Characterizing Static Analysis Alerts for Terraform Manifests: An Experience Report

Hanyang Hu* Yani Bu† Kristen Wong‡ Gaurav Sood§ Karen Smiley¶ Akond Rahman‖
* Company A, CA, USA
‖Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA
Email: *phenom.hu@gmail.com, †byn52602@gmail.com, ‡kristenwong16@berkeley.edu,
§sgaurav_res@yahoo.com, ¶kjsmiley@ieee.org, ‖akond@auburn.edu

***Abstract*—While Terraform has gained popularity to implement the practice of infrastructure as code (IaC), there is a lack of characterization of static analysis for Terraform manifests. Such lack of characterization hinders practitioners to assess how to use static analysis for their Terraform development process, as it happened for Company A, an organization who uses Terraform to create automated software deployment pipelines. In this experience report, we have investigated 491 static analysis alerts that occur for 10 open source and one proprietary Terraform repositories. From our analysis we observe: (i) 10 categories of static analysis alerts to appear for Terraform manifests, of which five are related to security, (ii) Terraform resources with dependencies to have more static analysis alerts than that of resources with no dependencies, and (iii) practitioner perceptions to vary from one alert category to another while deciding on taking actions for reported alerts. We conclude our paper by providing a list of lessons for practitioners and toolsmiths on how to improve static analysis for Terraform manifests.**

***Index Terms*—configuration as code, empirical study, experience report, infrastructure as code, static analysis, terraform**

I. Introduction

Infrastructure as code (IaC) is the practice of creating and managing computing infrastructure at scale [10]. Practitioners implement IaC with tools, such as Terraform. Use of Terraform has yielded benefits for organizations. For example, using Terraform, Asian Development Bank reduced its provisioning time for virtual machines from 4 days to less than 5 minutes [6]. As another example, use of Terraform helped GitHub to reduce its time to perform load balancing by 96% [5].

The above-mentioned benefits motivated Company A, a multinational proprietary organization, to adopt Terraform to automatically manage configurations and computing infrastructure. Building security activities, such as static analysis, into the development process of software artifacts is a priority for Company A. However, even though static analysis is pivotal for software security [14], prior work reports static analysis tools to have low actionability [4], [11], [26]. As a result, upon adoption of Terraform, practitioners at Company A sought evidence on the value of static analysis for Terraform manifests. As IaC is an emerging area [18], little is known about how static analysis is used in the context of Terraform manifests. A systematic characterization of static analysis alerts for Terraform manifests can address this gap by studying practitioner perceptions of Terraform-related static analysis alerts, studying the categories of static analysis alerts, and understanding how properties of entities, e.g., Terraform resources [17], are correlated with appearance of static analysis alerts. Such characterization can be helpful for: (i) *practitioners*, to learn about peer perspectives of Terraform-related static analysis alerts, (ii) *toolsmiths*, to improve static analysis for Terraform, and (iii) *researchers*, to identify new research directions in the domain of IaC.

Accordingly, we answer the following research questions:

- ***RQ1*: How do practitioners within a proprietary organization perceive the actionability of Terraform-related static analysis alerts?**
- ***RQ2*: What categories of static analysis alerts appear in open source and proprietary Terraform manifests? How frequently do alerts appear for each category?**
- ***RQ3*: How do static analysis alerts in Terraform manifests correlate with resources that have dependencies?**

We conduct an empirical study with 10 open source software (OSS) Terraform repositories and one proprietary repository maintained by Company A, to answer our research questions. First, we survey practitioners from Company A to quantify the actionability for Terraform-related alerts obtained for Company A's repository. Next, we mine the 11 repositories to quantify the frequency of Terraform-related alerts across multiple categories. Finally, we construct a Terraform entity dependency graph for each repository to quantify the relationship between resources with dependencies and appearance of static analysis alerts.

Contributions: The contributions of our paper are:

- An empirical characterization of Terraform-related static analysis alerts; and
- An evaluation of actionability of static analysis alerts from practitioners' perspectives.

The rest of the paper is organized as follows. Section II provides necessary background information and discusses related work. Section III presents the methodology to answer our research questions. We report the results in Section IV.

Section V discusses implications and limitations of our paper. Finally, we conclude the paper in Section VI.

II. Background and Related Work

We provide background and related work in this section.

A. Background on Terraform Manifests

Terraform is syntactically defined using the HashiCorp Configuration Language (HCL) [7]. There are two constructs in Terraform: arguments and blocks. Arguments are used for value assignments. Blocks are used to describe infrastructure-related entities, such as resources and variables.

A Terraform entity can be uniquely identified by its block types and labels. For example, Listing 1 shows 3 Terraform entities used for creating a docker infrastructure. The first Terraform entity, *var.nginx_ext_port*, is a variable that sets the external port of the container. The second entity, *docker_image.nginx*, is a resource that creates a docker image with tag "nginx:latest". The third Terraform entity, *docker_container.nginx*, is a resource that creates a docker container of name "mycontainer" whose image depends on *resource.docker_image.nginx* and whose external port depends on *var.nginx_ext_port*. Terraform resources represent actual infrastructure objects [17].

```
# In example_tf_repo/variables.tf
variable "nginx_ext_port" {
  type    = number
  default = 8000
}

# In example_tf_repo/main.tf
resource "docker_image" "nginx" {
  name = "nginx:latest"
}

resource "docker_container" "nginx" {
  image = docker_image.nginx.image_id
  name  = "mycontainer"
  ports {
    internal = 80
    external = var.nginx_ext_port
  }
}
```

Listing 1: An example Terraform manifest.

B. Related Work

Our paper is closely related with prior work in the domain of IaC-related static analysis. Sharma et al. [30] and Opendebeek et al. [15] applied static analysis to identify maintainability issues respectively, in Puppet and Ansible manifests. Researchers have also investigated security issues in IaC. In separate publications, Rahman et al. developed static analysis tools to identify security bugs in Ansible [21], Puppet [20], and Kubernetes manifests [22]. Reis et al. [23] and Saavedra et al. [25] pursued similar efforts for Ansible, Chef, and Puppet manifests, to facilitate better precision and recall compared to existing tools. Rahman et al. [20]'s paper was replicated by Hortlund [9], who reported the security weakness density to be less than reported by Rahman et al., due to false positives generated by SLIC. Bhuiyan and Rahman [3] reported similar observations: they manually inspected 2,764 Puppet manifests, and documented SLIC to generated 1,560 false positives. Rahman and Parnin [19] constructed TaintPup, which applies information flow analysis to reduce false positives.

The above-mentioned prior work show promise in applying static analysis tools for Ansible, Chef, and Puppet manifests. However, there is a lack of research on Terraform-related static analysis, which we address in our paper.

III. Methodology

This section provides the methodology to answer our research questions.

A. RQ1: How do practitioners within a proprietary organization perceive the actionability of Terraform-related static analysis alerts?

We use an online survey to answer RQ1. The survey includes three questions: (i) "*SQ1: Do you understand the alert message?*", (ii) "*SQ2: Does the alert message help you to understand the root cause of the alert that can help you to fix?*", and (iii) "*SQ3: Are you willing to fix the reported defect as determined by the alert?*".

Each of these three questions had two options: 'YES' and 'NO'. We derive these questions by taking motivation from prior research that has used the concept of understandability [27] and willingness to fix [8] as surrogate measures to determine actionability of static analysis alerts.

For each question, we presented definitions and examples for each alert message category identified by applying Checkov [2], a static analysis tool, on Company A's Terraform repository. We derive alert message categories by applying open coding on the collected 74 alert messages from Company A's Terraform repository. Open coding is a qualitative analysis technique that is used to identify commonalities in structured or unstructured text [28]. Upon application of open coding, we derive five alert message categories for which at least five instances were reported. We select the threshold of five instances to exclude alert message categories that were infrequent in Company A's Terraform repository. Table II presents the names and definitions for each derived alert message category.

Participants of the survey are practitioners working full-time at Company A. We invite all Terraform developers at Company A who worked on the proprietary repository to a Zoom channel where we inform them about voluntary participation of our survey.

B. RQ2: What categories of static analysis alerts appear in open source and proprietary Terraform manifests? How frequently do alerts appear for each category?

We answer RQ2 using the following steps:

TABLE I: Overview of the Studied Open Source and Proprietary Terraform Repositories

Name	# Commits	# TF Dirs.	# Tf Files	# TF Ent.	# TF Reso.
terraform-aws-modules/terraform-aws-eks	934	14	57	916	58
poseidon/typhoon	1,523	18	132	981	391
aws-ia/terraform-aws-eks-blueprints	1,182	97	455	2187	197
philips-labs/terraform-aws-github-runner	1,996	22	113	613	115
nozaq/terraform-aws-secure-baseline	344	17	85	523	103
GoogleCloudPlatform/cloud-foundation-fabric	3,756	162	756	4279	560
kube-hetzner/terraform-hcloud-kube-hetzner	1,083	2	18	231	34
maddevsio/aws-eks-base	490	8	63	431	74
kbst/terraform-kubestack	557	35	153	762	92
cattle-ops/terraform-aws-gitlab-runner	800	9	45	378	60
Proprietary Repository	1,367	57	227	1,941	383

TABLE II: Alert Message Categories Used in the Survey

Category	Definition
Key management	Messages related to key management
Logging	Messages related to logging
Permission of IAM Policies	Messages related to permission of identity access management (IAM)
Replication	Messages related to enabling cross-region replication
Versioning	Messages related to data versioning

1) Repository Collection: For answering RQ2, we use two sources: one proprietary Terraform repository created and maintained by Company A, and 10 OSS repositories with Terraform manifests. As use of one repository is limiting for answering RQ2, we also select top 10 OSS Terraform repositories hosted on GitHub based on star count. First and third authors of the paper use GitHub topics [1] to identify the top 10 repositories, so that repositories used for education purposes and with limited activity, as measured by one commit per month, are not included in our analysis. Table I provides an overview of the Terraform repositories used in our analysis. The 'Name', 'Commits', 'TF Dirs.', 'Tf Files', 'TF Enti.', and 'TF Reso.' columns respectively, represent the name, commit count, Terraform directories, entities, files, and resources for each repository.

2) Application of Static Analysis: We use Checkov [2] to apply static analysis. Listing 1 shows an example alert generated by Checkov in Static Analysis Results Interchange Format (SARIF) [1]. In Listing 1, *ruleId*, *level* and *message* respectively, provides the alert name, alert importance, and detailed information for the alert of interest. Also, using *uri*, *startLine*, and *endLine*, Checkov reports the location of the alert.

Manual Verification: Static analysis tools are susceptible to generating false positives. Checkov is no different. False positives generated by Checkov could influence results for RQ2. We mitigate this risk by manually inspecting each generated alert by Checkov for Company A's Terraform repository. Checkov generated 74 alerts in total for Company A's Terraform repository. The last author conduct manual verification and observe no false positive instances for the collected 74 alerts. Here, a false positive instance is defined as an alert that is not by any entity within a module.

[1]https://github.com/topics/terraform?l=hcl

```
{
    "ruleId": "CKV_AWS_21",
    "level": "error",
    "message": {
    "text": "Ensure all data stored in the
    S3 bucket have versioning enabled"
    },
    "locations": [
    {
      "physicalLocation": {
        "artifactLocation": {
        "uri":
        "terraform/layer1-aws/aws-cloudtrail.tf"
        },
        "region": {
          "startLine": 14, "endLine": 18
        }
      }
    }
    ]
}
```

Listing 1: Example partial static analysis alert generated by Checkov in SARIF format.

3) Alert Quantification: We use two metrics to answer RQ2. *First*, we report the proportion of alerts that belong to each alert category. *Second*, we report the distribution of alert count per module for all Terraform modules in our repositories with minimum, maximum, average, and standard deviation.

C. RQ3: How do static analysis alerts in Terraform manifests correlate with resources that have dependencies?

We answer RQ3 by *first* constructing a Terraform entity dependency graph. *Second*, we compute graph metrics to determine the correlation between static analysis alerts and entity dependencies. We describe these steps in the following subsections.

1) Construction of Terraform Entity Dependency Graph: The Terraform entity dependency graph is a directed graph with four types of nodes: (i) directory of the Terraform manifest, (ii) Terraform manifest, (iii) a non-Terraform file, and (iv) entity in Terraform manifest. An edge exists between nodes if there are 'has' or 'depends on' relationships between nodes, e.g., if a directory has a Terraform file, then an edge will exist between a directory node and a Terraform file node. The

Terraform entity dependency graph is constructed for each of the 10 OSS repositories and the Company A repository. We used the 'networkx' library [2] to construct the Terraform entity dependency graph.

Figure 1 is an illustration of a Terraform entity dependency graph based on the code snippet presented in Listing 1. In Figure 1, green, red, and blue nodes represent a file system directory, Terraform file and Terraform entity, respectively. Red and green edges represent the 'has' and 'depends_on' relationships. For example, our graph shows that *docker_container.nginx*, a Terraform resource in file main.tf, depends on *var.nginx_ext_port* in another Terraform file *variables.tf* in the same directory.

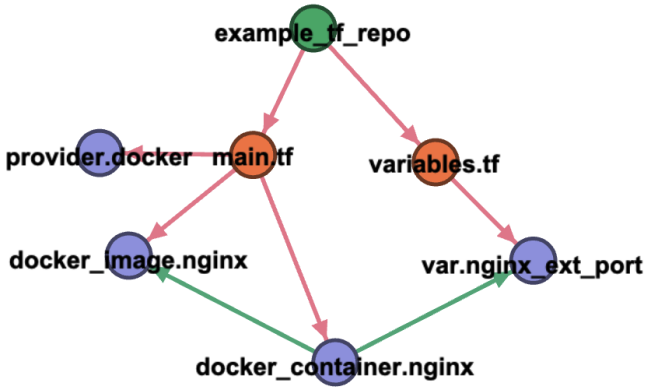

Fig. 1: Terraform entity graph for the code snippet presented in Listing 1.

2) Hypothesis Evaluation: We hypothesize that Terraform resources with dependencies are correlated with the count of static analysis alerts. We evaluate our hypothesis by *first* calculating a centrality metric called PageRank [16]. *Second*, we apply statistical tests to identify any quantitative correlation.

PageRank Calculation: PageRank is a metric used to measure the importance of nodes based on the structure of incoming links from other nodes. We compute PageRank for each Terraform resource included in the constructed dependency graph. We repeat this calculation for each of the 10 OSS repositories and the Company A repository. A relatively higher PageRank score for a resource shows the resource to have more dependencies than others.

Hypothesis Testing: We first apply Shapiro-Wilk test [29] to assess normality of obtained PageRank scores. The results of Shapiro-Wilk test show that the obtained PageRank scores do not follow normal distribution. Therefore, we apply Mann-Whitney U test [13] to answer if the difference between two groups of centrality scores are statistically significant. We state the following null and alternate hypotheses:

- *Null*: PageRank is not larger for Terraform entities with static analysis alerts.
- *Alternate*: PageRank is larger for Terraform entities with static analysis alerts.

If $p - value < 0.05$, we reject the null hypothesis, accept the alternative hypothesis, and report the difference to be statistically significant.

[2]https://github.com/networkx/networkx

IV. Results

In this section, we provide answers to our research questions.

A. RQ1: How do practitioners within a proprietary organization perceive the actionability of Terraform-related static analysis alerts?

In all, 10 practitioners agree to participate in our survey. Out of 10 practitioners, 3 reported 1 year, 5 reported 1-2 years, and 2 reported 3-4 years of experience in Terraform.

We answer RQ1 using Figure 2. From Figure 2, we observe majority of the survey respondents to understand all five categories. In the case of root cause analysis, we observe majority of the survey respondents to agree that the generated alert messages provide information to understand the root cause. For example $>= 8$ practitioners understood the root cause for logging, key management, replication, and versioning.

Compared to understandability and root cause analysis, the opinion is more divided for willingness to fix, i.e., whether or not the surveyed practitioners will act upon the reported alert, and fix the bug. While 9 out of 10 respondents agreed to fix logging-related alerts, 5 out of 10 respondents were not willing to fix permission of IAM policies and replication. The alert messages regarding IAM policies and replication are respectively, "*Ensure access is controlled through single sign-on (SSO) and not AWS IAM defined users*" and "*Ensure that S3 bucket has cross-region replication enabled*". As explanations, two practitioners mentioned that they do not have the authority to alter authentication or replication mechanism despite the alert messages being valuable. The implication of this finding is that static analysis tools for Terraform manifests need to account for practitioner context.

In short, from our survey analysis we observe that Terraform-related static analysis alerts support understandability and root cause analysis, but acting upon these alerts is limited for certain alert categories.

B. RQ2: What categories of static analysis alerts appear in open source and proprietary Terraform manifests? How frequently do alerts appear for each category?

We answer RQ2 by reporting the categories of static analysis alert categories and their corresponding frequency in Table III. We observe 5 of the 10 static analysis alert categories to be related to security: application security, encryption, general security, IAM, and supply chain. The total counts of alerts for Company A's repository and OSS repositories are respectively, 74 and 417.

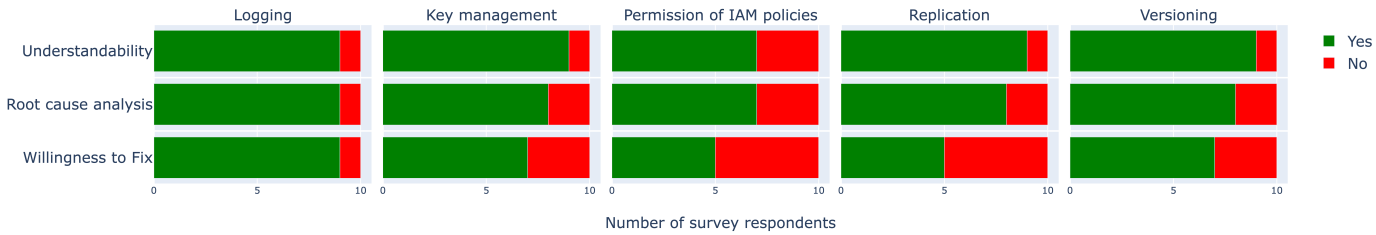

Fig. 2: Answer to RQ1: Practitioners' perceptions on understandability (SQ1), root cause information (SQ2), and willingness to fix (SQ3) for the five alert message categories.

TABLE III: Answer to RQ2: Alert Categories with Definitions and Frequency

		Alert Proportion		Alert Dist. (Min, Max, Avg., Std. Dev.)	
Category	**Definition**	**Company A**	**OSS**	**Company A**	**OSS**
Application security	Identify potential vulnerabilities in the application layer	0.0%	0.7%	(0.0, 0.0, 0.0, 0.0)	(0.0, 0.1, 0.0, 0.0)
Backup and recovery	Ensure proper backup and recovery practices	12.2%	1.7%	(0.0, 7.0, 1.0, 2.3)	(0.0, 1.3, 0.2, 0.5)
Convention	Checks for common best practices and conventions	1.4%	0.0%	(0.0, 1.0, 0.1, 0.3)	(0.0, 0.1, 0.0, 0.0)
Encryption	Ensures that data is properly encrypted both in transit and rest	21.6%	9.6%	(0.0, 7.0, 1.8, 2.2)	(0.1, 2.4, 0.7, 0.9)
General security	Covers a broad range of security checks that are not covered in other categories	12.2%	18.9%	(0.0, 7.0, 1.0, 2.3)	(0.1, 3.8, 1.0, 1.3)
IAM	Evaluates the management of user identities, permissions, and access control	18.9%	8.6%	(0.0, 5.0, 1.6, 1.7)	(0.0, 1.7, 0.5, 0.7)
Kubernetes	Evaluates Kubernetes configurations such as pod security, network policies, and role-based access control	0.0%	4.1%	(0.0, 0.0, 0.0, 0.0)	(0.0, 1.1, 0.3, 0.4)
Logging	Ensures proper logging and monitoring practices	25.7%	13.7%	(0.0, 8.0, 2.1, 3.1)	(0.0, 4.2, 0.7, 1.3)
Networking	Evaluates network configurations such as subnet setup, security group rules, and access control	8.1%	41.5%	(0.0, 3.0, 0.7, 1.1)	(0.0, 7.3, 1.5, 2.7)
Supply Chain	Checks for vulnerabilities in the software supply chain	0.0%	1.2%	(0.0, 0.0, 0.0, 0.0)	(0.0, 0.3, 0.1, 0.1)

According to Table III, the most frequently occurring alert category is Logging and Networking respectively, for Company A's repository and OSS repositories. We also observe certain alert categories to be present in OSS repositories but not in Company A's repository, namely, application security, Kubernetes, and supply chain.

For the OSS repositories, the standard deviation for alert/-module is higher than average for seven categories: backup and recovery, encryption, general security, IAM, Kubernetes, logging, and networking. For Company A's repository, the standard deviation for alert/module is higher than average for seven categories: backup and recovery, convention, encryption, general security, IAM, logging, and networking.

C. RQ3: How do static analysis alerts in Terraform manifests correlate with resources that have dependencies?

We provide answers to RQ3 using Table IV where we report the p-values and the PageRank score distribution for resources with alerts ('Stats [w/alerts]') and resources without alerts ('Stats [wo/alerts]').

Except for one, for all OSS Terraform repositories and for Company A's repository, Terraform resources with alerts have higher PageRank scores than Terraform resources without alerts. Results shown in Table IV provide empirical substantiation to our hypothesis that resources with dependencies are correlated with static analysis alert frequency. Considering all of the 8 OSS repositories, the median PageRank score for Terraform resources is 0.0016, which is 2.1 times higher than resources with no alerts. In the case of Company A's repository, the median PageRank score is 1.5 times higher for resources with alerts compared to resources for no alerts.

V. Discussion

We discuss our findings by first reporting the lessons learned from our empirical study. Next, we discuss Company A's future plans of using the constructed Terraform entity dependency graph. Finally, we discuss the limitations of our paper.

A. Lessons Learned

In the following subsections, we report the lessons learned from our empirical study.

1) Actions to Fix Identified Bugs: Figure 2 showcases majority of the survey respondents to find provided alerts understandable and helpful for root cause analysis. However, whether or not the reported alert will be fixed is dependent on the alert message category. This finding is consistent with Rahman et al. [20], [21]'s findings, who reported practitioners to not agree with all categories of security defects for IaC

TABLE IV: Answer to RQ3: Correlation between entity dependency and alert frequency

Repository	p-val	Stats [w/alerts] (Min, Median)	Max,	Stats [wo/alerts] (Min, Median)	Max,
aws-eks-base	0.012	(0.0012, 0.0014)	0.0063,	(0.0012, 0.0013)	0.0139,
cloud-foundation-fabric	0.001	(0.0001, 0.0002)	0.0011,	(0.0001, 0.0001)	0.0010,
terraform-aws-eks	0.001	(0.0007, 0.0026)	0.0099,	(0.0007, 0.0010)	0.0076,
terraform-aws-eks-blueprints	0.085	(0.0002, 0.0004)	0.0015,	(0.0002, 0.0003)	0.0018,
terraform-aws-github-runner	0.001	(0.0009, 0.0016)	0.0051,	(0.0007, 0.0008)	0.0067,
terraform-aws-gitlab-runner	< 0.001	(0.0031, 0.0079)	0.0106,	(0.0014, 0.0015)	0.0072,
terraform-aws-secure-baseline	< 0.001	(0.0012, 0.0025)	0.0088,	(0.0009, 0.0012)	0.0089,
terraform-kubestack	0.001	(0.0007, 0.0017)	0.0040,	(0.0005, 0.0007)	0.0037,
typhoon	< 0.001	(0.0004, 0.0004)	0.0171,	(0.0004, 0.0005)	0.0140,
Company A	< 0.001	(0.0001, 0.0003)	0.0019,	(0.0001, 0.0002)	0.0070,

manifests. The implication of this finding is that actionability of static analysis could be improved by accounting for alert categories.

Lesson#1 - The alert category is a contributing factor when practitioners take actions for Terraform-related static analysis alerts.

2) Implications of Resource Dependencies for Fix Prioritization: Results in Table IV show resource dependencies to correlate with alert frequency for 10 out of 11 repositories. This finding has implications for inspection and fix prioritization. When reporting static analysis alerts, tools can provide higher priority for resources that have dependencies. In this manner, practitioners can learn which alerts to address and fix first when multiple alerts are reported.

Lesson#2 - Terraform-related alerts appear more often for resources with higher dependencies than others.

3) Company A's Future Plans Regarding the Terraform Entity Dependency Graph: First author of the paper, who is a practitioner from Company A, have demonstrated the construction of Terraform entity dependency graph in an internal company meeting. The meeting generated interest among practitioners in applying the graph for the following utilities: (i) *visualization*: as visualization of infrastructure reduces complexity in design and implementation [12], [24], the Terraform entity dependency graph can aid Company A to design and implement computing infrastructure more efficiently; and (ii) *change impact analysis*: practitioners perceive the Terraform entity dependency graph as a useful aid in analyzing the impact of code changes, which in turn can be helpful for cost estimation and change propagation.

Lesson#3 - Practitioners from Company A perceive the Terraform entity dependency graph to be useful for visualization and change impact analysis for Terraform-based computing infrastructure.

B. Threats to Validity

This subsection discusses the limitations of our study.

- **Internal Validity** We use Checkov to obtain static analysis alerts. As static analysis tools tend to generate false positives, RQ2-related results could be biased. We mitigate this limitation by manually inspecting all 74 alerts obtained for Company A's repository, for which we observe no false positives.

  The Terraform entity dependency graph is susceptible to generate errors when a Terraform directory contains errors, which could impact the results for RQ3. We mitigate this risk by using 11 Terraform repositories with 441 Terraform directories, so that the impact of validation failures is low.

- **External Validity** We analyze 10 open source and 1 proprietary Terraform repositories using one tool called Checkov. Use of one tool is susceptible to external validity as it may miss defect categories. Findings from RQ1 are limited to the perceptions of 10 practitioners from one organization, which may not generalize for other practitioners. Also, results for RQ2 and RQ3 are limited to the 11 repositories that we analyzed.

VI. Conclusion

Despite Terraform's prominence in implementing IaC, characteristics of static analysis for Terraform manifests remains under-explored. In this empirical study, we have addressed this gap by analyzing the static analysis alerts that occur in 10 OSS repositories and one proprietary Terraform repository, which is owned by Company A. We have observed that the majority of the surveyed practitioners to understand alert messages across all categories. We have observed context of the alert to play a role when determining whether or not to fix the defect identified by the alert. We also have observed static analysis alerts to appear more frequently for Terraform resources with dependencies. Our findings lay the groundwork on how to improve static analysis alerts for Terraform manifests, by considering the alert message category and resource dependencies.

Acknowledgments

We thank the PASER group at Auburn University for their valuable feedback. This research was partially funded by the U.S. National Science Foundation (NSF) Award # 2247141, Award # 2310179, and Award # 2312321.

REFERENCES

[1] "Static Analysis Results Interchange Format (SARIF) Version 2.0," 2019.

[2] "bridgecrewio/checkov," May 2023, original-date: 2019-11-27T08:55:14Z. [Online]. Available: https://github.com/bridgecrewio/checkov

[3] F. A. Bhuiyan and A. Rahman, "Characterizing co-located insecure coding patterns in infrastructure as code scripts," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering Workshops*, ser. ASE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 27–32. [Online]. Available: https://doi.org/10.1145/3417113.3422154

[4] C. Dimastrogiovanni and N. Laranjeiro, "Towards understanding the value of false positives in static code analysis," in *2016 Seventh Latin-American Symposium on Dependable Computing (LADC)*, 2016, pp. 119–122.

[5] Hashicorp, "Cracking the code to global success." [Online]. Available: https://www.hashicorp.com/case-studies/github

[6] ——, "Lending a helping hand." [Online]. Available: https://www.hashicorp.com/case-studies/asian-development-bank

[7] ——, "Syntax - configuration language: Terraform: Hashicorp developer." [Online]. Available: https://developer.hashicorp.com/terraform/language/syntax/configuration

[8] S. Heckman and L. Williams, "A model building process for identifying actionable static analysis alerts," in *2009 International Conference on Software Testing Verification and Validation*, 2009, pp. 161–170.

[9] A. Hortlund, "Security smells in open-source infrastructure as code scripts: A replication study," 2021.

[10] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[11] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 672–681. [Online]. Available: http://dl.acm.org/citation.cfm?id=2486788.2486877

[12] P. Lourenço, J. P. Dias, A. Aguiar, and H. S. Ferreira, "Cloudcity: A live environment for the management of cloud infrastructures," in *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2019.

[13] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[14] G. McGraw, "Software security," *Building security in*, 2006.

[15] R. Opdebeeck, A. Zerouali, and C. De Roover, "Smelly variables in ansible infrastructure code: Detection, prevalence, and lifetime," in *Proceedings of the 19th International Conference on Mining Software Repositories*, ser. MSR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 61–72. [Online]. Available: https://doi.org/10.1145/3524842.3527964

[16] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[17] S. Pandya and R. Guha Thakurta, "Hands-on infrastructure as code with hashicorp terraform," in *Introduction to Infrastructure as Code: A Brief Guide to the Future of DevOps*. Springer, 2022, pp. 99–133.

[18] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Information and Software Technology*, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584918302507

[19] A. Rahman and C. Parnin, "Detecting and characterizing propagation of security weaknesses in puppet-based infrastructure management," *IEEE Transactions on Software Engineering*, pp. 1–18, 2023.

[20] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *Proceedings of the 41st International Conference on Software Engineering*, ser. ICSE '19. Piscataway, NJ, USA: IEEE Press, 2019, pp. 164–175. [Online]. Available: https://doi.org/10.1109/ICSE.2019.00033

[21] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams, "Security smells in ansible and chef scripts: A replication study," *ACM Trans. Softw. Eng. Methodol.*, vol. 30, no. 1, Jan. 2021. [Online]. Available: https://doi.org/10.1145/3408897

[22] A. Rahman, S. I. Shamim, D. B. Bose, and R. Pandita, "Security misconfigurations in open source kubernetes manifests: An empirical study," *ACM Trans. Softw. Eng. Methodol.*, dec 2022, just Accepted. [Online]. Available: https://akondrahman.github.io/files/papers/tosem-k8s.pdf

[23] S. Reis, R. Abreu, M. d'Amorim, and D. Fortunato, "Leveraging practitioners' feedback to improve a security linter," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3560419

[24] R. Rohan and D. Amey, "Visualize your aws infrastructure with amazon neptune and aws config," May 2021. [Online]. Available: https://aws.amazon.com/blogs/database/visualize-your-aws-infrastructure-with-amazon-neptune-and-aws-config/

[25] N. Saavedra and J. a. F. Ferreira, "Glitch: Automated polyglot security smell detection in infrastructure as code," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3551349.3556945

[26] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at google," *Commun. ACM*, vol. 61, no. 4, p. 58–66, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3188720

[27] ——, "Lessons from building static analysis tools at google," *Communications of the ACM*, vol. 61, no. 4, pp. 58–66, 2018.

[28] J. Saldana, *The Coding Manual for Qualitative Researchers*. SAGE, 2015.

[29] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[30] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 189–200. [Online]. Available: http://doi.acm.org/10.1145/2901739.2901761