Empirical Software Engineering (2025) 30:120
https://doi.org/10.1007/s10664-025-10672-8

Vulnerabilities in infrastructure as code: what, how many, and who?

Aicha War[1] · Alioune Diallo[1] · Andrew Habib[1,2] · Jacques Klein[1] · Tegawendé F. Bissyandé[1]

Accepted: 5 May 2025

Abstract
Infrastructure as Code (IaC) is a pivotal approach for deploying and managing IT systems and services using scripts, offering flexibility and numerous benefits. However, the presence of security flaws in IaC scripts can have severe consequences, as exemplified by the recurring exploits of Cloud Web Services. Recent studies in the literature have investigated IaC security issues, but they often focus on individual components (IaC tools or scripts), providing only preliminary insights. Our research extends the current knowledge by conducting a comprehensive investigation into various aspects of IaC security, encompassing its components. We explore vulnerabilities in terms of types, their predominant locations, contributor responsibilities for introducing vulnerabilities, and more. Our methodology relies on widely adopted static security testing tools, which analyze over 1600 repositories to identify IaC vulnerabilities. Our empirical study yields valuable observations, highlighting severe and recurrent vulnerabilities within IaC, while also categorizing their severity and types. We delve deeper into vulnerability patterns, examining source code, dependencies, and manifest files across IaC components, including tools, scripts, and add-ons (libraries or plugin tools). The study uncovers that IaC components are plagued by exploitable vulnerabilities that span all ten categories of security bugs outlined in the OWASP Top 10 2021. Furthermore, our investigation reveals that even when maintainers employ security tools to address vulnerabilities, they do not integrate them systematically into their automation routines. Consequently, we propose that IT teams need to foster stronger collaboration across DevOps profiles (developers and IT operators) and break down the boundaries with security operators to enhance Infrastructure as Code's security posture through the adoption of DevSecOps practices.

Keywords DevOps · DevSecOps · SecDevOps · Infrastructure as code · Security testing · Security smells · Software vulnerability

1 Introduction

In the realm of DevOps, Infrastructure as Code (IaC) has emerged as a pivotal practice, offering automation and code-driven management for infrastructure deployment and config-

Communicated by: Mehdi Mirakhorli

Extended author information available on the last page of the article uration. Infrastructure as Code is a DevOps practice where a wide array of IT infrastructure (servers, networks, Operating Systems, Virtual Machines, etc.) configuration and management tasks, such as provisioning servers or configuring networks, are automated and defined using code (Rahman 2018). As DevOps practices have gained widespread adoption, IaC has become an integral part of modern software development and deployment pipelines. It ensures benefits such as consistency across environments, the reduction of manual configuration errors, and facilitates faster deployment cycles. It empowers IT teams to adopt DevOps practices effectively, enabling scalability and reproducibility of systems, and enhances collaboration by allowing infrastructure to be version-controlled and treated like application code. On top of that, IaC helps IT teams to focus on their critical development tasks since their manual intervention is considerably reduced and leads to better quality software production. Consequently, safeguarding the security and integrity of the underlying infrastructure and systems has assumed paramount importance. However, IaC is not immune to vulnerabilities, and issues such as misconfigurations, inadvertent secret exposures, lax access controls, and resource leaks pose significant risks. The detection, comprehension, and mitigation of these security challenges are of utmost importance to protect the assets associated with deployed systems.

Understanding the vulnerabilities inherent in IaC and identifying the parties responsible for introducing them are crucial steps in preempting security incidents. To address these fundamental questions, this study pioneers a comprehensive exploration of IaC vulnerabilities, encompassing not only IaC scripts, as investigated in prior research (Rahman et al. 2019), but also the IaC tools themselves and add-ons. We refer to IaC tools, scripts, and add-ons as **IaC components**. **IaC tools** are software frameworks that enable the automation of IT infrastructure provisioning, configuration, and management through code. **IaC scripts** are the code written using IaC tools. We refer to **IaC add-ons** as any software or code that extends the functionalities of the infrastructure management when included in IaC tools or scripts. Section 2 provides a detailed breakdown of these three distinct components. Our methodology involves the examination of seven widely adopted IaC tools and over 1,600 IaC repositories, encompassing both scripts and add-ons. We employ two widely accepted static security testing (SAST) tools from the open-source community, namely Snyk [1] and Horusec [2], to conduct this investigation. Additionally, attributing the responsibility for introducing vulnerabilities in IaC is a nuanced challenge due to the diverse actors involved, including developers, operators, and maintainers, each contributing to different components of IaC. Therefore, we employ the blaming method that assess ones specialization (developer or IT operator in the context of IaC) depending on the types of files they edit frequently (Palix et al. 2011) coupled with our own manual investigations (on the types of edits inside of those files and the profiles of maintainers) to discern the origins of vulnerabilities within this multifaceted ecosystem.

In section 2, we elaborate on the difference between the three components. To perform the study, we collected seven widely used IaC tools and over 1,600 IaC repositories that contain scripts or add-ons, and we utilized two SAST tools: Snyk and Horusec. Another aspect of understanding the vulnerabilities in IaC is finding out who is responsible for introducing such vulnerabilities. Since we analyze several components of IaC, different actors such as developers, operators, and maintainers own and contribute to those components and hence,

[1] https://snyk.io/product/snyk-code/

[2] https://github.com/ZupIT/horusec

discerning “who is responsible for what” becomes a crucial part of understanding, avoiding, and fixing those vulnerabilities. Therefore, we apply our blaming method to answer this question.

Previous studies have investigated IaC security aspects (Verdet et al. 2023; Reddy Konala et al. 2023). For instance, there have been efforts to establish testing practices (Hasan et al. 2020) tailored to IaC, primarily revolving around infrastructure configurations (i.e., scripts), including the use and creation of linters, sandbox testing, and recently machine learning solutions (Cankar et al. 2023; Petrović 2023; Opdebeeck et al. 2023) among others (Reis et al. 2023). In our work, we further perform security testing on IaC components (IaC tools, scripts, and add-ons) using static analyzers. Additionally, other studies (Rahman 2018; Morris 2020) have characterized IaC scripts properties. However, because our study case involves two more components, we could not rely on those characteristics only. Hence, we consider three common artifacts from IaC repositories. Those artifacts are the source code, the manifest files, and the dependencies (libraries, packages, etc.). We apply Snyk and Horusec to perform security testing on the code, dependencies, and manifests of our IaC components. While prior work has investigated security flaws for some IaC tools (Rahman et al. 2021), our study has expanded to substantially more IaC tools. We also combine three IaC components and study them altogether to localize IaC vulnerabilities. Compared to previous research (Rahman et al. 2019; Rahman and Williams 2021), our security analysis provides new insights into IaC security, notably in terms of vulnerability occurrences, severity, component and defect origins (impacted files, code, dependencies, manifests). We finally apply the latest OWASP Top 10 categorization to present our list of vulnerabilities. It is noteworthy that the OWASP Top 10 includes DevOps security defects such as category eight (A08:2021-Software and Data Integrity Failures) that integrates DevOps pipelines security issues.

Overall, the main contributions of this paper are as follows:

- We provide a comprehensive view of vulnerabilities in IaC. In contrast to prior work, we investigate various different components that are relevant to IaC and analyze how they impact IaC security.
- We discuss in-depth where security issues are localized within IaC components.
- We supplement the collected data with a categorization of security issues reported by Static Analysis Security Testing tools, using the OWASP Top 10.

Among various insights, we note that:

- Vulnerabilities are widespread across all IaC components with every one of the OWASP Top 10 vulnerability types manifesting across each analyzed component;
- Although IaC maintainers demonstrate a proactive stance toward vulnerabilities, they do not seem to follow standard automatic practices and routines to check for the existence of such issues consistently and systematically;
- Manifest files emerge as the most vulnerable artifacts within the IaC ecosystem;
- IaC vulnerabilities frequently stem from non-specialized actors attempting modifications within components outside their expertise.

Paper Organization The rest of this paper is organized as follows: Section 2 introduces the 1background of this work, Section 3 discusses the study design, Section 4 presents the experimental results, Section 5 discusses the study implications and perspectives for research, Section 6 presents the threats to validity for our work, Section 7 investigates the related work, and finally, Section 8 concludes the paper.

2 Background

2.1 Infrastructure as Code and its Components

Infrastructure as Code (IaC) is a process that involves components, mainly IaC tools, scripts, and add-ons, for automatically managing and provisioning infrastructure. IaC tools (e.g., Ansible, Terraform, and Chef) are software frameworks that enable the automation of infrastructure provisioning, configuration, and management through declarative or imperative code. IaC scripts are the code or configuration files written by users to specify the desired infrastructure state or actions, typically in formats like YAML, HCL, or JSON. Finally, IaC add-ons (or plugins) extend the functionality of IaC tools by offering additional modules, libraries, or integrations for tasks such as resource provisioning, third-party API integration, or custom workflows. Many IaC scripts are written using Domain Specific Languages (DSLs), which are designed to model a specific domain: infrastructure in this case (Morris 2020). In general, IaC scripts have similar writing styles: declarative, scripted, or mixed (both declarative and scripted). Usually, the declarative code is used for configuration since it is more structured. It also allows faster management of infrastructure components. That declarative coding style involves the use of packages, services, files, user accounts, etc. Figure 1 provides an excerpt of an IaC declarative script for deploying a Tomcat server.

IaC tools can be categorized into two groups depending on their main tasks. There are orchestration tools (Terraform and Vagrant for example) that perform provisioning, organizing, and managing processes for infrastructure components. In addition, there are configuration management tools (Ansible, Chef, Puppet, for example) that allow installations, updates, or management of software in infrastructure components. In our study, we investigate both categories. Deploying IaC scripts into servers follows a generic automation process. This process implies a set of CI and CD tools (specifically IaC tools in our context). IaC tools are employed to write IaC scripts. We can see IaC tools as development kits for IaC scripts. Through Fig. 2, we explain the automation activities that lead to IaC scripts deployment:

1. The DevOps engineer writes the IaC script
2. The DevOps engineer commits the IaC script changes to a Version Control System (VCS)
3. The changes trigger the CI tool to run static testings
4. The CI tool incorporates the IaC script changes when tests are positive (and rejects changes otherwise)
5. The CI tool deploys the updated IaC script into the targeted server(s).

In this study, we rely on Jenkins as a CI tool for the experiments, and we collect source code from GitHub repositories of IaC components.

```
package: jdk
package: tomcat            file: /var/lib/tomcat/server.conf
                               owner:   tomcat
service: tomcat                group:   tomcat
    port:   8443               mode:    0644
    user:   tomcat             contents: $TEMPLATE(/src/appserver/tomcat/server.conf.template)
    group:  tomcat
```

Fig. 1 Excerpt from IaC declarative script written for deploying a Tomcat server

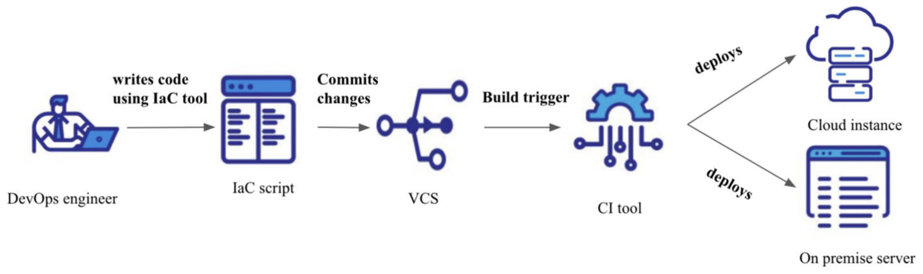
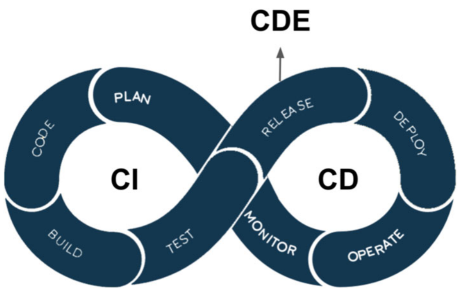


Fig. 2 IaC deployment schema

2.2 DevOps and DevSecOps

DevOps refers to the automation of development (Dev) tasks such as unit tests, integration tests, etc. as well as the automation of operation (Ops) tasks such as artifacts releases or deployments. Figure 3 illustrates how this automation fits into three main pipelines: Continuous Integration (CI) for development tasks, Continuous Delivery(CDE) for releases, and Continuous Deployment (CD) for IT operation tasks. Our study focuses on CD, specifically on the security aspects of Infrastructure as Code. To strengthen DevOps security, some practices and collaboration with security operators have been grafted to DevOps (Rahman and Williams 2016). And the generic terms for this evolution are DevSecOps and SecDevOps (Mohan and Othmane 2016). Despite possible nuances, in this paper, we refer to DevSecOps as an umbrella term for both DevSecOps and SecDevOps. We also refer to IaC maintainers as an umbrella term for developers, IT operators, and DevOps engineers. The application of DevSecOps entails several challenges (Myrbakken and Colomo-Palacios 2017) that often impede its adoption and effectiveness. These challenges can be grouped into three main categories. First, we have *speed and agility:* traditional, manual security practices are too slow for the fast-paced nature of DevOps (Goldschmidt and McKinnon 2016) Security methods need to be agile and seamlessly integrate with DevOps pipelines to avoid slowing down development and service delivery. Second, there are *organizational barriers:* implementing DevSecOps requires significant organizational changes such as cultural shifts (developers and operators often view security as a bottleneck, while security teams prioritize robust functionality over speed Akbar et al. 2022), skill gaps (organizations need expertise in encryption, logging standards, and other security measures Shackleford 2017), and cost perception (security is often seen as a cost center (Hornbeek 2015) rather than a value generator, which can limit investments in tools and training). Finally, there is the challenge related to tools and practices: the fast-changing DevOps environment demands security tools that

Fig. 3 DevOps automation and CI-CDE-CD pipelines



are automated, platform-compatible, and easy to use. However, many tools lack sufficient functionality (Rahman et al. 2019; Saavedra and Ferreira 2022) or automation, creating friction in the DevOps lifecycle (Shackleford 2017). Additionally, developers may struggle to write secure code due to inadequate training and poorly designed tools that do not keep up with DevOps speed, resulting in unsafe consumption of automation scripts (Sokolowski et al. 2024) (for instance, in August 2022, less than 1 % of the public Pulumi IaC scripts on GitHub implemented tests). Beyond classical DevSecOps concerns, our study explores other DevOps security aspects. Indeed, we investigate DevOps collaborating actors (developers, IT operators) to understand which profile is more likely to introduce security flaws in IaC environments. With this blaming/profiling method, we expect to provide insights for the development of tools that will help DevOps practitioners embrace DevSecOps and improve their security practices to build better quality IaC environments. Furthermore, we extend the comprehensiveness of DevOps security and explore IaC vulnerabilities using different SAST tools on code from multiple IaC components: tools, scripts, and add-ons. We study those defects with several parameters like their severity, propensity, and location (file types impacted) depending on those components.

2.3 OWASP Top 10

The OWASP Top 10 is a widely recognized standard awareness document in the field of web application security, first introduced by the Open Web Application Security Project (OWASP) in 2003. The Top 10 list is curated by a global team of security professionals based on data contributed by numerous organizations worldwide. This data is rigorously analyzed to reflect a consensus on the most critical security risks affecting web applications. The OWASP Top 10 has been adopted and utilized by numerous industry-leading organizations, including Google[3], Microsoft[4], and IBM[5], as a benchmark for identifying and mitigating vulnerabilities. In this work, we focus on vulnerabilities related to Infrastructure as Code and its interaction with the DevOps pipeline. While there exists a more specific OWASP Top 10 for CI/CD systems, IaC security risks can stem beyond CI/CD, affecting software involved in the configurations, such as web applications, Operating Systems, etc. Thus, we look into the general landscape of security risks, including insecure design, security misconfigurations, and vulnerable and outdated components within the OWASP Top 10, which are directly relevant to IaC vulnerabilities. This broader scope makes the OWASP Top 10 the most suitable framework for our analysis and ensures the generalizability of our approach (Table 1).

2.4 Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE)

Common Vulnerabilities and Exposures (CVE) (Chang et al. 2011) is a list of publicly disclosed cybervulnerabilities. Each entry, identified by a unique CVE ID (e.g., CVE-2024-12345), provides standardized information about a specific vulnerability, including its description, potential impact, and references to further details. The CVE system helps organizations and researchers to identify and discuss vulnerabilities in tools, databases, and reports. On the other hand, Common Weakness Enumeration (CWE) (Martin 2019) is a categorized

[3] https://cloud.google.com/architecture/owasp-top-ten-mitigation

[4] https://learn.microsoft.com/en-us/training/modules/owasp-top-10-for-dotnet-developers/

[5] https://www.ibm.com/fr-fr/topics/application-security

Table 1 OWASP Top 10 202 Security Vulnerability Risks

Rank	OWASP Top 10 2021 Category
A01	Broken Access Control
A02	Cryptographic Failures
A03	Injection
A04	Insecure Design
A05	Security Misconfiguration
A06	Vulnerable and Outdated Components
A07	Identification and Authentication Failures
A08	Software and Data Integrity Failures
A09	Security Logging and Monitoring Failures
A10	Server-Side Request Forgery (SSRF)

list of types of hardware and software weaknesses. Unlike CVE, which focuses on specific vulnerabilities, CWE addresses underlying flaws (e.g., buffer overflow or hardcoded credentials) that could lead to vulnerabilities. Each weakness is assigned a unique CWE ID (e.g., CWE-79 for Cross-Site Scripting). CWE provides a foundation for identifying, addressing, and mitigating security issues during software development and system maintenance.

Both CVE and CWE are widely used frameworks by security professionals to manage and communicate security risks. In the context of this study, CVE identifiers are used to reference specific vulnerabilities detected in Infrastructure as Code (IaC) projects, while CWE classifications are used to group and analyze these vulnerabilities based on their root causes. The OWASP Top 10 builds on these frameworks by grouping common security risks into categories, providing IT professionals with actionable guidance to mitigate critical threats in modern software systems. Together, they form an interconnected ecosystem for identifying, classifying, and addressing security challenges.

3 Methodology

We present the overall outline of our study methodology. We employ two Static Application Security Testing (SAST) tools to analyze datasets comprising infrastructure components, including tools, scripts, and add-ons. The study specifically aims to answer the following research questions:

3.1 RQ1: What types of vulnerabilities occur with IaC?

The primary focus of this research question is to identify and categorize the prevalent vulnerabilities associated with IaC. Understanding the specific vulnerabilities inherent in IaC environments is crucial for developing robust patterns for the detection and prevention of infrastructure-related security issues. We supplement our analysis by providing insights into the severity of vulnerabilities, their frequency of occurrence, and their origins.

3.1.1 Data Collection

In this study, we analyze the source code of IaC components: tools, scripts, and add-ons. Our data collection process is designed to comprehensively cover a large spectrum of IaC projects. Data were collected using the following steps:

Repository Search and Collection To ensure a broad and representative dataset, we primarily focused on widely used, open-source IaC tools commonly mentioned in the literature (Cottrell and Cottrell 2020; Valkeinen 2022; Paloviita et al. 2022; Castro Sánchez 2020; Houde et al. 2021; Rodríguez Couto 2022). These include tools such as Ansible, Terraform, Chef, Puppet, Saltstack, Pulumi, and Vagrant. We used the GitHub API to search for repositories associated with these tools. We did not apply filters based on repository popularity, as we aimed to avoid bias and capture a wide range of projects, including those from less popular but still relevant tools. This decision ensures that the analysis encompasses both mainstream and niche IaC components. Additionally, we collected projects that were actively maintained as of August 2022, which is the cut-off date for our data collection, through specifications in our requests to GitHub API. This helps ensure that the analysis is based on current tools and scripts in active use. The dataset contains repositories associated with:

- IaC Tools: Repositories containing the core code for the tools (e.g., Terraform, Ansible).
- IaC Scripts: Repositories related to IaC scripts developed using the identified tools.
- IaC Add-ons: Repositories containing add-ons and plugins that extend the functionality of the tools.

Scripts Identification Scripts were identified based on their association with the selected IaC tools, which we verified by including in our scrapping request keywords of files only present in IaC scripts. For example, scripts written with Chef are named *Cookbooks* and those written using Ansible are *Playbooks*. Then, we manually analyzed the metadata from their GitHub repositories to confirm that these scripts were indeed maintained.

Add-ons Identification Add-ons were also identified based on their association with the selected IaC tools, which we verified by manually checking their documentation and metadata. We also analyzed the metadata from the GitHub repositories to confirm that these add-ons were indeed maintained as part of the IaC tool ecosystem. Our methodology ensures that add-ons are identified accurately and only include those relevant to the analyzed IaC tools.

Automation of Data Collection with Jenkins We selected Jenkins for the automation of data collection and security testing due to its widespread use as a Continuous Integration (CI) tool that supports integration with various security tools and scripting (Armenise 2015; Cepuc et al. 2020). Jenkins facilitates automated workflows for managing the testing of multiple IaC components simultaneously, allowing our study to scale across the large volume of repositories collected.

Finally, we grouped IaC scripts and add-ons by their programming language, including Java, PHP, Go, JavaScript, Shell, Python, PowerShell, and Ruby. This step allows us later to analyze vulnerabilities within the file types of IaC components using file extensions. Overall, we collected the source code of 7 IaC tools and that of over 1,505 IaC scripts (215 for each IaC tool) and 105 add-ons (15 for each IaC tool, as Saltstack has a maximum of 17 GitHub add-on projects in August 2022).

3.1.2 Experimental Procedure

We explain in this experiment our static security testing techniques on the collected IaC components and the presentation of the data of the detected vulnerable parts.

3.1.3 Static Application Security Testing

In the literature, several security analyses have been conducted on IaC projects (Hortlund 2021; Almuairfi and Alenezi 2020; Ibrahim et al. 2022; Druta et al. n.d). However, these studies are generally limited. For example, some are focused on projects written in a specific programming language, while others rely on a single or a few IaC tools. Our work improves over these studies by widening the research focus to a substantially larger set of tools, to a large array of programming languages, and finally by considering IaC core components (tools and scripts) as well as add-ons. Indeed, it is noteworthy that previous studies have mainly characterized defective IaC scripts (Rahman 2018), while our work investigates also tools and add-ons. Our study, therefore, bears more constraints: the same analyses cannot be applied similarly to scripts, tools, and add-ons. We propose three characteristics that each repository of our IaC components generally possesses: a source code, dependencies, and manifest files. Manifests are files used to deploy and manage resources with Infrastructure as Code. When analyzing our components with our security application security testing (SAST) tools, we will take the vulnerabilities' origins as those enunciated characteristics (code, dependencies, manifests). We present our security testing architecture with Jenkins as a CI tool in Fig. 4:

- In Steps 1 and 2, Jenkins orchestrates the collection of IaC components.
- At stage 3, Jenkins performs the security testing with our selected SAST tools: Snyk and Horusec.
- At stage 4, Jenkins presents the detected vulnerabilities in a report for the components.

Selection of SAST Tools We selected Synk and Horusec as our Static Application Security Testing (SAST) tools after analyzing the benchmarking results of the literature (Di Stasio 2022; Cruz et al. 2023; Elrowayati and Fadeel 2024), focusing on tools that are compatible with IaC environments. Synk is known for its ability to scan code for known vulnerabilities, including issues related to dependencies and package management, source code, and IaC configurations. It also has a database of CVEs associated with our selected components that is continuously updated. Horusec, on the other hand, offers specific capabilities for detecting security flaws within IaC components, further enhancing our ability to conduct a comprehensive security analysis.

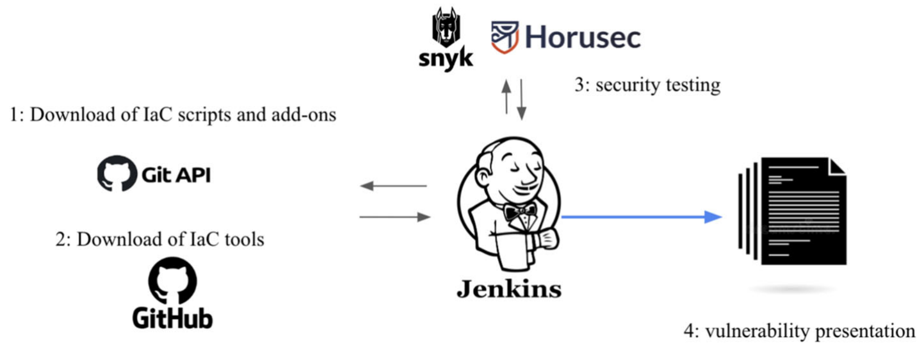


Fig. 4 Jenkins automated security testing on IaC components using Snyk, Horusec

3.1.4 Collection and Presentation of Vulnerabilities

Our static security testing with Snyk and Horusec on the code, dependencies, and manifests of IaC components identified a variety of vulnerability types. In this paper, we present the distribution and severity of IaC vulnerabilities based on these three artifacts. To categorize the detected vulnerabilities, we refer to existing taxonomies of infrastructure vulnerabilities. Previous research has categorized IaC vulnerabilities into eight groups (Rahman et al. 2020): conditional, configuration data, dependency, documentation, idempotency, security, service, and syntax. However, our study focuses exclusively on the security-related vulnerabilities of Infrastructure as Code rather than its overall vulnerabilities. For this purpose, we categorized the detected vulnerabilities using the OWASP Top 10 (2021) framework. This categorization is appropriate because each detected vulnerability could be mapped to a specific OWASP Top 10 category using CVE/CWE information. Due to space constraints, we focus on the ten most critical vulnerabilities under each OWASP Top 10 category. Here, "critical" does not exclusively refer to the severity level of vulnerabilities (e.g., Low, Medium, High, Critical). Instead, the selection considers the severity level as indicated by CWE classifications and warnings from the SAST tools (Snyk and Horusec) and the frequency of occurrence as the number of instances of each vulnerability type in our datasets. We combine these factors, with priority to the severity level, to identify the ten most "critical" (in terms of impact and representation) vulnerabilities under each OWASP Top 10 category. Thus, not all detected vulnerabilities are included in this paper. For each OWASP Top 10 category, we present the top vulnerabilities detected using our CI tool, Jenkins, combined with the analysis reports from Snyk and Horusec. The ranking process is automated, prioritizing vulnerabilities based on severity, origin (code, dependencies, manifests), and frequency of occurrence. For the final categorization, we manually assigned detected vulnerabilities to OWASP categories. The process involved the following steps:

1. We check whether the vulnerability type explicitly matches one in the OWASP category list.
2. If not, we map the vulnerability type using its CWE classification and locate the matching OWASP category.
3. If no explicit match is found, we read the CWE description to identify the best-fit OWASP category.

Additionally, we accounted for instances of overlapping vulnerabilities reported by both Snyk and Horusec. Using Jenkins, we automated the matching of vulnerable paths and line numbers to identify identical vulnerability instances flagged by both tools. This ensured that duplicate reports were consolidated in our analysis.

3.2 RQ2: How do vulnerabilities evolve in infrastructure components?

In this research question, we investigate the evolution of vulnerabilities across versions of IaC components. We assess to what extent vulnerabilities persist between versions.

3.2.1 Evolution of vulnerabilities across components versions

We follow the evolution of the detected vulnerabilities across different versions of our components' projects. We chose 5 projects for each of our IaC components: tools, scripts, and add-ons. These projects were identified based on their GitHub Watch, Fork, and Star numbers. The more, the better. Then, we selected 2 versions backward for each project sample. After our project data set is constructed, we automatically run security analysis using Snyk and Horusec in our CI tool, Jenkins. We kept tracking the history of the instances of the detected vulnerabilities using the paths, lines, and vulnerability descriptions that Horusec 5 and Snyk 6 report. We manually check if the warnings are fixed or if new vulnerability instances appear. Consequently, we could observe whether vulnerabilities have been fixed in those versions.

3.3 RQ3: How are Vulnerabilities Introduced in Infrastructure as Code?

Finally, with this research question, we investigate the locations (in terms of file type) where vulnerabilities appear, as well as the contributors associated with the vulnerable code. The profiles (i.e., developers and IT operators) of these contributors are also analyzed with respect to the introduced vulnerabilities: who introduces which vulnerabilities and where?

3.3.1 Security Responsibilities of DevOps Maintainers

Finding vulnerabilities' origins is an important task in IT systems. Hence, many Version Control Systems (VCS) tools like Git or automation tools like Jenkins provide functionalities for such tasks. For example, the `git-blame` command shows which revision an author made, who last modified which line of which file among repository maintainers, etc. Previous research can also help identify IT profiles that contribute the most to the introduction of vulnerabilities in source code. They have defined a method to estimate, according to the frequency of editing of a certain type of file, the specialization of the maintainers (Palix et al. 2011). Because DevOps is about collaborative work between development and operation activities, we thought it relevant to investigate the responsibilities of these maintainers in IaC vulnerabilities. Specifically in the types of file they manipulate the most. Unlike developers, IT operators are the ones specialized in managing the infrastructure and thus dealing with manifest files. Hence, we investigate if developers introduce more vulnerabilities in manifest files than in development files. First, we automatically collect information about the commits in IaC projects and about their maintainers. Then we were able to access the maintainers' profiles to check their specialization. Using our samples from each IaC component, we also

```
Language: Leaks
Severity: CRITICAL
Line: 98
Column: 18
SecurityTool: HorusecEngine
Confidence: MEDIUM
File: /Users/aicha.war/.jenkins/workspace/componentsevotestingsnyk/scripts/community.general/3/tests/unit
/plugins/modules/test_keycloak_realm.py
Code: 'auth_password': 'admin',
RuleID: HS-LEAKS-26
Type: Vulnerability
ReferenceHash: eb97965e94f0e19392916ea8d800295f99312b609fd9f49dc4c057d374e54611
Details: (1/1) * Possible vulnerability detected: Hard-coded password
The software contains hard-coded credentials, such as a password or cryptographic key, which it uses for
its own inbound authentication, outbound communication to external components, or encryption of internal
data. For more information checkout the CWE-798 (https://cwe.mitre.org/data/definitions/798.html) advisory.
```

Fig. 5 Hard-coded password vulnerability report by Horusec

```
[Low] Hardcoded Secret
Path: tests/unit/plugins/modules/test_java_keystore.py, line 88
Info: Avoid hardcoding values that are meant to be secret. Found a hardcoded string used in here.
```

Fig. 6 Hardcoded Secret vulnerability report with Snyk

investigate the types of files edited (readme files, JSON files, etc.) and the types of edits (e.g. comments, features) to match the relevant files and edits with the vulnerabilities reported by Snyk and Horusec. Finally, we assess the responsibilities of developers and IT operators in IaC components using the specialization of projects' maintainers, the types of files they have edited in those projects, and the types of edits that hosted vulnerabilities in these specific files.

3.3.2 Propensities and Distribution of Vulnerabilities in Components

Our study also gives insights into the location of the detected vulnerabilities in Infrastructure as Code. We automatically sort the vulnerabilities reported by our SAST tools by severity, IaC component, occurrence, and types of file. Localizing vulnerabilities with their severity and distribution in components gives better visibility on the most severe locations. It also naturally shows where to work in priority to build better security patterns for IaC components and reduce vulnerabilities.

4 Study Results

We present the investigation findings related to the research questions of our study. First, we summarize the key answers (cf. Section 4.4). Second, we overview the landscape of vulnerabilities affecting IaC (RQ1, cf. Section 4.1). Third, we dissect how vulnerabilities evolve with IaC code changes (RQ2, cf. Section 4.2). Fourth, we present the investigation results on where and who introduces vulnerabilities in IaC (RQ3, cf. Section 4.3).

4.1 IaC Vulnerabilities: Categories, Types, and Severity

The execution of Snyk and Horusec SAST tools on the datasets of IaC components yielded a large volume of reports (359,884 log entries from Jenkins) describing a variety of vulnerabilities for different components. These vulnerabilities appear with diverse occurrence rates across components and present different severity scores.

In every component dataset, we analyzed the source code, dependencies, and manifest files of projects using Snyk and Horusec. In Table 2 we present a taxonomy of these vulnerabilities using the OWASP Top 10 2021. Under each OWASP category, we sort the vulnerabilities based on the severity score. We then report the ten most severe vulnerabilities and their occurrence information. In our categorization process, we may find less than 10 categories of vulnerabilities under an OWASP category. Then, we do not consistently have 10 categories reported under each of the OWASP Top 10 categories. Furthermore, for each vulnerability type, we detail the propensities and severity of the detected vulnerabilities depending on the origin (or artefact) of the detected vulnerability (source code, dependencies, manifests) and the used SAST tool. We also provide the number of vulnerability instances that both Snyk and Horusec detect when analyzing the components.

Table 2 Top 10 of vulnerabilities detected by SAST tools applied on artefacts associated with IaC components for each category of the OWASP Top 10

	Source Code						Dependencies						Manifests					
Vulnerability	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST
OWASP A01:2021 Broken Access Control																		
Improper Access Control	0/0	0/0	0/0	0/0	0/0	0	4/0	0/0	3/0	2/0	9/0	0	-/-	-/-	-/-	-/-	-/-	-
Path Traversal	28/0	0/0	661/0	0/0	689/0	0	0/0	0/0	0/0	0/0	0/0	0	-/-	-/-	-/-	-/-	-/-	-
Sandbox Bypass	0/0	0/0	13/0	0/0	13/0	0	0/0	0/0	0/0	0/0	0/0	0	-/-	-/-	-/-	-/-	-/-	-
Privacy Leak	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	11/0	0/0	11/0	0	-/-	-/-	-/-	-/-	-/-	-
Cross-Site Request Forgery (CSRF)	150/0	96/0	8/0	0/0	254/0	0	0/0	11/0	1/0	0/0	12/0	0	-/-	-/-	-/-	-/-	-/-	-
Directory Traversal	0/0	0/0	0/0	0/0	0/0	0	44/0	16/0	8/0	0/0	68/0	0	-/-	-/-	-/-	-/-	-/-	-
OWASP A02:2021 Cryptographic Failures																		
Hardcoded Secret Transmission of Sensitive Information	85/0	0/0	81/0	0/166	166/166	166	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/6,169	0/6,169	0
No Use Weak Random Number Generator	0/0	0/0	0/200	0/0	0/200	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/561	0/0	0/561	0
Use of Password Hash With Insufficient Computational Effort	89/0	1,387/0	0/0	0/0	1,476/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
Improper Certificate Validation	19/0	142/0	0/0	0/0	161/0	0	19/0	0/0	52/0	0/0	71/0	0	0/0	0/0	0/0	0/0	0/0	0
Information Exposure	3/0	91/0	0/0	0/0	94/0	0	31/0	67/0	16/0	13/0	127/0	0	0/0	0/0	0/0	0/0	0/0	0
TLSCertVerificationDisabled	0/0	0/0	49/0	0/0	49/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
Insecure Data Transmission	0/0	0/0	33/0	0/0	33/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
Insecure Hash	0/0	183/0	0/0	0/0	183/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0

Table 2 continued

Vulnerability	Source Code						Dependencies						Manifests					
	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST
OWASP A03:2021 Injection																		
No Use Eval	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/406	0/406	0
Arbitrary Code Execution	0/0	0/0	0/0	0/0	0/0	0	0/0	9/0	21/0	25/0	55/0	0	0/0	0/0	0/0	0/0	0/0	0
SQL Injection	0/0	0/0	199/0	0/0	199/0	0	0/0	0/0	27/0	14/0	41/0	0	0/0	0/0	0/0	0/0	0/0	0
Remote Code Execution (RCE)	0/0	0/0	0/0	0/0	0/0	0	0/0	1/0	87/0	5/0	93/0	0	0/0	0/0	0/0	0/0	0/0	0
Command Injection	81/0	0/0	143/0	0/0	224/0	0	0/0	4/0	48/0	2/0	54/0	0	0/0	0/0	0/0	0/0	0/0	0
XML External Entity (XXE) Injection	14/0	0/0	13/0	0/0	27/0	0	0/0	9/0	484/0	1/0	494/0	0	0/0	0/0	0/0	0/0	0/0	0
Cross-site Scripting (XSS)	111/0	0/0	1,341/0	0/0	1,452/0	0	13/0	76/0	6/0	0/0	95/0	0	0/0	0/0	0/0	0/0	0/0	0
LDAP Injection	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/194	0/0	0/194	0
HTTP Header Injection	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	72/0	0/0	72/0	0	0/0	0/0	0/0	0/0	0/0	0
Code Injection	1/0	0/0	26/0	0/0	27/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
OWASP A04:2021 Insecure Design																		
Out-of-bounds Read	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	98/0	0/0	98/0	0	-/-	-/-	-/-	-/-	-/-	-
Prototype Pollution	0/0	6/0	0/0	0/0	6/0	0	3/0	45/0	44/0	0/0	92/0	0	-/-	-/-	-/-	-/-	-/-	-
HTTP Request Smuggling	0/0	0/0	0/0	0/0	0/0	0	1/0	83/0	0/0	1/0	85/0	0	-/-	-/-	-/-	-/-	-/-	-
Buffer Overflow	0/0	0/0	0/0	0/0	0/0	0	0/0	60/0	0/0	0/0	60/0	0	-/-	-/-	-/-	-/-	-/-	-

Table 2 continued

Vulnerability	Source Code						Dependencies						Manifests					
	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST
OWASP A05:2021 Security Misconfiguration																		
Privileged container in deployment	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	22/322	0/0	22/322	22
Default service account used in compute engine	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	104/0	0/0	104/0	0
Credentials are configured via provider attributes in provider	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	80/0	0/0	80/0	0
Hardcoded admin password in VM configuration in compute	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	71/0	0/0	71/0	0
S3 restrict public bucket control is disabled in S3	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	28/0	0/0	28/0	0
S3 block public policy control is disabled in S3	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	28/0	0/0	28/0	0
S3 block public acls control is disabled in S3	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	28/0	0/0	28/0	0
S3 ignore public acls control is disabled in S3	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	21/0	0/0	21/0	0
Rolebinding or clusterrolebinding is using a pre-defined role	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	13/0	0/0	13/0	0
Role with too wide permissions in clusterrole	-/-	-/-	-/-	-/-	-/-	-	-/-	-/-	-/-	-/-	-/-	-	0/0	0/0	13/0	0/0	13/0	0
OWASP A06:2021 Vulnerable and Outdated Components																		
Insecure API	56/0	2/0	0/0	0/0	58/0	0	0/0	0/0	0/0	0/0	0/0	0	-/-	-/-	-/-	-/-	-/-	-
Insecure Xml Parser	207/0	163/0	0/0	0/0	370/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
OWASP A07:2021 Identification and Authentication Failures																		
Use of Hardcoded Credentials	949/0	1,024/0	0/0	0/242,103	1,963/242,103	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
Returns Password	0/0	3/0	0/0	0/0	3/0	0	0/0	0/0	0/0	0/0	0/0	0	-/-	-/-	-/-	-/-	-/-	-

Table 2 continued

Vulnerability	Source Code						Dependencies						Manifests					
	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST	L	M	H	C	Total	∩SAST
OWASP A08:2021 Software and Data Integrity Failures																		
Deserialization of Untrusted Data	1/0	0/0	21/0	0/0	22/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
Resources Downloaded over Insecure Protocol	0/0	0/0	0/0	0/0	0/0	0	1/0	0/0	0/0	0/0	1/0	0	0/0	0/0	0/0	0/0	0/0	0
OWASP A09:2021 Security Logging and Monitoring Failures																		
Regular Expression Denial of Service (ReDoS)	10/0	0/0	49/0	0/0	59/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
Clear Text Logging	15/0	125/0	0/0	0/0	140/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
Improper Output Neutralization for Logs	0/0	0/0	0/0	0/0	0/0	0	32/0	0/0	0/0	0/0	32/0	0	0/0	0/0	0/0	0/0	0/0	0
Denial of Service (DoS)	0/0	1/0	0/0	0/0	1/0	0	0/0	0/0	0/0	0/0	0/0	0	0/0	0/0	0/0	0/0	0/0	0
OWASP A10:2021 Server-Side Request Forgery																		
Server-Side Request Forgery (SSRF)	2/0	0/0	114/0	0/0	116/0	0	0/0	0/0	0/0	0/0	0/0	0	-/-	-/-	-/-	-/-	-/-	-

L→ Low severity; M→ Medium severity; H→ High severity; ∩SAST: Intersection between Snyk and Horusec vulnerability reports. Results (#vulnerabilities) are encoded as ‘k/n‘ where ’k’ is the number of vulnerabilities reported by Snyk, and ’n’ is the number of vulnerabilities reported by Horusec
* **Artefacts** are the source code, the dependencies, and manifest files within projects associated with IaC components

4.1.1 Snyk vs Horusec

The execution results on our IaC datasets show that Snyk and Horusec are not systematically in agreement: they do not always detect the same types and instances of vulnerabilities in the same projects. Snyk has a database of CVEs with references to IaC components. In contrast, Horusec relies on a database of CWEs, which are not directly linked to the analyzed components. Furthermore, we observe that both SAST tools can associate different severity levels with the same vulnerability instances. Snyk also only shows warnings related to the vulnerabilities it has in its database of CVEs while Horusec not only presents the vulnerabilities related to its database of CWEs but also issues that it considers as potential vulnerabilities (ex: Hard-coded Password and *Potential* Hard-coded Password). Consequently, as revealed in Table 2, Horusec has thousands of vulnerability instances, while Snyk has only hundreds of vulnerability instances. Hence, using different static security tools may offer a good combination for finding more vulnerabilities in IaC components. However, it will create challenges in triaging a larger number of false positives.

4.1.2 Vulnerability of IaC Components

Data in Table 2 indicate that all categories of the OWASP Top 10 are reported in the execution of Horusec and Snyk on our IaC datasets. Despite the fact that both SAST tools may assert severity differently, there is one constant aspect of these vulnerabilities: they are all serious and exploitable security issues. This means that IaC tools, scripts, and add-ons are vulnerable and exploitable projects. IaC maintainers could benefit from adopting security testing automation in their DevOps routines and increase their security awareness and skills. If we take the example of the vulnerability "*container has no CPU limit*" (OWASP A05, Fig. 7) in the file misc/gce-federation/files/mongo-rs.yaml from the openshift project [6], it can easily be fixed by updating lightly the configuration parameters of the container (cf. Figure 8). We specify a request of 0.5 CPU and a limit of 2 CPUs in the use of the container. We also specify, under the args section, 2 CPUs that the mongo container should be allowed to use in case it needs more resources. From this example, we can highlight that some vulnerabilities represented in Table 2 can be avoided if the maintainers take better care of the basic configurations with IaC.

We observe that vulnerabilities, detectable by different SAST tools, spread across all IaC components. Furthermore, the security testing results revealed that these vulnerabilities are often severe, multiple, and various: all OWASP Top 10 categories are represented among vulnerabilities found in IaC scripts, IaC tools, and IaC add-ons. This finding underscores the gravity of vulnerabilities within the IaC ecosystem.

4.2 Vulnerabilities Lifecycle in IaC Components

Given a downloaded release of an IaC component, we investigate the evolution of the vulnerabilities comparing against prior versions(cf. Section 3.2.1). To assess this evolution across different components, we uniformly sample 5 projects per component type. For each sample, we consider its two preceding major release versions in which we also apply the SAST tools Snyk and Horusec. Our objective is then to find the changes in the sets of detected

[6] https://github.com/openshift/openshift-ansible-contrib.git

```
apiVersion: extensions/v1beta1
kind: ReplicaSet
metadata:
    labels:
        name: mongo
    name: mongo
spec:
    replicas: 3
    template:
        metadata:
            labels:
               name: mongo
        spec:
            containers:
            - image: mongo
              name: mongo
              args:
                - --replSet
                - rs0
              ports:
              - name: mongo
                containerPort: 27017
              volumeMounts:
                - name: mongo-db
                  mountPath: /data/db
            volumes:
                - name: mongo-db
                  persistentVolumeClaim:
                    claimName: mongo-storage
```

Fig. 7 Excerpt of script used in an Ansible project - Example of “Container has no CPU limits” vulnerability

vulnerabilities across projects’ versions. Note that we only focus on the evolution of vulnerability instances in the source code of sampled projects, irrespective, in this experiment, of the specialization of the maintainers involved.

```
-image: mongo
 name: mongo
 resources:
    limits:
        cpu:  "2"
    request:
        cpu: "0.5"
    args:
        --replSet
        --rs0
        --cpus
        --"2"
```

Fig. 8 Correction of the Ansible script excerpt (Fig. 7) for the vulnerability "Container has no CPU limits

Snyk results are broken down by manifest files, code, and dependencies, while the Horusec results are not due to the nature of the tools themselves. Snyk provides detailed vulnerability insights for each of these specific origins, enabling us to categorize the results accordingly. In contrast, Horusec aggregates vulnerabilities at the source code level without possible result separation. Our objective is then to find the changes in the vulnerabilities across projects' versions. In this experiment, we only focus on the evolution of vulnerability instances in the source code of sampled projects, no matter the IaC maintainer specialization (developer or IT operator). It appears that vulnerabilities evolve in different versions of the same project (cf. Tables 3, 4). We observe from one project version to another, that one vulnerability can have three statuses: it can be a new instance created, or it can be fixed in-between, or it can remain the same instance. We notice that, generally, IaC maintainers actually fix old vulnerabilities in their projects, in all components. The vulnerabilities that appear in newer projects' versions are mostly new vulnerability instances. We can, however, find the same instances of vulnerabilities in different versions of the same project for some components. This means that even if maintainers fix security issues in IaC projects, they do not systematically analyze them using security tools in automatic pipelines. Consequently, DevSecOps is not yet consistently applied by IaC maintainers.

Besides the evolution of vulnerabilities in IaC projects, our study investigates how maintainers actually address them. To that end, we mine commit messages that are associated with source code changes, as well as bug reports submitted on the project repository issue tracking system. It is noteworthy that some mentioned vulnerabilities are the same detected by the SAST tools: *Remote Code Execution (RCE), Arbitraty Code Execution, Directory Transversal, Deserialization of Untrusted Data, Cleartext Password, Insecure Hash, Hard-coded Admin, Container Auth.* We also observed that while some issues were reported to be fixed, some were just marked without any clear action. Due to the use of specific names, we also found that maintainers use linters in IaC to address vulnerabilities in specific tools, such as Ansible, Terraform, Chef, etc. (For more details on linter detection, cf. Methodology in Section 3). Furthermore, from GitHub commit messages, we observe that maintainers fix some vulnerabilities by updating or creating new features (mostly configuration features). They also add vulnerable code warnings for some dependencies.

Overall, mining IaC projects has revealed that IaC maintainers display, to some extent, a proactive approach towards addressing vulnerabilities. Unfortunately, the recurrence of some vulnerabilities within and across releases suggests that IaC development does build on established automatic practices and routines for consistent and systematic detection and fix of vulnerabilities.

4.3 IaC vulnerabilities: What, Where and Who?

In this section, we further dig into the details of IaC vulnerabilities to present the results in terms of IaC components, severity, files, programming languages, etc.

4.3.1 Occurrences and Severity of Vulnerabilities

We summarize the occurrences of vulnerabilities and their severity for IaC components based on Horusec and Snyk, where Snyk tools are used for source code (Snyk code), dependencies

Table 3 Evolution of vulnerabilities across the versions of IaC component samples using Snyk and Horusec Part 1

Snyk Manifests
* EATPINE: EC2 API termination protection is not enabled, EIAI: EC2 instance accepts IMDSv1, LBIIF: Load balancer is internet facing, NERBD: Non-Encrypted root block device, COPUCCWHU: Container's or Pod's UID could clash with host's UID, RCBD: Redis Cache backup disabled
* COPIRWPEC: Container or Pod is running without privilege escalation control, KVAPPD: Key Vault accidental purge prevention disabled, SADNELT: Storage Account does not enforce latest TLS
* CIRWML: Container is running without memory limit, CIRWLP: Container is running without liveness probe, COPIRWWRF: Container or Pod is running with writable root filesystem, CHNCL: Container has no CPU limit, CDNDADC: Container does not drop all default capabilities
* COPIRWRUC: Container or Pod is running without root user control, ROCIUAPR: RoleBinding or ClusterRoleBinding is using a pre-defined role, VNDPPD: Virtual Network DDoS protection plan disabled, ASAD: App Service authentication disabled, VKEDNS: Vault key expiration date not set
* CCBRWOI: Container could be running with outdated image, COPIRWRUC: Container or Pod is running without root user control, ASIM: App Service identity missing, VAINPAFW: VM Agent is not provisioned automatically for Windows, GRFACID: Geo replication for Azure Container Images disabled
* TMSATSAD: Trusted Microsoft Service access to storage account is disabled, ASMTD: App Service mutual TLS disabled, ASHD: App Service HTTP/2 disabled, CAAFD: CosmosDB account automatic failover disabled, CIIDFA: Container Insights is disabled for AKS, ANPD: AKS Network Policies disabled
* ASSINUSI: Azure Search Service is not using system-assigned identities, ASSEAID: Azure SQL server extended auditing is disabled, TMIPP: Traffic Manager insecure probing protocol, KVPPID: Key Vault purge protection is disabled, SAGD: Storage Account geo-replication disabled
* AASAHT: Azure App Service allows HTTP traffic, UTOMASPI: Use two or more App Service Plan instances, ETRAIRFTI: Ensure that RDP access is restricted from the internet, AMAAATDP: API Management allows anonymous access to developer portal, EDSCAC: Ensure Diagnostic Setting captures appropriate categories
* ETSAIRFTI: Ensure that SSH access is restricted from the internet, ANSGAPA: Azure Network Security Group allows public access, CEHNE: CDN Endpoint https not enforced, CAPNAE: CosmosDB account public network access enabled, RUATDOIACD: Restrict user access to data operations in Azure Cosmos DB
* RCMTV: Redis Cache minimum TLS version, ANSRAPA: Azure Network Security Rule allows public access, ASAFD: App Service allows FTP deployments, VMICWPAFA: Virtual machine is configured with password authentication for admin, LVMHPAE: Linux virtual machine has password authentication enabled
* ASSPNAE: Azure Search service public network access enabled, ASNRLNV: App Service not running latest .Net version, ASDNUPLS: App Service does not use production level SKU, ASNRLJV: App Service not running latest Java version, SCAPA: Storage container allows public access
* WVSSEAHD: Windows VM scale set encryption at host disabled, AKDE: AKS Kubernetes Dashboard enabled, ASRDE: App Service remote debugging enabled, DFPAE: Data Factory public access enabled
* AGDNUO3R: App Gateway does not use OWASP 3.x rules, PSMTV1.2: PostgreSQL server minimum TLS version 1.2, SFDNUADA: Service fabric does not use active directory authentication, ASAPA: API Server allows public access
* FABAD: Function App built-in authentication disabled, PALFSC&BIE: Public access level for storage containers & blobs is enabled, DFNEWCMK: Data Factory not encrypted with customer managed key, FADNEH: Function App does not enforce HTTPS

Name	V1	V2	V3	V1 Instances	V2 Fixed	V2 New	V2 Same	V3 Fixed	V3 New	V3 Same
Ansible (T)	v1.0	v2.0.0-0.1.alpha1	v2.15.2	0	0	0	0	0	0	0
Terraform (T)	v0.1.0	v1.0.0	v1.6.0-alpha20230719	EATPINE:91/ EIAI:91/LBIIF:1/NERBD:91	0	0	0	0	0	0

Table 3 *Continued*

Chef (T)	v17.0.0	v18.0.0	v18.2.44	0	0	0	0	0	0	0
Puppet (T)	7.0.0	8.0.0	upstream/0.25.5	0	0	0	0	0	0	0
Vagrant (T)	v1.0.0	v2.0.0	v2.3.7	0	0	0	0	0	0	0
ansible-for-devops (S)	1.0	2.0	2.2	0	0	COPUCCWHU:1/ COPIRWWRF:1/ CIRWML:1/ ROCIUAPR:1/ CHNCL:1/ COPIRWRUC:1/ COPIRWPEC:1/ CIRWLP:1/ CCBRWOI:1/ CDNDADC:1	0	0	0	COPUCCWHU:1/ COPIRWWRF:1/ CIRWML:1/ ROCIUAPR:1/ CHNCL:1/ COPIRWRUC:1/ COPIRWPEC:1/ CIRWLP:1/ CCBRWOI:1/ CDNDADC:1
vagrant-scripts (S)	2.0.4	2.3.0	3.0.0	0	0	0	0	0	0	0
terraform-generator (S)	v3.0.0	v4.0.0	v5.3.1	0	0	0	0	0	0	0
community.general (S)	6.0.0	7.0.0	7.2.1	0	0	0	0	0	0	0
NetBeansPuppet (S)	v1.2	v2.0.0	V2.0.3	0	0	0	0	0	0	0
soccer-stats (X)	v0.0.1	v0.0.2	NA	0	0	0	0	0	0	0
ansible-runner (X)	1.0.1	2.0.0	2.3.3	0	0	0	0	0	0	0

Table 3 *Continued*

terraform-provider- azurerm (X)	v2.0.0	v3.0.0	v3.67.0	CCBRWOI:2/CHNCL:2/ COPIRWWRF:3/ COPIRWRUC:3/ CDNDADC:2/ COPIRWPEC:3/ KVAPPD:2/ VNDPPD:14/TMSATS AD:11/ASAD:10/ ASIM:13/ASM TD:13/ASHD:13/ COPUCCWHU:3 /CIRWML:2/ CIRWLP:2/VAIN PAFW:3/GRFACID:1 /CAAFD:1/VKEDNS :1/CIIDFA:6/ANPD :6/RCBD:3/ASSIN USI:1/SAGD:7/ASSE AID:1/TMIPP:3/ KVPPID:2/ SADNELT:11/ASSE UTOMASPI:13 /AASAHT:13/ ETRAIRFTI:1 /ANSGAPA:1/CEHNE:1/ CAPNAE:2/RUATDOI ACD:2/ASAPA:7/R CMTV:3/ANSRAPA:1/ ASAFD:13/VMICWPA FA:3/LVMHPAE:1/ SCAPA:1/ASSPN AE:1/ASNRLNV:5 /ASDNUPLS:3/ ASNRLJV:1/ASRDE:4	0	CCBRWOI:1/ CHNCL:1/COPIRW WRF:1/COPIRW RUC:1/CDNDADC:1/ COPIRWPEC:1/ KVAPPD:1/VNDP PD:21/TMSATSAD:12 /AMAAATDP:1/ED SCAC:3/DFNEWC MK:2/COPUCCW HU:1/CIRWML:1/ CIRWLP:1/ CAAFD:2/VKEDNS:1/ CIIDFA:3/ANPD:2/ SAGD:10/ASSEAID:4/ KVPPID:2/ SADNELT:15/ UTOMASPI:3/ ANSGAPA:2/ CAPNAE:2/RUATD OIACD:2/ASAPA:3/ VMICWPAFA:4/ LVMHPAE:1/SCAPA:2 /DFPAE:2/AGD NUO3R:1/PSMTV1. 2:2/SFDNUADA:1/ WVSSEAHD:1 AKDE:16/FAD NEH:3/ FABAD:3/P ALFSC&BIE:2	CCBRWOI:2/ CHNCL:2/ COPIRWWRF:3/ COPIRWRUC:3/ CDNDADC:2/ COPIRW PEC:3/KVAPPD :2/VNDPPD:14/ TMSATSAD:11/ ASAD:10/AS IM:13/ASMTD: 13/ASHD:13/ COPUCCWHU:3/ CIRWML:2/ CIRWLP:2/VAINP AFW:3/GRFACI D:1/CAAFD:1/ VKEDNS:1/CIIDFA:6 /ANPD:6/RCBD:3/ ASSINUSI:1/SAGD :7/ASSEAID:1/ TMIPP:3/KVPP ID:2/SADNELT:11/ UTOMASPI:13/AAS AHT:13/ETRAIRFT I:1/ANSGAPA:1/ CEHNE:1/CAPNAE:2/ RUATDOIACD:2/ ASAPA:7/RCMTV:3/ ANSRAPA:1/ASAFD: 13/VMICWPAFA:3/ LVMHPAE:1/SCAPA: 1/ASSPNAE:1/ ASNRLNV:5/ ASDNUPLS:3/ ASNRLJV:1 /ASRDE:4	ASNRLNV:5/ ASDNUPLS:3/ ASNRLJV:1/ASR DE:4/AKDE:16/ FADNEH:3/ FABAD:3/ PALFSC&BIE: 2/ASAD:8/ASIM:11 /ASMTD:11/ ASHD:11/CA AFD:1/UTOMA SPI:14/AASAHT :11/CAPNAE:1/ RUATDOIAC D:1/ASAFD:11	KVAPPD:1/VND PPD:6/TMSAT SAD:3/CIIDFA:1/ KVPPID:1/SAG D:3/ETRAIRFTI:1/ ANSGAPA:1/VMICWP AFA:1/LVMHPAE:1	CCBRWOI:3/CHNCL:3/ COPIRWWRF:4/ COPIRWRUC:4/CDN DADC:3/COPIR WPEC:4/AMAAATDP:1/ KVAPPD:3/VNDPPD:35/ TMSATSAD:23/ ASAD:2/ASIM:2/ ASMTD:2/ASHD:2 /COPUCCWHU:4 /CIRWML:3/CIR WLP:3/EDSC AC:3/VAINP AFW:3/GRFA CID:1/CAAFD:2/ VKEDNS:2/DFNE WCMK:2/CIIDFA:9 /ANPD:8/RCBD:3/ ASSINUSI:1/S AGD:17/ASSEA ID:5/TMIPP:3/ KVPPID:3/SAD NELT:26/UTOM ASPI:2/AASAHT:2 /ETRAIRFTI:1/ANS GAPA:3/CEHNE:1 /CAPNAE:3/RUATD OIACD:3/ASAPA:10 /RCMTV:3/ANS RAPA:1/ASAFD:2/ VMICWPAFA:7/LVMHPAE:2/ SCAPA:3/ASSPNAE:1 /DFPAE:2/WNEOAG:1 /AGDNUO3R:1/PS MTV1.2:2/SFDN UADA:1/WVSSEAHD:1
cookstyle (X)	v6.0.0	v7.0.0	v7.32.3	0	0	0	0	0	0	0
pulumi-datadog (X)	v3.0.0	v4.0.0	v4.21.0	0	0	0	0	0	0	0

* Breakdown of vulnerabilities identified by Snyk across different IaC manifest files, code, and dependencies.
* Horusec results are presented at an aggregated level as the tool does not provide a location-wise breakdown.
* In the Name Column: T=tools, S= scripts, X=add-ons
* NA: Not Available

Table 4 Evolution of vulnerabilities across the versions of IaC component samples using Snyk and Horusec Part 2

Snyk Code
* PT: Path Traversal, PHWICE: Password Hash With Insufficient Computational Effort, HC: Hardcoded Credentials, CI: Cryptographic Issues, SSRF: Server-Side Request Forgery, IES: Inadequate Encryption Strength, BRCA: Broken or Risky Cryptographic Algorithm
* JAE: Jinja auto-escape set to false, IND: Improper Neutralization of Directives, ReDos: Regular Expression Denial of Service, DUD: Deserialization of Untrusted Data, NHV: no hostname verification, IXP: Insecure Xml Parser, ITF: Insecure Temporary File
* MPSSL: Missing protocol in ssl.wrap-socket, CTL: Clear Text Logging, HS: Hardcoded Secret, P2SC: Python 2 source code, ICV: Improper Certificate Validation, CInj: Command Injection, XSS: Cross-site Scripting, AFW: Arbitrary File Write

Name	V1	V2	V3	V1 Instances	V2 Fixed	V2 New	V2 Same	V3 Fixed	V3 New	V3 Same
Ansible (T)	v1.0	v2.0.0-0.1.alpha1	v2.15.2	P2SC:24/XSS:1/ICV:1/PHWICE:2/PT:4/JAE:2	0	DUD:1/ReDos:3/IND:1/AFW:2/HC:3/PT:2/IES:3/PHWICE:54	PT:4/PHWICE:2/XSS:1/P2SC:24/JAE:2	ReDos:2/IND:1/XSS:1/PHWICE:38/P2SC:24	PT:47/HC:7/JAE:6/P2SC:43/CInj:13/IXP:5/DUD:1/SSRF:6/HS:8/AFW:10/ITF:1	ReDos:1/IES:3/HC:3/JAE:2/PHWICE:18/DUD:1/AFW:2/PT:4
Terraform (T)	v0.1.0	v1.0.0	v1.6.0-alpha 20230719	HS:1/CInj:1/PT:1/PHWICE:2	CInj:1/PT:1	XSS:1/SSRF:1/CTL:1/ICV:1/HS:2/IES:1/PHWICE:19/PT:3	HS:1/PHWICE:2	0	CInj:2/PHWICE:2	XSS:1/SSRF:1/CTL:1/ICV:1/HS:3/IES:1/PHWICE:21/PT:3
Chef (T)	v17.0.0	v18.0.0	v18.2.44	HC:30/PT:1/P2SC:4	P2SC:3/PT:1	HC:6	HC:30	HC:4	0	HC:32
Puppet (T)	7.0.0	8.0.0	upstream/0.25.5	HC:17/BRCA:1	0	0	HC:17/BRCA:1	0	0	HC:17/BRCA:1
Vagrant (T)	v1.0.0	v2.0.0	v2.3.7	ICV:1/P2SC:1	ICV:1/P2SC:1	HC:13/BRCA:1	0	BRCA:1	HC:11	HC:13
ansible-for-devops (S)	1.0	2.0	2.2	P2SC:4	P2SC:4	0	0	0	0	0
vagrant-scripts (S)	2.0.4	2.3.0	3.0.0	0	0	0	0	0	0	0
terraform-generator (S)	v3.0.0	v4.0.0	v5.3.1	0	0	0	0	0	0	0

Table 4 continued

Snyk Code
* PT: Path Traversal, PHWICE: Password Hash With Insufficient Computational Effort, HC: Hardcoded Credentials, CI: Cryptographic Issues, SSRF: Server-Side Request Forgery, IES: Inadequate Encryption Strength, BRCA: Broken or Risky Cryptographic Algorithm
* JAE: Jinja auto-escape set to false, IND: Improper Neutralization of Directives, ReDos: Regular Expression Denial of Service, DUD: Deserialization of Untrusted Data, NHV: no hostname verification, IXP: Insecure Xml Parser, ITF: Insecure Temporary File
* MPSSL: Missing protocol in ssl.wrap-socket, CTL: Clear Text Logging, HS: Hardcoded Secret, P2SC: Python 2 source code, ICV: Improper Certificate Validation, CInj: Command Injection, XSS: Cross-site Scripting, AFW: Arbitrary File Write

community. general (S)	6.0.0	7.0.0	7.2.1	HC:3/PT:1/ HS:21/MPSSL:2/IXP:7/ NHV:4/PHWICE:8	0	0	HC:3/PT:1/ HS:21/MPSSL:2 /IXP:7/NHV:4 /PHWICE:8	0	0	HC:3/PT:1/ HS:21/MPSSL:2/ IXP:7/NHV:4/PHWICE:8
NetBeansPup pet (S)	v1.2	v2.0.0	V2.0.3	0	0	0	0	0	0	0
soccer-stats (X)	v0.0.1	v0.0.2	NA	0	0	0	0	0	0	0
ansible-runner (X)	1.0.1	2.0.0	2.3.3	PT:1	PT:1	PT:4	0	PT:4	CI:1	0
terraform-provider-azurerm (X)	v2.0.0	v3.0.0	v3.67.0	HC:23/PHWICE:25/PT:5 /IES:1/SSRF:1	HC:23/ PT:5	PT:1/ PHWICE:6	PHWICE :9/IES:1/SSRF:1	0	CTL:1/HC :20/PHWICE:3/PT:5	PHWICE:15/IES:1 /PT:1/SSRF:1
cookstyle (X)	v6.0.0	v7.0.0	v7.32.3	0	0	0	0	0	0	0
pulumi-datadog (X)	v3.0.0	v4.0.0	v4.21.0	0	0	0	0	0	0	0

Snyk Dependencies

* DoS: Denial of Service, RCE: Remote Code Execution, IPSp: IP Spoofing, IE: Information Exposure, AFEE: Arbitrary File Existence Exposure, DLL: DLL Loading Issue, DT: Directory Traversal, CSRF: Cross-site Request Forgery

* ACI: Arbitrary Code Injection, WCP: Web Cache Poisoning, IIV: Improper Input Validation, XEE: XML External Entity, ACB: Access Control Bypass, IHUDT: Improper Handling of Unexpected Data Type, NPD: NULL Pointer Dereference, IPM: Improper Privilege Management

Ansible (T)	v1.0	v2.0.0-0.1.alpha1	v2.15.2	0	0	0	0	0	0	0

Table 4 continued

Snyk Code
* PT: Path Traversal, PHWICE: Password Hash With Insufficient Computational Effort, HC: Hardcoded Credentials, CI: Cryptographic Issues, SSRF: Server-Side Request Forgery, IES: Inadequate Encryption Strength, BRCA: Broken or Risky Cryptographic Algorithm
* JAE: Jinja auto-escape set to false, IND: Improper Neutralization of Directives, ReDos: Regular Expression Denial of Service, DUD: Deserialization of Untrusted Data, NHV: no hostname verification, IXP: Insecure Xml Parser, ITF: Insecure Temporary File
* MPSSL: Missing protocol in ssl.wrap-socket, CTL: Clear Text Logging, HS: Hardcoded Secret, P2SC: Python 2 source code, ICV: Improper Certificate Validation, CInj: Command Injection, XSS: Cross-site Scripting, AFW: Arbitrary File Write

Terraform (T)	v0.1.0	v1.0.0	v1.6.0-alpha 20230719	ReDos:5/DoS :18/XSS:13/ IPSp:4/IE:4/ AFEE:1/ACI:3 /DLL:2/DUD :2/WCP:1	ReDos:5/ DoS:18/XSS :13/IPSp:4 /IE:4/AFEE:1 /ACI:3/DL L:2/DUD :2/WCP:1	0	0	0	0	0
Chef (T)	v17.0.0	v18.0.0	v18.2.44	ReDos:19/ DoS:8/XSS:1/ IIV:1/DUD:2 /ACI:3/WCP:1	ReDos:3/ DoS:2/ IIV:1	XSS:1/DU D:3/ACI:1	DUD:2/ ReDos:16 /DoS:6 /XSS:1/W CP:1/ACI:3	DUD:5/ ReDos:16 /DoS:6/ACI:4	WCP:1	XSS:2/WCP:1
Puppet (T)	7.0.0	8.0.0	upstream/ 0.25.5	0	0	0	0	0	0	0
Vagrant (T)	v1.0.0	v2.0.0	v2.3.7	0	0	ACI:1/XEE:3 /XSS:9/DT:3 /ReDos:3/Do S:11/ACB:1 /IE:1/CSRF:1 /IHUDT:1/WCP :1/NPD:1/ CInj:1/RCE:1	0	ACI:1/XEE:3/ XSS:9/DT:3/ ReDos:3/DoS:11/ ACB:1/IE:1 /CSRF:1/IHUDT:1 /WCP:1/NPD:1/ CInj:1/RCE:1	0	0
ansible-for-devops (S)	1.0	2.0	2.2	0	0	0	0	0	0	0
vagrant-scripts (S)	2.0.4	2.3.0	3.0.0	0	0	0	0	0	0	0

Table 4 continued

Snyk Code
* PT: Path Traversal, PHWICE: Password Hash With Insufficient Computational Effort, HC: Hardcoded Credentials, CI: Cryptographic Issues, SSRF: Server-Side Request Forgery, IES: Inadequate Encryption Strength, BRCA: Broken or Risky Cryptographic Algorithm
* JAE: Jinja auto-escape set to false, IND: Improper Neutralization of Directives, ReDos: Regular Expression Denial of Service, DUD: Deserialization of Untrusted Data, NHV: no hostname verification, IXP: Insecure Xml Parser, ITF: Insecure Temporary File
* MPSSL: Missing protocol in ssl.wrap-socket, CTL: Clear Text Logging, HS: Hardcoded Secret, P2SC: Python 2 source code, ICV: Improper Certificate Validation, CInj: Command Injection, XSS: Cross-site Scripting, AFW: Arbitrary File Write

terraform-generator (S)	v3.0.0	v4.0.0	v5.3.1	IPM:1/ReDos:2	0	0	IPM:1/ ReDos:2	IPM:1/ ReDos:2	0	0
community. general (S)	6.0.0	7.0.0	7.2.1	0	0	0	0	0	0	0
NetBeansPuppet (S)	v1.2	v2.0.0	V2.0.3	0	0	0	0	0	0	0
soccer-stats (X)	v0.0.1	v0.0.2	NA	0	0	0	0	0	0	0
ansible-runner (X)	1.0.1	2.0.0	2.3.3	0	0	HP:1	0	0	HP:2	HP:1
terraform-provider-azurerm (X)	v2.0.0	v3.0.0	v3.67.0	APK:7/ASK: 17/HP:260 /PHC:48	ASK:10	HP:22/P HC:1/LSK:3 /APK:10/TCI:5	APK:7/ ASK:7/ HP:260/PHC:48	LSK:1	LCI:1/APK:15 /HP:93/PHC :53/ASK:87	APK:17/ HP:282/PH C:49/ASK:7 /LSK:2/TCI:5
cookstyle (X)	v6.0.0	v7.0.0	v7.32.3	0	0	0	0	0	PHC:3	0
pulumi-datadog (X)	v3.0.0	v4.0.0	v4.21.0	APK:6/ PHC:5	APK:6/ PHC:5	LSK:12	0	LSK:12	HP:1/APK :3/NUE:1/PHC:33	0

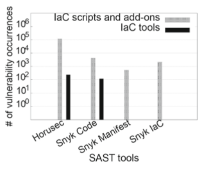
* Breakdown of vulnerabilities identified by Snyk across different IaC manifest files, code, and dependencies.
* Horusec results are presented at an aggregated level as the tool does not provide a location-wise breakdown.
* In the Name Column: T=tools, S= scripts, X=add-ons
* NA: Not Available

(Snyk manifest), and configuration of IaC manifest files (Snyk IaC), respectively. From Fig. 9, we note large differences in terms of vulnerability occurrences depending on the SAST tool and component on which it is applied.

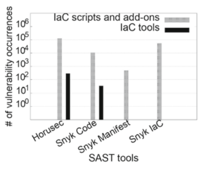
Although Horusec and Snyk agree on some instances of vulnerabilities (cf. Table 2) in IaC components, Horusec reveals substantially more occurrences of issues. Indeed, Horusec not only reports any vulnerabilities that are recognized to be associated with a given weakness (CWE) but also the issues that it considers as potential vulnerabilities(ex: Hardcoded-password vs Potential Hardcoded-password). On the other hand, Snyk will only report vulnerabilities that relate to CVEs. The distribution of vulnerabilities within tools, scripts, and add-ons (cf. Figure 9) reveal the following information:

- The SAST tools we used to test our datasets of tools, scripts, and add-ons utilize different security rules for source code analysis. We observe that Horusec and Snyk report different occurrences of vulnerabilities and offer different analysis capabilities. Horusec investigates the provided source code and reports the identified vulnerabilities and security smells (potential vulnerabilities), while Snyk offers three types of command lines for testing the security of the provided IaC projects: Snyk iac for infrastructure configuration testing, Snyk code for source code testing, and Snyk manifest for dependency testing.
- There are no vulnerabilities related to infrastructure configuration in our dataset of IaC tools because such configurations are only found in IaC scripts.
- Compared to vulnerabilities found in the code and infrastructure configurations of our components, vulnerabilities found in dependencies are almost imperceptible. These results suggest that practitioners should focus more on detecting and fixing vulnerabilities during the development process of these components.

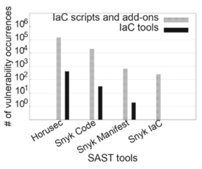
In terms of severity(cf. Figure 10), the results of the analysis show that all IaC tools are affected largely more by vulnerabilities that are tagged as "critical" than others. Actually, vulnerabilities tagged with high to low severity scores are minimal. On the other hand, IaC



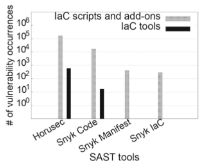
(a) For components associated with Ansible



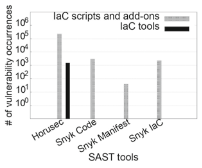
(b) For components associated with Terraform



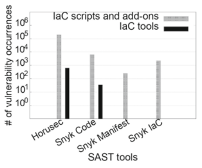
(c) For components associated with Chef



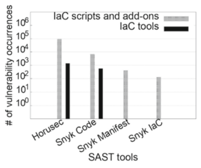
(d) For components associated with Puppet



(e) For components associated with Pulumi



(f) For components associated with Vagrant



(g) For components associated with Saltstack

Fig. 9 Vulnerabilities reported by SAST tools Applied on IaC components * The analyzed source code repository versions of IaC tools are the following: Ansible stable-2.11-branchpoint, Terraform v1.4.0-alpha20221207, Chef v18.0.197, Puppet v7.21.0, Pulumi v3.49.0, Vagrant v2.3.4, Saltstack v3005.1

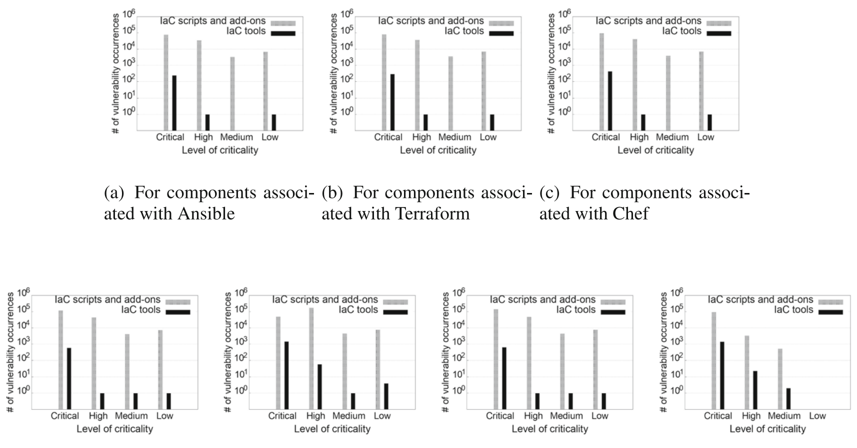
scripts and add-ons have more critical and high-severity vulnerabilities than medium to low-security ones. Despite the fact that Snyk results (cf. Table 2) have more Medium to Low severity vulnerabilities, Horusec's huge vulnerability proportions make it insignificant in the overall results.

We further observe that while SAST tools may find the same types of vulnerabilities, they assess their severity based on different standards and parameters. Nevertheless, all the vulnerabilities reported by Snyk and Horusec either involve CVEs or are associated with known CWEs. This suggests that they are significant in terms of potential exploitability. Therefore, regardless of their severity, exploiting these vulnerabilities can lead to a significant impact on IaC.

4.3.2 Vulnerable Files and Programming Languages Propensities

In previous sections, the vulnerability distributions have been presented in terms of occurrences by origin, i.e., within tools, scripts, and add-ons. We propose to investigate another dimension, i.e., localization in terms of file types and affected code by programming language.

Figure 11 summarizes the average occurrences of vulnerabilities per file type in IaC tools. Python and JSON files appear to be the most affected by IaC vulnerabilities. In contrast, certificate files (.ts, .tf, .crt) contain few vulnerabilities. It is noteworthy that in absolute numbers, most vulnerabilities are in Python, JSON, and Ruby files, which are common programming languages used in IaC. Furthermore, Python and JSON files are commonly used as configuration files in IaC tools. These files frequently define key infrastructure settings and operations, which may explain their susceptibility to vulnerabilities. Table 5 highlights the predominant programming languages used to develop the IaC tools in our dataset, where Python and JSON are highly represented. File types (e.g., JSON vs Java) have different importance depending on the IaC components: A JSON file, from the perspective of an IaC add-on will host the project dependencies (e.g., package versions). In contrast, from the



(a) For components associated with Ansible (b) For components associated with Terraform (c) For components associated with Chef

(d) For components associated with Puppet (e) For components associated with Pulumi (f) For components associated with Vagrant (g) For components associated with Saltstack

Fig. 10 Severity of Reported Vulnerabilities in IaC Components. * The analyzed source code repository versions of IaC tools are the following: Ansible stable-2.11-branchpoint, Terraform v1.4.0-alpha20221207, Chef v18.0.197, Puppet v7.21.0, Pulumi v3.49.0, Vagrant v2.3.4, Saltstack v3005.1

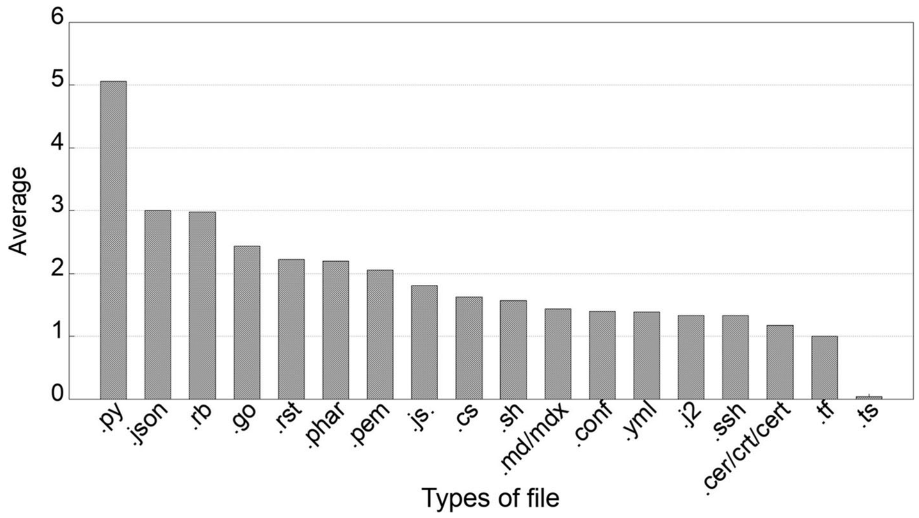


Fig. 11 Average Vulnerability Instances associated with Affected Types of File in IaC tools

perspective of an IaC script, it will host the main configurations (e.g., user creation, and privilege settings, etc.). This means that the types of vulnerabilities that will be detected will be different, and thus overall, the file types that make up most of the component files will have an impact on the vulnerability landscape of that component.

Figure 12 illustrates how, in contrast to the case of IaC tools, vulnerabilities of IaC scripts and add-ons are mainly located in general-purpose programming language files (e.g., .js, .java, .rb, .py, etc).

Configuration files, which are core IaC files, are significantly affected by vulnerabilities in IaC. Our study further reveals that such vulnerabilities are often labeled as being severe. We postulate that a community effort to systematically address vulnerabilities in Manifest files will substantially improve IaC security in various axes, including authentication, access control, etc.

4.3.3 DevOps profiles investigation

After investigating the origin and location of the widespread vulnerabilities within IaC, we study who (between developers and IT operators) bears the responsibility of introducing them.

We explore the list of edited files in our dataset of IaC tools repositories. The most frequently edited file types (Fig. 13) include typical development files (e.g., .rb, .go, .js,

Table 5 Languages used to program our dataset of IaC tools

IaC tools	Ansible	Terraform	Chef	Puppet	Pulumi	Vagrant	Saltstack
Programming Languages	Python	Go	Ruby	Ruby	Go	Ruby	Python

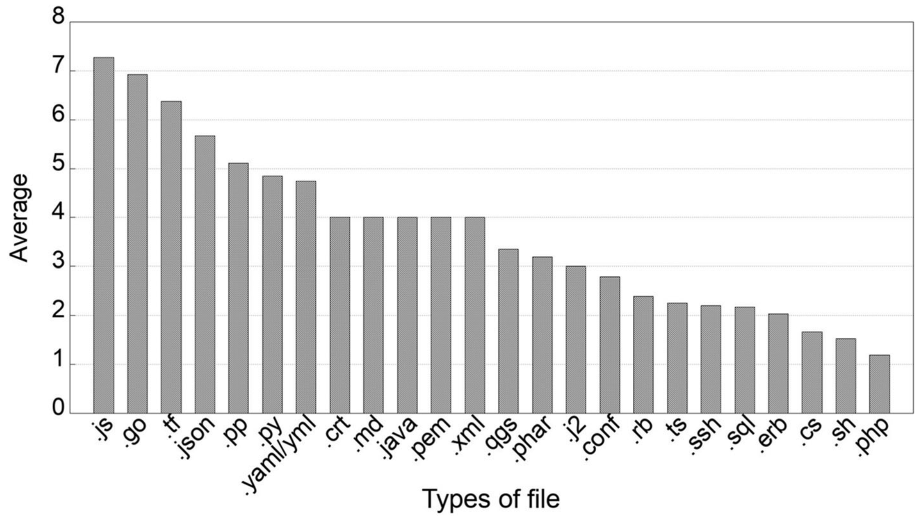


Fig. 12 Average Vulnerability Instances associated with Affected Types of File in IaC scripts and add-ons

.html), manifest files (e.g., .yml, .yaml, .json), specification/configuration files (e.g., .toml, .spec), and script files (e.g., .sh), listed here in descending order of occurrence. These results show that development files are more involved in the making of IaC tools than operation files. Therefore, the profile of those who create infrastructure tools is probably that of developers. However, we observe that development files (.js, .go, etc.) contain the least vulnerabilities inside IaC tools in Fig. 11 compared to manifest file types except for Python and Ruby files. This is because, in the context of Infrastructure as Code, these two specific files are strongly used for configuration purposes (cf. Figure 5). Consequently, manifest files and a significant portion of development files, particularly those frequently edited in IaC tool repositories, are likely handled by IT operators. We validated this hypothesis from samples of IaC tools' projects (cf. Tables 3, 4) by manually investigating the GitHub profiles of maintainers who edited manifest files to determine their specialization. Furthermore, after we manually investigated the edits from these maintainers on those IaC tools, we could match them with the vulnerabilities that Snyk and Horusec had detected in those projects. Overall, we observed that IT operators are mainly responsible for introducing vulnerabilities in IaC tools' environments.

On the other hand, we investigated the list of edited files in IaC scripts and add-ons' source code. Notably, we have 5 types of files that are often affected as illustrated in Fig. 14. Those are Python files, reStructuredText files (.rst), Nullsoft Scriptable Install System files (.nsi), YAML/YML files, and shell files. We thus note that the most edited files in IaC scripts and add-on components are all operation files. Nevertheless, they were not the most vulnerable files reported (cf. Figure 12).

In addition, we observe that on the OWASP list of vulnerabilities, manifest vulnerabilities (Table 2) barely have high-severity vulnerabilities and zero critical vulnerabilities. We also note that there are more files that match the developer profile, such as Java, PHP, or JavaScript files, in Fig. 14 than in Fig. 13. These results suggest that developers are the most responsible for vulnerabilities inside IaC scripts and add-on environments. We again manually investigated the specialization of maintainers who edited development files from the samples of scripts and add-ons in Tables 3 and 4 based on their GitHub profiles, and confirmed that

Files extensions	Ansible	Terraform	Chef	Puppet	Pulumi	Vagrant	Saltstack
.yml/.yaml	3,259	127	1,918	538	999	5,775	171
.json	0	966	95	0	170	174	0
.py	102,108	0	0	0	1,245	0	394,750
.rb	0	50	945,912	1,235	0	95,748	0
.go	0	97,844	0	0	53,463	1,313	0
.js	234	2	0	0	76	0	0
.sh	55	44	0	0	110	56	132
.gemspec	0	0	576	208	0	298	0
.erb	0	2,328	0	78	0	300	0
.mod	0	365	0	0	820	106	0
.sum	0	370	0	0	426	107	0
.cs	0	0	0	0	138	0	0
.proto	0	0	0	0	142	68	0
.spec	118	0	0	8	0	28	0
.ts	0	0	0	0	3,768	0	0
.rst	58	0	0	0	0	0	5,252
.toml	0	0	53	0	0	0	0
.html	2,100	0	0	0	0	0	0

Fig. 13 List of file types edited by IaC tools maintainers

these maintainers indeed mainly have developer profiles. From the manual checking of their edits on development files, we could match some of them with the vulnerabilities our SAST tools had detected. Overall, we can confirm that developers introduce more vulnerabilities in scripts and add-ons.

Investigations into contributors of IaC vulnerability reveal that IT operators are mainly responsible for vulnerabilities in IaC tools, while developers are more responsible for those that occur in IaC scripts and add-ons. Therefore, we confirm that in IaC, similarly to other settings, vulnerabilities are likely to be introduced when an actor (e.g. an IT operator) modifies components outside of his/her area of expertise (e.g. IaC tool).

Files extensions	Ansible	Terraform	Chef	Puppet	Pulumi	Vagrant	Saltstack
.yml/.yaml	60,162	5,612	960	590	2,960	1,891	111
.json	1,104	13,493	401	147	181	333	172
.py	162	270	905	1332	107	400	212
.rb	63	896,448	2,012	1,448	1,512	604	1,296
.go	3,077	996	189	0	222	546	954
.sh	305	604	1,400	194	82	1,890	320
.php	126	738	4,719	111	119	0	25
.java	1,090	244	322	1,030	200	484	19,740
.js	201	214	396	2,058	94	168	204
.xml	2,408	140	189	1,440	45	744	51,420

Fig. 14 List of file types edited by IaC scripts and add-ons maintainers

4.4 Summary of Key Findings

- Answer to RQ1: IaC is affected by all OWASP Top 10 categories of vulnerabilities. The distribution of these vulnerabilities, as well as their severity, underscores the significant security concerns within the IaC realm.
  ⇒ Vulnerabilities in IaC components do not appear to be largely addressed in research and practice. Yet, given the importance of IaC in today's IT infrastructure, those vulnerabilities, which are easily discoverable by existing tools could be exploited by malicious parties. It is, therefore, essential to develop pattern-based tooling for fixing IaC vulnerabilities at scale with adequate documentation for practitioner adoption.
- Answer to RQ2: The analysis of development artifacts reveals that IaC maintainers detect vulnerabilities, using security linters, for example, and actually fix them in their projects. However, we also note that a variety of vulnerability instances across the severity spectrum survive changes that evolve IaC components.
  ⇒ We conclude that while IaC maintainers demonstrate a certain degree of proactive behavior in addressing vulnerabilities, they do not seem to adhere to established automated practices and routines for the consistent and systematic detection of IaC vulnerabilities.
- Answer to RQ3: The security analysis data reveal that the most severe and recurrent vulnerabilities in IaC components are mainly located in manifest files, database files, and other configuration files in this order. Our results further indicate that DevOps actors are associated with vulnerable components which they are the least experienced with.

5 Implications for Practice and Research

The findings of this study constitute valuable insights with implications for practitioners and researchers.

Implications for Practitioners

- *SAST Tools Limitations:* In contrast to prior work, this study explored beyond a single type of IaC component and applied two different SAST tools, providing new insights into their limitations: first, a given SAST tool may apply to a specific type of component but not another: e.g., Horusec can find vulnerabilities in source code but leaves out vulnerabilities of configuration files, which are yet important in IaC. Second, SAST tools may not consistently cover the same type of vulnerabilities, making applying a single tool a risk of having higher false negatives in detecting vulnerabilities.
  ⇒ This implication is in line with the best practices detailed in the literature (Afaneh et al. 2023; Rajapakse et al. 2021) and the conclusions of the review by Aslan et al. (2023): Practitioners should account for tool compatibility challenges when planning for static security analysis of IaC.
- *Standardization of IaC Security Risks:* Our study covering various components associated with seven IaC tools highlights common (i.e., shared across various IaC tools), recurrent (i.e., frequent within components), and severe types of vulnerabilities. This implies that beyond the vulnerabilities arising from mistakes in writing code in different languages or writing configuration files for different IaC tool types, there are IaC design flaws that practitioners do not manage adequately. e.g., practitioners often forget to ensure the security of the link for updating packages in IaC.

⇒ This implication confirms the relevance of the very recent efforts to propose an OWASP Top 10 for CI/CD (the first OWASP Top 10 for CI/CD Security Risks was released on March 21, 2023).

- *Security Awareness in IaC:* Our investigation of vulnerability authorships from DevOps profiles demonstrates that all contributors introduce vulnerabilities into IaC components regardless of their specialization. The results suggest that these contributors, during the development, either do not systematically apply the existing SAST tools as part of their CI/CD pipelines, or the applied SAST tools are not effective, or the outputs of the SAST tools are ignored to some extent. In any case, this implies a strong need for the IaC practice to consider the sensitization of contributors on the importance of security.
  ⇒ Through cross-disciplinary training, the establishment of "security champions", or security certifications within teams, practitioners can bridge gaps between development, operations, and security disciplines (Sánchez-Gordón and Colomo-Palacios 2020; Akbar et al. 2022; Rajapakse et al. 2022), fostering better practices.
- *Open-Source Community Initiatives:* Each of the seven IaC tools involved in this study is maintained by specific open-source communities. Our results imply that different communities work on improving the security of their utilized components, and yet they face common security problems. This suggests substantial opportunities to mutualize effort in community-driven contributions.

Implications for Research

- *Addressing Tool Limitations:* Our results show that, by design, SAST tools do not consistently detect the same types of vulnerabilities in the same analyzed projects or components. They are thus complementary. When they do detect the same types of vulnerabilities, they may not find them in the same locations. This implies that there are inconsistencies between tools with respect to the definition of a given vulnerability or the implementation of the detection rule/pattern.
  ⇒ Further research on how to harmonize the definition and description of vulnerabilities for SAST tools would be beneficial for accurate vulnerability detection and enhanced remediation in IaC components.
- *Addressing Vulnerability Persistence:* Our results show that some vulnerabilities remain present across several versions of the project. Such persistence can stem from different situations: no SAST tools are used by IaC practitionners; the used SAST tools do not detect the types of vulnerabilities that are persistent; vulnerabilities detected by SAST tools are ignored by IaC maintainers.
  ⇒ Remediation of vulnerabilities in IaC could be facilitated through research on end-to-end approaches investigated in the repair community (Chen et al. 2019; Jin et al. 2023; Leotta et al. 2016) for the detection, classification, localization, and fixing of vulnerabilities.
- *Confirming Conclusions from Prior Work:* Our investigation findings are also in line with previous studies which discussed how "non-experts" are more likely to introduce bugs into code (Bird et al. 2011; Palix et al. 2011).
  ⇒ Sociotechnical dynamics (Thomas et al. 2018; Tahaei and Vaniea 2019), including team structures and expertise levels, have an impact on the introduction and remediation of vulnerabilities in the domain of IaC, and thus should be further investigated by the research community.
- *Expanding the Analysis Scope for IaC:* while current literature largely focuses on improving the security of IaC scripts (Rahman et al. 2019; Rahman and Williams 2021; Rahman

et al. 2021), the results from our comprehensive study suggests that security efforts should be extended to IaC tools and add-ons as we find redundant and severe security vulnerabilities in those components. Additionally, the vulnerability types emerging from our results suggest that IaC has specificities due to its internal paradigms, such as serverless (with Virtual Machines) and containerized architectures (Sultan et al. 2019; Bila et al. 2017). Those paradigms are associated with specific vulnerability patterns that deserve targeted research.

Broader Impact The insights provided by this study align with technological trends such as AI-driven IaC tools and DevOps automation (Camacho 2024; Pakalapati et al. 2023). Improved IaC security practices, for example, through the use of Large Language Models (LLMs) as static analyzers (Yadav et al. 2021), have the potential to enhance the resilience of critical IT infrastructures and reduce the risk of misconfigurations in such environments.

6 Threats to Validity

Our study bears a number of threats to validity, which we try to mitigate:
Infrastructure as Code's Components. Our study focuses on a sample number of tools for collecting IaC. We considered a set of free and open-source ones. Our results may, therefore, be only representative of these tools. Nevertheless, we attempted to mitigate this threat by considering popular tools which are used even in industry.
Vulnerability assessment. As an external threat to validity, we have collected our subject projects from GitHub default repository branches. Since such master branches are supposed to host clean, tested, and ready-for-deployment code, it is possible that our study was not able to capture the entire variety of vulnerabilities in IaC. While our results indicate that the use of static security analysis in CI pipelines is not a systematic practice among maintainers, if maintainers performed static security analysis before merging code into the default branches, some vulnerabilities may have been addressed prior to our analysis. Nevertheless, this threat is mitigated by the use of various tools.

When presenting the vulnerabilities in infrastructure components, we only considered the top 10 under each OWASP category. This is the threat to conclusion validity since it may be perceived that other levels of severity are not sufficiently problematic. Actually, we could not display all vulnerability types for space concerns. However, repeated Low to Medium vulnerabilities could have a severe impact on aviation and medical fields, for example.
Static Analyzers. Also, static analyzers are known to produce false positives (Habib and Pradel 2018; Cadar and Donaldson 2016). Although we did not manually verify each reported warning, we mainly reported warnings flagged by two mature and widely adopted tools that have databases of CVEs and CWEs, respectively: Snyk and Horusec. In our security analysis, Snyk shows more types of vulnerabilities than Horusec. Likewise, other existing SAST tools could also show new types of vulnerabilities that our selected tools did not because they are built differently. Hence, we may have missed vulnerabilities in analyzing IaC's components.
Maintainers security responsibilities Moreover, our blaming method finds the possible authors of the detected vulnerabilities according to their specialization and expertise. We assume that their DevOps profile is based on the files they manipulate the most (Palix et al. 2011) and on the types of edits they make to the source code. We cannot prove the accuracy of this method since we did not manually check all hundreds of repositories from the scripts, tools, and add-ons. However, we relied on samples of each IaC component to manually eval-

uate the types of edits (simple comments or real features) and profiles of their contributors (using their GitHub profile information).

7 Related Work

Broadly, the security of IaC has been mentioned in recent literature as a key concern of modern development (Rajapakse et al. 2022; Omoike et al. 2024). In relation to our work, some prior studies have provided insights into the application of vulnerability detectors to IaC components. We discuss these studies to highlight the novel insights that stem from our large-scale investigation.

7.1 IaC Security

The security of Infrastructure is a recent but emerging area of focus in DevSecOps (Castro Sánchez 2020; Reddy et al. 2021). The existing literature explores various dimensions of IaC security, including integrating security into DevOps pipelines and statistically analyzing IaC scripts. Mohan and Othmane (Mohan and Othmane 2016) provide an overarching perspective on integrating security into DevOps, emphasizing the collaboration among development, operations, and security teams. Their work identifies key challenges such as cultural shifts in IT organizations (Thomas et al. 2018), the integration of security into DevOps automation (Ahmed and Francis 2019), and tooling (Cankar et al. 2023), which resonate with the fundamental DevSecOps principles. Our results, based on the execution of SAST tools, highlight evidence of the use (or lack thereof) of tools for IaC security and the profiles of the contributors of these vulnerabilities. Dynamic analysis and sandbox testing methods have been used to assess the security of IaC deployments (Hasan et al. 2020). However, dynamic analysis is expensive, does not scale, and is challenging for DevSecOps automation. Our work focuses on static security analysis as we look to run large-scale experiments.

Recent research on IaC security (Petrović 2023) has explored LLM-enabled static analysis for IaC scripts, demonstrating novel applications of AI in DevSecOps pipelines. However, the scope of such studies is limited to specific IaC tools (e.g., Ansible, Terraform), and the findings are not generalized across IaC components (scripts, add-ons, tools).

Overall, although the referred literature highlights the importance of secure IaC environments and addresses critical aspects of DevOps and IaC security, it often focuses on specific components (generally IaC scripts), limiting their applicability across the whole spectrum of components (scripts, add-ons, tools) in IaC. Our work attempts to fill this gap.

7.2 Static Vulnerability Detection and Categorization in IaC

The integration of security into DevSecOps automated pipelines requires security tools that are platform-compatible and easy to use Myrbakken and Colomo-Palacios (2017); Goldschmidt and McKinnon (2016). Our work starts with the assumption that static security testing offers the best compromise between effectiveness in discovering vulnerabilities and efficiency in terms of execution in the context of infrastructure management.

In the literature, several studies (Reis et al. 2023) have employed static analysis techniques, e.g., linters, to detect vulnerabilities in IaC scripts. While these tools can effectively detect certain types of vulnerabilities, they are only applicable to specific types of IaC scripts: SLAC (Rahman et al. 2021) was designed for Ansible and Chef scripts and SLIC (Reis et al.

2023) for Puppet scripts. Our work extends the static analysis of IaC components beyond scripts, to tools and add-ons.

Unlike prior work, which focuses on source code, we also analyze manifest files (main configuration files in IaC scripts) and dependencies using the capabilities of the leveraged SAST tools.

In terms of vulnerability categorization, our work extends prior works (Rahman et al. 2019; Rahman and Williams 2021; Rahman et al. 2021) by providing insights based on the combined analysis across seven different IaC tools, covering more issues, and leveraging a standard (OWASP Top 10) for providing a view on the prevalence of vulnerabilities and their distribution in IaC file types, components, and origins (manifest, code, dependency).

7.3 Evolution of vulnerabilities and Identification of Their Authors

The specialization of software contributors has been investigated in the literature for various purposes. For example, for Linux, Palix et al. (2011) have proposed to investigate who introduces faults in the many versions of the Linux kernel. They used static analysis tools (Coccinelle) for fault detection and developed a heuristic based on the type of files a maintainer frequently edited to associate specialization. We follow their methodology in our work on IaC vulnerability introduction. Instead of Coccinelle, we leverage specialized IaC security analyzers.

8 Conclusion

In this study, we have provided a comprehensive investigation into the security landscape of Infrastructure as Code through its components: tools, scripts, and add-ons. Leveraging the analysis capabilities of Snyk and Horusec, we analyzed a large number of component repositories. In every repository, we analyzed three main vulnerability origins: the code, the dependencies, and the manifests, and we categorized the different security issues according to the OWASP Top 10 taxonomy. Our findings have unveiled key insights: i) Vulnerabilities are widespread across all IaC components with most of the OWASP Top 10 vulnerability types manifesting across each analyzed component; ii) Although IaC maintainers demonstrate a proactive stance toward vulnerabilities, they do not seem to follow standard automatic practices and routines to check for the existence of such issues consistently and systematically, iii) Manifest files emerge as the most vulnerable artifacts within the IaC ecosystem, iv) IaC vulnerabilities frequently stem from non-specialized actors attempting modifications within components outside their expertise. These findings collectively underscore the critical importance of reinforcing security awareness among developers and IT operators. The imperative is to galvanize a paradigm shift towards the systematic adoption of DevSecOps within DevOps teams. By integrating security checks into the heart of IaC component development, we can facilitate the continual evolution of secure infrastructure environments. As future work, we will focus on advancing techniques for the more precise characterization and detection of vulnerabilities within Infrastructure as Code manifests.

Acknowledgements This work is funded by the Ministére des Affaires Etrangéres et Européennes (MAEE) of Luxembourg through their Digital4Development (D4D) portfolio under the project LuxWAyS (Luxembourg/West-Africa Lab for Higher Education Capacity Building in CyberSecurity and Emerging Topics in ICT4Dev).

Author Contributions Aicha War: Methodology, Experiments and Writing.
Alioune Diallo: Contributions to Review, Editing, and Writing.
Andrew Habib: Contributions to Review, and Writing.
Jacques Klein: Contributions to Review, and Writing.
Tegawendé F. Bissyandé: Contributions to Review, and Writing.

Data Availability The datasets used in the present study are available in our repository (War et al. n.d.).

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

Ethical approval This study does not involve human participants, animals, or other entities requiring ethical oversight. Consequently, no ethical approval was required.

Informed consent No human participants were involved in this study, and informed consent was therefore not applicable.

References

Afaneh S, Al-Mousa MR, Al-hamid HS, Bara'h Suliman A-A, Alia M, Almimi H, Alkhatib AA (2023) Security challenges review in agile and devops practices. In: 2023 International conference on information technology (ICIT). IEEE, pp 102–107

Ahmed Z, Francis SC (2019) Integrating security with devsecops: techniques and challenges. In: 2019 International Conference on digitization (ICD). IEEE, pp 178–182

Akbar MA, Smolander K, Mahmood S, Alsanad A (2022) Toward successful devsecops in software development organizations: A decision-making framework. Inf Softw Technol 147:106894

Almuairfi S, Alenezi M (2020) Security controls in infrastructure as code. Comput Fraud Secur 2020(10):13–19

Armenise V (2015) Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery. In: 2015 IEEE/ACM 3rd International Workshop on Release Engineering. IEEE, pp 24–27

Aslan Ö, Aktuğ SS, Ozkan-Okay M, Yilmaz AA, Akin E (2023) A comprehensive review of cyber security vulnerabilities, threats, attacks, and solutions. Electronics 12(6):1333

Bila N, Dettori P, Kanso A, Watanabe Y, Youssef A (2017) Leveraging the serverless architecture for securing linux containers. In: 2017 IEEE 37th international conference on distributed computing systems workshops (ICDCSW). IEEE, pp 401–404

Bird C, Nagappan N, Murphy B, Gall H, Devanbu P (2011) Don't touch my code! examining the effects of ownership on software quality. In: Proceedings of the 19th ACM SIGSOFT symposium and the 13th european conference on foundations of software engineering. ESEC/FSE '11. Association for Computing Machinery, New York, NY, USA, pp 4–14. https://doi.org/10.1145/2025113.2025119

Cadar C, Donaldson AF (2016) Analysing the program analyser. In: Proceedings of the 38th international conference on software engineering companion, pp 765–768

Camacho NG (2024) Unlocking the potential of ai/ml in devsecops: effective strategies and optimal practices. J Artif Intell Gen Sci (JAIGS) 3(1):106–115

Cankar M, Petrovic N, Pita Costa J, Cernivec A, Antic J, Martincic T, Stepec D (2023) Security in devsecops: applying tools and machine learning to verification and monitoring steps. In: Companion of the 2023 ACM/SPEC international conference on performance engineering, pp 201–205

Cankar M, Petrovic N, Pita Costa J, Cernivec A, Antic J, Martincic T, Stepec D (2023) Security in devsecops: applying tools and machine learning to verification and monitoring steps. In: Companion of the 2023 ACM/SPEC international conference on performance engineering. ICPE '23 Companion, pp. 201–205. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3578245.3584943

Castro Sánchez JE (2020) Devsecops: implementación de seguridad en devops a través de herramientas open source

Cepuc A, Botez R, Craciun O, Ivanciu I-A, Dobrota V (2020) Implementation of a continuous integration and deployment pipeline for containerized applications in amazon web services using jenkins, ansible and kubernetes. In: 2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet). IEEE, pp 1–6

Chang Y-Y, Zavarsky P, Ruhl R, Lindskog D (2011) Trend analysis of the cve for software vulnerability management. In: 2011 IEEE Third international conference on privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing. IEEE, pp 1290–1293

Chen Z, Kommrusch S, Tufano M, Pouchet L-N, Poshyvanyk D, Monperrus M (2019) Sequencer: sequence-to-sequence learning for end-to-end program repair. IEEE Trans Softw Eng 47(9):1943–1959

Cottrell N, Cottrell N (2020) Deployment and monitoring. MongoDB Topology Design: Scalability, Security, and Compliance on a Global Scale, 151–171

Cruz DB, Almeida JR, Oliveira JL (2023) Open source solutions for vulnerability assessment: a comparative analysis. IEEE Access 11:100234–100255

Di Stasio V (2022) Evaluation of static security analysis tools on open source distributed applications. PhD thesis, Politecnico di Torino

Druta R, Botosan-Bora N, Iovan M, Cruzes DS An analysis of infrastructure as code security in an industrial setting. SSRN 4461951

Elrowayati A, Fadeel A (2024) Sast tools and manual testing to improve the methodology of vulnerability detection in web applications. Int J Eng Inf Technol (IJEIT) 12(1):79–83

Goldschmidt M, McKinnon M (2016) Devsecops-agility with security. Technical report, Technical report, Sense of Security

Habib A, Pradel M (2018) How many of all bugs do we find? a study of static bug detectors. In: 2018 33rd IEEE/ACM international conference on automated software engineering (ASE), pp 317–328. https://doi.org/10.1145/3238147.3238213

Hasan M, Bhuiyan FA, Rahman A (2020) Testing practices for infrastructure as code, pp 7–12. https://doi.org/10.1145/3416504.3424334

Hornbeek M (2015) Devops makes security assurance affordable. https://devops.com/devops-makes-security-assurance-affordable

Hortlund A (2021) Security smells in open-source infrastructure as code scripts: a replication study

Houde L, Jacob D, Rabemanantsoa T, Rey J-F (2021) Gestion automatique d'environnement virtuel (gaev). PhD thesis, INRAE

Ibrahim A, Yousef AH, Medhat W (2022) Devsecops: a security model for infrastructure as code over the cloud. In: 2022 2nd International mobile, intelligent, and ubiquitous computing conference (MIUCC). IEEE, pp 284–288

Jin M, Shahriar S, Tufano M, Shi X, Lu S, Sundaresan N, Svyatkovskiy A (2023) Inferfix: end-to-end program repair with llms. In: Proceedings of the 31st ACM joint european software engineering conference and symposium on the foundations of software engineering, pp 1646–1656

Leotta M, Clerissi D, Ricca F, Tonella P (2016) Approaches and tools for automated end-to-end web testing. In: Advances in computers. Elsevier, vol 101, pp 193–237

Martin B (2019) Common vulnerabilities enumeration (cve), common weakness enumeration (cwe), and common quality enumeration (cqe) attempting to systematically catalog the safety and security challenges for modern, networked, software-intensive systems. ACM SIGAda Ada Lett 38(2):9–42

Mohan V, Othmane LB (2016) Secdevops: is it a marketing buzzword? - mapping research on security in devops. In: 2016 11th International conference on availability, reliability and security (ARES), pp 542–547. https://doi.org/10.1109/ARES.2016.92

Mohan V, Othmane LB (2016) Secdevops: is it a marketing buzzword?-mapping research on security in devops. In: 2016 11th International conference on availability, reliability and security (ARES). IEEE, pp 542–547

Morris K (2020) Infrastructure as Code. O'Reilly Media. https://books.google.lu/books?id=R24NEAAAQBAJ

Myrbakken H, Colomo-Palacios R (2017) Devsecops: a multivocal literature review. In: Software Process Improvement and Capability Determination: 17th International Conference, SPICE 2017, Palma de Mallorca, Spain, October 4–5, 2017, Proceedings. Springer, pp 17–29

Omoike O et al (2024) Devsecops in aws: embedding security into the heart of devops practices. Int J Sci Res Arch 13(2):1309–1313

Opdebeeck R, Zerouali A, De Roover C (2023) Control and data flow in security smell detection for infrastructure as code: Is it worth the effort? In: 2023 IEEE/ACM 20th international conference on mining software repositories (MSR), pp 534–545. https://doi.org/10.1109/MSR59073.2023.00079

Pakalapati N, Konidena BK, Mohamed IA (2023) Unlocking the power of ai/ml in devsecops: strategies and best practices. J Knowl Learn Sci Technol 2(2):176–188

Palix N, Thomas G, Saha S, Calvès C, Lawall JL, Muller G (2011) Faults in linux: ten years later. In: ASPLOS 2011 - 16th international conference on architectural support for programming languages and operating systems. ACM, Newport Beach, California, United States, pp 305–318. https://doi.org/10.1145/1950365.1950401https://hal.archives-ouvertes.fr/hal-00940355

Paloviita O, et a (2022) Infrastructure as code for managed service providers: a case study

Petrović N (2023) Chat gpt-based design-time devsecops. In: 2023 58th International scientific conference on information, communication and energy systems and technologies (ICEST). IEEE, pp 143–146

Petrović N (2023) Chatgpt-based design-time devsecops

Rahman A (2018) Anti-patterns in infrastructure as code. In: 2018 IEEE 11th international conference on software testing, verification and validation (ICST), pp 434–435. https://doi.org/10.1109/ICST.2018.00057

Rahman A (2018) Characteristics of defective infrastructure as code scripts in devops. In: 2018 IEEE/ACM 40th international conference on software engineering: companion (ICSE-Companion), pp 476–479

Rahman AAU, Williams LA (2016) Security practices in devops. Proceedings of the symposium and bootcamp on the science of security

Rahman A, Williams L (2021) Different kind of smells: security smells in infrastructure as code scripts. IEEE Secur Priv 19(3):33–41. https://doi.org/10.1109/MSEC.2021.3065190

Rahman A, Williams L (2021) Different kind of smells: security smells in infrastructure as code scripts. IEEE Secur Priv 19(3):33–41

Rahman A, Rahman MR, Parnin C, Williams L (2021) Security smells in ansible and chef scripts: a replication study. ACM Trans Softw Eng Methodol (TOSEM) 30(1):1–31

Rahman A, Farhana E, Parnin C, Williams L (2020) Gang of eight: a defect taxonomy for infrastructure as code scripts. In: 2020 IEEE/ACM 42nd international conference on software engineering (ICSE), pp 752–764. https://doi.org/10.1145/3377811.3380409

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

Rahman A, Rahman MR, Parnin C, Williams L (2021) Security smells in ansible and chef scripts: a replication study. ACM Trans Softw Eng Methodol 30(1). https://doi.org/10.1145/3408897

Rajapakse RN, Zahedi M, Babar M. (2021) An empirical analysis of practitioners' perspectives on security tool integration into devops. In: Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM), pp 1–12

Rajapakse RN, Zahedi M, Babar MA, Shen H (2022) Challenges and solutions when adopting devsecops: a systematic review. Inf Softw Technol 141:106700

Reddy Konala PR, Kumar V, Bainbridge D (2023) Sok: static configuration analysis in infrastructure as code scripts. In: 2023 IEEE international conference on cyber security and resilience (CSR), pp 281–288. https://doi.org/10.1109/CSR57506.2023.10224925

Reddy AK, Alluri VRR, Thota S, Ravi CS, Bonam VSM (2021) Devsecops: integrating security into the devops pipeline for cloud-native applications. J Artif Intell Res Appl 1(2):89–114

Reis S, Abreu R, d'Amorim M, Fortunato D (2023) Leveraging practitioners' feedback to improve a security linter. In: Proceedings of the 37th IEEE/ACM international conference on automated software engineering. ASE '22. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3551349.3560419

Rodríguez Couto A (2022) Ferramenta para automatización de traballos por lotes con apache spark

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

Sánchez-Gordón M, Colomo-Palacios R (2020) 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

Shackleford D (2017) The devsecops approach to securing your code and your cloud. SANS institute infosec reading room a devsecops playbook

Sokolowski D, Spielmann D, Salvaneschi G (2024) Automated infrastructure as code program testing. IEEE Trans Softw Eng

Sultan S, Ahmad I, Dimitriou T (2019) Container security: issues, challenges, and the road ahead. IEEE Access 7:52976–52996

Tahaei M, Vaniea K (2019) A survey on developer-centred security. In: 2019 IEEE european symposium on security and privacy workshops (EuroS&PW). IEEE, pp 129–138

Thomas TW, Tabassum M, Chu B, Lipford H (2018) Security during application development: an application security expert perspective. In: Proceedings of the 2018 CHI conference on human factors in computing systems, pp 1–12

Valkeinen M (2022) Cloud infrastructure tools for cloud applications: infrastructure management of multiple cloud platforms. Master's thesis

Verdet A, Hamdaqa M, Da Silva L, Khomh F (2023) Exploring security practices in infrastructure as code: an empirical study. arXiv:2308.03952

War A, Habib A, Diallo A, Klein J, Bissyandé TF (n.d.) Security Vulnerabilities in Infrastructure as Code: What, How Many, and Who? https://github.com/Sherlock0001/empirical-study-iac.git

Yadav B, Choudhary G, Shandilya SK, Dragoni N (2021) Ai empowered devsecops security for next generation development. In: Frontiers in Software Engineering: First International Conference, ICFSE 2021, Innopolis, Russia, June 17–18, 2021, Revised Selected Papers 1. Springer, pp 32–46

Authors and Affiliations

Aicha War[1] · Alioune Diallo[1] · Andrew Habib[1,2] · Jacques Klein[1] · Tegawendé F. Bissyandé[1]

✉ Aicha War
aicha.war@uni.lu

Alioune Diallo
alioune.diallo@uni.lu

Andrew Habib
andrew.a.habib@gmail.com

Jacques Klein
jacques.klein@uni.lu

Tegawendé F. Bissyandé
tegawende.bissyande@uni.lu

1 University of Luxembourg, Esch-sur-Alzette, Luxembourg

2 ABB Corporate Research Center of Germany, Baden, Switzerland