible	community.aws.elasticache and elasticache	cache_subnet_group
CloudFormation	AWS::ElastiCache::CacheCluster	Properties.CacheSubnetGroupName
Terraform	aws_elasticache_cluster	subnet_group_name

4.2.12 [AWS_NS_51] ElastiCache Using Default Port & [AWS_NS_58] Relational Database Service (RDS) Using Default Port & [AWS_NS_61] Redshift Using Default Port

Query description: When using a default port, an attacker can trivially guess it, which could pose a risk to the network security of the environment. Although replacing the default port does not prevent port scanning and subsequent attacks, it makes malicious actions a little harder to carry out.

Query approach: For each environment, this query verifies if the correspondent default port is in use. Table 29 lists the default port for the contexts in question.

Table 29: Default ports.

Environment	Default Port	Environment	Default Port
ElastiCache (memcached)	11211	RDS (oracle-se2)	1521
ElastiCache (redis)	6379	RDS (oracle-se2-cdb)	1521
RDS (aurora)	3306	RDS (postgres)	5432
RDS (aurora-mysql)	3306	RDS (sqlserver-ee)	1433
RDS (aurora-postgresql)	3306	RDS (sqlserver-se)	1433
RDS (mariadb)	3306	RDS (sqlserver-ex)	1433
RDS (mysql)	3306	RDS (sqlserver-web)	1433
RDS (oracle-ee)	1521	Redshift	5439
RDS (oracle-ee-cdb)	1521		

Table 30: [AWS_NS_51] ElastiCache Using Default Port.

Platform	Resource(s)	Attributes
Ansible	community.aws.elasticache and elasticache	engine and cache_port
CloudFormation	AWS::ElastiCache::ReplicationGroup	Properties.Engine and Properties.Port
Terraform	aws_elasticache_cluster	engine and port

Table 31: [AWS_NS_58] Relational Database Service (RDS) Using Default Port.

Platform	Resource(s)	Attributes
Ansible	community.aws.rds_instance and rds_instance	engine and port
CloudFormation	AWS::RDS::DBInstance	Properties.Engine and Properties.Port
Terraform	aws_db_instance	engine and port

Table 32: [AWS_NS_61] Redshift Using Default Port.

Platform	Resource(s)	Attributes
Ansible	community.aws.redshift and redshift	port
CloudFormation	AWS::Redshift::Cluster	Properties.Port
Terraform	aws_redshift_cluster	port

4.2.13 [AWS_NS_56] VPC Without Network Firewall

Query description: AWS Network Firewall provides additional network protection to a VPC, such as traffic filtering and intrusion detection. That said, each VPC should be associated with an AWS Network Firewall.

Query approach: For each resource related to VPC, this query verifies if there is a Network Firewall attached to it.

Table 33: [AWS_NS_56] VPC Without Network Firewall.

Platform	Resource(s)	Attributes
CloudFormation	AWS::EC2::VPC and AWS::NetworkFirewall::Firewall (1)	Properties.Vpclid (1)
Terraform	aws_vpc and aws_networkfirewall_firewall (1)	vpc_id (1)

4.2.14 [AWS_NS_57] RDS Associated with Public Subnet

Query description: When a Relational Database Service (RDS) is associated with a public subnet, it can become a target for malicious requests since it is exposed to the entire Internet.

Query approach: Each RDS is associated with a subnet group constituted by a set of subnets. This query checks if at least one of them is public (CIDR block or IPv6 CIDR block set to "0.0.0.0/0" or "::/0", respectively).

Table 34: [AWS_NS_57] RDS Associated with Public Subnet.

Platform	Resources	Attributes
Ansible	community.aws.rds_instance (1), rds_instance (1), community.aws.rds_subnet_group (2), rds_subnet_group (2), amazon.aws.ec2_vpc_subnet (3), and ec2_vpc_subnet (3)	db_subnet_group_name (1), subnet_group (1), name (2), subnets (2), cidr (3), and ipv6_cidr (3)
CloudFormation	AWS::RDS::DBInstance (1), AWS::RDS::DBSubnetGroup (2), and AWS::EC2::Subnet (3)	Properties.DBSubnetGroupName (1), Properties.SubnetIds (2), Properties.CidrBlock (3), and Properties.Ipv6CidrBlock (3)
Terraform	aws_db_instance (1), aws_db_subnet_group (2), and aws_subnet (3)	db_subnet_group_name (1), subnet_ids (2), cidr_block (3), and ipv6_cidr_block (3)

4.2.15 [AWS_NS_74] Shield Advanced Not In Use

Query description: AWS Shield protects against Distributed Denial of Service (DDoS) attacks on applications. Two plans enable this service: AWS Shield Standard and AWS Shield Advanced.

As the name suggests, AWS Shield Standard provides defense against common attacks. This protection occurs in the network and transport layer (layers 3 and 4 of the OSI model, respectively).

In contrast, AWS Shield Advanced, in addition to these layers, also offers protection at higher levels of the OSI model and detects uncommon attacks. For that reason, to achieve the maximum protection possible for an application, AWS Shield Advanced should be used.

Query approach: AWS Shield Advanced is available for Amazon Route 53 hosted zone, AWS Global Accelerator accelerator, Elastic IP Address, Elastic Load Balancing, and Amazon CloudFront Distribution. That said, this query should verify if any of them, when defined, is not protected by AWS Shield Advanced.

Table 35: [AWS_NS_74] Shield Advanced Not In Use.

Platform	Resources	Attributes
CloudFormation	AWS::CloudFront::Distribution, AWS::ElasticLoadBalancing::LoadBalancer, AWS::GlobalAccelerator::Accelerator, AWS::EC2::EIP, AWS::Route53::HostedZone, and AWS::FMS::Policy (1)	Properties (1)
Terraform	aws_cloudfront_distribution, aws_lb, aws_globalaccelerator_accelerator, aws_eip, aws_route53_zone, and aws_shield_protection (1)	resource_arn (1)

4.2.16 [AZURE_AC_21] Role Assignment Not Limit Guest User Permissions

Query description: Role Assignment should limit guest user permissions in order to grant the least privileges.

Query approach: It checks if each resource related to Role Assignment does not limit permissions for guest users. For that, it is necessary to check the associated resource related to Role Definition.

Table 36: [AZURE_AC_21] Role Assignment Not Limit Guest User Permissions.

Platform	Resources	Attributes
Terraform	azurerole_assignment (1) and azurerole_definition (2)	role_definition_id (1) and permissions.not_actions (2)

4.2.17 [AZURE_AC_22] Role Definition Allows Custom Role Creation

Query description: Role Definition should not allow custom role creation in order to grant the least privileges.

Query approach: It checks if each resource related to Role Definition has all permissions set (represented by '*') or has the permission to write custom roles ('Microsoft.Authorization/roleDefinitions/write').

Table 37: [AZURE_AC_22] Role Definition Allows Custom Role Creation.

Platform	Resource	Attribute
Ansible	azure_rm_roledefinition	permissions.actions
Terraform	azurerole_definition	permissions.actions

4.2.18 [AZURE_AC_29] Storage Share File Allows All ACL Permissions

Query description: Storage Share File should set the least privileges. In other words, it should not allow all ACL (Access Control List) permissions – r (read), w (write), d (delete), and l (list).

Query approach: The resource related to the Storage Share File is associated with the resource related to Storage Share (where the permissions are set). It is important to check if the Storage Share allows all ACL (Access Control List) permissions – r (read), w (write), d (delete), and l (list).

Table 38: [AZURE_AC_29] Storage Share File Allows All ACL Permissions.

Platform	Resources	Attributes
Terraform	azurerm_storage_share_file (1) and azurerm_storage_share (2)	storage_share_id (1) and acl.access_policy.permissions (2)

4.2.19 [AZURE_AC_39] Storage Table Allows All ACL Permissions

Query description: Storage Table should set the least privileges. In other words, it should not allow all ACL (Access Control List) permissions – r (read), w (write), d (delete), and l (list).

Query approach: It checks if the resources related to Storage Table allow all ACL (Access Control List) permissions – r (read), w (write), d (delete), and l (list).

Table 39: [AZURE_AC_39] Storage Table Allows All ACL Permissions.

Platform	Resource	Attribute
Terraform	azurerm_storage_table	acl.access_policy.permissions

4.2.20 [AZURE_NS_29] Virtual Network with DDoS Protection Plan Disabled

Query description: Virtual Network should have a Distributed Denial-of-Service (DDoS) Protection Plan enabled to be protected against denial of service (DoS) attacks.

Query approach: It checks if the resource related to Virtual Network disables the DDoS Protection Plan. There are two ways to verify this: (1) DDoS Protection Plan configuration is not defined; (2) DDoS Protection Plan configuration is defined but disabled.

Table 40: [AZURE_NS_29] Virtual Network with DDoS Protection Plan Disabled.

Platform	Resource	Attribute
Terraform	azurerm_virtual_network	ddos_protection_plan.enable

4.2.21 [GCP_AC_13] Service Account With Improper Privileges

Query description: Service Account should not have admin, editor, owner, or write privileges in order to grant the least privileges.

Query approach: It checks if service account users have improper privileges: admin, owner, or editor.

Table 41: [GCP_AC_13] Service Account With Improper Privileges.

Platform	Resources	Attributes
Terraform	google_iam_policy (1), google_project_iam_binding (2) and google_project_iam_member (2)	binding.role (1) and role (2)

4.2.22 [GCP_AC_16] IAM Role Assigned to User

Query description: As a best practice, it is better to assign a role to a group. Adding or removing members to a group seems easier than creating, updating, or removing an IAM role to a user.

Query approach: It is only necessary to check if a resource related to IAM Role is assigned to a user.

Table 42: [GCP_AC_16] IAM Role Assigned to User.

Platform	Resources	Attributes
Terraform	data.google_iam_policy, google_project_iam_binding, and google_project_iam_member	binding and role

4.2.23 [GCP_AC_17] User with KMS Admin and CryptoKey Roles

Query description: IAM Policy should not have KMS admin and CryptoKey roles. This check ensures the separation of responsibilities. The KMS admin role ('roles/cloudkms.admin') allows full access to Cloud KMS resources, except to the operations allowed by CryptoKey roles ('roles/cloudkms.cryptoKeyDecrypter', 'roles/cloudkms.cryptoKeyEncrypter', or 'roles/cloudkms.cryptoKeyEncrypterDecrypter').

Query approach: It checks if users have both KMS admin and CryptoKey roles set.

Table 43: [GCP_AC_17] User with KMS Admin and CryptoKey Rules.

Platform	Resources	Attributes
Terraform	google_project_iam_policy	policy_data

4.2.24 [GCP_AC_21] KMS Crypto Key is Publicly Accessible

Query description: The policy associated with the KMS crypto key should restrict public access. If not, anyone can access the KMS crypto key and the data encrypted with them.

Query approach: In order to verify if the KMS crypto key is publicly accessible, it is necessary to check if the policy attached to it allows "allUsers" or "allAuthenticatedUsers".

Table 44: [GCP_AC_21] KMS Crypto Key is Publicly Accessible.

Platform	Resources	Attributes
Terraform	google_kms_crypto_key_iam_policy (1) and google_iam_policy (2)	policy_data (1) and binding.members (2)

4.2.25 [GCP_AC_23] Container Cluster Using Default Service Account

Query description: When using a default service account, the principle of least privilege is not assured, which can lead to malicious attacks on the container cluster. That said, a container cluster should use a custom service account that has the least privileges to run it.

Query approach: This query verifies if the resource related to the container cluster does not define the service account or uses a service account named as default.

Table 45: [GCP_AC_23] Container Cluster Using Default Service Account.

Platform	Resource(s)	Attribute
Ansible	google.cloud.gcp_container_cluster and gcp_container_cluster	node_config.service_account
Terraform	google_container_cluster	node_config.service_account

4.2.26 [GCP_NS_06] Google Compute Network Using Default Firewall Rule

Query description: Using a default firewall rule can undermine the principle of least privilege since default firewall rules are not defined for a specific context. That said, a Compute Network should not use default firewall rules.

Query approach: This query verifies if a Google Compute Network is associated with a Google Compute Firewall named as default.

Table 46: [GCP_NS_06] Google Compute Network Using Default Firewall Rule.

Platform	Resources	Attributes
Ansible	google.cloud.gcp_compute_firewall, gcp_compute_firewall, google.cloud.gcp_compute_network (1), and gcp_compute_network (1)	name (1) and network (1)
Terraform	google_compute_network and google_compute_firewall (1)	network (1) and name (1)

4.2.27 [GCP_NS_07] Google Compute Network Using Firewall Rule that Allows All Ports & [GCP_NS_08] Google Compute Network Using Firewall Rule that Allows Port Range

Query description: When a Google Compute Network uses a firewall rule that allows all ports, an attacker can trivially guess the port and perform malicious attacks. The same can happen when the firewall rule allows a port range since some ports can be unintentionally exposed.

Query approach: For each resource related to Google Compute Network, these queries verify if it is associated with a firewall rule that allows all ports (matches 0-65535) or a range of ports (matches the regex [0-9]+-[0-9]+), respectively.

Table 47: [GCP_NS_07] Google Compute Network Using Firewall Rule that Allows All Ports & [GCP_NS_08] Google Compute Network Using Firewall Rule that Allows Port Range.

Platform	Resources	Attributes
Ansible	google.cloud.gcp_compute_firewall, gcp_compute_firewall, google.cloud.gcp_compute_network (1), and gcp_compute_network (1)	direction (1), allowed.ports (1), and network (1)
Terraform	google_compute_network and google_compute_firewall (1)	direction (1), allow.ports (1), and network (1)

4.2.28 [GCP_NS_43] Compute Subnetwork with Private Google Access Disabled

Query description: Virtual machines in a Compute Subnetwork without external IP addresses can only send traffic to the internal network. However, they can access Google APIs and services through Private Google Access enablement.

Query approach: This query verifies if Private Google Access is enabled to ensure that VMs without external IP addresses can access Google APIs and services.

Table 48: [GCP_NS_43] Compute Subnetwork with Private Google Access Disabled.

Platform	Resources	Attributes
Ansible	google.cloud.gcp_compute_subnetwork and gcp_compute_subnetwork	private_ip_google_access
Terraform	google_compute_subnetwork	private_ip_google_access

4.3 Discarded Queries

Around 77% of the collected security queries could not be implemented in KICS. This value reflects the discarding process in action.

The discarding process raises a few sets of reasons responsible for discarding 77% of the collected security queries. These sets of reasons can be categorized in the following scopes:

- **Dynamic scope:** Scope related to information that can be only accessible in a dynamic environment. In other words, the data is only available in a run-time environment. As a case in point, the information related to the cloud account and the availability of its services.

Regarding Access Control, the time of user inactivity in the account, for example, is not accessible for an IaC static approach. Concerning Network Security, knowing if a VPC tunnel is "UP" is also impossible since there is no access to the real-time state of the VPC in an IaC static approach.

- **Known IaC scope:** Scope related to similar information already available in an IaC static approach for the same resource.

Regarding Access Control, there are already available IaC queries that verify if a user has admin privileges. So, for the context of the gap, the verification of several users with admin privileges is not that relevant since some static analysis tools for IaC already handle similar cases.

Concerning Network Security, the study case reports that the target IaC tools do not verify if an FTP port is "open" in an AWS Security Group. However, some of the target IaC tools check it for an SSH port. For the context of bridging this gap, verifying if an FTP port is open in an AWS Security Group is not that relevant for the same reason indicated above.

- **Inapplicable IaC scope:** Scope related to information that does not make sense to implement in an IaC static approach.

Regarding Access Control, the developers writing IaC scripts, keep in mind which specific users can access the defined environment. That said, custom queries, where the IaC developer needs to refer to the "allowed" users, do not make sense.

Concerning Networking Security, as an assumption, no IaC developer will create an AWS Auto Scaling Group and associate it to an inactive AWS Security Group. And even if the developer does that, there is no way to infer it by an IaC static approach.

- **Unknown scope:** Scope related to information that seems to not be accessible in an IaC static approach. Several reasons can justify this statement, such as the lack of service support, information about a specific service, or documentation from the target IaC technologies.

Table 49 classifies the queries according to "dynamic", "known", "inapplicable", and "unknown" scopes. Queries are represented according to the ID defined in Section 3.1.2.

Table 49: List of the discarded queries according to "dynamic", "known", "inapplicable", and "unknown" scopes.

Dynamic	Known IaC	Inapplicable IaC	Unknown
[AWS_AC_37]	[AWS_AC_12]	[AWS_AC_14]	[AWS_AC_01]
[AWS_AC_56]	[AWS_AC_25]	[AWS_AC_15]	[AWS_AC_02]
[AWS_AC_62]	[AWS_AC_27]	[AWS_AC_17]	[AWS_AC_36]
[AWS_AC_67]	[AWS_AC_54]	[AWS_AC_19]	[AWS_AC_58]
[AWS_AC_72]	[AWS_AC_79]	[AWS_AC_30]	[AWS_AC_61]
[AWS_AC_75]	[AWS_NS_16]	[AWS_AC_31]	[AWS_AC_65]
[AWS_NS_67]	[AWS_NS_17]	[AWS_AC_33]	[AWS_NS_04]
[AWS_NS_81]	[AWS_NS_20]	[AWS_AC_59]	[AWS_NS_08]
	[AWS_NS_25]	[AWS_AC_68]	[AWS_NS_55]
	[AWS_NS_27]	[AWS_AC_78]	[AWS_NS_63]
	[AWS_NS_30]	[AWS_AC_80]	[AWS_NS_64]
	[AWS_NS_35]	[AWS_AC_81]	[AWS_NS_65]
	[AWS_NS_43]	[AWS_AC_82]	[AWS_NS_66]
	[AWS_NS_45]	[AWS_NS_01]	[AWS_NS_68]
	[AZURE_NS_08]	[AWS_NS_53]	[AWS_NS_69]
	[AZURE_NS_09]		[AWS_NS_70]
	[AZURE_NS_12]		[AWS_NS_71]
	[AZURE_NS_17]		[AZURE_AC_01]
	[AZURE_NS_20]		[AZURE_AC_02]
	[AZURE_NS_22]		[AZURE_AC_03]
	[GCP_NS_17]		[AZURE_AC_04]
	[GCP_NS_18]		[AZURE_AC_06]
	[GCP_NS_19]		[AZURE_AC_07]
	[GCP_NS_20]		[AZURE_AC_08]
	[GCP_NS_21]		[AZURE_AC_09]
	[GCP_NS_22]		[AZURE_AC_10]
	[GCP_NS_24]		[AZURE_AC_11]
	[GCP_NS_25]		[AZURE_AC_12]
	[GCP_NS_26]		[AZURE_AC_13]
	[GCP_NS_27]		[AZURE_AC_14]
	[GCP_NS_28]		[AZURE_AC_15]
	[GCP_NS_30]		[AZURE_AC_16]
	[GCP_NS_31]		[AZURE_AC_17]
	[GCP_NS_32]		[AZURE_AC_18]
	[GCP_NS_33]		[AZURE_AC_19]
	[GCP_NS_34]		[AZURE_AC_20]
	[GCP_NS_35]		[AZURE_AC_23]
	[GCP_NS_36]		[AZURE_AC_31]
	[GCP_NS_37]		[AZURE_AC_32]
	[GCP_NS_38]		[AZURE_AC_35]
	[GCP_NS_39]		[AZURE_AC_37]
	[GCP_NS_40]		[GCP_AC_05]
	[GCP_NS_41]		[GCP_AC_11]
	[GCP_NS_42]		[GCP_AC_14]
	[GCP_NS_44]		[GCP_AC_15]
	[GCP_NS_45]		[GCP_AC_20]
	[GCP_NS_46]		

4.4 Summary

This chapter presents the discussion about the availability and current (wherever possible) development of the 150 collected security queries (as a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure) to KICS (Section 3). It involves the implementation process and the subjacent discarding process.

Around 23% (34) of the 150 collected security queries have been found to have practical solutions. Although this number can be considered of low value, it reflects the possibility to adapt the security queries "only" collected by dynamic analysis tools for cloud-based infrastructure to static analysis tools for IaC, in the context of the present dissertation.

On the other hand, around 77% (116) of the collected security queries does not have viable solutions. This value reflects the discarding process in action. From it, a few sets of reasons that result in four scopes can be noticed: (i) **dynamic scope** related to information that can be only accessible in a dynamic environment, (ii) **known IaC scope** related to similar information already available in an IaC static approach for the same resource, (iii) **inapplicable IaC scope** related to information that does not make sense to implement in an IaC static approach, (iv) **unknown scope** related to information that seems to not be accessible in an IaC static approach until the moment.

EXTENDING KICS

This chapter addresses necessary contributions to the KICS GitHub repository to implement the security queries described in Chapter 4. Therefore, it lists all the solutions implemented and contributed to KICS, including the KICS GitHub pull request related to each one, as can be seen in Table 50. Note that all the security queries listed in the table are already in the KICS GitHub repository master branch.

5.1 Setup of the KICS Development Environment

Before the contribution of all the solutions to KICS, it is necessary to prepare the KICS development environment. For that, it is necessary to check the following steps:

1. **Fork the KICS GitHub repository:** Click on the "Fork" button of the KICS GitHub repository and create the fork.

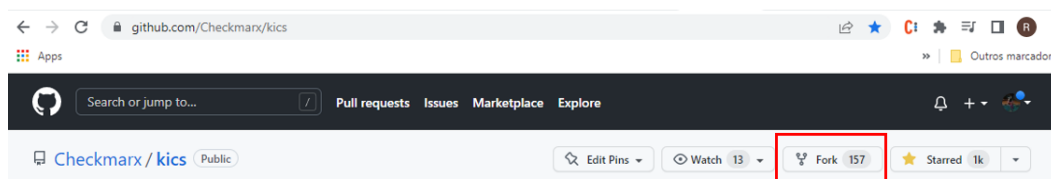


Figure 5: KICS GitHub repository fork.

2. **Clone the fork locally:** In the forked repository, click on the "Code" button and copy paste the link, as can be seen in Figure 6. After that, it is necessary to run the command `'git clone <clone_link>'`.

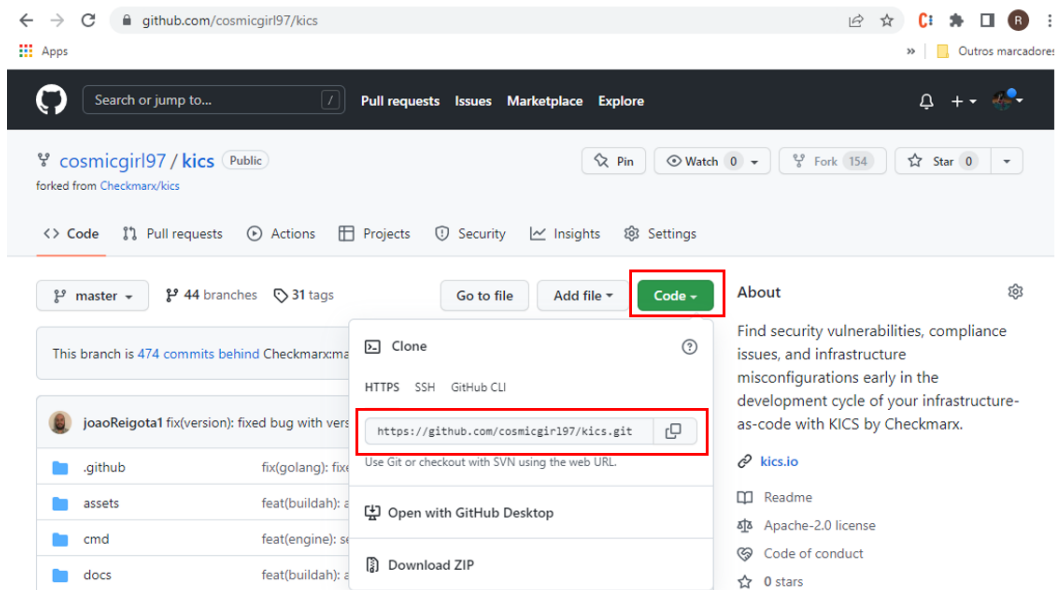


Figure 6: KICS GitHub repository fork clone link.

3. **Verify if the setup is ready:** Check for any necessary dependencies by running `'go run -tags dev ./cmd/console/main.go scan'`, in the KICS folder, for example.

5.2 Development of the Security Queries

As a best practice, the development of each solution to KICS requires a specific branch. The branch can be created by running `'git checkout -b <branch_name>'` inside the KICS cloned folder. However, before the creation of the branch, another best practice is to run `'git pull'` in the KICS master branch.

Before initiating the development of the security queries, it is necessary to keep in mind the guidelines presented in Section 3.3.5 and understand the gold of the security query. Each security query should be composed of `metadata.json`, `query.rego`, and the test folder.

5.2.1 Metadata File

The `metadata.json` documents all the relevant aspects of the security query. See a brief explanation of them below:

- ID: Should be unique and can be generated by the command `'go run ./cmd/console/main.go generate-id'`.
- Query Name: Should clearly indicate what the security query finds.

- **Severity:** Should indicate the severity related to the security query. KICS considers INFO, LOW, MEDIUM, and HIGH for this field.
- **Category:** Should indicate what security domain fits better for the security query. KICS considers the following: Access Control, Availability, Backup, Best Practices, Build Process, Encryption, Insecure Configurations, Insecure Defaults, Networking and Firewall, Observability, Resource Management, Secret Management, and Supply-Chain.
- **Description text:** Should suggest how to remediate the configuration.
- **Description URL:** Should point to the IaC platform documentation.
- **Platform:** Should point to the target IaC platform.

5.2.2 Query File

The security queries in KICS are written in REGO (See Section 3.3.1). Since the input of the REGO policies requires structured data, KICS parses all the IaC platforms files in a JSON payload (See Section 3.3.3).

Although it is possible to develop the security queries directly in KICS and test them, the use of the REGO Playground¹ is very useful and intuitive as a first step. A helpful REGO Playground setting is the "Coverage" button that indicates whether or not the statements have been evaluated, as can be seen in Figure 7.

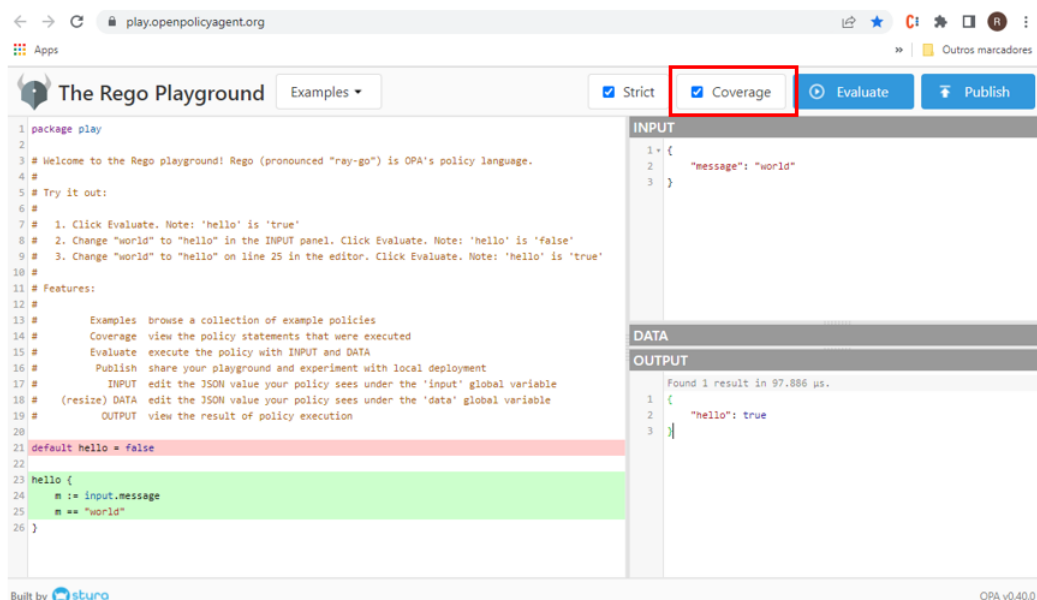


Figure 7: REGO Playground.

¹ <https://play.openpolicyagent.org/>

The content of the query.rego is composed of all the policies necessary to cover the goal of the target security query. Each policy should return a result composed of:

- Search key: Applies Levenshtein distance to find where the "vulnerability" occurs in the original file.
- Issue type: indicates the issue found by the policy: (i) IncorrectValue, (ii) MissingAttribute, (iii) RedundantAttribute.
- Key expected value: Presents a recommendation as an expected configuration.
- Key actual value: Presents the actual configuration found by the policy.

In addition to the guidelines presented above, there are a few more that should be considered during the development of the security queries. Among them it highlights:

- Understand the goal(s) of the security query.
- Create the payload of an IaC script related to the target IaC technology to understand how KICS parses the specific IaC technology files. This guideline is important to understand how to access the information in the development of the security queries.
- Study the existing KICS queries to understand its specifications: each platform have a specific set of security queries.
- Use of the 'package Cx'.
- Explore the KICS queries libraries and use them (if necessary).

5.2.3 Security Query Development Example

As a use case, focus on the Terraform solution for the security query [AWS_AC_24] "Neptune Cluster With IAM Database Authentication Disabled", presented in Section 4.2.5. The security query should verify if the field 'iam_database_authentication_enabled' is enabled in the resource 'aws_neptune_cluster'.

When using the REGO Playground, the input should contain the configuration target of analysis. In the query example, the input should point to the configuration of the resource 'aws_neptune_cluster'. In KICS, the input can be generated through the following command 'go run ./cmd/console/main.go scan -p <file_path> -d payload'. As an example, see the payload of the file presented in Listing 14:

```
{
  "document": [
    {
```

```

"file": "positive.tf",
"id": "388a0fa0-bbda-49f9-bbe3-aec598321743",
"resource": {
  "aws_neptune_cluster": {
    "positive1": {
      "apply_immediately": true,
      "backup_retention_period": 5,
      "cluster_identifier": "neptune-cluster-demo",
      "engine": "neptune",
      "preferred_backup_window": "07:00-09:00",
      "skip_final_snapshot": true,
      "storage_encrypted": true
    }
  }
}
]
}

```

Listing 14: Payload example.

After obtaining the payload, it is time to develop the query. For this query, it is necessary to develop two policies. One that verifies if the field 'iam_database_authentication_enabled' is undefined and another one that checks if the field 'iam_database_authentication_enabled' is set to false in the resource 'aws_neptune_cluster'. Since the payload presents a configuration with the field 'iam_database_authentication_enabled' undefined in the resource 'aws_neptune_cluster', it should satisfy the first policy. So, it is expected that the output presents results. See Figure 8.

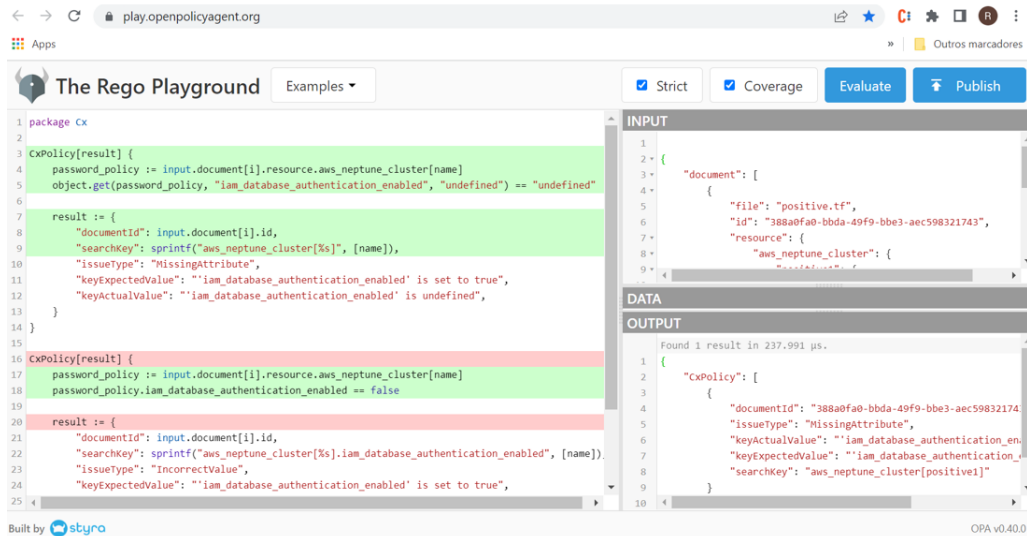


Figure 8: REGO Playground.

After using the REGO Playground and validating all the policies, it is necessary to create a branch in KICS and start to add the query. The query should be added according to the platform and cloud provider. In this case, it should be added in `'kics/assets/queries/terraform/aws'`. Remember the query file tree structure in Section 3.3.5 and see the content of the `metadata.json` and the `query.rego` for this query:

- **metadata.json**

```
{
  "id": "c91d7ea0-d4d1-403b-8fe1-c9961ac082c5",
  "queryName": "Neptune Cluster With IAM Database Authentication Disabled",
  "severity": "MEDIUM",
  "category": "Access Control",
  "descriptionText": "Neptune Cluster should have IAM Database Authentication enabled",
  "descriptionUrl": "https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/neptune_cluster#storage_encrypted",
  "platform": "Terraform",
  "descriptionID": "88b26e61",
  "cloudProvider": "aws"
}
```

Listing 15: `metadata.json` of the query "Neptune Cluster With IAM Database Authentication Disabled".

- **query.rego:** In this query example, to verify if the field 'iam_database_authentication_enabled' is enabled in the resource 'aws_neptune_cluster', it is necessary to check if the field is undefined or set to false.

```

package Cx

CxPolicy[result] {
    password_policy := input.document[i].resource.aws_neptune_cluster[name]
    object.get(password_policy, "iam_database_authentication_enabled",
    "undefined") == "undefined"

    result := {
        "documentId": input.document[i].id,
        "searchKey": sprintf("aws_neptune_cluster [%s]", [name]),
        "issueType": "MissingAttribute",
        "keyExpectedValue": "'iam_database_authentication_enabled' is set to true",
        "keyActualValue": "'iam_database_authentication_enabled' is undefined",
    }
}

CxPolicy[result] {
    password_policy := input.document[i].resource.aws_neptune_cluster[name]
    password_policy.iam_database_authentication_enabled == false

    result := {
        "documentId": input.document[i].id,
        "searchKey": sprintf("aws_neptune_cluster [%s].iam_database_authentication_enabled", [
            name]),
        "issueType": "IncorrectValue",
        "keyExpectedValue": "'iam_database_authentication_enabled' is set to true",
        "keyActualValue": "'iam_database_authentication_enabled' is set to false",
    }
}

```

Listing 16: query.rego of the query "Neptune Cluster With IAM Database Authentication Disabled".

5.3 Tests

As mentioned in the previous section and in Section 3.3.5, a KICS query is also composed of the test folder. For testing purposes, KICS requires that each query should have a folder named "test". This folder should contain positive and negative IaC samples as test cases. Additionally, it should have a JSON file with the expected results of the query against the samples. See the Listing 17.

```
- test
  |- positive<.ext>
  |- negative<.ext>
  |- positive_expected_result.json
```

Listing 17: Test folder tree.

The positive samples present vulnerable configurations that the security query should find. Focusing on query [AWS_AC_24] "Neptune Cluster With IAM Database Authentication Disabled" for Terraform, for example, "positive1.tf" should set a resource 'aws_neptune_cluster' with 'iam_database_authentication_enabled' undefined. See the Listing 18.

```
resource "aws_neptune_cluster" "positive1" {
  cluster_identifier      = "neptune-cluster-demo"
  engine                  = "neptune"
  backup_retention_period = 5
  preferred_backup_window = "07:00-09:00"
  skip_final_snapshot     = true
  apply_immediately       = true
  storage_encrypted       = true
}
```

Listing 18: Positive sample example ('positive1.tf').

Additionally, "positive2.tf" should set a 'aws_neptune_cluster' with 'iam_database_authentication_enabled' set to false, for example. See the Listing 19.

```
resource "aws_neptune_cluster" "positive2" {
  cluster_identifier      = "neptune-cluster-demo"
  engine                  = "neptune"
  iam_database_authentication_enabled = false
  backup_retention_period = 5
}
```

```

preferred_backup_window      = "07:00-09:00"
skip_final_snapshot         = true
apply_immediately           = true
storage_encrypted           = true
}

```

Listing 19: Positive sample example ('positive2.tf').

On the other hand, negative samples suggest a recommended configuration to avoid the vulnerability. In the present case, 'negative.tf' should set a 'aws_neptune_cluster' with 'iam_database_authentication_enabled' set to true. As an example, see the Listing 20.

```

resource "aws_neptune_cluster" "negative" {
  cluster_identifier      = "neptune-cluster-demo"
  engine                 = "neptune"
  iam_database_authentication_enabled = true
  backup_retention_period = 5
  preferred_backup_window = "07:00-09:00"
  skip_final_snapshot    = true
  apply_immediately      = true
  storage_encrypted      = true
}

```

Listing 20: Negative sample example ('negative.tf').

Finally, it should also contain the 'positive_expected_result.json', which indicates where the positives files have the vulnerability. As an example, see the Listing 21.

```

[
  {
    "queryName": "Neptune Cluster With IAM Database Authentication Disabled",
    "severity": "MEDIUM",
    "line": 1,
    "fileName": "positive1.tf"
  },
  {
    "queryName": "Neptune Cluster With IAM Database Authentication Disabled",
    "severity": "MEDIUM",

```



```

    "line": 4,
    "fileName": "positive2.tf"
  }
]

```

Listing 21: Positive expected result sample example ('positive_expected_result.json').

The test folder is essential for the validation of the query. When running the test folder of the query against the target query, it is expected that the scan returns the results presented in the 'positive_expected_result.json'. To test that, the command 'go run -tags dev ./cmd/console/main.go scan -p <query_query> -q <query_path>' needs to be run inside the 'kics' folder.

Additionally to that, it is necessary to run the command 'go test ./test' to ensure that the query follows all the KICS requirements. However, in the KICS GitHub pull request, there are KICS GitHub actions that verify if the query follows all the requirements.

5.4 Creation of the Pull Request

The security query should pass all the KICS queries tests to be ready for the pull request. After that, the following steps are required:

1. Commit and push the changes.
2. Submit the pull request on KICS GitHub repository.

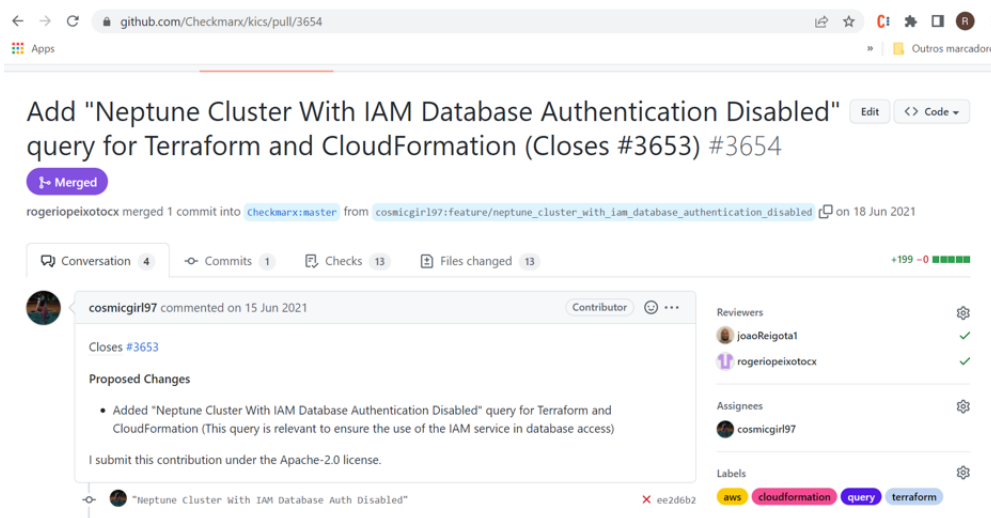


Figure 9: Pull request.

3. Wait for the pull request review.
4. Wait for the merge of the pull request.

5.5 Contribution Overview

In total, the work done in this dissertation contributes 71 new security queries to KICS, 34 of them for Terraform, 20 for Ansible, and 17 for CloudFormation. Table 50 presents all the security queries contributed to KICS, including the KICS GitHub pull request(s) related to each one.

The comparison of these values can only be considered fair between the same cloud provider(s) as the target of the dissertation study (AWS, AZURE, and GCP). From the 34 queries implemented for Terraform, 20 are related to AWS, 5 to AZURE, and 9 to GCP. On the other hand, from the 20 queries implemented for Ansible, 14 are related to AWS, 1 to AZURE, and 5 to GCP. As CloudFormation only covers AWS, all the 17 queries are related to AWS.

Overall, there are no significant discrepancies between the number of queries per cloud provider. However, Terraform stands out in all of them. The higher value for Terraform seems to suggest that this platform provides more relevant information about cloud services than the others.

Table 50: List of security queries contributed to KICS.

	Platforms			Pull Request(s)
	Ansible	CloudFormation	Terraform	
[AWS_AC_08] API Gateway without WAF	✓	✓	✓	#4547
[AWS_AC_10] API Gateway Without Configured Authorizer	✓	✓	✓	#2960 #4727
[AWS_AC_11] Certificate Has Expired	✓		✓	#2960 #4044
[AWS_AC_18] Elasticsearch Without IAM Authentication		✓	✓	#4689
[AWS_AC_24] Neptune Cluster With IAM Database Authentication Disabled		✓	✓	#3654
[AWS_AC_32] SES Policy With Allowed IAM Actions	✓		✓	#4548
[AWS_AC_35] IAM Access Analyzer Undefined		✓	✓	#4660 #4772
[AWS_AC_41] IAM Group Without Users	✓	✓	✓	#3815
[AWS_AC_53] Cross-Account IAM Assume Role Policy Without ExternalId or MFA	✓	✓	✓	#4546
[AWS_AC_66] Certificate RSA Key Bytes Lower Than 256	✓	✓	✓	#2960 #3089
[AWS_NS_05] Default EC2 security group are in use	✓	✓	✓	#4570
[AWS_NS_07] Default VPC in use for EC2 instance	✓	✓	✓	#4738
[AWS_NS_49] Elastic MapReduce Without VPC		✓	✓	#4571
[AWS_NS_50] ElastiCache Without VPC	✓	✓	✓	#4572
[AWS_NS_51] ElastiCache Using Default Port	✓	✓	✓	#4524
[AWS_NS_56] VPC Without Network Firewall		✓	✓	#4569
[AWS_NS_57] RDS Associated with Public Subnet	✓	✓	✓	#4737
[AWS_NS_58] Relational Database Service (RDS) Using Default Port	✓	✓	✓	#4522
[AWS_NS_61] Redshift Using Default Port	✓	✓	✓	#4656
[AWS_NS_74] Shield Advanced Not In Use		✓	✓	#4545
[AZURE_AC_21] Role Assignment Not Limit Guest User Permissions			✓	#3805
[AZURE_AC_22] Role Definition Allows Custom Role Creation	✓		✓	#3660 #5417
[AZURE_AC_29] Storage Share File Allows All ACL Permissions			✓	#3656
[AZURE_AC_39] Storage Table Allows All ACL Permissions			✓	#3658
[AZURE_NS_29] Virtual Network with DDoS Protection Plan Disabled			✓	#4509
[GCP_AC_13] Service Account With Improper Privileges			✓	#4516
[GCP_AC_16] IAM Role Assigned to User			✓	#4517
[GCP_AC_17] User with KMS Admin and CryptoKey Roles			✓	#4657
[GCP_AC_21] KMS Crypto Key is Publicly Accessible			✓	#4514
[GCP_AC_23] Container Cluster Using Default Service Account	✓		✓	#4515
[GCP_NS_06] Compute Network uses a default firewall rule	✓		✓	#4513
[GCP_NS_07] Google Compute Network Using Firewall Rule that Allows All Ports	✓		✓	#4512
[GCP_NS_08] Google Compute Network Using Firewall Rule that Allows Port Range	✓		✓	#4511
[GCP_NS_43] Compute Subnetwork with Private Google Access Disabled	✓		✓	#4510

CONCLUSIONS AND FUTURE WORK

This chapter presents the conclusions of this dissertation, focusing on contributions to KICS and the prospect for future work.

6.1 Conclusions

By analyzing misconfiguration and non-compliance problems in Infrastructure as Code, this dissertation proves the viability of adapting relevant security queries that were only collected by dynamic analysis tools for cloud-based infrastructure to static analysis tools for IaC. The 150 security queries that the present study reports as a gap between static analysis tools for IaC and dynamic analysis tools for cloud-based infrastructure in Access Control and Network Security context supports this statement.

This study contributes around 23% of the collected security queries to KICS for at least one platform, resulting in 71 new security queries to this open source tool. Although the percentage is one-third of the total, this value validates the dissertation study.

This discussion also raises relevant observations regarding 77% of the collected security queries. This value reflects the discarding process that identified a set of reasons that result in four scopes: (i) **dynamic scope** related to information that can be only accessible in a dynamic environment, (ii) **known IaC scope** related to similar information already available in an IaC static approach for the same resource, (iii) **inapplicable IaC scope** related to information that does not make sense to implement in an IaC static approach, (iv) **unknown scope** related to information that seems to not be accessible in an IaC static approach until the moment.

All the scopes mentioned above sustain strong motives responsible for 77% of the collected security queries without a solution. It also reveals how challenging it can be to find the exploited scope that fills the collected security queries with solutions (in this study, represented by 23%).

In conclusion, the contribution of 23% of the collected queries to KICS not only validates this work but can also be considered a successful achievement.

6.2 Future Work

As a prospect for future work, the case study of this dissertation can take the following approaches:

- **Cover other fields:** The present case study covers Access Control and Network Security contexts, which mainly focuses on security query categories like Access Control and Network Security. It would also be relevant to cover many others, such as Backup, Encryption, etc. (see Section 2.1.4).
- **Cross results in the same context:** The results of the case study are grouped in tables accordingly to the cloud provider and security query category, as can be seen in Section 3.1.2. These tables are analyzed individually.

Another approach can be crossing the tables related to the same security query category. For example, focus on the tables associated with Access Control context (Tables 6, 7, 8, and 9). Tables 6 and 7 are related to AWS, Table 8 to AZURE, and Table 9 to GCP. The collected security queries in Table 6 and Table 7 (AWS), if not implemented either in Table 8 (AZURE) or Table 9 (GCP), can be collected for implementation in either AZURE or GCP context and vice-versa.

- **Cover other platforms:** Platforms like Docker and Kubernetes, for example, are potential targets to compare the security queries covered by dynamic analysis for cloud-based infrastructure and static analysis tools for IaC.
- **Repeat the same case study in the future:** Technology is moving at a fast pace. Updates of the tools are constant, which almost certainly include new security queries. The repetition of the case study, some time from now, can result in the collection of new security queries only covered by dynamic analysis tools for cloud-based infrastructure.

REFERENCES

- Almuairfi, S. and Alenezi, M. (2020), 'Security controls in infrastructure as code', *Computer Fraud Security* (10), 13 – 19.
URL: <https://www.sciencedirect.com/science/article/pii/S1361372320301093>
- Bai, X., Li, M., Chen, B., Tsai, W. and Gao, J. (2011), Cloud testing tools, in 'Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)', pp. 1–12.
- Brikman, Y. (2019), *Terraform: Up & Running: Writing Infrastructure as Code*, O'Reilly Media.
URL: <https://books.google.pt/books?id=57ytDwAAQBAJ>
- Guerriero, M., Garriga, M., Tamburri, D. A. and Palomba, F. (2019), Adoption, support, and challenges of infrastructure-as-code: Insights from industry, in '2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)', pp. 580–589.
- Rahman, A., Mahdavi-Hezaveh, R. and Williams, L. (2019), 'A systematic mapping study of infrastructure as code research', *Information and Software Technology* pp. 65 – 77.
URL: <http://www.sciencedirect.com/science/article/pii/S0950584918302507>
- Rahman, A., Parnin, C. and Williams, L. (2019), The seven sins: Security smells in infrastructure as code scripts, in '2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)', pp. 164–175.
- Schwarz, J., Steffens, A. and Lichter, H. (2018), Code smells in infrastructure as code, in '2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)', pp. 220–228.
- Sharma, T., Fragkoulis, M. and Spinellis, D. (2016), Does your configuration code smell?, pp. 189–200.
- Silva, C. E. and Campos, J. C. (2013), 'Combining static and dynamic analysis for the reverse engineering of web applications', *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems - EICS 13* .