A Large-Scale Study on the Security Vulnerabilities of Cloud Deployments

Iosif Andrei-Cristian[1,2][0000−0003−1867−1542], Tiago Espinha Gasiba[1,3][0000−0003−1462−6701], Tiange Zhao[1,3][0000−0003−1518−473], Ulrike Lechner[3][0000−0002−4286−3184], and Maria Pinto-Albuquerque[4][0000−0002−2725−7629]

[1] Technische Universität München, Munich, Germany
andrei.iosif@tum.de
[2] Siemens AG, Munich, Germany
{andrei-cristian.iosif,tiago.gasiba,tiange.zhao}@siemens.com
[3] Universität der Bundeswehr München, Germany
{tiago.gasiba,tiange.zhao,ulrike.lechner}@unibw.de
[4] Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Lisboa, Portugal
maria.albuquerque@iscte-iul.pt

Abstract. As cloud deployments are becoming ubiquitous, the rapid adoption of this new paradigm may potentially bring additional cyber security issues. It is crucial that practitioners and researchers pose questions about the current state of cloud deployment security. By better understanding existing vulnerabilities, progress towards a more secure cloud can be accelerated. This is of paramount importance especially with more and more critical infrastructures moving to the cloud, where the consequences of a security incident can be significantly broader. This study presents a data-centric approach to security research – by using three static code analysis tools and scraping the internet for publicly available codebases, a footprint of the current state of open-source infrastructure-as-code repositories can be achieved. Out of the scraped 44485 repository links, the study is concentrated on 8256 repositories from the same cloud provider, across which 292538 security violations have been collected. Our contributions consist of: (1) understanding on existing security vulnerabilities of cloud deployments, (2) contributing a list of *Top Guidelines* for practitioners to follow to securely deploy systems in the cloud, (3) providing the raw data for further studies.

Keywords: Cloud · Security · Industry · Critical Infrastructures · Awareness · Infrastructure as Code · Terraform · Secure Coding

1 Introduction

Cloud computing, as per the NIST definition [14], is "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources", with rapid provisioning and minimum management requirements. Industry needs and trends correlate well with the advantages of

using cloud-based resources – on-demand availability, reliability, and a strong accent on flexibility of the cloud have all contributed to the accelerated adoption of the cloud paradigm.

Cloud computing can be seen as a combination of existing successful technologies (i.e. virtualization, distributed storage, etc.). Recent demands for more robust infrastructure, as well as the underlying paradigm shift to *Everything-as-Service* (XaaS) have only heightened the adoption rate of companies using cloud resources for software deployment, due to both economic and performance considerations [4,1].

All previous points make cloud deployment inherently resilient to DDoS attacks, natural disasters, and large-scale power cuts – which also makes it an appropriate and viable implementation option for critical infrastructures. On this topic, The Cloud Security Alliance points out, notwithstanding all advantages, that cloud critical infrastructures need to account for *concentration of resources*, i.e. having many resources in the same data center rather than distributed [18]. This can be an issue since exploits in widely used software are awaiting to be discovered, and the impact of a cyber attack can therefore be devastating. Cloud adoption within critical infrastructure sectors accentuates the imperative for ensuring resilience and security of deployment practices.

On the other hand, ensuring the security of cloud resources is an emerging research field, offering ample space for fruitful research. Whereas software vulnerabilities would be intrinsically related with secure coding, secure cloud deployment is more amorphous and multifaceted by comparison. Recent developments in terms of cloud practices have given birth to *Infrastructure as Code* (IaC), which allows to capture the architecture and configuration details of the deployment infrastructure with code (e.g. type and number of required resources). IaC as a practice thus enables version control and automation of operations.

Codebases, in the traditional sense of programming, can be developed and compared against a set of Secure Coding Guidelines, which are principles and practices for developers to follow for ensuring proper, verifiable, security [9]. Deployment code, on the other hand, namely IaC, has yet to currently mature enough to encompass similar directionality in its own ecosystem. *DevSecOps* [19] is a first step in constructing a similar set of principles, serving as a security-centric production lifecycle paradigm. The cornerstone of *DevSecOps* is that all development stages keep best practices into consideration and share responsibility, which consequentially leads to a smaller bridge to gap between development and operations teams.

Past cloud security incidents point to a consistent pattern, in which even a small misconfiguration can incur catastrophic consequences. Incidents focusing on Amazon Web Services (AWS) Scalable Storage Solution (S3) buckets have offered various news stories in the past, as the data breaches usually impress in size and negativity of outcome [15], usually due to improper access control policies. Moreover, searching publicly accessible S3 buckets is trivial [10]. As a notable example, the U.S. Department of Defense unintentionally disclosed login credentials and government intelligence data [21].

To the best of our knowledge, no study looks at real-world problems of Infrastructure as Code. Namely, the question of which most prominent problems exist in cloud deployments is seldom explored systematically. This current work addresses this question – our work focuses on Terraform as an IaC solution, and explores publicly available repositories with the help of three static code analysis tools to understand existing problems. A total of 8256 repositories have been analysed, all of which contain Terraform deployment code and use AWS as a cloud provider.

The main contributions of our study are the following:

- Providing an understanding of the most prominent security vulnerabilities for cloud environments.
- Deriving a list of *Top IaC Guidelines for Security* for Terraform projects
- Publishing the data gathered from all repositories, for further studies of similar nature.

This paper is organized as follows: Section 2 will introduce related work pertaining to standards and guidelines related to cloud security. Section 3 provides an overview of the experimental methodology employed for completing this study. Section 4 follows by presenting the technical details behind the methodology. The results are showcased and examined in section 5. Based on these results, the section also includes a discussion centered around recommendations given the currently available security toolkits. Finally, section 6 epitomizes through our work and presents further tangent research directions based on our findings.

2 Related Work

While the security services are the same between traditional and cloud deployments (Integrity, Confidentiality, Availability, Accountability, Authenticity, Privacy), the means of achieving them can differ due to the underlying distributed implementation. Furthermore, the convenience and deployment speed gained through the inherent abstraction provided by IaC may lead to *quick bugs* and vulnerabilities [17].

A generalized taxonomy of cloud security issues has been elaborated by Tabrizchi et al. [20], in which threats, attacks and solutions are discussed. While this security classification may serve as a consistent introductory reading to cloud architectures and their potential issues, it does not address the real-world probability of cloud security. Another study conducted by Rahman et al. provides an onlook of open-source software (OSS) repository security and outlines the top 8 malpractices for IaC. Nonetheless, the authors' study is limited to the OSS repositories of solely one organization, and thus paints an incomplete picture of the *overall* security of OSS projects [17].

There is a number of ongoing standardisation efforts to refine the issue of cloud security into a practical, applicable formula. For example, The Cloud Controls Matrix (CCM) is a cybersecurity control framework for cloud computing, based on best practices recommended by the Cloud Security Alliance, "that is considered the de-facto standard for cloud security and privacy"[2]. The CCM is

constantly updated to reflect the newest developments and threats in the Cloud ecosystem. Additionally, government agency branches are developing usage and security guidelines alongside these efforts – one such example is the BSI (German Federal Office for Information Security), which released two documents in 2021, pertaining to the secure usage of cloud services [8,7]. Similarly to the Cloud Security Alliance, the BSI Whitepaper on Cloud Security draws a stark warning about *concentration of resources* [6], with respect to critical infrastructures deployed in the cloud.

Other industry branches also employ their own set of security guidelines: IEC 62443 [12] is a set of standards, developed starting in 2009, focusing around Industrial Automation and Control Systems (IACS) security. However, the standard does not currently cover cloud security explicitly, which poses a severe risk – as more critical infrastructures begin relying on cloud resources, their certification will be only on par with an outdated standard.

On the issue of assessing the robustness of IaC deployment scripts, there are no standardised coding guidelines and best practices yet. According to industry practitioners surveyed by Guerriero el al. [11], lack of testability is quoted as the leading drawback of IaC solutions. The IaC ecosystem provides, however, a set of static code analyisis (SAST) tools, which serve as a (semi-)automated way to ensure that a given piece of code does not present vulnerabilities. The practice of augmenting development pipelines with such SAST tools in the early stages is refered to as the *shift-left* security paradigm, as it offsets potential threats earlier before the code gets deployed.

All SAST tools accounted for in this study (`tfsec`, `terrascan`, `checkov`) share the following common information in the output format for a security violation: the tool would provide an error ID, a short description, a documentation link with secure and insecure code examples pertaining to the detected vulnerability, and a vendor-specific severity rating. All aforementioned tools are under constant development, with rapidly evolving security policy indexes, reflecting the underlying emergence of cloud technologies. The integration of the SAST tools into the methodology pipeline shall be presented in the following section.

3 Method

To assess real-world security of IaC implementations, we decided to focus on open-source repositories. The rationale behind this decision relies on the fact that openly available projects are very often used as either starting points, or directly as a *paste-in-place* solution. By analysing open-source repositories, we can estimate a vulnerability footprint of IaC based on the results of the security assessment each repository shall be subjected to. The collated results shall thus serve as a security snapshot of the open-source IaC ecosystem.

Figure 1 introduces an overview of the employed methodology, which we will describe in the following. The methodology can be divided into 3 phases, as shown in the same figure.

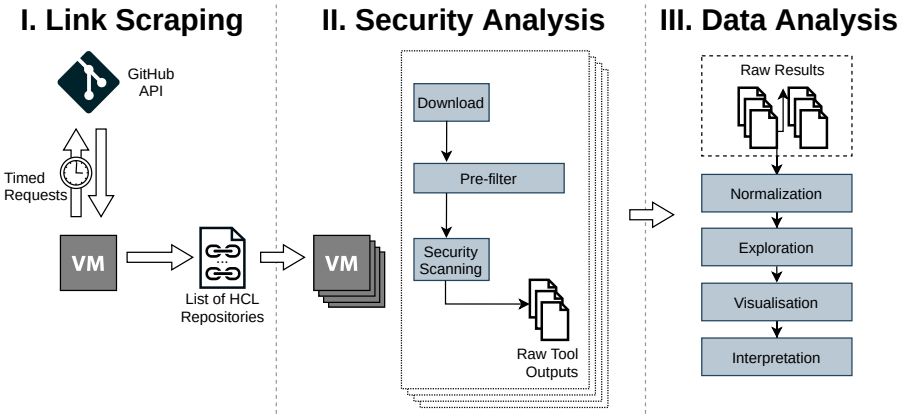

Fig. 1. Research design

I. Link Scraping: A list of GitHub repositories containing Terraform code was constructed by querying the website's API. The underlying syntax of Terraform itself is Hashicorp Configuration Language (HCL). The paper will refer to HCL code as Terraform code, as the latter term provides a more transparent understanding. GitHub's API documentation provides useful options for searching within a given timespan and filtering for the HCL language. In terms of limitations, each query is limited to 10 pages of 100 results each, totalling a possible maximum of 1000 results. The constructed query for collecting the links, with string-interpolated query parameters, is:

```
api.github.com/search/repositories?q=language:HCL
+pushed:{start}..{end}&per_page=100&page={page}
```

We have explored and collected results from the GitHub database in one-month increments. The result of this step is a collated list of 44485 repository links and their associated metadata, labeled by GitHub as Terraform code.

II. Security Analysis: Starting from the aforementioned list, the next step centers on static code analysis. Each repository is downloaded, then subjected to a number of sanity checks and filters. First, we assess whether the repository contains Terraform code or not, as one discovery consisted in uncovering that GitHub's labeling mechanism is prone to a small number of false positives (i.e. non-Terraform and empty repositories were initially included in the scraping results set). Next, the repository is evaluated by three security evaluation tools: `tfsec`, `terrascan` and `checkov`. Particular tools timed out on various repositories, which lead to the need of pruning such cases out of the evaluation loop. The process was parallelized across 4 machines, by splitting the list of repositories in 4, each quarter being further split to maximize the number of available threads in the used machines. The scanners' output is collected *as-is*, with further processing steps being left for the next stages of the processing pipeline.

III. Data Analysis: Since three different static code analysis tools are employed, all developed by different teams, the initial program output is heterogeneous, and thus initially inadequate for analyzing. Therefore, a first step for analysis consists of *normalizing* the tool's output – this comprises of manually evaluating each tool's output format and transforming it under a unified scheme that captures the essential information from each tool report.

The captured information for each reported security violation is: repository metadata (ID, name, creation date, date of last update, stargazers count, forks count), violation name, resource type, guideline, and severity of the violation. The consolidated dataset encompasses 292538 security violations, over 13627 repositories. The final steps of this research consists of using the available data to formulate conclusions, which shall be presented in Section 5. Data exploration, analysis and visualisation was performed in both R (v4.1.1) and Python (v3.8.10), with some intermediary data transformations being obtained through bash scripting and `jq`.

4 Experiment

This section will describe the relevant technical details behind the first two stages of the implementation effort of the present study, in order to present a complete and transparent outlook on the resource and time requirements of the study.

The execution time of the webscraper for collecting links was approximately 3 hours. This step resulted in the construction a dataset consisting of 44485 repository links, along with their relevant metadata: date created, date of last update, number of stars, number of forks, etc. The links were collected on July 15 2021, and their 'last pushed date' ranges from 07-2015 to 06-2021. After pruning for AWS-only providers and non-Terraform downloads, this list was narrowed down to 8256 repository links. A total of 70 repositories were rejected for not containing any Terraform code, and less than 10 repositories from the list of collected links had either turned private or been deleted in the timeframe between link acquisition and repository cloning. As the operation of querying the API is not resource-intensive, the machine running the script was sized accordingly – a `t3.small` AWS instance was used, with 2GB RAM and 2 virtual CPU's (up to 3.1GHz Intel Xeon Platinum).

The second stage, constituting of scanning individual repositories for potential security malpractices, was parallelized across 4 machines. Each of the 4 machines ran 16 parallel threads, thus reducing the overall time of this operation 64-fold. The execution time was approximately 36 hours. The employed resources for running the static code analysis tools and collecting results were AWS `t3.2xlarge` instances with 8 vCPU's, 2 threads per core (up to 3.1GHz Intel Xeon Platinum) and 16GB RAM. The software versions of the employed security scanning tools were: `tfsec v0.48.7`, `terrascan v1.4.0`, and `checkov v2.0.46`.

5 Results and Discussions

The consolidated dataset has been explored and analysed to determine the distribution of resource types and subtypes, which widespread security malpractices reside in the most widely used resource types, and to determine underlying security trends in IaC repositories. These findings shall be presented in the first part of this section.

5.1 Cloud Vulnerability Trends

Looking at the yearly statistics outlined in Figures 2 and 3, we observe that the the number of Terraform repositories has increased over time, indicating the rise in popularity of the IaC paradigm. However, we also observe that the number of security violations has risen alongside the number of repositories over the years. Analysing the number of average violations per repository, our results indicate that the average count of violations for a given public repository has stabilised in the past two years around a value of 15 violations per repository. We find this result to be alarming, since it can imply that security awareness for IaC is stagnating, a trend which is reflected by the lack of overall decline in the yearly statistic of average vulnerabilities per repository. The three static code analysis tools employed for this study offer subjective *severity* ratings – the ranking between the severity ratings does not change over the years, with security findings ranked as high-severity being the most prevalent, and critical-severity ones being the least prevalent.

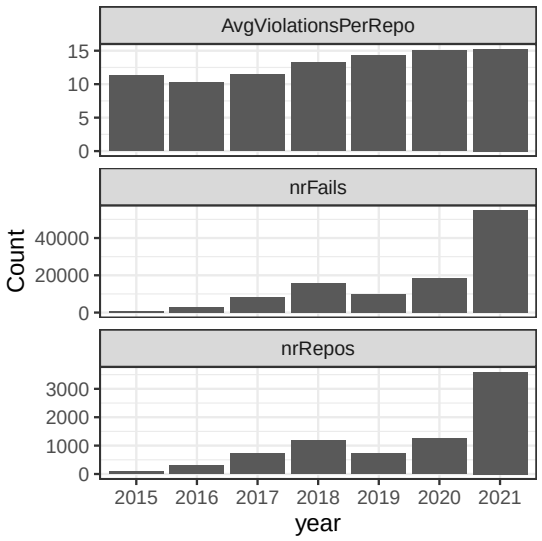

Fig. 2. Yearly Evolution - Repositories, Violations

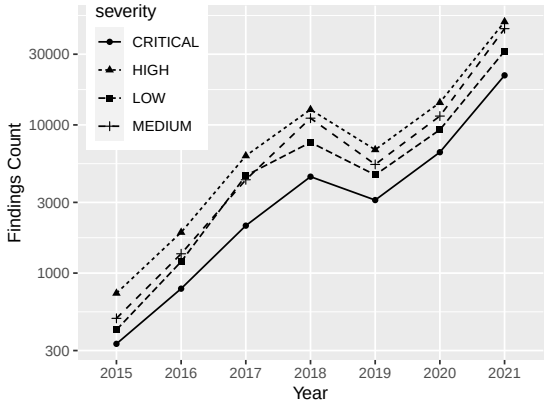

Fig. 3. Yearly Evolution - Severity

Resource distribution (Fig. 4) highlights the fact that variables (6.32%) and outputs (5.03%) account together for more than 10% of the overall used types in Terraform code. Despite the notably high prevalence of variable and resource usage in Terraform code, there is almost no ruleset built into the security scanners employed in this study to account and spot for security violations in these generic types – `tfsec` is the sole exception, providing four rules that scan for secret values stored in attributes, templates, default values and variables.

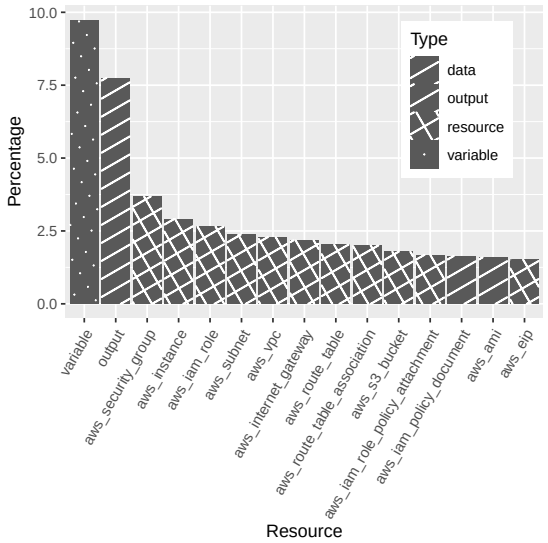

Fig. 4. Top 15 Used Types

Accounting only for resource subtypes, figure 5 highlights the top 10 most vulnerable AWS resources, based on the count of total associated observed security violations from the three SAST tools.

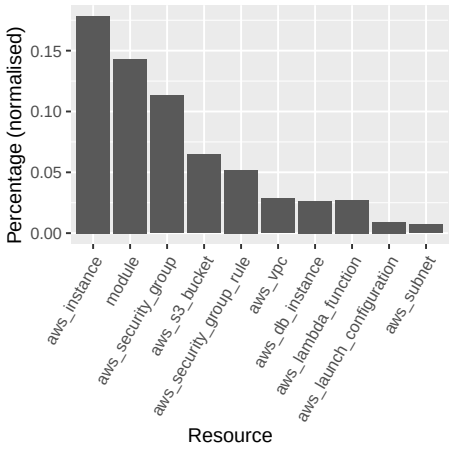

Fig. 5. Top 10 Vulnerable Resources

The security of a given public Terraform repository can also be discussed from a probability standpoint. For this, the cumulative probability of the number of total tool findings was analysed. The results are presented in Figure 6 for up to 15 tool findings. One may see that the probability distribution resembles a sigmoid function, with a sharp increase in probability starting at the count of 3 vulnerabilities. An alarming conclusion based on this analysis is that the average rounded median value is **5**, which is an alarmingly high average number of potential vulnerabilities in code used for deploying infrastructure.

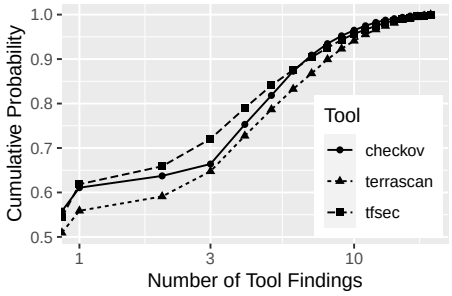

Fig. 6. Cumulative Probability - SAST security violations

5.2 Most commonly observed malpractices

We present the most prevalent security findings from the three SAST tools, ranked by occurrence, in table 1. The last column clusters the results across different areas for interest. The data was collected such that each violation's rule ID is only collected once per repository. The rationale is that this approach allows for conveying the security of a given repository, irrespective of the size of its codebase – if two repositories of vastly different size and complexity trigger the same rule violation, regardless of the number of observations from the SAST tools, they are considered equally insecure.

Table 1. Most Observed Security Violations across SAST tools

Ranking	Rule	Tool	Category
1	AWS079	tfsec	Hardening
2	AWS009	tfsec	Insecure defaults
3	CKV_AWS_8	checkov	Encryption
4	AC-AWS-NS-IN-M-1172	terrascan	Hardening
5	AWS.AI.LM.HIGH.0070	terrascan	Logging
6	AWS.CloudTrail.Logging.Medium.008	terrascan	Best practices
7	AWS008	tfsec	Insecure defaults
8	CKV_AWS_79	checkov	Insecure defaults
9	AWS.VPC.Logging.Medium.0470	terrascan	Logging
10	AWS018	tfsec	Code Documentation
11	CKV_AWS_135	checkov	Best practices
12	CKV_AWS_126	checkov	Logging
13	AWS002	tfsec	Logging
14	AWS098	tfsec	Hardening
15	AWS005	tfsec	Access control
16	AWS.S3Bucket.LM.MEDIUM.0078	terrascan	Logging
17	AWS017	tfsec	Encryption
18	AWS099	tfsec	Access control
19	AWS077	tfsec	Logging
20	AC_AWS_0228	terrascan	Access control (AC)
21	AC_AWS_0227	terrascan	AC & Insecure defaults
22	AWS.S3Bucket.EKM.High.0405	terrascan	Encryption
23	CKV_AWS_24	checkov	Logging
24	AWS.S3Bucket.IAM.High.0370	terrascan	Logging
25	AC-AW-IS-IN-M-0144	terrascan	Insecure defaults
26	AWS014	tfsec	Encryption
27	AC-AW-CA-LC-H-0439	terrascan	Insecure defaults
28	AWS012	tfsec	Access control
29	CKV_AWS_144	checkov	Hardening
30	CKV_AWS_52	checkov	N/A (deprecated)
31	AC_AWS_0276	terrascan	Access control
32	CKV_AWS_18	checkov	Logging

Overall, our findings about the nature of the most frequently occurring security violations correlate well with the ones from Rahman et al. [17]. Namely, based on the top seven observed malpractices, the clustering revealed the following areas of blindsidedness in IaC scripts, which list, in the following, from what we consider most to least severe:

1. **Encryption**: Unencrypted resources deserve a category of their own , separate from *insecure defaults*, due to the implications of not using encryption – a data breach on an unencrypted database can have consequences ranging from downtime to identity theft and privacy violations.
2. **Access control**: Misconfigured access control consequences range from simple data breaches/loss/corruption to intellectual property implications and ransomware. This topic is refered to as CWE-200 [3] and results in exposing sensitive information to an actor that is not explicitly authorized to have access to that information. The OWASP project has rated Broken Access Control as the first-ranking web vulnerability in their 2021 Top 10 ranking of vulnerability classes [16].
3. **Insecure defaults**: Some resources come with very relaxed default security options, such as public access and no login gatekeeping. These defaults must be overridden before any other configuration options are accounted for.
4. **Enabling readily available hardening**: As the cloud providers' API need to account for more and more usecases, the configuration options are evolving to include more and more security hardening options. Some of these options are not enabled by default, and their addition is not always brought to the forefront of the providers' update notes. Therefore, the duty of routinely investigating into the addition of such options and enabling them in already deployed infrastructure lies with the practitioner.
5. **Logging**: As resources are no longer *in-house*, quantitative logging gains greater importance within the cloud paradigm.
6. **Best practices – enabling readily available options**: Similar to enabling security options, cloud providers offer out-of-the box customizations which can result in great performance boosts
7. **Best practices – documented code**: Due to the inherent complexity of large infrastructure, having a well-documented codebase related to the deployed architecture can ease security auditing and prevent unintended misconfigurations along the time.

5.3 SAST Tools Comparison

In the following, we provide a comparison between the static code analysis tools, based on the gathered results and experience, that focuses on their performance and clarity of delivering results. In terms of scanning duration, measuring the time between the three tools revealed that `tfsec` is two orders of magnitude faster than the other two SAST tools (avg 0.01s/scan), with the latter being comparable in this regard (avg. 1s/scan).

Documentation-wise, both `tfsec` and `terrascan` offer built-in, proprietary *severity* ratings, whereas `checkov` only does so on the website of its parent

company. Each severity rating of a `checkov` rule had to be compiled from the webpage of the rule, since the link and rule nomenclature lacked homogeneity.

Furthermore, a deeper look into the tools' ruleset documentation revealed that `terrascan` exhibits duplicate rules, listed under different rule ID's (i.e `AWS.S3Bucket.IAM.High.[0378|0379|0381]`). Similarily, `checkov` exhibits mislabeled rules (i.e. `CKV_AWS_21`, labeled as an S3 bucket rule, states that *"all IAM users should be members of at least one IAM group"*, which refers to an unrelated resource). Another, less concerning finding about documentation is that rules which get deprecated are no longer included in the documentation in the case of `checkov` (i.e. `CKV_AWS_52`). Further metrics which the discussion about the SAST tools will rely on are presented in Tables 2 and 3.

Table 2. Tools - Detection Overlap

	Number of Errors Found by Tool			
Tool	**0**	**1**	**2**	**3**
tfsec	44.02%	1.65%	19.56%	34.77%
terrascan		3.10%	18.11%	
checkov		2.04%	19.17%	

Table 3. Tools - Violation Metrics

Metric	**tfsec**	**terrascan**	**checkov**
zero violations	50.8%	51.22%	58.05%
avg. violation #	6.66	4.87	3.14
# total rules	90	296	118

We observe that the distribution of available rules for each resource type varies between all three tools – some tools offer good coverage for certain resource types, while others do not. Due to this fact, we conclude that, from the tools we have analyzed, none offers can promise better *overall* coverage than the others. Tables 2 and 3 comprise of notable metrics between the tools and point to a similar conclusion – there is a great discrepancy between the available number of rules of each tool, and table 2 further cements the intuitive conclusion that the tools have no unclear overlap between their coverage.

Having these points considered, our recommendation is that any IaC security pipeline should include all three. A security scan shall thus rather report duplicate violations from multiple tools, or include false positives, rather than the incident-prone alternative of having a false negative scan from a smaller number of SAST tools.

5.4 Discussion

Improving the current situation in terms of cyber security of cloud environments can be addressed through raising awareness surrounding the security of cloud deployments. This can be done through campaigns, either informative or structured as hands-on trainings [5,22].

Furthermore, increasing the popularity of the already-existing SAST tools for IaC can not only boost the security of codebases, but also enforce pressure on the development teams to improve the tools' coverage and better tackle false positives/negatives.

Considering the rapid, constant evolution of cloud solutions, security practitioners and researchers alike must always stay informed about the latest changes to resource deployment methodologies, reasons behind the deprecation of security options and the introduction of updates. Ideally, cloud providers and SAST tool development teams should congregate to provide up-to-date, holistic changelogs that always reflect the current conditions.

Based on the priorly introduced results, we elaborate the following findings to augment our discussion:

- As the number of Terraform repositories is increasing, so does the number of vulnerabilities. Whereas the first conveys an increase in adoption of IaC, we must argue about the latter trend observed in security malpractices is a problem for cloud deployments, especially in the case of critical infrastructures.
- As the number of Terraform repositories is increasing over time, while the average count of vulnerabilities stays constant (flattening out at 15), we can conclude that security awareness is lacking for cloud deployment using IaC.
- An unexpected result from analysing the data came in the form of the discrepancy between the most vulnerable observed resources, when compared to news outlets – the S3 bucket was not the highest finding, being ranked fourth in our assessment. The top 3 vulnerable resources are `instance`, `module` and `security group`. As these resources are more widely used than the S3 bucket, the security impact of malpractices concerning them can be higher.
- Another unexpected conclusion is the high vulnerability ranking of the `module` Terraform type. However, malpractices concerning these type are firstly violations in terms of *best practices* and *DevOps*, rather than introducing problems in the deployed infrastructure.
- We observed two classes of problems:
  1. Before-deployment, related to best practices (i.e. logging of variables)
  2. After-deployment, succeeding the application of a Terraform configuration, and resulting in potentially misconfigured and/or vulnerable resources (i.e. unencrypted S3 bucket)
- We observe that `tfsec` outputs more finding than the other tools, followed by `checkov` and `terrascan`. This is an unanticipated finding, as `tfsec` employs the least amount of rules in its policy index, yet results in the highest amount of findings (cf. table 2. Overall, this indicates that either the quality of the

employed SAST tools can differ significantly, or that `tfsec`'s rules are more targeted.
- Our experience has shown that tools are constantly evolving and changing. According to our experience, such ever-changing requirements can be seen as a *wicked problem* for software developers, in which practitioners struggle to always be up to date. Based on the present work, as well as experience in the field of cyber security, we thus recommend practitioners to take the following guidelines into consideration:
  1. Relying on a single SAST tool can be perilous, as our results indicate that findings between them are disperse and the overlap in coverage varies.
  2. *Paste-in-place* solutions should be heeded cautiously, as our study revealed a multitude of problems in open source repositories, and drastically variable quality between repositories. Code should be analysed for potential vulnerabilities before being inserted into a deployable architecture.
  3. Practitioners may use table 1 to guide their decisions on tool selection.
  4. In practical deployments, we recommend the use of a baseline of tools, in order to avoid introducing unstable requirements due to rule deprecation and/or creation.
  5. Implementing awareness campaigns in an industrial setting with the goal of trying to raise awareness of secure cloud deployment problems can have a benefic impact. In a non-corporate setting, the use of employing SAST tools can provide the same exposure to security malpractices and enhance one's code quality

5.5 Threats to Validity

In terms of threats to validity, the study cannot account for false positives and negatives – the data spans across too many repositories to properly quantify and asses such metrics. The closest figures are presented in table 2, where coarse detection overlap is highlighted between the three employed SAST tools. Furthermore, the data is specialised, in the sense that only repositories that have AWS as a provider have been considered, as there is no one-to-one mapping between cloud providers. Nonetheless, we believe that our work can extend to other cloud providers, with different results between them. Another underlying feature of the data consists in the fact that some repositories are "collections of snippets" (i.e. book examples), which might be incomplete, thus unintentionally increasing the number of findings. This latter point is easily refutable, since published snippets should include security measures nonetheless.

6 Conclusions and Further Work

As trends in information technology are all congregating towards cloud computing, distributed infrastructures and off-site deployment become more and more

ubiquitous. The industry is exerting sufficient pressure in this direction for cloud solutions to become more widely adopted, due to their inherent attractiveness in terms of lower, variable, costs and significantly shortened operations lifecycle. In terms of security, the deployment of infrastructure is currently ongoing a discovery phase before a unified body of security standards emerges.

Since deploying architectures is faster and more straightforward through the use programming languages for Infrastructure as Code, the pre-existing security problems of cloud deployment can potentially be amplified. Furthermore, due to the multitude of openly available, ready-to-use IaC repositories for any general use-case, the issue of propagating security malpractices thus gains fundamental importance.

In this work we look at the most prominent security vulnerabilities in cloud deployments. The study focuses on Terraform as an IaC solution, and AWS as a cloud provider. By using three tools for static code analysis, we conducted a security analysis on 13627 public repositories, over which 292538 security violations were detected.

Analysing the collected data, we conclude on a list of the most wide-spread vulnerable cloud resources. We clustered the most prevalent categories of vulnerabilities across seven categories, and on their basis propose a security guidelines list, acting as a security footprint snapshot for the current state of cloud deployment security across publicly available Terraform code.

We intend that the proposed list of seven cloud security guidelines may act as a valuable source of information for industry practitioners, such that the security of their own deployments may improve, along with the overall security awareness in the cloud deployment landscape.

In a further work, the authors would like to explore mechanisms on how to raise the awareness of security vulnerabilities in IaC using Terraform. Additionally, the conclusions of the study can be adapted for cloud service providers apart from AWS, to explore whether the observed vulnerability trends are universally aplicable or vendor-specific. Furthermore, as development concerning the static code analysis progresses, and as the number of newly created Terraform repositories is on the rise, this study provides a high degree of repeatability, in terms of assessing a time evolution of the security of open-source projects relying on Terraform.

Supporting Research Data

The consolidated dataset, which is the cornerstone of this work, is publicly available for inspection and further research on the Zenodo Platform [13]. The data is provided in Comma Separated Values (CSV) format, and comprises 292538 security violations collected across 13627 Terraform repositories (AWS-only provider).

References

1. Achilleos, A.P., Georgiou, K., Markides, C., Konstantinidis, A., Papadopoulos, G.A.: Adaptive Runtime Middleware: Everything as a Service. In: Nguyen, N.T., Papadopoulos, G.A., Jedrzejowicz, P., Trawinski, B., Vossen, G. (eds.) Computational Collective Intelligence. pp. 484–494. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-67074-4_47
2. Cloud Security Alliance: Cloud Controls Matrix. https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v4/ (2021)
3. Common Weakness Enumeration: Exposure of Sensitive Information to an Unauthorized Actor (2021), https://cwe.mitre.org/data/definitions/200.html
4. Duan, Y., Fu, G., Zhou, N., Sun, X., Narendra, N.C., Hu, B.: Everything as a Service (XaaS) on the Cloud: Origins, Current and Future Trends. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 621–628 (2015). https://doi.org/10.1109/CLOUD.2015.88
5. Espinha Gasiba, T., Andrei-Cristian, I., Lechner, U., Pinto-Albuquerque, M.: Raising Security Awareness of Cloud Deployments using Infrastructure as Code through CyberSecurity Challenges. In: The 16th International Conference on Availability, Reliability and Security. pp. 1–8 (2021)
6. Federal Office for Information Security: Security Recommendations for Cloud Computing Providers – Minimum information security requirements. White Paper - (06 2011), https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/CloudComputing/SecurityRecommendationsCloudComputingProviders.html
7. Federal Office for Information Security: OPS.2: Cloud-Nutzung. White Paper - (2021), https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium_Einzel_PDFs_2021/04_OPS_Betrieb/OPS_2_2_Cloud-Nutzung_Edition_2021.pdf?__blob=publicationFile&v=2
8. Federal Office for Information Security: Sichere Nutzung von Cloud-Diensten. White Paper - (2021), https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Broschueren/Sichere_Nutzung_Cloud_Dienste.pdf?__blob=publicationFile&v=1
9. Gasiba, T., Lechner, U., Cuellar, J., Zouitni, A.: Ranking Secure Coding Guidelines for Software Developer Awareness Training in the Industry. In: First International Computer Programming Education Conference (ICPEC 2020). OpenAccess Series in Informatics (OASIcs), vol. 81, pp. 11:1–11:11. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). https://doi.org/10.4230/OASIcs.ICPEC.2020.11, https://drops.dagstuhl.de/opus/volltexte/2020/12298
10. Greyhat Warfare: Public S3 Buckets (2021), https://buckets.grayhatwarfare.com/
11. Guerriero, M., Garriga, M., Tamburri, D.A., Palomba, F.: 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. IEEE, Cleveland, OH, USA (2019)
12. International Standard Organization: Industrial communication networks - Network and system security. Standard, International Electrical Commission (2009-2021)
13. Iosif Andrei-Cristian: Raw Results on the Study on the Security Vulnerabilities of Cloud Deployments. https://doi.org/XX.XXXX/zenodo.XXXXXXX, https://zenodo.org/record/XXXXXXX, online, Accessed XX. MONTH 2021

14. Mell, P., Grance, T.: The NIST Definition of Cloud Computing (2011-09-28 2011). https://doi.org/https://doi.org/10.6028/NIST.SP.800-145
15. Nag Media: List of AWS S3 Leaks (2021), https://github.com/nagwww/s3-leaks
16. Open Web Application Security Project: OWASP Top 10 (2017), https://owasp.org/Top10/A01_2021-Broken_Access_Control/
17. Rahman, A., Parnin, C., Williams, L.: The Seven Sins: Security Smells in Infrastructure as Code Scripts. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). pp. 164–175. ACM, Montreal Quebec Canada (2019). https://doi.org/10.1109/ICSE.2019.00033
18. Samani, R.: Critical Infrastructure and the Cloud (2013), https://cloudsecurityalliance.org/blog/2013/02/01/critical-infrastructure-and-the-cloud/
19. Sánchez-Gordón, M., Colomo-Palacios, R.: Security as Culture: A Systematic Literature Review of DevSecOps. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. pp. 266–269. IEEE, Seoul Republic of Korea (2020)
20. Tabrizchi, H., Rafsanjani, M.K.: A survey on security challenges in cloud computing: issues, threats, and solutions. The Journal of Supercomputing **76**(12), 9493–9532 (Feb 2020). https://doi.org/10.1007/s11227-020-03213-1, https://doi.org/10.1007/s11227-020-03213-1
21. UpGuard Team: Black Box, Red Disk: How Top Secret NSA and Army Data Leaked Online. https://www.upguard.com/breaches/cloud-leak-inscom (2017)
22. Zhao, T., Gasiba, T.E., Lechner, U., Pinto-Albuquerque, M.: Exploring a Board Game to Improve Cloud Security Training in Industry. In: Henriques, P.R., Portela, F., Queirós, R., Simões, A. (eds.) Second International Computer Programming Education Conference (ICPEC 2021). Open Access Series in Informatics (OASIcs), vol. 91, pp. 11:1–11:8. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/OASIcs.ICPEC.2021.11, https://drops.dagstuhl.de/opus/volltexte/2021/14227