Empirical Software Engineering (2025) 30:74
https://doi.org/10.1007/s10664-024-10610-0

Assessing the adoption of security policies by developers in terraform across different cloud providers

Alexandre Verdet[1] · **Mohammad Hamdaqa**[1,2] · **Leuson Da Silva**[1] · **Foutse Khomh**[1]

Accepted: 29 December 2024

Abstract
Cloud computing has become popular thanks to the widespread use of Infrastructure as Code (IaC) tools, allowing the community to manage and configure cloud infrastructure using scripts. However, the scripting process does not automatically prevent practitioners from introducing misconfigurations, vulnerabilities, or privacy risks. As a result, ensuring security relies on practitioners' understanding and the adoption of explicit policies. To understand how practitioners deal with this problem, we perform an empirical study analyzing the adoption of scripted security best practices present in Terraform files, applied on AWS, Azure, and Google Cloud. We assess the adoption of these practices by analyzing a sample of 812 open-source GitHub projects. We scan each project's configuration files, looking for policy implementation through static analysis (Checkov and Tfsec). The category *Access policy* emerges as the most widely adopted in all providers, while *Encryption at rest* presents the most neglected policies. Regarding the cloud providers, we observe that AWS and Azure present similar behavior regarding attended and neglected policies. Finally, we provide guidelines for cloud practitioners to limit infrastructure vulnerability and discuss further aspects associated with policies that have yet to be extensively embraced within the industry.

Keywords Security vulnerabilities · Infrastructure as code · Policy misconfiguration

Communicated by: Valeria Pontillo,Coen De Roover, Riccardo Scandariato,Emanuele Iannone

This article is part of the Topical Collection: *Special Issue on Security Testing for Complex Software Systems (SECUTE).*

✉ Alexandre Verdet
alexandre.verdet@polymtl.ca

Mohammad Hamdaqa
mhamdaqa@polymtl.ca

Leuson Da Silva
leuson-mario-pedro.da-silva@polymtl.ca

Foutse Khomh
foutse.khomh@polymtl.ca

1 Polytechnique Montreal, Montreal, Canada

2 Reykjavik University, Reykjavík, Iceland

1 Introduction

The current need for software development focused on reducing the space, time, and efforts between the software development itself and its delivery is known as DevOps (Ebert et al. 2016). Its adoption is directly related to adopting security practices, which refer to actions, procedures, and policies adopted to protect information, assets, and systems (Ur Rahman and Williams 2016). In DevOps, previously adopted practices are further explored in order to orchestrate and manage the execution and intersection of tasks, leading to the concept of Infrastructure as Code (IaC) (Morris 2020). For example, IaC has been integrated into production pipelines to further improve automation (Spinellis 2012; Humble and Farley 2010).

Overall, IaC represents a set of practices responsible for automatically managing and provisioning computing infrastructure to local and remote instances through machine-readable script files (Artac et al. 2017). Based on these scripts, practitioners may define and manage the infrastructure required for different activities, like deploying large cloud infrastructures, while retaining the benefits of software development, such as code reuse, collaboration, testing, and static code analysis.

Over time, practitioners and researchers have been interested in IT security and privacy, considering the rise of new user concerns following major data leaks and surveillance. In order to address those related challenges, several data residency regulations have been introduced, such as the GDPR, CCPA, PIPEDA, HIPAA (European Commission 2016; California State Assembly 2018; Office of the Privacy Commissioner of Canada 2000). Knowing that more systems are moved and/or deployed to different types of cloud, like public, private, and hybrid, there is a growing demand for secure IaC solutions. In order to help practitioners comply with specific regulations, organizations and consortiums establish lists of security best practices. With time, some of those lists gained recognition among practitioners, becoming industry standards.

This way, security guidelines from those regulations establish sets of best practices, which actual implementations can be called policies. While IaC tools such as Terraform (HashiCorp 2022), AWS CloudFormation (Services 2023), and Azure Resource Manager (ARM) (Azure 2023) simplify software infrastructure provisioning, they shift responsibility for security risks to operational teams (Sharma et al. 2016). However, writing security policies as code can be challenging. For instance, improper configuration can compromise security and put sensitive cloud data at risk (Sengupta et al. 2011). In order to mitigate or avoid these issues, new standards have been proposed that IaC configurations must meet (Kemp 2018). However, implementing these standards remains uneven because not all compliance guidelines can be translated into configuration implementation policies (actual identifiable code patterns), and not all policies are uniformly enforced.

As a result, the responsibility of infrastructure security is shared between practitioners and cloud providers and is often targeted by data protection regulations like GDPR or CCPA. Insecure infrastructure can lead to unauthorized access to data or server instances, compromising the overall system. To understand the usage and associated challenges with IaC, Guerriero et al. (2019) perform a qualitative study interviewing practitioners. The authors report that maintaining IaC code is one of the current challenges, while Terraform was observed as the most popular infrastructure provisioning tool. In the same way, Iosif et al. (2022) investigated security vulnerabilities in AWS repositories where the Terraform component is major. However, besides only focusing on vulnerabilities, in real-world projects, IaC components tend to co-exist with other types of files and represent only a small percentage of the full project

(Jiang and Adams 2015). Although previous studies have focused on general security issues, when it comes to issues caused by errors in IaC scripts, that is an avenue open for investigation (Rahman et al. 2019b). In this study, we investigate the adoption of best practices related to the implementation of security policies by Infrastructure as Code (IaC). For that, we propose, categorize, and check the implementation of security policies in repositories adopting IaC.

First, we identify, select, and propose a categorization of recognized security policies, and then examine the degree to which each type is observed in real-world deployments. Second, based on the popularity of IaC environment on GitHub, we perform an empirical study evaluating the implementation of the previous policies on GitHub repositories exploring Terraform files. Although cloud providers support different IaC tools, Terraform stands out as one of the most used tools in practice. Regarding the usage of Terraform, GitHub currently indexes 109K Terraform files with AWS configurations, compared to 35.7K with Azure and 51.5K with Google Cloud. For that, we mine open-source GitHub repositories based on three different cloud providers (AWS, Azure, and Google Cloud), and then we scan each project, looking for the presence or absence of security policies (checkov and tfsec).

To summarize, we investigate the following research questions in this study:

1. ***RQ1:*** *Which security guidelines have matching Terraform implementation policies, and how can they be categorized?*
   The objective of this research is to examine how different security guidelines are implemented by various cloud providers through their Terraform implementations and to gain insights into their adoption. Given the multitude of policies available, it is challenging to derive meaningful insights from individual policy implementations. Therefore, we categorize the policies into well-adopted groups and evaluate whether this categorization comprehensively covers all existing policies. We conduct an analysis based on policy categories, which allows us to uncover policy patterns and enhance practitioners' awareness. Our research presents a catalog of security policies classified into eight categories, informed by both previous research and our current findings. This approach enables practitioners to understand which security policies are supported by various cloud providers, aiding them in making informed decisions. Additionally, it highlights opportunities for cloud providers to implement support for currently unsupported policies, thereby enhancing their offerings and addressing gaps in security policy implementation.
2. ***RQ2:*** *How are common security best practices being adopted in Terraform files?* In this RQ, we focus on practices that tend to be well adopted in projects with Terraform components to spotlight patterns that foster implementation. Additionally, we also intend to investigate configuration flaws through neglected practices to suggest improvements in those areas from providers and practitioners. We observe, based on the previous categorization, that the category *Access policy* emerges as the most widely adopted in all evaluated cloud providers. Since we consider different cloud providers in our evaluation, we aim to compare the findings obtained through the previous questions between cloud providers to generalize and validate some findings, as well as highlight reasons for observation from the characteristic differences between providers.

Based on our findings, we provide general guidelines for cloud practitioners to limit infrastructure vulnerability and discuss further aspects associated with policies that have yet to be extensively embraced within the industry. As contributions of this study, we list:

– We evaluate the categorization of security policies related to IaC and propose three new categories, based on standard industry-recognized patterns, which could be used for future studies and also be extended;

– Empirical investigation exploring the adoption of these policies in Terraform files based on three different cloud providers;
– Comparison involving Checkov and Tfsec, static analysis tools for policy checking, and mapping of their supported checks;
– We provide our datasets and methods to evaluate the implementation of security policies online to support further studies as well as replications of our current work (Verdet et al. 2023).

2 Background and Motivation

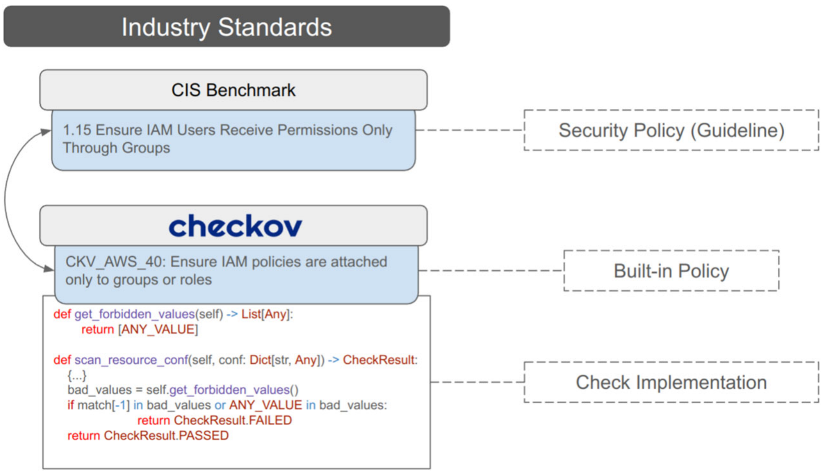
This section presents background information and motivates our work with context and real-world examples. Initially, we discuss Infrastructure as Code (IaC), providing more details about Terraform, the IaC tool we focus on in this paper. Then, to illustrate the problems related to policy implementation through Terraform, we discuss two scenarios below where practitioners implemented such policies.

2.1 Infrastructure as Code

IaC is the concept of defining computing infrastructure requirements with source files in the form of code. Based on these files, IaC tools parse the defined infrastructure and automatically provide the expected output. For example, regarding the setup of a cloud environment, the IaC tool parses the configuration file, holding all the pre-defined restrictions of the desired environment, and then deploys the environment in the cloud and/or on-premises environment. During this process, the infrastructure definition can go through the DevSecOps lifecycle, enabling compliance and security checks before deployment. As popular IaC tools, we may cite Terraform, Chef, Puppet, Ansible, and CloudFormation.

Overall, IaC tools can address different types of needs and layers in an infrastructure. IaC tools can create, modify, and destroy infrastructure resources like computation instances, storage, and networking components; these tools are conventionally known as *Infrastructure Management tools*. Other tools can be used to deploy and update the application running on the infrastructure (Morris 2020). Most popular frameworks can be used to deploy not only the infrastructure but also the application. It is possible broad tools use more specific components to take care of the infrastructure management step; for instance, Ansible can use Terraform for the infrastructure needs (Nayak 2019). Our study focuses on the infrastructure management stage of IaC, which directly interacts with cloud providers to define and provision cloud infrastructures. In previous work, Guerriero et al. (2019) investigated the adoption of IaC tools in the industry. Their findings report Terraform as the most popular IaC tool focusing on infrastructure management (59%).

Terraform (HashiCorp 2022) is a leading infrastructure management tool with extensive compatibility across providers and integration with other tools, such as Ansible. It is an open-source project led by HashiCorp and released in 2014. The tool supports managing infrastructures from public cloud providers such as AWS, Azure, and GCP, as well as private cloud frameworks like OpenStack. The main purpose of Terraform files is to declare the infrastructure resources, commonly known as manifest. Infrastructure provisioning allows users to automatically deploy and configure resources like servers, storage, networks, and services directly in the cloud (Juve and Deelman 2011). Resources are declared in blocks, with all the relevant configurations within Terraform files written in HCL (HashiCorp Con-


Industry Standards
CIS Benchmark
1.15 Ensure IAM Users Receive Permissions Only Through Groups
Security Policy (Guideline)
checkov
CKV_AWS_40: Ensure IAM policies are attached only to groups or roles
Built-in Policy
Check Implementation

```
def get_forbidden_values(self) -> List[Any]:
      return [ANY_VALUE]

def scan_resource_conf(self, conf: Dict[str, Any]) -> CheckResult:
    {...}
    bad_values = self.get_forbidden_values()
    if match[-1] in bad_values or ANY_VALUE in bad_values:
            return CheckResult.FAILED
    return CheckResult.PASSED
```

Fig. 1 Associating Security and Built-in Policies

figuration Language). For example, a block declares a computing instance with the required computing power, the size of the allocated, and the image to deploy on it. Modules can ease the definition of the infrastructure by reusing code, like programming libraries. Once the infrastructure is declared, Terraform converts the code into API calls to deploy the resources. The tool acts as an interface between the cloud practitioners and provider API to ease the deployment.

Over time, different industry standards have been proposed focusing on how to write and maintain IaC code. Overall, these standards group a set of best practices, reported as *security policies*, aiming to guide practitioners when defining their IaC blocks. For example, in Fig. 1, we present an example of a security control best practice guideline as recommended by the CIS Benchmark, a widely recognized framework (Center for Internet Security 2021). Associated with that benchmark, practitioners can find or define a set of security policies' checks or implementations; for instance, in Fig. 1, we present CKV_AWS_40 an automated policy check that Checkov uses to scan AWS infrastructure code and identify instances where IAM permissions are not granted via groups, thereby helping organizations adhere to the security best practices outlined by CIS. The rationale behind such policy best practices advocates that adopting such a guideline unifies permissions management to a single and flexible method while reducing excessive permissions. With these security policies best practices, SCA tools, like Checkov and Tfsec, have also been proposed to automatically detect errors when implementing these policies. These tools group *built-in policies*, which are predefined checks to evaluate the implementation correctness of the target security policies. For the example under discussion, Checkov presents the built-in policy `CKV_AWS_40`, which checks whether the IaC block provides permission to groups or roles instead of specific users.[1]

2.2 Motivating Example

To illustrate the implementation of policies in Terraform, we discuss two examples adapted from real projects on GitHub. First, consider a practitioner adopting Terraform to define and

[1] Checkov - Policy ID CKV_AWS_40

manage an AWS cloud infrastructure used to deploy their application. [2] For that, it writes down a new block with the required information, as represented in Listing 1. Initially, `region` to `us-east-1` is defined, specifying the AWS region where their resources will be provisioned. Next, the `access_key` and `secret_key` are informed. These two credentials are used for authentication on the AWS service. Mocking or using predefined keys represents good practice, as informing the valid values would expose the user's sensitive information. This way, anyone could access and misuse those credentials, representing a security risk.

```
provider "aws" {
  region                      = "us-east-1"
  access_key                  = "mock_access_key"
  secret_key                  = "mock_secret_key"
}
```

Listing 1 Correct Policy Implementation on Terraform

To ensure the policy is correctly implemented, a developer could review it or even ask another teammate to do it. However, to optimize the team's productivity, SCA tools like Checkov and Tfsec might be adopted to automatically perform such tasks, and eventually report problematic implementations. For the policy reported in Listing 1, Checkov evaluates whether the values for the user credentials `access_key` and `secret_key` match with a predefined pattern. [3] Since the information provided in Listing 1 does not match the expected pattern, Checkov would not report a problem with the implemented policy.

In the same way, another practitioner decided to use Terraform to define an AWS security group resource named `lb` in the context of AWS (see Listing 2). In the new rule `ingress`, while the first three properties provide information about the supported protocol and the starting/ending ports (`protocol`, `from_port`, and `to_port`, respectively), the last property may currently introduce a possible security risk. The property `cidr_blocks` is responsible for defining the IP range that is allowed to access the associated resource. By setting that property to `"0.0.0.0/0"`, the practitioner allows incoming traffic from any IP address, meaning that any device could potentially communicate with the specified resource. To address the potential threats, it is recommended to specify the trusted IP ranges that should communicate with the associated web server.

```
resource "aws_security_group" "lb" { {...}
  ingress {
    protocol    = "tcp"
    from_port   = 80
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Listing 2 Example of an AWS S3 Bucket resource in Terraform

Calling again Checkov to check the implementation of the previously presented policy (Listing 2), we observe the tool focuses on the contents of the rule `ingress`, specifically in the property `cidr_blocks`. After checking the ports declared in the rule and additional general information, Checkov gets to the point of checking whether `"0.0.0.0/0"` is reported in the property `cidr_blocks`, eventually indicating a problem in the policy. [4] As a result, the tool reports the implementation is not correctly implemented, as a security

[2] Link to commit

[3] Checkov - Policy ID CKV_AWS_41

[4] Checkov - Policy ID CKV_AWS_260

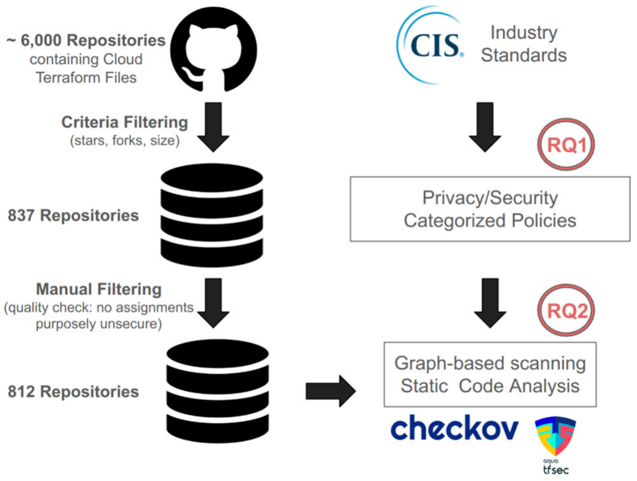


Fig. 2 Research process overview

issue was detected. Those two examples show, in practice, the type of configuration we will be looking for when trying to measure the adoption of security policies by practitioners in Terraform files.

3 Study Setup

Our methodology comprises two main steps (Fig. 2). First, to answer RQ1, we select and categorize Terraform security policies promulgated in the industry for popular cloud providers, such as AWS, Azure, and Google Cloud. Second, to address RQ2, we select open-source repositories hosted on GitHub to assess the adoption of the previously categorized practices.

3.1 Policies Categorization

To address our first research question (RQ1), we carefully select and compile a list of industry-recognized security policies, map them to their related security best practices, and further categorize them based on their individual goals. The method adopted for this process is synthesized and visualized in Fig. 3.

3.1.1 Selecting and Mapping Industry-Recognized Security Policies

The first step in our approach involves the systematic selection of security policies from well-established industry frameworks known for their effectiveness in compliance and security assurance. We focus on identifying policies that are relevant to Infrastructure as Code (IaC) implementations using Terraform, with an emphasis on those applicable to different major cloud providers. For each cloud provider evaluated in this study, we identified relevant policies from industry-recognized frameworks to ensure comprehensive coverage of security best practices. (Step 1 in Fig. 3). For the AWS cloud provider, we observe that the CIS Amazon Web Services Foundations (Center for Internet Security 2021)[5] and the AWS Foundational

[5] Version v1.4.0

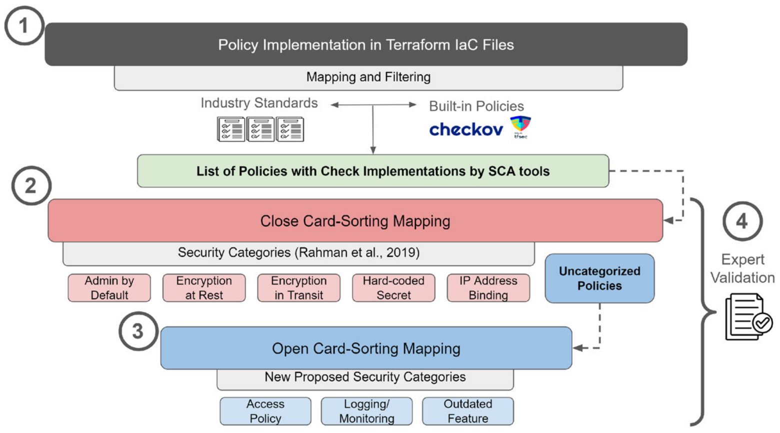


Fig. 3 RQ1 policies selection and categorization process

Security Best Practices (Amazon Web Services 2022) [6] are the two most widely recognized and deployed industry-standard frameworks (Stultiens 2020). For the Azure, the Center for Internet Security (CIS), which established the CIS Amazon Web Services Foundations, also defined an Azure equivalent, the CIS Microsoft Azure Foundations Benchmark (Center for Internet Security 2023a).[7] Finally, for the Google Cloud provider, we consider the CIS Google Cloud Platform Foundation Benchmark (Center for Internet Security 2023b). Considering that we select two sources of information for AWS, we combine them in one single set, resulting in 298 policies. For Azure and GCP, we consider one single source, resulting in 247 and 137 policies, respectively.

After selecting the appropriate policies from industry-recognized frameworks, the next step was to filter and map them to corresponding best practices implemented by Static Code Analysis (SCA) tools. In our study, we considered two state-of-the-art (SOTA) SCA tools: Checkov and Tfsec. These tools were chosen based on their popularity, widespread adoption in various projects, and relevance in previous related work. The selected SCA tools provide a wide range of built-in policies that verify compliance with security guidelines and standards across major cloud providers Each built-in policy is associated with a configuration snippet in Infrastructure as Code (IaC) files, implementing a particular security feature within the provisioned cloud resource.

In the scope of this study, we focused exclusively on policies with existing implementations, as our goal is to trace the actual prevalence and adoption of these policies by practitioners. Custom policies, which may not provide consistent insights, were not included in this analysis. This way, we explore Terraform files that define cloud infrastructures for the cloud providers assessed in this study

To ensure accurate alignment between policies collected from the industry-recognized frameworks and SCA tool implementations, we conducted a thorough review of the documentation provided by each SCA tool to understand the specific checks and controls related to the selected policies. Given the complexity of the mapping process, manual inspection by Infrastructure as Code (IaC) experts was necessary to ensure accuracy and validity. This

[6] Version v1.0.0

[7] Version v2.0.0

Table 1 Supported Standards Specifications and Policies by SCA tools

Cloud Providers	Tools					
	Checkov			Tfsec		
	Covered	Selected	Detected	Covered	Selected	Detected
AWS	298	121	117	137	101	94
Azure	247	84	79	50	34	33
GCP	137	82	82	62	45	45
Total	682	287	278	249	180	172

involved cross-referencing each policy with SCA tool capabilities and verifying the mappings.

The manual mapping process was conducted between the policies related to each cloud provider, thoroughly exploring the documentation provided by each SCA tool. For example, for the policy `EC2.21` reported on the AWS Foundational Security Best Practices (*IP Address Binding* category). The policy recommends that *Network ACLs should not allow ingress from 0.0.0.0/0 to port 22 or port 3389*[8]. Based on the title of the policy and its description and by analyzing the Checkov documentation, we observe that check `CKV_AWS_231` is responsible for evaluating such a policy.[9] For each policy, we established a mapping to the corresponding checks in the SCA tools. If no mapping could be established, the policy was discarded. This analysis ensures that each policy might be supported by at least one of the SCA tools, though a single policy may be supported by none, one, or both of the evaluated tools. By applying the mapping process described in this section, we selected a set of standard security policies that cloud providers should support or consider for compliance, with existing implementations in SCA tools. The summary of these policies is presented in Table 1.

3.1.2 Policy Categorization

Classifying each configuration into a category is essential to retrieving broad knowledge from the database scanning analysis. This way, we decide to explore grouping security policies into categories instead of studying the adoption of security policies alone, as they might lead to restricted results. To achieve this goal, we manually categorize the previously filtered policies into categories (Steps 2 and 3 in Fig. 3) using a closed and open card-sorting method. We adopt the same process for the policies supported by each tool evaluated in this study. Regarding the policies supported by Checkov, starting with the AWS policies, we aim to categorize the policies into seven (7) categories originally proposed by Rahman et al. (2019c). To the best of our knowledge, the categorization proposed by Rahman et al. (2019c) is the first one regarding security smells in IaC. Although some categories could be merged into one single category, we consider all provided categories individually, a decision further supported by external evaluation from experts. During the first mapping round (Steps 2 in Fig. 3), 72 policies were mapped into five categories: (*Hard-coded secret*, *IP Address binding*, *Admin by default*, *Encryption at rest*, *Encryption in transit*). For example, consider the previously discussed policy EC2.21, regarding guidance for allowing ingress from specific IP addresses. Among the seven categories, *IP Address binding* is the target one, as it is responsible for dealing with address assignment for a specific cloud service/instance. Consider now a new

[8] Policy EC2.21

[9] Checkov check - Ensure no NACL allow ingress from 0.0.0.0:0 to port 3389

policy responsible for *Ensuring EBS Volume Encryption is Enabled in all Regions*. Checking additional information from this policy, the CIS AWS Benchmark advises that encrypting data at rest reduces the likelihood of unintentionally exposing and nullifying the impact of disclosure if the encryption remains unbroken. Relying again on the previous seven categories, we can clearly assign the discussed policy to *Encryption at rest*. For some categories reported by Rahman et al. (2019c), we did not map any policies to them (*Suspicious comment* or *Empty password*). Although some policies under analysis explore multiple subjects, their main goals could be mapped in one of the previous five used categories.

The remaining 49 policies were further evaluated in a second round (Step 3 in Fig. 3), leading to the addition of two new categories: *Access Policy* and *Logging/Monitoring*. Since our study focuses on Terraform, new categories are required as they focus on the infrastructure provisioning stage of IaC. These new categories were created based on the mutual goal associated with their individual policies. For example, consider the policies responsible for *ensuring that S3 bucket access logging is enabled on the CloudTrail S3 bucket* and CloudTrail is enabled in all regions. Although the first policy relates to the activity of logging, the second policy focuses on using previously produced information for future monitoring. This way, we group the policies in the new category *Logging/Monitoring*. In the end, 45 policies were grouped into the new two categories. Finally, the remaining four policies were further evaluated in a third round (still in Step 3 in Fig. 3). By analyzing them, we observed that the commonality between all the remaining policies is that they focus on preventing the use of outdated, deprecated, or vulnerable versions that could pose security risks. Specifically, they address preventing the use of outdated versions and automatic updates and patching. For example, in AWS, ensuring Instance Metadata Service Version 1 is not enabled and RDS PostgreSQL instances use non-vulnerable versions with the log_fdw extension for AWS. For Azure and GCP, we observe general concerns about ensuring the latest required resources are used, like getting the latest Java version to run the web app and using the latest version of the SQL database, respectively. Then, we propose the creation of the category *Outdated Feature*. Overall, the total number of interactions in the process is 174 = 121 + 49 + 4. All the categories are explained in details in Section 4.

Next, we follow the same approach to map the policies associated with Azure and Google Cloud providers, as well as for all the policies supported by Tfsec. All policies could be mapped to one of the eight categories during these mappings in one round (Steps 2 and 3 in Fig. 3). One researcher with strong knowledge of Terraform configuration and cloud providers performed this mapping process. The final distribution of the policies for all providers is presented in Table 2, and the full categorization can be found in our online Appendix (Verdet et al. 2023).

Since each category groups a different number of policies, we generate guidelines based on the prevalence of their failing policies for each category. Although the policies have individual goals, grouping them into categories allows us to explore the same overall goal they shared, supporting us when providing security guidelines. Based on the failing policies, we analyze them and provide a guideline covering their individual goals. Although we analyze different cloud providers, our guidelines are not specific for each provider. Rather we focus on the categories previously used to group related policies.

To investigate the potential to offer more robust and scalable results regarding categorizing the policies, we explored LLM to leverage the manual categorization of security policies. For that, we collect the previously selected built-in policies from Checkov and Tfsec associated with the AWS provider and prompt the LLM to categorize them based on the eight categories considered for this study. For each built-in policy, we extract the available information provided in the documentation of each tool and feed it into the GPT-4o-mini with temperature

Table 2 Distribution of Policy Category Mapping

Policy Categories	Cloud Providers					
	AWS		Azure		GCP	
	Checkov	Tfsec	Checkov	Tfsec	Checkov	Tfsec
Access Policy	29	18	25	11	31	14
Admin by Default	6	3	2	1	5	8
Encryption at Rest	35	35	16	2	12	3
Encryption in Transit	15	8	21	7	3	3
Hard-coded Secret	4	5	1	1	-	1
IP Address Binding	12	7	4	4	7	5
Logging/Monitoring	16	19	9	7	23	11
Outdated Feature	4	6	6	1	1	-
Total	121	101	84	34	82	45

0.2, as such a value results in more accurate and deterministic results compared to higher temperatures (Lin et al. 2024). For each category, we provide their name and description; for the categories originating from Rahman et al. (2019c), we use the description provided by the authors, while for the categories proposed by this study, we consider our description (see Section 4). We repeated this process three times; each time computing the Cohen's Kappa agreement between our manual and the LLM categorization and, finally, computing the mean of the three executions (Cohen 1960).

3.1.3 Validation by Independent Experts

Since our mapping of policies relies on the subjective judgment of one researcher, we mitigate potential bias by inviting independent Security Consultants for each cloud provider (Step 4 in Fig. 3). For that, we invite two experts who currently work on one of the most adopted cloud providers by practitioners. The selection requirements were based on (i) previous solid experience implementing security best practices and (ii) using automated security software scanning tools like Checkov in several projects. Once the consultants were selected, instructions were given by video conference, and possible questions were also addressed. Next, we provided the experts with a description of each policy and the category previously mapped in the form of a spreadsheet. Additionally, they had access to the full documentation of the policies with their code implementations. For each policy the experts were asked to agree or disagree with its mapping by checking an empty checkbox. Related threats to this step are discussed in Section 6. For the AWS provider, two experts were recruited with six and two years of experience, respectively, while for the Azure and Google Cloud providers, two other experts were recruited with four years of experience each. As a result, we observe a Cohen's Kappa score of $\kappa = 1$, in all validation processes, showing that our categorization represents a valid way to generalize and group security policies based on IaC.

3.2 Selecting Open-Source Repositories based on Cloud Providers

In order to assess the adoption of IaC security best practices, we evaluate the implementation of the previously selected policies. For that, first, we systematically select repositories hosted on GitHub based on changes in specific configuration files.[10] Next, we filter these projects based on some metrics. Based on each cloud provider, this process was repeated, leading to the sample of projects evaluated in this study (Fig. 2).

3.2.1 Datasets Collection

Jiang and Adams (2015) report that IaC components co-exist with other types of files in open-source projects. The median of IaC files turned out to be around 11% of the total project file number. Considering this small frequency of IaC scripts, the most representative way to collect relevant repositories is by searching for Terraform files corresponding to our chosen cloud provider.

The standard file extension for Terraform is '.tf'. This way, we expect most targeted files to follow this guideline. Moreover, the selected cloud provider has to be introduced in one file with the code snippet '`provider "cloud_provider"`'. As such, for each cloud provider, we query the GitHub Code Search API and successfully collect the last 1,000 indexed corresponding Terraform files. The results are sorted by *Last indexed files* to avoid bias from the GitHub *Best match* algorithm. Since 1,000 Terraform indexed files at a certain point in time become no longer representative, we ran the same query and collected the repositories periodically each day for specific consecutive days. Eventually, duplicated repositories were removed, leaving only distinct repositories.

For the AWS provider, we run the associated query for three weeks (from September 12th to October 5th, 2022). As a result, 3,245 distinct GitHub repositories with at least one AWS Terraform file were selected. For the Azure provider, the data collection period also lasted one month (from January 20th to February 20th, 2023), reporting 1,308 repositories. Finally, for the Google Cloud provider, the data collection period lasted one month from March 20th to April 20th, 2023. In total, 1518 repositories were selected. Although the sample selection process was performed in different periods for each cloud provider, we observe a saturation point regarding the list of projects associated with the changed files for all providers. Since we collect the first 1000 files associated with each provider daily (sorted by relevance and indexation date) and later extract the associated projects during the last iterations, we observe this saturation point. Such an observation shows that our methodology was valid for selecting active projects. Associated threats to the validity of this step are addressed in Section 6.

3.2.2 Repository Filtering

With the initial collected projects, we filter the repositories, selecting those relevant to our study. For that, we adopt the same filter metrics adopted by previous studies (Das et al. 2022; Gonzalez et al. 2020) that examined GitHub repositories for Blockchain and Machine Learning projects, respectively. This way, we collect the next metrics (Gousios and Spinellis 2017):

- Activity: the last project indexation must be later than September, 1^{st} 2022. Specifically, in our context, we believe inactive projects might not reflect the issues investigated here, as these projects could be seen as obsolete and possibly introduce noise in our results.
- Size: a repository must have a non-null size (e.g., > 0 KB).
- Originality: a repository must not be a fork of another project.

[10] In the context of the Terraform IaC tool, we call interchangeably manifest files Terraform configuration files.

- Popularity: a repository must have at least ≥ 2 stars.
- Data Availability: the repository must be publicly accessible via the GitHub API.
- Content: the project must not be a course assignment, tutorial, or purposefully insecure project.

The first five criteria are inherent to our sample collection method, while the sixth requires manual filtering. For the AWS sample, after applying the first five criteria, 413 were remaining. Next, based on our sixth criterion, a manual analysis was conducted, filtering out 12 projects and leading to the final sample of 401 relevant repositories. More specifically, we remove two (2) purposefully insecure projects, six (6) workshops/assignments, and four (4) course materials/tutorials. For the Azure sample, after applying the first five criteria, 143 projects were returned. Next, after the manual analysis, 6 projects were removed, leading to the final sample of 137 Azure projects. The discarded repositories refer to workshops/assignments (3) and course materials/tutorials (3). Finally, for the Google Cloud sample, 281 repositories passed the first five criteria. After the manual analysis, 274 repositories compose the final sample. Regarding the manual analysis, the removed repositories refer to workshops/assignments (4), course materials/tutorials (2), and purposefully insecure projects (1). Finally, we save the selected projects as individual textual files grouped by the associated cloud providers. For each project, we save their names and associated owner IDs (username).

3.3 Running the Study

Once we select our sample, we call the SCA tools Checkov and Tfsec to check the implementation correctness of the security policies. Since our analysis is focused on checking the passed rates of security policies, we compute the passed rate, which is computed based on the number of passed checks compared to the number of passed and failed checks. The formula below better illustrates it:

$$passed\_rate = passed\_checks/(passed\_checks + failed\_checks) \quad (1)$$

In the end, for a given project, we save the number of failed and passed checks, their associated policy IDs (informed by each tool, see Fig. 1), and the date the analysis was conducted. This information was grouped in different CSV files grouped by SCA tools and cloud providers.

3.3.1 Comparing SCA Tools

Since we invoke the SCA tools for performing their analysis considering an entire project, it is expected that one single check can *pass* and *fail*. Such behavior is expected as the projects might have different Terraform files, and the tools perform their analysis considering each file individually. Associated threats to validity are discussed in Section 6. However, these tools might present different results for the same security policy. This way, we aim to evaluate the accuracy of the SCA tools evaluated in our study. For that, we first need to filter the policies supported by both tools. Second, we consider the checks associated with the previous policies. Third, we select the cases in which the tools present different outputs for the same policies. Our goal with such analysis is to understand the ≥ reasons for such disagreements and further explore guidelines for tool maintainers and users.

Once we collect the policies with conflicting evaluations, we check their associated check results and perform our manual analysis. For that, we randomly select a subsample of the conflicting evaluations. The manual analysis was conducted by the third author of this work and further validated by the first one. For each case, an analysis was done based on the official documentation of each tool and further information when available, like example implementations and associated code. Since such an analysis was based on a straightforward approach, we decided that one researcher would perform all the required checks while the other would double-check. We understand such a decision might introduce noise in our results due to personal bias, and we now discuss such a threat (see Section 6).

To investigate the impact of the reported alerts on different cloud providers, we identified policies commonly implemented by all providers. For that, we considered the initial mapping of policies supported by Checkov and Tfsec for each provider. Since our goal was to investigate how the structure of Terraform impacts the occurrence of failures, we filtered out cases in which single policies are associated with one provider while other providers require multiple policies to reach the same goal. As a result, we select a set of 5 policies from four categories.

Finally, to further assess the support of the tools, we calculate two weighted averages. The first one is based on the range of selected and detected policies, while the second one considers the number of detected and failed policies (see Table 1). While the first average focuses on the coverage provided by the tools, the second one focuses on the effectiveness of the tools to detect failures in policy implementations. Below, we explain how we compute these metrics.

Let:

- R_{ij} be the number of relevant policies (either detected or failed) by tool j for cloud provider i,
- C_{ij} be the number of covered policies by tool j for cloud provider i,
- N be the total number of cloud providers,
- M be the total number of tools being evaluated.

For each cloud provider i and tool j, the relevant rate (either detection or failing detection) is calculated as:

$$\text{Rate}_{ij} = \frac{R_{ij}}{C_{ij}} \times 100$$

To compute the *generalized weighted average rate* for tool j across all cloud providers, we calculate:

$$\text{Weighted Average Rate}_j = \frac{\sum_{i=1}^{N} (R_{ij} \times \text{Rate}_{ij})}{\sum_{i=1}^{N} R_{ij}}$$

Where:

- R_{ij} can represent the number of detected policies or the number of failed detections for cloud provider i and tool j,
- $\sum_{i=1}^{N} (R_{ij} \times \text{Rate}_{ij})$ is the sum of relevant policies (detected or failed) multiplied by their corresponding rate for each cloud provider,
- $\sum_{i=1}^{N} R_{ij}$ is the total number of relevant policies across all cloud providers for tool j.

4 Results

In this section, we report the results addressing our previous research questions.

4.1 RQ1: Which Security Guidelines have Matching Terraform Implementation Policies, and how can they be Categorized?

Overall, we can observe a similar distribution of mapped policies to categories across the different tools (see Table 1). For example, considering the top two categories with more associated policies by each cloud provider, we observe consistent support by the SCA tools. Such observation shows that SCA tools focus on common categories for implementing checkings. Although we observe a regular distribution of policies over the categories, none of the selected policies have been mapped to the categories *Hard-coded Secrets* and *Outdated Feature* regarding the implementation of Checkov and Tfsec, respectively.

Since we are using different tools for conducting our study, it is important to evaluate them by exploring their supported policies (see Table 1). For that, we manually analyze the previously selected policies and get the intersection of policies supported by Checkov and Tfsec. This manual checking was done based on the official documentation provided by each tool for their supported checks (descriptions and implementation examples). The third researcher of this study conducted the initial mapping, while the first author further validated the mapping. A total of 121 policies are found to overlap between the two tools, with 76, 20, and 25 policies corresponding to AWS, Azure, and GCP, respectively. While this overlap demonstrates that the tools implement a common set of policies, it is also evident that each tool uniquely targets specific policies. This finding suggests that relying on a single tool may lead to incomplete policy coverage, as certain policies are enforced by only one tool. Thus, combining both tools is recommended to achieve more comprehensive policy enforcement across different cloud providers.

Regarding using LLMs to perform such a categorization, overall, we observe a moderate and high agreement for the categorization of built-in policies by Tfsec and Checkov (0.778 and 0.869, respectively), underscoring the reliability of our manual classification. Furthermore, it demonstrates the potential of LLMs to automate security policy categorization in future research.

Next, we discuss in detail the eight categories previously introduced.

Admin by Default The creation of action statements with all privileges is very rare. Most IAM, KMS key, and SQS policies specify which actions are allowed. Such results can be interpreted as a good implementation of the least privilege principle. The automation of infrastructure provisioning with IaC allows the creation of policies with the minimal required permissions automatically.

Hard-Coded Secrets This category and associated policy results highlight the presence of secrets in the Terraform source code. For example, access keys and passwords present in the Terraform source code and EKS (Elastic Kubernetes Service) Secrets Encryption, which enables containers to access secrets securely without needing to hard-code the secret in the container definition (e.g., the Dockerfile). GitHub now scans repositories for known types of secrets, preventing these types of misconfiguration flaws (GitHub 2022a).

IP Address Binding Most resources interacting with one another are properly configured to verify the least privilege principle. This applies to resource-specific policies as well as IP Address binding, allowing access from all sources (0.0.0.0:0 IP address). Public access is often disabled, and resource policies restrict non-required authorizations. The key-referencing feature of IaC enables practitioners to refer to other resources in the file code easily. Therefore, binding resources access is easily done with IaC leaving open-access authorizations useless and vulnerable.

Access Policy The *Access Policy* category refers to any misconfiguration of *Identity and Access Management Roles*, as well as *Access Control Lists*. Managing Access policy is an important part of cloud infrastructure security, as it defines which resources can access one another.

Encryption in Transit This category ensures all communications use secure protocols over Transport Layer Security (TLS). The communication has to be properly configured in the infrastructure configuration, which defines how resources communicate with one another. 63.1% of communication links used secure protocols, allowing for improvement.

Encryption at Rest Good data encryption helps prevent leaks by third parties and ensures data integrity. Cloud providers now offer fully managed server-side encryption at a low cost, which eases the use of such technologies. Only one-quarter of resting data resources are encrypted, representing a worrying amount of vulnerable storage. However, our study only considers Terraform configuration and can only check for server-side encryption. Resting data can always be encrypted locally before uploading them to the cloud.

Logging/Monitoring Policies in this category check if logs and detailed monitoring are enabled on the deployed resources. We find log use is quite low for AWS and Azure. Further investigation may clarify why practitioners do not enable and use logs when deploying and maintaining cloud infrastructures despite evidence of the usefulness of logs in system security. Ávila et al. (2021), and Vaarandi and Pihelgas (2014) show that even if the absence of logs does not represent a direct vulnerability to the infrastructure, logs can be very useful to collect metrics and detect security issues.

Outdated Feature Cloud providers often implement new service versions and security fixes automatically with no need to modify the infrastructure definition. However, major changes may not be backward compatible, so some cloud practitioners disable automatic minor updates or stay with vulnerable versions.

Our selection and mapping process exhibited relevant configuration snippets in Terraform files for AWS, Azure, and GCP cloud providers (287 and 180 policies, reported by Checkov and Tfsec, respectively). Those policies are all related to recognized cloud provider security industry standards specifications, supporting practitioners when adopting any of the previous providers. The policies have been mapped to 8 categories related to common security policies, such as *Hard-coded secret*, *IP Address binding*, *Admin by default*, *Encryption at rest*, *Encryption in transit*, *Access Policy*, *Logging/Monitoring* and *Outdated feature*. Table 2 shows the category distribution of the policies.

4.2 RQ2: How are Common Security best Practices Being Adopted in Terraform Files?

To answer this research question, we perform an empirical study evaluating the implementation of the previously selected security policies across different GitHub repositories. Before

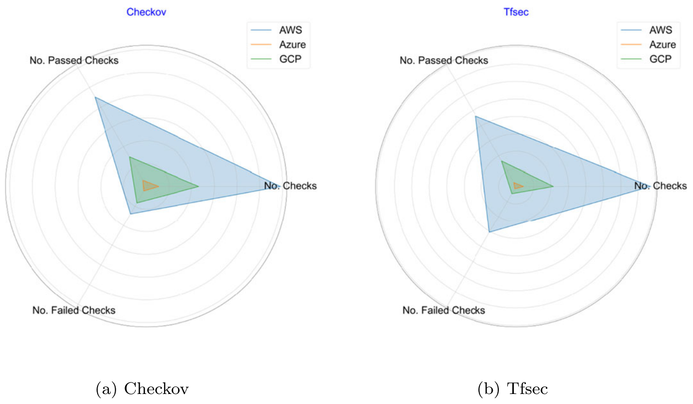


Fig. 4 Overall check results per provider for checkov and Tfsec

presenting the results for this research question, we discuss the overall data analyzed during our study.

4.2.1 Overall Results

Figure 4 presents an overview of the number of checks performed when checking the security policies performed during our empirical study (number of checks, passed, and failed checks). Overall, similar behavior of the tools across the different cloud providers can be observed, as Checkov performs more checking with success status. AWS is the cloud provider that performed more checks individually and grouped by the tools under analysis; 59,045 and 38,152 checks are performed exploring the 121 and 101 selected policies from Checkov and Tfsec, respectively.[11]

To assess the completeness of supported policies by the tools, we compute a weighted average based on the range of supported and detected policies. Regarding the overall detection rate, both tools exhibit high performance, with Checkov slightly outperforming Tfsec at 96% and 95%, respectively. This high level of coverage highlights the effectiveness and comprehensiveness of both tools in detecting relevant policies. When examining the detection rates for individual cloud providers, the tools demonstrate strong coverage. However, some notable differences emerge: Checkov performs particularly well for AWS, achieving a 96% detection rate, while Tfsec excels in Azure, with a detection rate of 93%. These results suggest that while both tools provide robust coverage across cloud platforms, they complement each other by offering superior detection for specific providers.

Regarding the detection rate of failing policy implementation, we observe the tools present similar rates, with a slight advantage for Checkov compared to Tfsec (87% and 85%, respectively). However, when looking for the individual cloud providers, we observe that Checkov performs better in AWS and GCP (91% for both), while Tfsec outperforms in the Azure cloud (79%). First, this suggests that each tool may be better suited for specific cloud environments, and the choice of tool could depend on the cloud provider being analyzed. Therefore, using both tools together can maximize detection coverage and mitigate any gaps in policy enforcement for particular cloud environments.

[11] A check here means evaluating a project's defined policy against Checkov's supported policies; each check passes if the target policy is well-implemented and fails if there is a policy violation.

Regarding the popularity of the projects, we observe a tendency for an increasing rate of passed checks with the number of stars. Manually checking some projects, we observed that tendency in practice. However, after further investigation into whether there is a correlation between these metrics, we did not observe a positive, strong, or significant correlation. Further investigation should be done to properly identify the factors correlated with successfully adopting security policies.

4.2.2 Commonly Adopted Security Policies

Overall, we observed some policy categories perform well in all providers, while other providers presented some particularities. Figure 5 presents the overall checking and failing rate for the evaluated policies for each of the evaluated cloud providers by the different SCA tools. For the AWS provider, we can observe that the *Admin by default* and *Hard-coded secret* categories performed very well according to both tools (pass rates higher than 80%, see Fig. 5a and b). For the remaining categories with more than 50% of pass rates, we observe some categories placing on different positions, but still consistent when evaluated across the different tools. For example, *IP Address Binding* places with the third-highest pass rate on Checkov, while it places in the fourth position based on the Tfsec analysis.

For Azure, we observe similar results to those of AWS. The *Hard-coded secret* category does not report any failure (see Fig. 5c and d), which contains one policy in both SCA tools. That category looks for credentials in Virtual Machines data, meaning that no sensitive credentials have been found in the 47 definitions of Azure Virtual Machines with Terraform in our dataset. Exploring the pass rates reported by Tfsec, we observe *Admin by Default* and *Encryption at Rest* with the high pass rate (100% and 94.4%, respectively); they are implemented by at most 2 policies, possibly impacting the reported rates (see Table 2). Then, the *Access policy* and *IP Address binding* categories performed similarly in both SCA tools. The *IP Address binding* only contains 4 policies showing good adoption results, notably all 47 scanned AKS cluster nodes not having public IP addresses.

For the GCP provider, some previous policy categories also present good passing rates. For example, we observe that the *Admin by default* category performs well in both tools, with an overall pass rate of 98.8% and 92.7%, respectively (see Fig. 5e and f). However, *Hard-coded Secret* category, which has only one single associated policy, is the category with the highest pass rate on Tfsec (99.6%). The second best performing category for Checkov with 72.4% pass rate is the *Access policy* category. This category contains the most policies in our GCP policy taxonomy. Overall, policies restricting access to services to unnecessary parties are well adopted in GCP Terraform files. Similarly, the *IP Address binding* category resulted in a 62.3% and 60.1% overall pass rate for Checkov and Tfsec, respectively.

Overall, *IP Address binding* category has very successful pass rates in all evaluated cloud providers. Such a result shows the current support for these categories by cloud providers is consistent, and practitioners are aware of and exploring them in their applications. AWS and Azure providers present similar results in other categories when compared to GCP; while they present good implementation of *Encryption in transit* and *Hard-coded secrets*, GCP shows relatively good adoption of *Logging/Monitoring* policies, even though the individual policy results show mixed results.

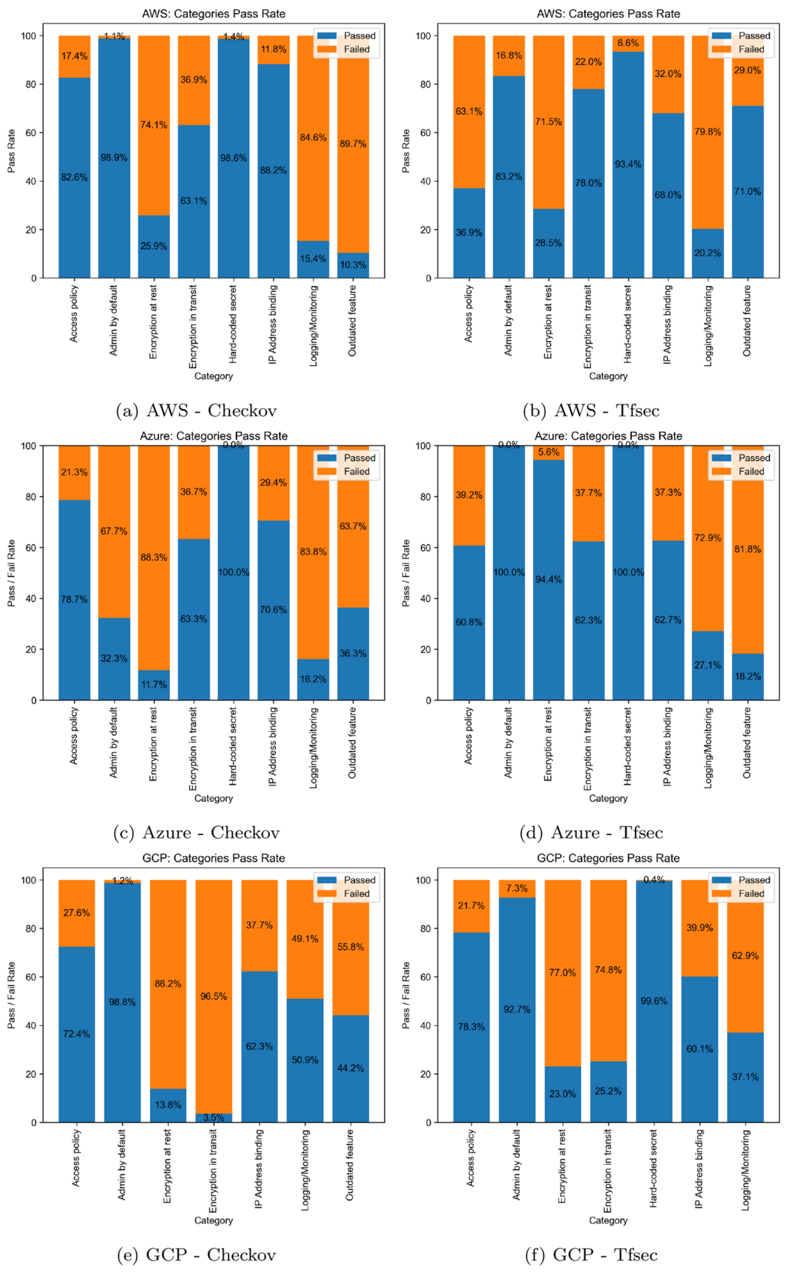


Fig. 5 Policy categories pass rates for checkov and Tfsec

4.2.3 Commonly Neglected Security Policies

We observe some general failing categories among all providers and further particularities. For the AWS provider, we observe that the *Encryption at rest* and *Logging/Monitoring* categories have the lowest pass rates across both evaluated tools. Next, for the *Outdated Feature* category, Tfsec presents a high pass rate (71%), while Checkov reports the lowest rate for a category in AWS (10.3%). Someone might argue that such a result might occur due to the low number of policies associated with this category on Tfsec; however, we observe that Checkov has fewer policies when compared with Tfsec (4 and 6, respectively).

For the Azure provider, we observe similar behavior. Overall, *Logging/Monitoring* and *Outdated Feature* categories present lower pass rates in both tools (see Fig. 5c and d). However, we observe different reports regarding *Admin by default* and *Encryption at Rest* categories; while Checkov reports lower pass rates, Tfsec reports higher ones. Like the previous discussion for AWS, Tfsec has fewer policies associated with these categories. Finally, for the GCP provider, we observe a general agreement regarding the categories with the lowest pass rates dealing with encryption (*Encryption at rest* and *Encryption in Transit*).

Overall, we observe Checkov and Tfsec present a general agreement for the policy categories evaluated here. Although they present some discrepancies regarding specific categories, most of these cases are motivated by the different number of policies associated with these categories. As a result, categories with fewer policies tend to present higher pass rates.

Overall, AWS and Azure cloud providers neglect the same categories, while GCP has its own particularities. While *Logging and monitoring* and *Outdated feature* categories are commonly neglected by AWS and Azure, GCP presents a good implementation of these policies. On the other hand, *Encryption in transit* is expressively neglected by GCP. These results show cloud service maintainers might provide further support to practitioners by guiding them during the implementation of important missing policies.

4.2.4 Differences Between Cloud Providers

In this section, we explore the differences between the three studied cloud providers, AWS, Azure, and Google Cloud, and reflect on the potential reasons for the previous observations.

4.2.5 Supported Services and Taxonomy Differences

AWS, Azure, and GCP provide cloud instances publicly, but the services they offer and their configuration highly differ (Saraswat and Tripathi 2020). As a result, the Terraform configurations and the related security policies may also differ. The built-in policies selected from Checkov and Tfsec are associated with security policies (guidelines) that can be leveraged through Terraform files (see Table 2). However, the services offered by the three cloud providers are different. For example, although AWS EC2 Instances are equivalent to Azure Virtual Machines, their usage, and configuration, as well as security specifications, are in high contrast. For each service, the freedom of configuration left to the user will result in different implementable security policies. Likewise, the default configurations will likely be different, leading to fewer security policies that have to be implemented by the customer. For instance, GCP applies most `Encryption at rest` and `Encryption in transit` best practices by default, without the need for practitioners to specify such configuration in Terraform (Google 2022a, b).

The policy distribution in categories itself reveals a few insights. Since encryption is hard-coded into GCP services, its Encryption categories are considerably smaller than the AWS and Azure taxonomies. On the other hand, GCP tends to have more policies related to *Logging and Monitoring* representing 28% and 24.4% of the taxonomy distribution on Checkov and Tfsec, respectively. The distribution of the AWS and Azure policies is more similar. The absence of *Hard-coded secrets* policies for GCP on Checkov and just one on Tfsec suggests such a category might be susceptible to experiencing security issues in IaC code. First, we consider that GCP does not require the usage of secrets in its code, preventing this type of vulnerability, as GCP already provides such support as a default. On the other hand, it is also reasonable that the SCA tool's maintainers might still not support the discussed policies (ongoing support). Finally, further analysis is required to understand the reasons for the previous observations.

To further investigate how the structure of Terraform impacts the occurrence of failures, we analyze some policies commonly implemented by all providers. Overall, we observe that different constructions on Terraform impact different cloud providers. For example, for policies requiring single keys for their implementation, like *encryption*, Azure reported a high occurrence of successful checks, while AWS and GCP reported two and three times more failures as compared to successful ones, respectively. For policies requiring advanced Terraform constructs, like *restricting public IP addresses*, Azure and AWS present twice the number of successful checks compared with failed ones, while GCP reports more failed checks. We understand that these results might be influenced by the number of projects analyzed in this study, requiring further studies investigating the impact of Terraform constructions.

4.2.6 Adoption of Security Policies: Comparison

Now, we focus on comparing the results of the security policies' checking on the Terraform components of the different cloud providers under two perspectives: number of checks and pass-checking rates.

Number of Checks Comparison Table 3 presents the reported metrics associated with the performed checks for the three datasets. As we can see, the reports of Checkov and Tfsec show that the GCP dataset executed more than ten (10) and three (3) times more checks per policy than the analysis of the Azure dataset, respectively. For all metrics showcased in Table 3, the Azure Terraform files produced fewer checks (67% and 72% fewer checks per repository, and 86% and 84% fewer checks per policy compared to AWS, for Checkov and Tfsec, respectively). The lower number of checks (per repository or policy) does not imply weaker security or practice adoption. Instead, it reflects how often the policy can be

Table 3 Overall check results per provider

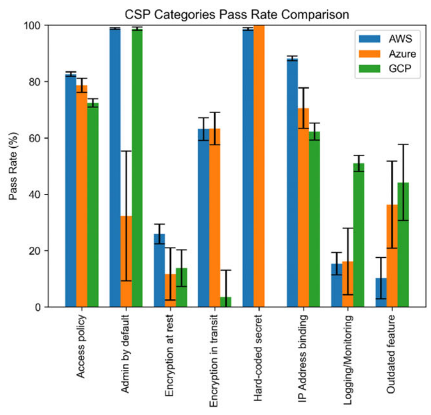
Metrics	Tools					
	Checkov			Tfsec		
	AWS	Azure	GCP	AWS	Azure	GCP
Median Checks/Policy	174.0	20.0	201.0	51.5	32.0	90.0
Mean Checks/Policy	504.66	68.16	282.04	409.70	62.09	243.27
Skewness Checks/Policy	0.40	0.44	0.24	0.23	0.40	0.32
Median Checks/Repo	56.0	17.5	22.0	34.0	8.5	21.0
Mean Checks/Repo	160.01	52.81	108.66	125.58	34.16	88.22
Skewness Checks/Repo	0.33	0.45	0.25	0.28	0.34	0.27
Median Pass Rate/Repo	71.43	50.0	57.14	49.42	50.0	52.08
Mean Pass Rate/Repo	69.11	51.90	55.92	44.28	53.21	59.10

implemented in the Terraform file, which is related to the number of cloud resources deployed and policies related to those resources.

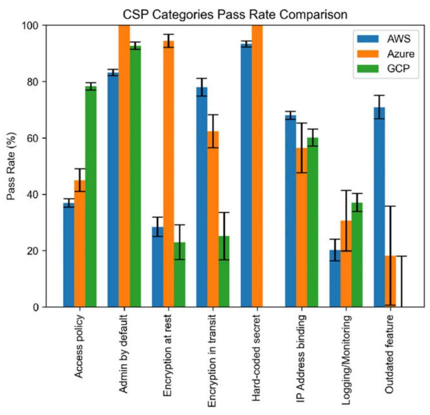
Regarding the distribution of checks per policy, we observe the distribution of checks is approximately symmetric based on the `Skewness Checks/Policy` for Checkov (6th row). Meanwhile, AWS and Azure providers report higher values when compared to GCP, leading us to conclude that the policies for these providers do not have the same weight in the results, as some only apply to specific resources rarely deployed. For Tfsec, we do not make the same observation as the reported values are consistently different. We also observe that the `median` and `average` pass rates per repository (10-11th rows) are close for all cloud providers, suggesting that the results are likely to be distributed among the dataset and not biased by a few exceptionally well or badly-performing projects.

While these findings do not give insights into adopting security policies, fewer checks could impact confidence in the results. This way, we consider a confidence interval related to the low number of checks into account when comparing the category's pass rate performances (a 95% confidence level, see Fig. 6). The GCP dataset results are more symmetrically distributed than the AWS and Azure datasets regarding repository check distribution. Finally, we remark that the median and average pass rate per repository is higher on the AWS dataset, which can be interpreted as higher adoption of security policies by AWS practitioners.

Categories Pass Rates Comparison To better understand the difference among cloud providers and their statistical impact, we added 95% confidence intervals to the categories' pass rate results (Fig. 6). Regarding the *Admin by Default* category, while the AWS taxonomy only contains 6 and 3 policies in this category (Checkov and Tfsec, respectively), that is the category responsible for producing one of the highest numbers of checks on the AWS dataset (9,914 and 14,552, for Checkov and Tfsec, respectively, see Table 4). On the other hand, the *Admin by default* category on Azure contains three policies, producing only 107 checks, combining the number of checks of Checkov and Tfsec. Considering the default configuration of the Azure service is secure, there is no need for the practitioners to implement the security policies in the Terraform configuration, leading to no existing related policies, which results in the analysis producing fewer checks.



(a) Checkov



(b) Tfsec

Fig. 6 Categories results comparison with 95% confidence intervals

Table 4 Overall number of checks across policy categories

Categories	Cloud Providers					
	AWS		Azure		GCP	
	Checkov	Tfsec	Checkov	Tfsec	Checkov	Tfsec
Access Policy	20,570	7,753	2,569	612	9,594	3,593
Admin by Default	9,914	14,552	96	11	3,275	4,625
Encryption at Rest	4,575	4,696	768	36	1,522	283
Encryption in Transit	1,716	499	824	430	792	159
Hard-coded Secret	4,143	423	47	140	-	273
IP Address Binding	12,766	6,216	428	702	3,256	813
Logging/Monitoring	4,107	2,428	452	96	4,457	958
Outdated Feature	1,254	1,945	201	22	231	-
Total	59,045	38,512	5,385	2,049	23,127	10,704

The categories *Encryption in transit*, *Hard-coded secrets*, and *Logging/Monitoring* have very close pass rates between AWS and Azure in both SCA tools. Figure 6a shed light on the *Admin by default* category, which shows a very high adoption on AWS and GCP, found on Azure with a difference of 66 percentage points based on Checkov's report. Although we observe the same behavior for AWS and GCP on Tfsec's report, Azure presents the highest pass rate. However, it is important to mention that only one check is available for that category in Tfsec, resulting in 11 checks that were all successful. Apart from that, such a finding suggests managing privileges on Azure to be significantly less convenient for Terraform practitioners, leading to neglect in this field. Likewise, adopting *Encryption in transit* policies highly differs between GCP and AWS/Azure. The latter has already been explained by the use of encryption by default on GCP services (Google 2022a, b). Finally, we remark that adopting *Logging/Monitoring policies* is significantly higher on GCP, with a similar number of checks in this category in AWS. Investigating how log adoption differs from the three cloud providers could reveal some interesting factors fostering adoption.

Overall, the three evaluated cloud providers are different under different perspectives. Regarding implementing security policies supported by the cloud providers, we observe a different distribution by the evaluated projects. The results also highlight the impact that default configurations of cloud providers have on the adoption of security policies by practitioners, like *encryption by default* on GCP.

4.2.7 Differences between Checkov and Tfsec

In this Section, we compare the SCA tools aiming to gain a deeper understanding of their functionality and identify areas for improvement. As previously reported in Section 4.1, we identify an intersection of 121 policies supported by Checkov and Tfsec. Associated with these policies, we observe 3,026 checks performed by both tools, which 326 of them (10%) report conflicting evaluations. At this point, it is important to note that the data collected for this analysis represents a specific context. Specifically, Checkov reported passed checks for 155 cases, while Tfsec reported them as failed ones; on the other hand, Tfsec reported 171 passed checks, classified as failed checks by Checkov. Since we only considered the checks performed by both tools on the same policies, and ultimately, a small portion of these checks

Table 5 Accuracy, related metrics and comparison between checkov and Tfsec

Metrics	Tools	
	Checkov	Tfsec
Precision	0.97	0.26
Recall	0.73	0.35
F1 Score	0.84	0.30
Accuracy	0.81	0.18
Units	97	97

presented different results, we may conclude that the tools produced similar reports for most cases, indicating a general consistency between them.

Moving forward with our analysis, based on the 326 cases of conflicting reports, we randomly select a subsample of 97 cases. For that, we select 48 and 49 cases reported with passed checks by Checkov and Tfsec, respectively, aiming for a 90% confidence level and a 10% margin of error. Since the selected cases conflict with each other regarding the tools' report, a given passed check in one tool is already known as failed by the other, and vice-versa. In case we consider the passed check reported by a given tool A is correct, we consider that case as a *true positive* for tool A, and we automatically consider it as a *false negative* for tool B. In the same way, if the failed check reported by a given tool A is wrong, we consider that case as a *false positive* for tool A, and we automatically consider it as a *true negative* for tool B. Then, we compute *precision, recall, accuracy* and *F1 Score*. Table 5 presents the final metrics computed for both evaluated tools.

We observe better results for the Checkov when compared to Tfsec. For some cases, Checkov presents better results since the tool provides a broader and encompassing strategy to evaluate policy implementation, while Tfsec is restricted to unique strategies. For example, in the Listing 3, we have an attempt to protect the current traffic by adopting the HTTPS protocol.[12] For that, the developer must explicitly assign `HTTPS` to the `protocol` key, which supports encrypted connection. Although the first defined resource adopts protected traffic, there is a chance that the second resource adopts the general `HTTP` protocol, while the third one surely does it. In this case, Checkov correctly reports a failure when implementing the policy, while Tfsec reports the policy was well implemented. In other cases, although the tools adopt similar ways to check the same policy, Tfsec fails to report the correct evaluation. For example, when checking for *ensuring that S3 bucket has block public policy enabled*, both tools check for the value associated with the key `block_public_policy`. [13] However, Tfsec erroneously reports an error associated with the user's implementation, although the tool presents a similar code to illustrate a secure implementation as part of its documentation. [14] Based on the previously discussed issues, we may conclude that Tfsec struggles in specific policies, resulting in higher numbers of false positives, and consequently impacting its accuracy and F1 score. Although both tools report false positives impacting their precision, the high number of false negatives reported by Tfsec directly impacts its recall and accuracy.

Further investigating the conflicting evaluations, we observe that Azure and GCP reported fewer conflicting evaluations of policies when compared to AWS (8.17%, 10.71%, and 11.29%, respectively). However, we must consider that Azure is supported by fewer policies when compared with the other providers (see Section 4.1). Checking the categories associated

[12] Link to commit

[13] Link to target file

[14] Tfsec: S3 Access block should block public policy

with these policies, the *Access Policy* category was consistently reported with conflicting evaluations for all cloud providers. Further investigating these policies, we observed that Tfsec faced the same issues previously reported across the evaluated cloud providers.

```
resource "aws_alb_listener" "https" { {...}
  protocol           = "HTTPS"
}

resource "aws_alb_listener" "this" { {...}
  protocol           = each.key == "443" ? "HTTPS" : "HTTP"
}

resource "aws_alb_listener" "http" { {...}
  protocol           = "HTTP"
}
```

Listing 3 Correct Policy Implementation on Terraform

5 Discussion

In this section, we reflect on the previous results to suggest new guidelines for cloud practitioners and providers. Overall, we provide evidence of the adoption of security best practices across major cloud providers by analyzing a diverse range of GitHub projects. Regarding the adoption of these practices, we observe high and consistent use of key security measures such as *IP Address Binding*, *Hard-coded Secrets*, *Admin by Default*, and *Access Policies*, regardless of the differences in services offered by each cloud provider. This finding suggests that cloud providers offer strong support for implementing these security practices, and practitioners are generally well aware of their importance. However, we also identified some neglected security practices, including *Logging/Monitoring*, *Outdated Feature* and *Encryption in Transit*, though this lack of implementation was not consistent across all providers. These gaps highlight the challenges cloud providers face in promoting the adoption of certain security measures, which may be influenced by provider-specific configurations and industry regulations that shape security priorities and enforcement.

Additionally, our evaluation of two static code analysis (SCA) tools demonstrated a high coverage of policy implementation across both. While there was an overall agreement between the tools for most policies under investigation, we noted that Checkov stands out for its capability to handle provider-specific configurations and industry regulations. This flexibility makes Checkov a valuable tool for dynamic and evolving cloud environments, where practitioners have more freedom to implement policies in a customizable manner. A detailed discussion about the generalizability and associated threats to validity is presented in Section 6.

5.1 Security Policies of Cloud Provider

Cloud providers may offer the same services, but the support is packaged in different products. Therefore, the configuration power left to the practitioner differs from one provider to another. Section 4 compared adopting best practices among cloud providers. Although the adoption results match in certain categories, we observe some differences for other categories. For example, differences in the way permissions are defined (*IAM Policies* on AWS vs. Direct Links on Azure) and how encryption is enabled (by default on GCP) might impact how good practices are implemented through Terraform.

By investigating the policies' definitions, we can pinpoint potential reasons for the results observed in Section 4.

- Cloud providers' default configurations seem to highly impact the adoption/use of best practices. If cloud default configurations change over time, it would be possible to measure its actual impact.
- The number of lines of code needed to implement the practice seems to be related to its adoption. Practitioners might be more inclined to implement practices requiring only a single line of code rather than a whole new Terraform block.
- The key referencing feature of IaC makes easier the implementation of the least privilege principle, especially in both the *Access Policies* and *IP Address Binding* categories.

Extended research on the actual impact of those observations in adopting the practices could lead to more insights into the different features of IaC. As a disclaimer, our study aims to compare practitioners' adoption of security policies through Terraform configurations. Therefore, the study does not compare the actual security of the cloud infrastructure between cloud providers as this is a more complex topic to address, closely related to the cloud services design themselves. Moreover, cloud providers constantly work on improving the security of their products. For instance, AWS recently changed the configuration of S3 buckets to implement some security best practices by default (AWS 2023). Since the findings of this study could be impacted by future changes in cloud services designs, further future studies might be done to evaluate the current status of security policy implementation over time.

5.2 Categories Policy Synthesis

Analyzing the policies that address the security best practice standards and their adoption through the Static Code Analysis of Terraform files of real-world projects reveals key principles when designing secure infrastructures. To provide guidelines to cloud practitioners, In Table 6, we synthesize the policies contained in each category considering the prevalence of failing checks and further extend them to the potential benefits and impact of implementing them.

5.3 Policy Set Completeness

In Section 4.1, we carefully identified which security guidelines have matching Terraform implementation policies that can be directly mapped to recognized industry standards. Even though the process aims to build a qualitative set of recognized security policies to ensure the relevance of the scanning results, not all cloud security policies might have been identified and selected. Therefore, our policy set could be limited in two ways, which we discuss now.

5.3.1 Unsupported Security Guidelines on Terraform

Some security guidelines specified in cloud regulations can not be implemented through Terraform for several reasons. For example, guidelines related to account configuration rather than the infrastructure definition itself or those for which the interface provided by Terraform does not support the required feature. The CIS standards specify whether one guideline can be automatically implemented or if it requires manual action. One such policy is ***1.21 Ensure IAM users are managed centrally via identity federation or AWS Organizations for multi-account environments***. This specification deals with multi-account environments,

Table 6 Practical guidelines for secure terraform configurations

Security Policy Categories	Practical Guidelines
Admin by Default	Avoid wildcard permission preventing unauthorized access
	Apply the least privilege principle via the key-referencing
Encryption in Transit	Enable HTTPS and HTTP redirects to HTTPS to ensure that all communication is secure and encrypted
	Use a secure communication protocols (SSL and TLS) further enhances security by providing a secure channel for data transfer and authentication
Encryption at Rest	Encrypt all data at rest even when encryption is implemented in another layer of the sistem
Access Policy	Limit public access to back-end resources is crucial in preventing unauthorized access and potential breaches
	Use restrictive policies and groups to implement the least privilege principle
Logging / Monitoring	Enable and use logs to prevent and detect security threats
IP Address Binding	Avoid ingress traffic from 0.0.0.0/0 and assigning public IPs helps to limit attacks and reduce unauthorized accesses
	Ensure only necessary traffic is allowed by using key-referencing in IaC
Hard-Coded Secrets	Use secret manager services to ensure sensitive information are securely stored and not committed to source code
	Use secrets scanning tools before uploading IaC files to identify and remove any potential secrets in the code
Outdated Feature	Enable automatic upgrades to ensure the infrastructure is always up to date with the latest security patches

where different users can access the same environment or changes of access through role assumption. This way, centralizing the IAM users facilitates user control while reducing complexity and possibly avoiding unauthorized access.

5.3.2 Unsupported Security Policies in Regulations and/or Industry Standards

On the other hand, practitioners can implement security policies that are not contained in industry standards and/or cloud regulations. Investigating such practices can generate interesting findings to understand why practitioners implement security policies that do not come from compliance requirements.

For example, the set of built-in policies in Checkov contains the following policy: ***Ensure that auto Scaling groups that are associated with a load balancer, are using Elastic Load Balancing health checks***. This policy could not be mapped to either one of the two industry standards, suggesting that no cloud regulations require the adoption of such practice. However, using health checks ensures that the instance is up and ready to perform its task before receiving traffic from a load balancer. This can be seen as a security practice, ensuring that no unprepared (and potentially vulnerable) instance receives external requests.

As we saw through those two examples, the set of policies extracted from industry standards used to scan the IaC components of our datasets might not contain all relevant security policies. Investigating the gap between standards and actual practices could lead to interesting findings. Practices not contained in standards but commonly implemented by practitioners could suggest new revisions of cloud regulations to implement more recent and secure practices. Likewise, studying the security policies that can not be implemented through Terraform could lead to qualitative findings on the practitioners' security habits that can not be empirically measured with code artifacts. Such paths could lead to insightful future research in the field.

5.3.3 Security Vulnerability Detection based on Different SCA Tools

When investigating the reported discrepancies between the analyzed SCA tools in this study, we observe some challenges regarding detecting security vulnerabilities. First, as discussed in Section 4.2.7, the tools might have different understandings about how the security policies should be implemented. Our findings reveal that combining these different tools is the best approach, considering their coverage and accuracy. We believe that one should not rely solely on the results of one single tool. Combining the strengths of both tools would lead to better recommendations. By understanding the contexts in which each tool excels or falters, we can offer more reliable insights and suggest potential improvements for future tool development. Furthermore, it is also possible that a given policy implemented by a practitioner is correct, but both tools provide wrong reports for different reasons. First, if the tool reports the security implementation is wrong, such a report could be caused by the wrong understanding and implementation of how the policy should be implemented. Second, it is also possible the tool erroneously reports the policy is well implemented, which could occur due to limited ways to check the policy (as we previously reported by Tfsec). Finally, further investigation is required to better understand the reasons for such conflicts.

6 Threats to Validity

Our empirical analyses and evaluations naturally leave open a set of potential threats to validity, which we explain in this section.

Construct Validity We acknowledge the occurrence of potential methodological threats. For establishing our sample of GitHub projects, we used cloud provider definitions to Terraform file code snippets, although IaC components form a small percentage of a project's total size. For that, we had to look for cloud provider imports on Terraform files. We know there are more common ways of collecting GitHub repositories based on projects' titles, descriptions, associated programming languages, or topics. However, these approaches would not have worked for us, as general GitHub Search does not track content code file information.

In the same way, due to GitHub API limitations, the data collected is linked to the data collection periods. To mitigate this risk, we adopted rounds of data collection periods that lasted around a month for each cloud provider. By the end, we observed that after each new round, the number of new projects added to the datasets was becoming low, showing that most active projects with Terraform files may have been collected for each cloud provider. On top of that, even though the collection period lasted around one month for all three datasets, the collections have not been executed simultaneously, as we adopt an interactive and incremental approach. Such a decision could possibly impact the ability to properly compare the results between providers. Aiming to mitigate this risk, we reduced the overall extent of the collection period to less than 6 months. Regarding the popularity filter adopted when calling the API, we understand that popularity might not be a relevant metric to filter GitHub repositories. This filter helped reduce the number of repositories to analyze manually. The popularity threshold has been set quite low ($>$ 1 star) to mitigate the risk of bias in the dataset.

The mapping of policies into their associated categories was a manual process conducted by one researcher. Eventually, some misjudgments may have happened, considering this process was based on one single author's judgment. To address this threat, external consultants were individually asked to evaluate the proposed mapping. Although we have reported a high agreement among the consultants regarding the mapping, we acknowledge that providing the mapping could have influenced the consultants' analyses to some extent. However, given the self-explainable nature of the policies and the distinctiveness of the categories, we believe new mapping attempts to be straightforward, resulting in no or small differences from our current proposed mapping. The comparison between the SCA tools was also a manual analysis conducted by one researcher and further validated by a second one. However, such a manual analysis was straightforward by comparing the reports against the tools' implementations.

Internal Validity Knowing that projects might consider multiple Terraform files to define their resources, we call the SCA tools to analyze all possible `.tf` files available in a given project. By doing that, it is possible that one check can pass in one file, but it could also fail in another one. As a result, when the tools report a *passed* or *failed* check, such a report is related to a specific check performed on a given list of resources. A possible way to address this threat would be combining all tf files into one and then calling the SCA tools.

Different security policies can be leveraged at different levels. For example, weak server-side encryption can be mitigated by users who encrypt their data locally before sending it to the cloud. As a result, it is expected that encryption-related policies represent, at most, a lower bound of these policies' implementation. As with any scanning software, the results are likely to contain false negatives and positives. Due to the popularity of Checkov and Tfsec, these tools are continuously updated to address the reported issues. Furthermore, based on our analysis, we compare the tools under analysis aiming to understand the reasons behind these false negatives and positives. Ultimately, we observe that the tools report different results in a small number of cases (10%).

External Validity We acknowledge other external considerations that might represent a threat to validity. GitHub, the source of our dataset, hosts publicly accessible, often open-source repositories and, therefore, might not fully represent the industry context in which projects are often closed-sourced. Since security may not be a priority for some projects, we addressed this threat by manually removing projects declared purposefully insecure. However, for the remaining projects, we can not guarantee that practitioners always aim to maximize security. Still, regarding our sample, although we have selected repositories with different sizes, programming languages, and domains, our results might not reflect the reality of projects out of these properties.

Our results are tool-based; running new studies with other tools might present different findings. However, to mitigate this threat, we considered two tools when conducting our study. Regarding the applicability of our new proposed categories of security risks, we believe future studies could benefit from them. If running studies with other tools, new mappings among the policies supported by the tools and the Industry Standards are required.

In the same way, it is possible that we miss policies in our study commonly adopted by the community. Although the open-source nature of SCA tools mitigates the risk of omission as contributors may regularly add new policies, we would still miss these policies considering they are not present in these standard guidelines yet. Likewise, our reliance on only four industry standards is mitigated by their broad acceptance (Stultiens 2020). Finally, we categorize the investigated policies by importing categories from previous work (Rahman et al. 2019c) and also proposing new ones.

Although we have adopted and selected standard industry-recognized cloud security and privacy practices, we may not have identified all of them at the infrastructure level. We selected only Terraform secure configuration checks implemented by Checkov and Tfsec policies that map to a specification of one of the evaluated Industry Standards. This way, it is expected that other security best practices might exist. To address this threat, we consider recent versions of these standard guidelines that might reflect the current status of these policies' adoption/usage by the community.

7 Related Work

In this Section, we summarize and discuss some related work.

7.1 Infrastructure as Code & Terraform

Few empirical studies have investigated IaC code maintainability. van der Bent et al. (2018) and Schwarz et al. (2018) have examined *Puppet* and *Chef* configurations. A systematic mapping study by Rahman et al. (2019a) highlighted the need for research into quality issues such as security flaws in IaC scripts. In further studies, the authors have also worked on characterizing configuration defects (Rahman and Williams 2018), identifying seven common security flaws in Puppet scripts through qualitative analysis (Rahman et al. 2019c). However, those studies focus on system configurations and installations and are not specific to the infrastructure layer. Ntentos et al. (2023) take a different route by investigating the occurrence of architecture smells in IaC-based deployments. Based on a set of best practices, the authors present a method to detect smells and propose associated fixes. Begoug et al. (2024b) investigate the prediction of defects in IaC code. For that, the authors explore different ML algorithms.

The closest work to our study has been done by Iosif et al. (2022). As we do here, the authors performed an empirical study investigating security vulnerabilities in 8256 GitHub repositories of Terraform files but only deploying AWS infrastructures. Next, they used three scanning tools to check for security vulnerabilities. Due to their collection method, their dataset only contains repositories where HCL (Terraform language) is the most used language of the project (e.g., $> 50\%$ of the project size), As a result, they focused on malpractices based on the tools' full lists of built-in policies without investigating the industrial relevance of the policies.

Unlike the previous related work, in this study, we focus on a smaller dataset of 812 GitHub repositories from three different cloud providers. Moreover, we identify and categorize industry-recognized security best practices while investigating the adopted and neglected practices. Therefore, despite the growing prevalence of projects containing Terraform configuration files on hosting platforms like GitHub, we are unaware of empirical studies besides our own investigating the adoption of security practices (good or bad) at the infrastructure level in *real-world* projects.

7.2 Cloud Security & AWS Security Best Practices Standards

To help practitioners comply with cloud infrastructure security and privacy regulations, providers and independent organizations have created sets of best practices and security benchmarks. Some form the basis of security certifications, of both cloud deployment audits and practitioners' knowledge. These help build customer trust and have become industry standards for secure cloud infrastructure (Ryoo et al. 2014; Choo et al. 2017). The two main standards for AWS are the Center for Internet Security (CIS) AWS Foundations Benchmark (Center for Internet Security 2021) and the AWS Foundational Security Best Practices standard (Amazon Web Services 2022). In the same way, the CIS also developed security best practices standards on Azure (Center for Internet Security 2023a) and Google Cloud (Center for Internet Security 2023b). In this study, we used the proposed standard practices to check their implementation on a sample of repositories. Although our goal is not to evaluate the applicability and coverage of these practices, we discuss possible fields that could be added to these benchmarks.

7.3 IaC Static Code Analysis (SCA) Tools

Guerriero et al. (2019) investigate the usage of IaC by practitioners, the current support for IaC, and the current needs of the community. For that, the authors performed 44 semistructured interviews with practitioners from different roles and companies. As a result, they highlight the difficulty of testing and maintaining IaC code. However, with the rise of IaC, several code-scanning tools have been released to help practitioners and automate tests in integration pipelines (Tfsec 2023; Terrascan 2023; Semgrep 2023; Bridgecrew 2022). These tools provide built-in misconfiguration catalogs and can be implemented in delivery pipelines. Among those tools, *Checkov* provides the larger supported compatibility with other IaC tools and cloud providers.

Different from the previous studies, Begoug et al. (2024a) present TerraMetrics, a tool for checking the quality of Terraform artifacts. For that, the authors parse the Terraform code, and based on the associated AST, they explore a set of forty structural metrics. Although this study focuses on security smells, further studies exploring TerraMetrics could provide insights into structural factors that influence security smells. Saavedra and Ferreira (2022) present GLITCH, a smell detector tool supporting different IaC languages, later extended to Terraform as well (Saavedra et al. 2023). Opdebeeck et al. (2023) go beyond by presenting GASEL and comparing it with SOTA tools. Such a tool explores the adoption of control and data flow for detecting security smells in the Ansible IaC language. Extending GASEL for Terraform language and further comparing it with SOTA tools is an open venue for research.

No prior work focuses on the infrastructure provisioning and management stage of IaC in real-world projects. In our evaluated sample of projects, the IaC component co-exists with other types of code, which excludes Terraform templates and tutorials not deploying actual applications. Closely related to the cloud provider platform and the practitioners building cloud architectures, this layer often involves Terraform configuration files. SCA tools compatible with Terraform could be used to build relevant empirical studies.

8 Conclusions

Infrastructure as Code (IaC) provisioning tools are useful but do not automatically preclude misconfiguration and security risks. This work analyzes security policies for Terraform from different cloud providers (AWS, Azure, and GCP), categorizing them into eight groups and empirically investigating the prevalence of their use in 812 open-source GitHub repositories through SCA tools. As a result, we found that some security policy categories, such as *Access policy* and *IP Address binding*, were consistently applied by practitioners in all evaluated cloud providers. We also observe that AWS and Azure providers present similar results in other categories compared to GCP. For example, they present good implementation of

Encryption in transit and *Hard-coded secret*. Regarding the neglected categories, we observe that *Encryption at rest* is the lowest implemented category. Once again, we observe AWS and Azure cloud providers neglect some same categories while GCP has its own particularities. For example, AWS and Azure neglect *Logging/Monitoring* and *Outdated feature*, while GCP neglects *Encryption in transit* and *Hard-coded secret*. Based on these findings, we provide guidelines that cloud practitioners could adopt to enhance the security of their IaC code.

Our findings indicate that IaC configuration poses a major risk and confirm that, despite the availability of security scanning tools, there is a lack of adoption of best practices in open-source projects. More effort is required to raise awareness about the significance of applying IaC best practices. In the same way, we believe these practices would become reachable by making IaC security scanning tools more widely available and accessible to developers as part of frameworks for continuous delivery and deployment. In addition, efforts should be made to give education and resources on how to use these technologies successfully and to encourage cloud application developers to prioritize security in their deployment processes.

In conclusion, IaC tools are powerful instruments benefiting from the upsides of writing code like collaboration, code analysis, or pipeline integration. Building secure cloud infrastructures can be challenging for practitioners. The adoption of security policies in Terraform components still varies according to the best practice category across different cloud providers. We hope this study might support cloud practitioners and providers in improving towards more secure infrastructures through valuable insights into SOTA practices. As future work, we highlight investigating how developers deal with known and reported vulnerabilities in security policies, and how to automate the mapping process between security policies from Industry Standards and built-in policies supported by SCA tools.

Acknowledgements This work is partly funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), Fonds de recherche du Québec (FRQ), and the Canadian Institute for Advanced Research (CIFAR).

Data Availability To promote open science and facilitate reproducibility, we make all our artifacts available to the community. This includes the datasets used in our experimentations, the source code for the scripts that were used, the results produced, and any other artifacts related to our study: https://github.com/averdet/master-terraform-sec-adoption.

Declarations

Conflicts of Interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Amazon Web Services (2022) Aws foundational security best practices standard. Accessed 12-November-2022

Amazon Web Services (2023) Aws cloudformation. https://aws.amazon.com/fr/cloudformation/. Accessed 11-June-2023

Artac M, Borovssak T, Di Nitto E, Guerriero M, Tamburri DA (2017) Devops: introducing infrastructure-as-code. In: 2017 IEEE/ACM 39th international conference on software engineering companion (ICSE-C). IEEE, pp 497–498

Ávila R, Khoury R, Khoury R, Petrillo F (2021) Use of security logs for data leak detection: a systematic literature review. Secur Commun Netw 2021:612–622. https://doi.org/10.1155/2021/6615899. ISSN 1939-0114

AWS (2023) Amazon s3 beginning to apply two security best practices to all new buckets by default. https://aws.amazon.com/fr/about-aws/whats-new/2023/04/amazon-s3-two-security-best-practices-buckets-default/. Accessed 13-April-2023

Begoug M, Chouchen M, Ouni A (2024a) Terrametrics: An open source tool for infrastructure-as-code (IaC) quality metrics in terraform. In: Proceedings of the 32nd IEEE/ACM international conference on program comprehension pp 450–454

Begoug M, Chouchen M, Ouni A, Abdullah Alomar E, Mkaouer MW (2024b) Fine-grained just-in-time defect prediction at the block level in infrastructure-as-code (IaC). In: 2024 IEEE/ACM 21st international conference on mining software repositories (MSR). IEEE, pp 100–112

Bridgecrew. Prevent cloud misconfigurations and find vulnerabilities during build-time in infrastructure as code, container images and open source packages with checkov by bridgecrew., 2022. URL https://github.com/bridgecrewio/checkov. Accessed 12 November 2022

California State Assembly (2018) California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa

Center for Internet Security (2021) Cis amazon web services benchmarks version 1.4.0. https://www.cisecurity.org/benchmark/amazon_web_services. Accessed 12-November-2022

Center for Internet Security (2023a) Cis microsoft azure foundations benchmark v2.0.0. https://www.cisecurity.org/benchmark/azure. Accessed 23-March-2022

Center for Internet Security (2023b) Cis google cloud platform foundation benchmark v1.3.0. https://www.cisecurity.org/benchmark/google_cloud_computing_platform. Accessed 30-April-2022

Choo KK, Rana OF, Rajarajan M (2017) Cloud security engineering: theory, practice and future research. IEEE Trans Cloud Comput 5(3):372–374. https://doi.org/10.1109/TCC.2016.2582278

Cohen J (1960) A coefficient of agreement for nominal scales. In: Educational and psychological measurement, vol 20, no 1, pp 37-46. Sage Publications

Das A, Uddin G, Ruhe G (2022) An empirical study of blockchain repositories in github. In: Proceedings of the international conference on evaluation and assessment in software engineering 2022, EASE '22, pp 211-220, New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/3530019.3530041. ISBN 9781450396134

Ebert C, Gallardo G, Hernantes J, Serrano N (2016) Devops. IEEE Softw 33(3):94–100

European Commission (2016) General data protection regulation (GDPR). https://gdpr-info.eu/

GitHub (2022a) About secret scanning. https://docs.github.com/en/code-security/secret-scanning/about-secret-scanning. Accessed 12-November-2022

GitHub (2022b) Search code. https://docs.github.com/fr/rest/search?apiVersion=2022-11-28#search-code. Accessed 13-March-2023

Gonzalez D, Zimmermann T, Nagappan N (2020) The state of the ml-universe: 10 years of artificial intelligence & machine learning software development on github. In: 2020 Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20. Association for Computing Machinery, pp 431-442. https://doi.org/10.1145/3379597.3387473

Google (2022a) Default encryption at rest. https://cloud.google.com/docs/security/encryption/default-encryption. Accessed 5-May-2023

Google (2022b) Encryption in transit. https://cloud.google.com/docs/security/encryption-in-transit. Accessed 5-May-2023

Gousios G (2013) The ghtorent dataset and tool suite. In: 2013 10th Working conference on mining software repositories (MSR), pp 233–236. https://doi.org/10.1109/MSR.2013.6624034

Gousios G, Spinellis D (2017) Mining software engineering data from github. In: 2017 IEEE/ACM 39th international conference on software engineering companion (ICSE-C), pp 501–502. https://doi.org/10.1109/ICSE-C.2017.164

Guerriero M, Garriga M, Tamburri DA, 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. https://doi.org/10.1109/ICSME.2019.00092

HashiCorp (2022) Terraform. https://www.terraform.io/. Accessed 12-November-2022

Humble J, Farley D (2010) Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education

Jiang Y, Adams B (2015) Co-evolution of infrastructure and source code - an empirical study. In: 2015 IEEE/ACM 12th working conference on mining software repositories, pp 45–55. https://doi.org/10.1109/MSR.2015.12

Juve G, Deelman E (2011) Automating application deployment in infrastructure clouds. In: 2011 IEEE third international conference on cloud computing technology and science, pp 658–665. https://doi.org/10.1109/CloudCom.2011.102

Kemp R (2018) Legal aspects of cloud security. Comput Law Secur Rev 34(4):928–932. https://doi.org/10.1016/j.clsr.2018.06.001https://www.sciencedirect.com/science/article/pii/S0267364918302358. ISSN 0267-3649

Lin F, Kim DJ et al (2024) When LLM-based Code Generation Meets the Software Development Process. arXiv:2403.15852

Microsoft Azure (2023) Azure resource manager. https://azure.microsoft.com/en-us/get-started/azure-portal/resource-manager/. Accessed 11-June-2023

Morris K (2020) Infrastructure as code. O'Reilly Media

Nayak R (2019) When to use which infrastructure-as-code tool. https://medium.com/cloudnativeinfra/when-to-use-which-infrastructure-as-code-tool-665af289fbde. Accessed 12-November-2022

Ntentos E, Zdun U, Falazi G, Breitenbücher U, Leymann F (2023) Detecting and resolving coupling-related infrastructure as code based architecture smells in microservice deployments. In: 2023 IEEE 16th international conference on cloud computing (CLOUD). IEEE, pp 201–211

Office of the Privacy Commissioner of Canada (2000) The personal information protection and electronic documents act (PIPEDA). https://www.priv.gc.ca/en/privacy-topics/privacy-laws-in-canada/the-personal-information-protection-and-electronic-documents-act-pipeda/

Opdebeeck R, Zerouali A, De Roover C (2023) Control and data flow in security smell detection for infrastructure as code: Is it worth the effort?. In: 2023 IEEE/ACM 20th international conference on mining software repositories (MSR). IEEE, pp 534–545

osif AC, Gasiba TE, Zhao T, Lechner U, Pinto-Albuquerque M (2022) A large-scale study on the security vulnerabilities of cloud deployments. In: Ubiquitous security: first international conference, UbiSec 2021, Guangzhou, China, December 28–31, 2021, Revised Selected Papers. Springer, pp 171–188. https://doi.org/10.1007/978-981-19-0468-4_13

Rahman A, Mahdavi-Hezaveh R, Williams L (2019) A systematic mapping study of infrastructure as code research. Inf Softw Technol 108:65–77

Rahman A, Mahdavi-Hezaveh R, Williams L (2019) A systematic mapping study of infrastructure as code research. Inf Softw Technol 108:65–77. https://doi.org/10.1016/j.infsof.2018.12.004https://www.sciencedirect.com/science/article/pii/S0950584918302507. ISSN 0950-5849

Rahman A, Parnin C, Williams L (2019c) The seven sins: Security smells in infrastructure as code scripts. In: 2019 IEEE/ACM 41st international conference on software engineering (ICSE), pp 164–175. https://doi.org/10.1109/ICSE.2019.00033

Rahman A, Williams L (2018) Characterizing defective configuration scripts used for continuous deployment. In: 2018 IEEE 11th international conference on software testing, verification and validation (ICST), pp 34–45. https://doi.org/10.1109/ICST.2018.00014

Richter F (2022) Infographic: Amazon leads $200-billion cloud market. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. Accessed 12-November-2022

Ryoo J, Rizvi S, Aiken W, Kissell J (2014) Cloud security auditing: challenges and emerging approaches. IEEE Secur Privacy 12(6):68–74. https://doi.org/10.1109/MSP.2013.132

Saavedra N, Ferreira JF (2022) Glitch: automated polyglot security smell detection in infrastructure as code. In: Proceedings of the 37th IEEE/ACM international conference on automated software engineering, pp 1–12

Saavedra N, Gonçalves J, Henriques M, Ferreira JF, Mendes A (2023) Polyglot code smell detection for infrastructure as code with glitch. In: 2023 38th IEEE/ACM international conference on automated software engineering (ASE). IEEE, pp 2042–2045

Saraswat M, Tripathi RC (2020) Cloud computing: Comparison and analysis of cloud service providers-aws, microsoft and google. In: 2020 9th International conference system modeling and advancement in research trends (SMART), pp 281–285. https://doi.org/10.1109/SMART50582.2020.9337100

Schwarz J, Steffens A, 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. https://doi.org/10.1109/QUATIC.2018.00040

Semgrep (2023) Semgrep: code analysis at ludicrous speed. https://semgrep.dev/. Accessed 13-March-2023

Sengupta S, Kaulgud V, Sharma VS (2011) Cloud computing security–trends and research directions. In 2011 IEEE world congress on services, pp 524–531. https://doi.org/10.1109/SERVICES.2011.20

Sharma T, Fragkoulis M, Spinellis D (2016) Does your configuration code smell? In: 2016 IEEE/ACM 13th working conference on mining software repositories (MSR), pp 189–200. https://doi.org/10.1145/2901739.2901761

Spinellis D (2012) Don't install software by hand. IEEE Softw 29(4):86–87. https://doi.org/10.1109/MS.2012.85

Stultiens R (2020) Compliant but vulnerable: fixing gaps in existing aws security frameworks

Terrascan (2023) Detect compliance and security violations across infrastructure as code (iac) to mitigate risk before provisioning cloud native infrastructure. https://runterrascan.io/. Accessed 13-March-2023

Tfsec (2023) tfsec uses static analysis of your terraform code to spot potential misconfigurations.https://github.com/aquasecurity/tfsec. Accessed 13-March-2023

Ur Rahman AA, Williams L (2016) Security practices in devops. In: Proceedings of the symposium and bootcamp on the science of security, pp 109–111

Vaarandi R, Pihelgas M (2014) Using security logs for collecting and reporting technical security metrics. In: 2014 IEEE military communications conference, pp 294–299. https://doi.org/10.1109/MILCOM.2014.53

van der Bent E, Hage J, Visser J, Gousios G (2018) How good is your puppet? an empirically defined and validated quality model for puppet. In: 2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER), pp 164–174. https://doi.org/10.1109/SANER.2018.8330206

Verdet A, Hamdaqa M, Da Silva L, Khomh F (2023) Exploring security practices in infrastructure as code: An empirical study - replication package. https://github.com/averdet/master-terraform-sec-adoption

Alexandre Verdet currently is a Professional Services Graduate at Amazon Web Services (AWS). Previously, he was a Research Assistant at the Department of Computer Engineering and Software Engineering at Polytechnique Montréal (Canada). In 2023, he received a Master of Applied Science (M.A.Sc - Computer Engineering) from Polytechnique Montréal, Canada, as well as a Master of Engineering (M.Eng Ingénieur Civil des Mines - Computer Science) from École des Mines de Saint-Étienne, France. He is interested in implementing security and privacy-preserving practices in distributed systems, especially in cloud infrastructures.

Mohammad Hamdaqa is an assistant professor at the Department of Computer Engineering and Software Engineering at Polytechnique Montréal (Canada), where he leads the Software and Emerging Technologies Lab. He is also an adjunct professor at the Department of Computer Science and a co-founder of the Centre of Financial Technology at Reykjavik University (Iceland). Hamdaqa received a Ph.D. in software engineering from the University of Waterloo in Canada in 2016. He holds a Bachelor's degree in Computer Engineering, a Master of Applied Science (Software Engineering), and a Master of Business Administration (MBA) with a minor in Management Information Systems (MIS). Hamdaqa's work focuses on the interplay between emerging technologies and software applications, particularly how software engineering approaches, methods, and practices can be tailored to tame the complexities of building and deploying applications for the new emerging platforms. He is interested in exploring how models and model-driven engineering can empower the development of intelligent systems, as well as how machine learning and generative AI work in tandem with models to revolutionize the conceptualization of trustworthy software systems, ultimately enabling software development through user-friendly low-code platforms accessible to citizen developers. Hamdaqa has made contributions and published papers in the areas of model-driven software engineering, cloud computing, and blockchain. He is a member of the IEEE Computer Society (CS), the Association for Computing Machinery (ACM), and the Icelandic Blockchain Foundation.

Leuson Da Silva is a Postdoctoral Fellow at Polytechnique Montreal (Canada). He received his Ph.D. in Computer Science with a focus on Software Engineering from the Federal University of Pernambuco, Brazil, in 2022. He also holds a Bachelor's degree in Software Engineering and a Master's degree in Computer Science. Leuson has published his research in leading software engineering venues, including MSR, ICSME, JSS, and JSEP. He is an active contributor to the academic community, serving on program committees for conferences and as a reviewer for renowned journals such as TSE, TOSEM, and JSEP. His research interests include code integration conflicts, software modularity, software engineering for machine learning, and empirical software engineering.

Dr. Foutse Khomh is a Full Professor of Software Engineering at Polytechnique Montréal, a Canada Research Chair Tier 1 on Trustworthy Intelligent Software Systems, a Canada CIFAR AI Chair on Trustworthy Machine Learning Software Systems, an NSERC Arthur B. McDonald Fellow, an Honoris Genius Prize Laureate, and an FRQ-IVADO Research Chair on Software Quality Assurance for Machine Learning Applications. He received a Ph.D. in Software Engineering from the University of Montreal in 2011, with the Award of Excellence. He also received a CS-Can/Info-Can Outstanding Young Computer Science Researcher Prize for 2019. Khomh is a leader in Software Engineering (SE) with strong emphasis on the use of Machine Learning (ML) in supporting SE activities, and the use of SE to engineer world-class trustworthy ML systems. His work uniquely combines his deep knowledge on software quality and ML to tackle the multifaceted problem of engineering trustworthy ML-powered software systems. Khomh's research activities contribute theories, methods, and tools to support the development, testing, and release of dependable and trustworthy ML-powered software systems. His work has received four ten-year Most Influential Paper (MIP) Awards, six Best/Distinguished Paper Awards at major conferences, and two Best Journal Paper of the Year Awards. He initiated and co-organized the Software Engineering for Machine Learning Applications (SEMLA) symposium and the RELENG (Release Engineering) workshop series. He also co-organized the FM+SE Summit series (https://fmse.io/), a platform where leading industrial and academic experts discuss and reflect on the challenges associated with the adoption of foundation and large models in software engineering. He is co-founder of the NSERC CREATE SE4AI: A Training Program on the Development, Deployment, and Servicing of Artificial Intelligence-based Software Systems and one of the Principal Investigators of the DEpendable Explainable Learning (DEEL) project. He is also a co-founder of Quebec's initiative on Trustworthy AI (Confiance IA Quebec) and Scientific co-director of the Institut de Valorisation des Données (IVADO). He is on the editorial board of multiple international software engineering journals (e.g., TOSEM, IEEE Software, EMSE, SQJ, JSEP) and is a Senior Member of IEEE.