Fortifying Cloud DevSecOps security using terraform infrastructure as code analysis tools

Fortifying Cloud DevSecOps Security Using Terraform Infrastructure as Code Analysis Tools

1st Rashika Singh
School of Computing and Eng
University of West London
United Kingdom
rashikasingh64@gmail.com

1st Abel Yeboah-Ofori
School of Computing and Eng
University of West London
United Kingdom
Abel.yeboah.ofori@uwl.ac.uk

2nd Saurabh Kumar
BTech in Computer Sc.and Eng
University of Allahabad
India
saurabhkumar4829@gmail.com

3rd Aishat Ganiyu
School of Eng, Phys and Math
Royal Holloway University
United Kingdom
aishat.ganiyu.2021@live.rhul.ac.uk

***Abstract*—Fortifying Cloud Security has become inevitable due to challenges such as misconfigurations, coding errors, and compromised secrets or passwords that impact infrastructure as a service during infrastructure such as code automation (IaC). These challenges require code analysis tools to enhance security during infrastructure automation. Setting up a simple cloud architecture is quick, but human errors are still common, especially when cloud infrastructure can be deployed with just a few clicks. Terraform provides a ready-made infrastructure as code modules to build and scale cloud-hosted applications. However, cyber attackers exploit these vulnerabilities and gain access to sensitive data or resources without authorization due to configuration errors, inadequate storage, and infrastructure manipulation, resulting in unauthorized deployments or alterations. That affects the availability of resources during infrastructure deployment using attacks such as DoS attacks, injection attacks, Man in the Middle (MITM), malware spread, remote code execution (RCE), and phishing attacks to penetrate the cloud infrastructures. The paper aims to analyze Terraforms infrastructure as code in cloud security to fortify codes and assist DevSecOps engineers in identifying misconfiguration in Terraform scripts. The paper's contributions are threefold. First, we explore cloud security by securing IaC solutions on Terraform. We consider security issues, including misconfigurations and coding errors, present in Terraform IaC. Secondly, we implement a static analysis tool for terraform by comparatively analyzing existing tools. Finally, we provide a comparative analysis of terraform IaC on tools including Checkov, Tfsec, Tflint, and Terrascan for suitability based on their key features and performance metrics to enhance security.**

Keywords— Terraform, Cloud Security, IaC, Coding Error, Cyber Security, Static Analysis Tool, DevSecOps

I. Introduction

Ensuring cloud security is critical in securing operating systems, web servers, servers, storage systems, and database servers in a cloud network environment to ensure connectivity without interference from attacks or errors during operational processes [1]. Cloud Security in Infrastructure as a Service (IaaS) with tools like Terraform entails safeguarding cloud infrastructure against potential threats and vulnerabilities. It involves implementing security measures to protect resources, continuous monitoring for anomalies, and automated recovery mechanisms in case of failures. HashiCorp. developed the infrastructure-as-code software tool Terraform, which has been used to improve security during cloud infrastructure deployment to support business IaaS, SaaS, and PaaS [1][2][3]. Built-in security and compliance variables in Terraform modules help harden and safeguard cloud infrastructure when configured according to best practices for cloud security. Resources could be publicly exposed if these variables mistakenly remain undefined or contain any misconfiguration error, thus compromising the production cloud environment, ultimately leading to risk. Several static code analysis tools are available to examine Terraform code for security flaws and violations of best practices. As an infrastructure-as-code tool, Terraform plays a critical role by enabling the codification of security best practices, making it easier to maintain a secure and resilient IaaS environment [1][2][3]. Figure 1 depicts the terraform IaC security model, how the model could be implemented on various cloud infrastructures, and how cyberattacks can be deployed to exploit the vulnerabilities.

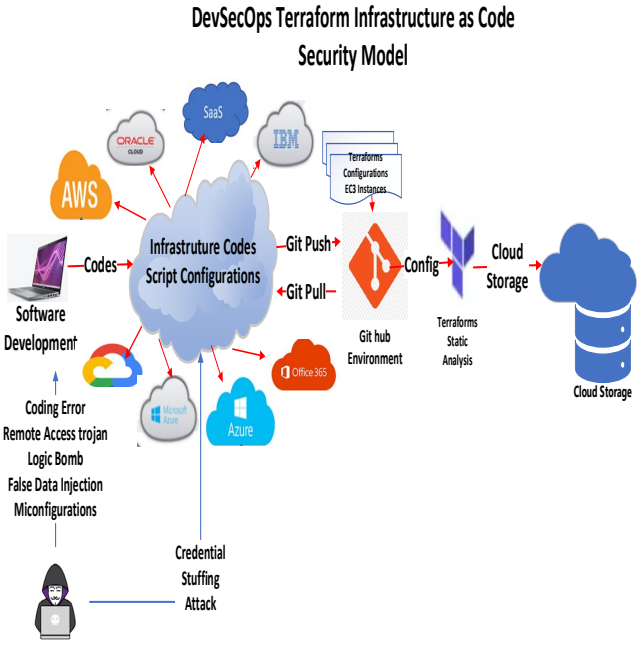

Fig. 1. Terraforms Infrastructure as Code Security Model

This approach ensures that infrastructure remains robust and can withstand disruptions, providing a dependable foundation for applications and services hosted in the cloud. However, there are existing challenges with IaC solutions like Terraform that impact the cloud infrastructure, leading to various vulnerabilities. Some of these challenges include:

- Misconfigurations
- Coding errors
- Hardcoded secrets and passwords
- Excessive data access permissions for employees who don't require it.
- Unrestricted access to a critical S3 bucket.
- Unnecessary open network ports.

- Neglecting essential security patch updates

A significant issue in Terraform frameworks is that DevSecOps engineers often write code configurations without being fully aware of potential misconfigurations and their associated risks, which can lead to unnoticed vulnerabilities and potential attacks such as Denial of Service (DoS) attacks, Injection attacks, Man in the middle (MITM) , Malware spread, Remote code execution (RCE), Phishing attacks to penetrate the cloud infrastructures. In cloud platforms, misconfigurations are one of the critical reasons that result in significant data breaches. The goal is to address such problems to secure the cloud infrastructure from data breaches. The paper aims to analyze Terraforms infrastructure as code in cloud security to fortify codes and assist DevSecOps engineers in identifying misconfiguration in Terraform scripts. The paper's contributions are threefold. First, we explore cloud security by securing IaC solutions on Terraform. We consider security issues, including misconfigurations and coding errors, present in Terraform IaC. Secondly, we implement a static analysis tool for terraform by comparatively analyzing existing tools. Finally, we provide a comparative analysis of terraform IaC on tools including Checkov, Tfsec, Tflint, and Terrascan for suitability based on their key features and performance metrics to enhance security.

II. RELATED WORKS

The section discusses the state-of-the-art and existing literature about exploring cloud security to secure IaC solutions on Terraform. We consider security issues, including misconfigurations and coding errors, present in Terraform IaC. Furthermore, we discuss a static analysis tool for terraform configuration platforms by analyzing existing tools. For instance, Andrei-Cristian Iosif [3] examines vulnerabilities in cloud deployments, focusing on the most vulnerable resources. Analyzing AWS as a cloud provider and Terraform as an IaC framework, the study reviewed 8256 public repositories using tfsec, checkov, and terrascan, uncovering 292,538 security breaches. The researchers identified instances, modules, and security groups as the top three most vulnerable resources, with S3 buckets ranking fourth. However, the study's limitation is that it only considers AWS, leaving open the question of whether these findings apply to other cloud providers [3]. The term “code smells,” coined by Kent Beck and Martin Fowler, refers to code defects that might cause issues. T. Sharma's paper was the first to introduce this concept in infrastructure as code (IaC). The study aimed to detect common implementation and design issues in Puppet scripts, analyzing 4621 Git repositories. They identified 24 issues, using Puppet-lint for implementation defects and a custom tool, "Puppeteer," for design issues. Common implementation problems included improper quote usage, misalignment, and lengthy statements, while design issues involved deficient modularization and multilayered abstraction. The study did not explore the correlation between code smells and actual defects in IaC scripts and was limited to security defects in Puppet scripts [4]. J. Schwarz [5] extended T. Sharma's research on security issues in Infrastructure as Code (IaC) scripts by analyzing Chef scripts. They examined Chef scripts from over 3200 official cookbooks and 35 industrial partner repositories using the "Foodcritic" linting tool. The study identified common security issues such as improper alignment, lengthy statements, and misplaced attributes, similar to findings in Puppet scripts by T. Sharma. However, unlike Puppet, improper quote usage was less common in Chef scripts. The paper focused exclusively on configuration management tools like Puppet and Chef, without investigating configuration orchestration tools like Terraform [5]. J. Lepiller [6] introduced the intra-update sniping vulnerability in Infrastructure as Code (IaC) services. This type of vulnerability occurs when an infrastructure update process moves through unsafe intermediate stages despite transitioning between secure ones, such as updating components out of sequence. They developed Hayha, a tool focused on identifying and recommending secure update practices within AWS CloudFormation. While effective for its purpose, Hayha's evaluation was limited to CloudFormation templates. It did not address broader configuration vulnerabilities or other IaC orchestration tools like Terraform. A. Rahman [7] conducted in-depth research on secret management in Infrastructure as Code (IaC) scripts focusing on best practices to enhance security in DevOps workflows. They analyzed 38 artifacts from grey literature sources like blogs and videos to identify 12 practices for IaC secret management. These practices include both tool-agnostic approaches like access control and tool-specific methods such as using Hashicorp Vault. The study recommends leveraging language-specific tools like Hiera for Puppet scripts and universal solutions like Hashicorp Vault, which is compatible across all IaC languages. However, it acknowledges that these practices may not cover all possible approaches, and their effectiveness can vary depending on specific IaC implementations. The study by M. Chiari reviews [8] static analysis methods for Infrastructure as Code (IaC) scripts, highlighting popular approaches like model verification, machine learning, and string-pattern rules. It outlines targeted platforms and defect categories but lacks a thorough evaluation of tool efficacy and dynamic analysis methods. Further research is needed to fill these gaps and provide a comprehensive understanding of current practices. The research by Antunes [9] analyzed Docker's security vulnerabilities and the effectiveness of static code scanners in its codebase. They evaluated security reports, categorizing vulnerabilities by causes, impacts, and risks, revealing risks like bypass and privilege escalation. However, the scope was limited to a few security reports and issues within Docker, potentially not representing all vulnerabilities. It also focused solely on static code analyzers without exploring other security solutions. The paper by Lawall [10] discusses enhancing infrastructure software security using the code-matching and transforming tool Coccinelle. The authors advocate for increased use of static analysis to detect programming flaws before software deployment. Coccinelle simplifies the creation of static analysis algorithms and automates source code inspection. However, the study lacks empirical evidence on Coccinelle's effectiveness in improving software security and does not compare it with similar tools or techniques. The paper by A. Rahman [11] examines how infrastructure as code (IaC) scripts can inadvertently introduce vulnerabilities, termed "security smells," leading to potential security breaches. The study introduces the Security Linter for Infrastructure as Code Scripts (SLIC) tool through empirical analysis and static analysis techniques. SLIC identifies seven security smells in IaC scripts, detecting 21,201 instances across a dataset of 15,232 scripts from 293 open-source repositories. The study submitted bug reports for 1,000 instances, receiving 212 responses, with 148 acknowledging and addressing the issues. However, the research focuses exclusively on security smells, limiting coverage of all possible vulnerabilities and relying solely on open-source data, potentially affecting broader applicability. The study also lacks detailed remediation strategies and does not explore underlying causes comprehensively, highlighting these as important considerations. The paper by L. Williams [12] focuses on identifying and categorizing security smells in Infrastructure as Code (IaC) scripts as indicators of potential security vulnerabilities. Using static analysis and the IaC-Sec tool, the study analyzed 1,000 IaC scripts from GitHub, identifying

67,801 instances of security smells, including 9,175 occurrences of hard-coded passwords, deprecated functions, and weak cryptography. However, the study's exclusive focus on security smells limits its evaluation of broader script quality aspects such as performance optimization and maintainability. These dimensions are crucial for ensuring efficient script execution and long-term adaptability in dynamic cloud environments, suggesting a more comprehensive analysis approach is needed. Another paper by A. Rahman [13] examines common errors in Infrastructure as Code (IaC) scripts through three initial studies based on defect data from open-source repositories. It quantifies the frequency and categorizes defects, primarily focusing on syntax and configuration assignments. The study identifies three consistent operations indicative of defective IaC scripts, laying the groundwork for proposed studies on process anti-patterns and security-related anti-patterns in IaC [12]. However, the paper does not present outcomes or results from these proposed investigations, which is a notable limitation. The paper by Alghofaili [14] conducts a survey addressing security concerns across various tiers of cloud infrastructure and reviews existing literature solutions for mitigation. It emphasizes the pervasive security challenges in cloud computing, highlighting gaps in current research and suggesting areas for further exploration to enhance cloud system safety. The study provides an overview of existing literature on cloud infrastructure security but does not present new research findings or empirical data. It suggests potential solutions without comprehensive effectiveness analysis and may not cover all possible security issues. Despite its focus on prominent concerns, it offers a broad perspective on current research in cloud infrastructure security

III. Approach

This section discusses the implementation process and the approach used for the paper from data collection to the security assessment on static analysis tools for terraform. The paper systematically evaluates misconfiguration issues across SadCloud, CloudGoat2, and TerraGoat cloud environments. It rigorously assesses the effectiveness of four static code analysis tools—tfsec, tflint, checkov, and terrascan—using diverse datasets and vulnerable cloud setups. This research requirement underscores our selection of a quantitative methodology to ensure meticulous analysis and reliable findings. The chosen quantitative approach facilitates determining tool effectiveness by calculating absolute values, true positives, and false positives from security checks conducted by each tool. The evaluation of these metrics, coupled with the large dataset and varied cloud environments, aligns with the quantitative approach's objective of generating reliable, numeric insights. By quantifying the results and employing statistical analysis, the research aims to provide objective and generalized findings, making the quantitative methodology suitable for deriving concrete conclusions and contributing valuable insights to the field.

A. Data Collection Approach

Data utilized in this study is sourced from open-source libraries. The methods described below are employed to execute the data collection process: Search and selection or Terraform script were accomplished using one of two web scraping techniques.

GitHub's API query: The exploration of the GitHub repository database was conducted to obtain a list of repositories that contain terraform code. This was done through the website's API, using Terraform's Hashicorp Configuration Language (HCL) syntax. GitHub's API allows targeted searches based on timeframes and language filters, with each query yielding up to 1000 results spread across ten pages [1]. The query for gathering the links is constructed using string-interpolated query parameters as follows:

api.github.com/search/repositories?q=language:HCL&per_page=100&page={page_number}

This technique resulted in a comprehensive list of 269190 repository links identified as Terraform code, along with their relevant metadata [1].

GitHub code search interface: Files with the extensions .tf or .hcl were acquired directly from the GitHub database via the GitHub code search interface at https://github.com/search.

The query employed to retrieve the list of .tf and .hcl files is:

path:*.tf or path:*.hcl

B. Data Pre-processing

- Downloaded repositories undergo various sanity checks and filters. First, we verify if the repository contains Terraform code. This step is crucial because GitHub's labelling may yield a few false positives, such as non-Terraform or empty repositories initially appearing in the scraping results [1].
- A Python script developed automates the download process of Terraform scripts from a public repository. This collection comprises around 1000 .tf files, which were meticulously selected in the prior phase of web scraping.
- Eliminating any redundant and unrelated data

All this pre-processing and download process is carried out using a python. Subsequently, these downloaded scripts will be input for the analysis and testing process.

C. Tool Creation

In this paper, we also introduce 'terraformsolutions,' a web application-based tool developed for the analysis of Terraform scripts (.tf). This tool assesses code configurations to identify vulnerabilities and offers recommended remediations. It is implemented in Python, with the user interface (UI) built using the Django framework. Our tool encompasses checks for AWS, GCP, and Azure cloud providers, enhancing its versatility and applicability.

D. Security Analysis

This section defines the approach followed for conducting a security analysis of each tool

- **Step 1:** Metrics calculation involves analyzing each tool's precision and false discovery rates using a confusion matrix.
- **Step 2:** Tool Key feature exploration compares features like IaC platform support, security checks, cloud provider compatibility, adoption rates, Docker support, and more.
- **Step 3:** Identifying configuration issues in vulnerable cloud environments (Sadcloud, CloudGoat2, TerraGoat) through penetration testing and configuration reviews.
- **Step 4:** Comparative analysis of tools uses results from Steps 1, 2, and 3 to assess performance.
- **Step 5**: Data Visualization presents calculations and analyses using tables and graphs.
- **Step 6:** Interpretation uses results to select the best static analysis tool for Terraform.

IV. Implementations

This section discusses the implementation process and highlights the tools used to achieve objectives.

A. Security Analysis - Static Code Analysis Tools for Terraform: The paper evaluates four static code analysis tools (tfsec, tflint, checkov, terrascan) across three vulnerable cloud environments (SadCloud, CloudGoat2, TerraGoat). Performance is assessed by running these tools on 1000 Terraform files, measuring efficacy through precision, false discovery rate, true positives, and false positives from security checks. The formula for absolute value is defined as:

Absolute V = True Positive + False Positive.

This approach offers a solid basis for comparing and ranking the tools' performance in identifying security issues. Table 1 below summarizes configuration issues identified through penetration testing and configuration reviews conducted across multiple vulnerable cloud environments.

TABLE 1: Configuration Issues In Various Vulnerable Cloud Environments

	SadCloud	Cloud Goat 2	TerraGoat
Cloud Provider Supported	AWS	AWS	AWS, Azure, GCP
	Config Review/ PT	Config Review/ PT	Config Review/ PT
Total no. of Vulnerabilities	84	7	108
Storage	13	2	24
Network			4
Virtual Machine	24	1	15
Database	5		14
Access Management	15	2	2
Serverless Function		1	2
Certificate Management	1		
CI/CD Pipelines		1	
Secret Management	2		5
Container Service	2		2
Kubernetes	4		23
App Service (PaaS)	1		10
Security Center			6
Mail Service	2		
Notification Service	3		
Analytics	2		1
Management & Governance	10		

B. Tool Implementation Algorithm

This section defines the algorithm employed for the implementation of customized tool “terraformsolutions” The tool initializes critical security pattern recognition using regular expressions (regex) and utilizes Python's 'os' module for secure file upload, creating dedicated directories for isolation. It parses Terraform (.tf) files, analyzing them for S3 and EC2 security configurations, presenting results as 'failed checks' for vulnerabilities and 'passed checks' for error-free configurations.

Figure 2 below shows a Python function, `s3_Checks`, that takes `tf_data` as input and performs checks using regular expressions to find S3 bucket configurations like logging, server-side encryption, and MFA-Delete settings. The function appends the results to the `s3_Checks` list for further processing or reporting.

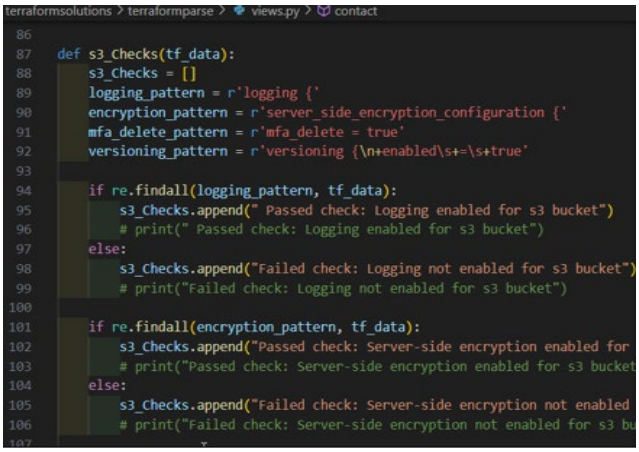

Fig. 2. Code for outlining S3 bucket rule definitions for the tool

Figure 3 illustrates a Django view function, process_file(request), managing file upload and processing. It saves the file, verifies its .tf extension, and conducts AWS S3 bucket and security group checks using predefined Terraform patterns. Results are stored in s3_checks_response and security_group_checks_response and then displayed to users via an HTML template.

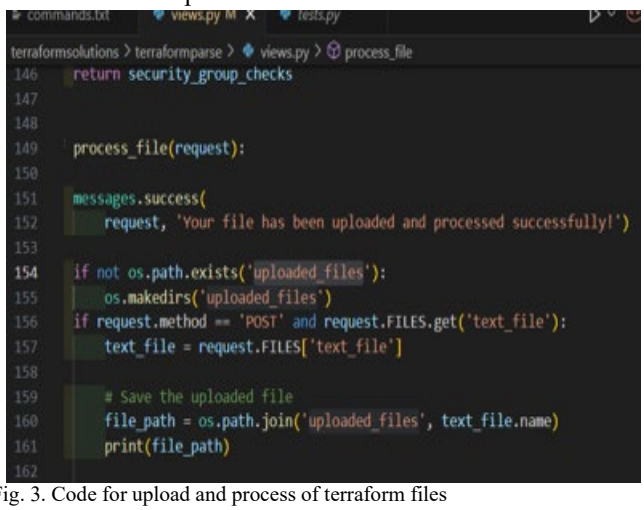

Fig. 3. Code for upload and process of terraform files

V. Result and Discussion

This section discusses how terraform security is becoming increasingly important for DevSecOps engineers to learn and implement. Static code analysis of Terraform code provides a detailed report highlighting identified issues along with their descriptions and recommended solutions. This process involves applying an extensive set of security policies and best practices, ultimately fortifying the quality and security of cloud infrastructure services.

A. Comparative Analysis of Tools Based on Key Features

Checkov, Tfsec, Tflint, and Terrascan are vital tools used for Infrastructure as Code (IaC) security analysis. Checkov, maintained by BridgeCrew, supports multiple IaC formats, including Terraform, CloudFormation, Kubernetes, and others, with over 1000 built-in policies and fast execution (<5s). Tfsec by Aqua Security focuses on Terraform with fast execution (<0.5s) and 380 built-in checks. Tflint, supporting Terraform, offers seven checks and is noted for its ease of use across different platforms. Terrascan, now Tenable, supports AWS, providing over 500 built-in checks, but is more complex to configure due to its Rego language. Each tool supports integration with CI/CD pipelines and various output formats, ensuring compatibility and ease of adoption across different environments.

B. *Customized tool "Terraformsolutions" result*

The output in Figure 4 displays the scan results showing 4 failed checks for S3 bucket vulnerabilities and 1 failed check for security group vulnerabilities detected in the uploaded sample.tf file. The sample.tf file reports 5 vulnerabilities:

- Logging not enabled for S3: Crucial for access details and security monitoring.
- Server-Side Encryption not enabled for S3: Enhances data security.
- MFA-Delete not enabled for S3: Mitigates account takeover risks.
- Versioning not enabled for S3: Prevents accidental data loss.
- Security group allows ingress from 0.0.0.0 to port 22: Risks unauthorized access; avoid unrestricted access to uncommon ports like 22.

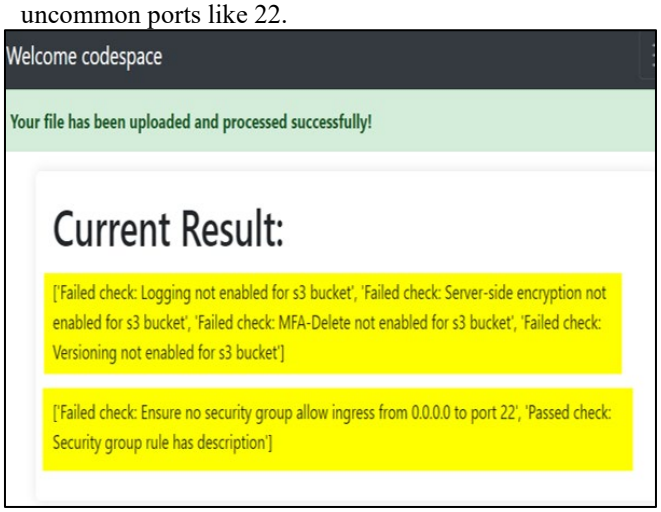

Fig 4. The scan results of Terraform file

C. *Calculations Precision and False Discovery Rates*

This research section evaluates each tool's performance through precision and false discovery rate calculations, using a confusion matrix to analyze true positives, true negatives, false positives, and false negatives. This method allows for informed comparisons to determine the most effective tool for vulnerability identification. Table 2 defines values as follows:

- True positive (TP): Issues identified by each tool against a vulnerable environment or in open-source libraries (1000 .tf files).
- False positive: Absolute value – True positive
- N/A (Not Applicable): The tool has given no result.

Absolute values (total number of vulnerabilities) of different vulnerable environments and open-source libraries:

- SadCloud = 84 misconfigurations
- CloudGoat2 = 200 misconfigurations
- TerraGoat = 108 misconfigurations
- Open-Source Libraries= 800 misconfigurations

TABLE 2: Calculated TP and FP of All the Tools

True positive (TP) and False positive (FP) rates of issues identified by each tool		Tfsec		Checkov		Terrascan		TfLint	
		TF	**FP**	**TF**	**FP**	**TP**	**FP**	**TP**	**FP**
Vulnerable Environments	SadCloud	50	34	44	40	1	83	N/A	N/A
	CloudGoat2	33	167	99	101	46	154	25	175
	TerraGoat	20	88	52	56	20	88	15	93
Open-Source Libraries	1-500 .tf files	263	537	515	285	110	690	N/A	N/A
	500-1000 .tf files	198	602	441	359	132	668	N/A	N/A

After obtaining True Positive (TP) and False Positive (FP) values for each tool, our next step involves calculating and comparing two key metrics: Precision (Positive Predictive Value - PPV) and False Discovery Rate (FDR). PPV assesses the accuracy of positive predictions, while FDR quantifies the rate of false positives. Precision or positive predictive value (PPV) = TP / TP+FP. False discovery rate (FDR) = FP/ (TP+FP) or 1- PPV. The False Discovery Rate graph in Figure 5 shows that Checkov has the lowest False Discovery Rate among all the tools studied.

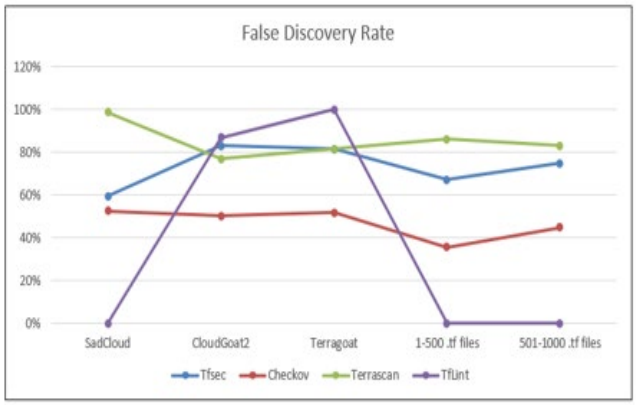

Fig. 5. False Discovery Rate Graph

Similarly, the Precision Rate graph in Figure 6 indicates that Checkov has the highest Precision Rate. Tfsec ranks second in precision, followed by Terrascan and TfLint.

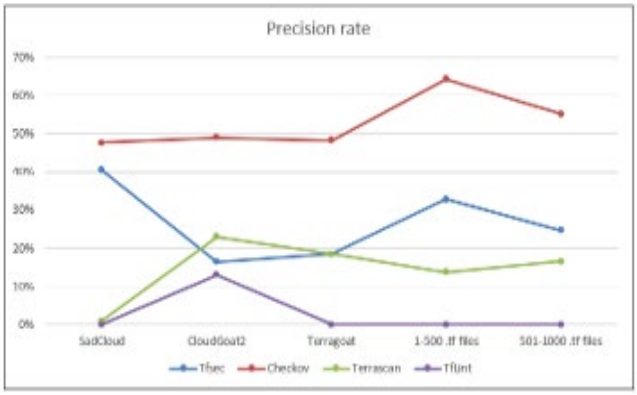

Fig.6. Precision Rate Graph

D. *Outcome of the Analysis Tools*

Based on our study, Checkov by BridgeCrew emerges as the optimal tool for Terraform IaC static code analysis among those evaluated. It excels for several reasons: Checkov supports multiple IaC platforms and cloud providers (AWS, GCP, Azure), offers an extensive library of policies, integrates seamlessly into CI/CD pipelines, and features a user-friendly interface with comprehensive documentation covering over 1000 built-in policies. Checkov maintains high precision rates and exceptional accuracy in identifying configuration issues while minimizing false positives, ensuring reliable performance. In contrast, Tfsec, Tflint, and Terrascan show less versatility, policy coverage, and documentation, with varying precision rates and inconsistent performance.

TABLE 3: Tools and their Analysis Outcome

TOOL	Remark
Checkov	Supports roughly 85 AWS and/or TF specific rules with rule suppression and GitLab CI Integration. May introduce high false positives so customization and tuning are required. No documented dependency on the TF version. BEST
TFSec	Supports roughly 24 AWS specific rules with no guidance on customization. Rule suppression but no direct GitLab integration. Need to identify how to seamlessly integrate into DevOps pipeline. Results appear on lower to average side across the tools. BETTER
TFLint	Robust ruleset, highly customizable but focusing almost solely on structural errors and invalid inputs. Does not focus on security issues. OK (this might be good to use in conjunction with a security specific tool)
Terrascan	Limited ruleset, primarily CLI support with no proper documented ability to customize. GOOD

CONCLUSION

Cloud security incidents, such as AWS S3 bucket misconfigurations, highlight how minor errors can lead to major data breaches due to inadequate access controls. The U.S. Department of Defense's accidental data leak serves as a notable example. Secure Coding Guidelines (SCGs) frameworks are crucial for secure codebases, but Infrastructure as Code (IaC) currently lacks such standards. DevSecOps integrates security throughout development stages, addressing these vulnerabilities. This study examines Terraform with tools like Checkov, Terrascan, TFlint, and TFsec, applying DevSecOps to enhance IaC security. The paper intends to find an optimal Terraform IaC static code analysis tool by evaluating existing tools. Using these tools, we conducted a security analysis on 1000 open-source terraform files from GitHub repositories and three vulnerable cloud environments: SadCloud, CloudGoat2, and TerraGoat. We calculated each tool's performance metrics, like true positives, false positives, and precision rates. Based on the evaluation results of key features and performance metrics, the study concluded that Checkov surpasses other tools and stands out as the best Terraform static code analysis tool for enhancing cloud infrastructure security. This paper aims to provide valuable insights for DevSecOps Engineers, helping them select the appropriate Infrastructure as Code (IaC) tool for integrating security early in the development process. We aim to reduce the need for repetitive exploration and research, enhancing efficiency, productivity, and decision-making. Ultimately, this will strengthen the overall security of cloud deployments.

A limitation of the current static analysis approach in this research is its inability to detect vulnerabilities that arise only during runtime, such as configuration deviations and behavioral anomalies, underscoring the need for dynamic analysis in future work. To address this, we plan to enhance Terraform analysis tools with behavioral analysis capabilities to monitor configuration behavior patterns, enabling more effective detection of security risks and deviations from best practices. That will include evaluating tool performance by identifying security checks with high false-positive rates, and refining detection accuracy and tool efficiency. Furthermore, we also aim to investigate methods for increasing awareness of security vulnerabilities in Infrastructure as Code (IaC) with a focus on Terraform and also explore the automation of remediation processes triggered by tools such as Checkov, to reduce time overhead and mitigate the potential for human error.

REFERENCES

[1] A Large-Scale Study on the Security Vulnerabilities of Cloud *Deployments. 10.1007/978-981-19-0468-4_13.*

[2] JavaTpoint: Resiliency in Cloud Computing. Available at: https://www.javatpoint.com/resiliency-in-cloudcomputing

[3] Iosif, Andrei-Cristian & Gasiba, Tiago & Zhao, Tiange & Lechner, Ulrike & Albuquerque, Maria. (2022).

[4] *T. Sharma, M. Fragkoulis and D. Spinellis, "Does Your Configuration Code Smell?,"* 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, Austin, TX, USA, 2016, pp. 189-200.*

[5] *J. Schwarz, A. Steffens and H. Lichter, "Code Smells in Infrastructure as Code," 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), Coimbra, Portugal, 2018, pp. 220-228, doi: 10.1109/QUATIC.2018.00040.*

[6] *J. Lepiller, R. Piskac, M. Sch¨af, and M. Santolucito, "Analyzing infrastructure as code to prevent intra-update sniping vulnerabilities," in Proc. 27th Int. Conf. Tools Alg. for the Constr. and Anal. of Syst*., TACAS'21, Part II, ser. LNCS, vol. 12652. Springer, 2021, pp. 105–123

[7] A. Rahman, F. L. Barsha and P. Morrison, "Shhh!: 12 Practices for Secret Management in Infrastructure as Code," 2021 IEEE Secure Development Conference (SecDev), Atlanta, GA, USA, 2021, pp. 56-62, doi: 10.1109/SecDev51306.2021.00024.

[8] M. Chiari, M. De Pascalis and M. Pradella, "Static Analysis of Infrastructure as Code: a Survey," 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C), Honolulu, HI, USA, 2022, pp. 218-225, doi: 10.1109/ICSA-C54293.2022.00049.

[9] A. Duarte and N. Antunes, "An Empirical Study of Docker Vulnerabilities and Static Code Analysis Applicability," *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, Foz do Iguacu, Brazil, 2018, pp. 27-36, doi: 10.1109/LADC.2018.00013.

[10] Lawall, Julia & Hansen, René & Palix, Nicolas & Muller, Gilles. (2010). Improving the Security of Infrastructure Software using Coccinelle. ERCIM News. 2010. 54.

[11] A. Rahman, C. Parnin and L. Williams, "The Seven Sins: Security Smells in Infrastructure as Code Scripts," *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, Montreal, QC, Canada, 2019, pp. 164-175, doi: 10.1109/ICSE.2019.00033.

[12] A. Rahman and L. Williams, "Different Kind of Smells: Security Smells in Infrastructure as Code Scripts," in *IEEE Security & Privacy*, vol. 19, no. 3, pp. 33-41, May-June 2021, doi: 10.1109/MSEC.2021.3065190.

[13] A. Rahman, "Anti-Patterns in Infrastructure as Code," *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, Västerås, Sweden, 2018, pp. 434-435, doi: 10.1109/ICST.2018.00057.

[14] Alghofaili, Y., Albattah, A., Alrajeh, N., Rassam, M.A., Al-rimy, B.A.S. (2021) Secure Cloud Infrastructure: A Survey on Issues, Current Solutions, and Open Challenges. Applied Sciences. 11(19), 9005.

[15] M. Artac, T. Borovšak, E. Di Nitto, M. Guerriero, D. Perez-Palacin and D. A. Tamburri, "Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach," *2018 IEEE International Conference on Software Architecture (ICSA)*, Seattle, WA, USA, 2018, pp. 156-15609, doi: 10.1109/ICSA.2018.00025

[16] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba, "Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry," *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Cleveland, OH, USA, 2019, pp. 580-589, doi: 10.1109/ICSME.2019.00092.

[17] A. Yeboah-Ofori, S. K. Sadat and I. Darvishi, "Blockchain Security Encryption to Preserve Data Privacy and Integrity in Cloud Environment," *2023 10th International Conference on Future Internet of Things and Cloud (FiCloud)*, Marrakesh, Morocco, 2023, pp. 344-351, doi: 10.1109/FiCloud58648.2023.00057.

[18] A. Yeboah-Ofori, I. Darvishi and A. S. Opeyemi, "Enhancement of Big Data Security in Cloud Computing Using RSA Algorithm," *2023 10th International Conference on Future Internet of Things and Cloud (FiCloud)*, Marrakesh, Morocco, 2023, pp. 312-319, doi: 10.1109/FiCloud58648.2023.00053