Metadata Assisted Supply-Chain Attack Detection for Ansible

Pandu Ranga Reddy Konala, Vimal Kumar, David Bainbridge, and Junaid Haseeb

School of Computing and Mathematical Sciences,
University of Waikato,
Hamilton, New Zealand 3240
{pkonala, vkumar, davidb, jhaseeb}@waikato.ac.nz

Abstract. This study examines metadata-assisted detection of supply chain attacks in Infrastructure as Code (IaC), focusing on metadata's role in identifying security smells. Metadata, including dependency relationships and author records, provides insights into IaC scripts but remains underutilized by detection tools. The evaluation of static IaC smell detection tools highlights their limitations in incorporating metadata analysis. To address this, a methodology integrating metadata and dependency analysis was developed to identify security smells in dependency chains. An analysis of 482 Ansible Galaxy repositories identified vulnerabilities in 45 dependency chains, including reliance on deprecated dependencies (CWE-477), hard-coded credentials (CWE-798), and improper file permissions (CWE-280). Additionally, three repositories contained security vulnerabilities associated with output (CVE-2024-8775) and logging (CVE-2017-7550). The findings highlight the necessity of integrating metadata analysis with static code analysis for detecting security smells. This approach enhances IaC security and mitigates risks related to supply chain attacks.

Keywords: Infrastructure as Code (IaC), Ansible, Dependency management, Vulnerability detection, Supply chain attacks, Metadata.

1 INTRODUCTION

Software supply chain attacks/poisoning occur when malicious code is injected into the software development and distribution process. Notable incidents include the 2018 attack on the Node.js *eslint-scope* package, where attackers exploited a stolen *npm* token to release a malicious version designed to exfiltrate *npm* tokens from dependent machines [51]. Similarly, in 2022, a large-scale attack on a WordPress plugin and theme developer compromised 40 themes and 53 plugins, impacting over 360,000 active websites [20].

While such previous attacks primarily targeted traditional software, the scope of software development has expanded significantly. Developers now engage in

operations such as infrastructure management, which is managed through Infrastructure as Code (IaC). It shares similarities with traditional software development such as presence of rich metadata and versioning, particularly in its reliance on external components and repositories. Developers rarely create all components independently, instead utilizing prebuilt IaC dependencies, which may originate from the same or different authors. One analysis of GitHub repositories showed that open source projects, on average, rely on 180 external components [16]. These dependencies, in turn, rely on other dependencies, creating a more complex software supply chain and consequently a larger attack surface.

Despite the increased adoption of IaC in software development workflows, existing research on software supply chain security primarily focuses on traditional software, leaving gaps in understanding how supply chain vulnerabilities manifest in IaC environments. IaC repositories contain rich metadataincluding version history, dependency relationships, licensing, platform compatibility, and authorshipyet current security mechanisms do not fully leverage this metadata for supply chain vulnerability assessment.

This work investigates supply chain attacks in the context of IaC by analyzing dependencies and metadata associated with IaC repositories. Metadata plays a critical role in identifying potential attack vectors, such as tracing transitive dependencies and detecting poisoned repositories. By systematically examining metadata-driven relationships, this work provides a methodology for detecting supply chain vulnerabilities in IaC, addressing a critical gap in current security approaches.

To achieve this, we begin by examining the extent to which existing IaC smell detection tools incorporate metadata parameters in their analyses. This allows us to assess whether these tools effectively capture metadata-driven security concerns or if there are omissions that leave IaC repositories vulnerable. Next, we explore how dependency information within metadata fields can be leveraged to identify repositories that exhibit security smells, shedding light on how metadata can serve as a valuable asset in security assessment. Finally, we evaluate the susceptibility of Ansible software to supply chain attacks, identifying possible vulnerabilities to supply chain attacks by analyzing the dependency chain through metadata.

By structuring our study around these investigative directions, we aim to provide a comprehensive understanding of how metadata can enhance security assessments in IaC.

2 BACKGROUND

Supply chain attacks represent a cybersecurity risk, as demonstrated by incidents that reveal vulnerabilities across industries. The Colonial Pipeline ransomware attack, which leveraged a compromised password, disrupted fuel distribution in the United States, affecting gasoline and jet fuel along the East Coast [33]. Likewise, JBS, a meat producer, faced a supply chain ransomware attack attributed

to the REvil group, interrupting operations in Australia, Canada, and the United States [34].

The PyPI supply chain attack in 2024 targeted the Python Package Index (PyPI), a repository for Python libraries [41]. Attackers uploaded malicious packages that appeared legitimate, exploiting the trust embedded in open-source ecosystems. These packages included malware intended to exfiltrate information from compromised systems, posing risks to organizations that inadvertently integrated them. This incident exposed weaknesses in open-source supply chains and emphasized the need to verify dependencies prior to their inclusion in software projects.

Hossain Faruk et al. [18] investigated software supply chain security by examining vulnerabilities that adversaries can exploit at multiple points of the software development lifecycle. Their study highlights that reliance on open-source or third-party code necessitates measures to protect the software supply chain from malicious activities. The authors identified issues in software supply chain security to enhance organizational awareness and best practices in this field. They reviewed existing methods and frameworks for securing the software supply chain, offering insights into prevention, detection, assessment, and remediation of security issues.

Duan et al. [15] examined package managers for interpreted languages, specifically PyPI (Python), *npm* (Node.js), and RubyGems (Ruby), to detect malicious packages. Their approach employed use of regular expressions to build heuristic rules derived from known supply chain attacks and malware studies, integrating both static and dynamic code analysis. Additionally, metadata-based heuristics, including package names, authorship, popular packages with different authors, and version information, were incorporated. A dependency analysis further identified vulnerabilities such as single points of failure and risks associated with unmaintained packages.

Ohm et al. [35] conducted a comprehensive review of 20 publications focused on identifying and detecting software supply chain attacks. The reviewed studies were categorized into six groups: rules and heuristics, typosquatting, differential analysis, machine learning methods, anomaly detection, and clustering. Notably, six of these studies integrated metadata analysis with both static and dynamic code analysis, highlighting the significance of metadata in detecting supply chain attacks. The authors concluded that no single approach is universally effective; instead, employing multiple strategies can help identify malicious behaviors across various environments. They emphasized the need for future research adopting a multimodal approach for detecting security-breaching packages. Despite this progress in software supply chain security, similar studies focusing on IaC scriptsparticularly those that incorporate metadata for attack detectionare currently lacking, represents a promising area for further research.

2.1 Infrastructure as Code

Although existing studies are primarily focused on software code and security incidents, yet the use of IaC for building, orchestrating, and maintaining soft-

ware systems prompts an inquiry into whether IaC may experience supply chain attacks in the same manner as code. This issue arises because IaC and code share many common features. This subsection considers two aspects: the general concept of IaC and the ways developers and system administrators interact with it. The IaC technology stack is divided into three categories [50]:

- *Infrastructure provisioning* automates the allocation and management of hardware resources for deploying infrastructure components.
- *Configuration management* encodes system setups to maintain software environments.
- *Image building* generates machine and container images in standardized formats.

These categories collectively support automation in provisioning, configuration, and image-building procedures, facilitating IT operations. IaC is generally set up using either a standalone script or a group of scripts organized within a repository, collectively designed to accomplish a specific task. These scripts can be executed on a single system or across multiple systems. A typical example [26] of an IaC script (`main.yml`) is shown in Listing I:

Listing I: Sample Ansible Script

```
- name: Install Nginx
  hosts: web_servers
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
```

These IaC scripts are typically shared by developers or system administrators through technology specific code bases such as Ansible Galaxy or platforms such as GitHub. When scripts are downloaded using the software's command line interface, metadata is retrieved before the script itself. An example of metadata obtained from Ansible Galaxy [2] is shown in Listing II:

Listing II: Metadata of Sample Ansible Script


```
{"id": 56789,"upstream_id": null,
  "created": "2024-02-05","modified": "2024-02-05",
  "username": "devops_admin",
  "github_repo": "ansible-role-nginx","github_branch": "main",
```

```
"name": "nginx","description": "Install Nginx",
"summary_fields": {"dependencies": [
    {"id": 10, "name": "security.hardening"},
    {"id": 11, "name": "common.utils"}],
  "namespace": {"id": 3050,"name": "devops_admin"},
  "provider_namespace": {
    "id": 205,"name": "devops_admin",
    "repository": {"name": "ansible-role-nginx"},
    "tags": ["webserver", "nginx", "http"],
    "versions": [{"name": "1.1"},{"name": "1.0"}]},
  "download_count": 342560}}
```


The metadata retrieved from Ansible Galaxy provides detailed information about the IaC script through various key fields. This data is necessary for ensuring management of infrastructure automation processes. It provides essential information about the script, such as its dependencies, supported platforms, version requirements, and licensing, which enables developers and system administrators to assess compatibility and compliance before deployment. Metadata also facilitates traceability by identifying the script's origin, authorship, and version history, helping teams address issues and implement updates effectively. Additionally, usage metrics like download counts and community engagement provide valuable information about a script's reliability and level of adoption. While these metrics promote script reuse and collaboration, poor management can expose systems to potential vulnerabilities.

3 STATE-OF-THE-ART IaC SMELL DETECTION TOOLS

Software tools employ different mechanisms for processing input and generating output. Ohm et al. [35] conducted a survey of traditional software tools, where each tool in their respective studies described its approach to handling input and output data [35]. Similarly, we examine how static vulnerability analysis tools for IaC scripts process input and output data to identify security vulnerabilities. We define the term '*level of analysis*' where the levels indicate the assessment scope and influence the types of vulnerabilities that can be detected. This study identifies two levels of analysis in static vulnerability assessment: single file analysis, and repository-level analysis.

3.1 Level 1: Single File Analysis

Single file analysis, as shown in Fig. 1, evaluates individual IaC files in isolation, identifying localized misconfigurations, security smells, syntactical errors,

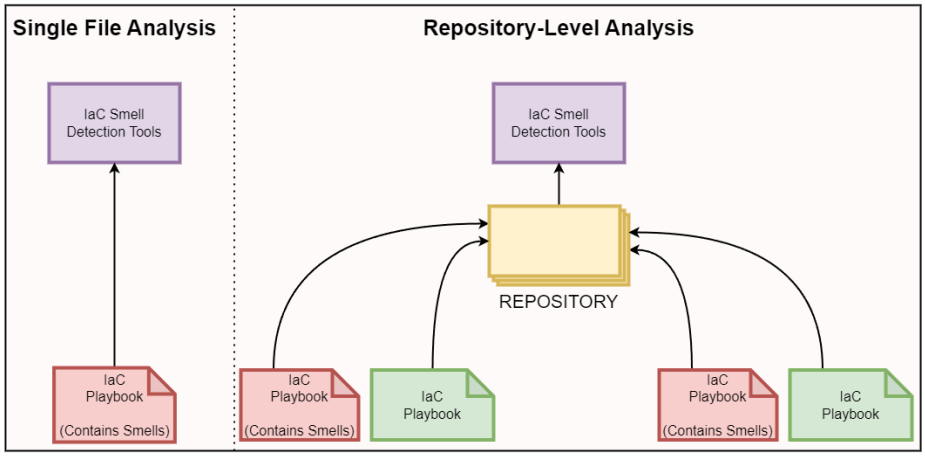

Fig. 1. Levels of Analysis

etc. This level examines hardcoded secrets, insecure configurations, and non-compliant syntax. Tools such as `ansible-lint` [49] and `yamllint` [12] verify that Ansible scripts conform to fundamental standards, while policy-as-code frameworks, including Checkov [6] and KICS [9], ensure that each file meets predefined security and compliance benchmarks.

This approach does not consider interdependencies between files or the broader operational context. It establishes a baseline of file-level security, ensuring that vulnerabilities are addressed before integration into larger systems.

3.2 Level 2: Repository-Level Analysis

Repository-level analysis, as shown in Fig. 1, expands the scope to assess an entire repository as a cohesive unit. This level detects cross-file issues that may not be identifiable through single file analysis, such as variable reuse, conflicting configurations, and role or module misconfigurations.

Policy-as-code tools, including Open Policy Agent (OPA) [36] and Checkov [6], are used at this level to enforce compliance and maintain consistency with organizational standards. By analyzing interactions between IaC components, this level provides insights into security issues that emerge when multiple files operate together.

3.3 Analysis of IaC Smell Detection Tools

An analysis of static IaC smell detection tools from both industry and academic sources, initially conducted by Konala et al. [22], was expanded to examine the mechanisms employed by these tools. Our study was guided by two questions:

1. What levels of analysis are implemented by these tools? (To understand how these tools process input and output data.)

Table 1. Static Code Analysis Tools to Detect Smells for IaC Software

Tool Name	Year	IaC Software Category	Availability	Levels of Analysis	Metadata Analysis
ACID[38]	2020	Configuration Management	Opensource	Repository Level	No
BARREL[7]	2018	Infrastructure Provisioning	Opensource	Single File	No
⋆ Checkov[6]	2021	Infrastructure Provisioning	Hybrid	Single File, Repository Level	No
⋆ CloudSploit[3]	2020	Infrastructure Provisioning	Proprietary	Repository Level	Unknown
CookStyle[10]	2016	Configuration Management	Opensource	Single File, Repository Level	No
DeepIaC[5]	2020	Configuration Management	Opensource	Single File	No
Foodcritic[43]	2011	Configuration Management	Opensource	Repository Level	No
GLITCH[42]	2022	Configuration Management	Opensource	Single File, Repository Level	No
Häyhä[25]	2021	Infrastructure Provisioning	Opensource	Single File	No
⋆ KICS[9]	2020	All	Opensource	Single File	No
Puppeteer[46]	2016	Configuration Management	Opensource	Repository Level	No
RADON[14]	2021	Configuration Management	Opensource	Single File, Repository Level	No
Rehearsal[45]	2016	Configuration Management	Opensource	Repository Level	No
SecGuru[19]	2014	Infrastructure Provisioning	Proprietary	Unknown	No
SecureCode[13]	2020	Configuration Management	Proprietary	Repository Level	No
⋆ Semgrep[44]	2020	All	Opensource	Single File	No
SLAC[40]	2021	Configuration Management	Opensource	Repository Level	No
SLIC[39]	2019	Configuration Management	Opensource	Repository Level	No
⋆ SNYK IaC[47]	2023	Infrastructure Provisioning, Image Building	Proprietary	Single File, Repository Level	No
SODALITE[24]	2020	Infrastructure Provisioning	Opensource	Single File, Repository Level	No
Sommelier[8]	2017	Infrastructure Provisioning	Opensource	Single File	No
⋆ SonarQube[48]	2016	Infrastructure Provisioning, Image Building	Hybrid	Repository Level	No
TAMA[17]	2022	Configuration Management	Opensource	Repository Level	No
⋆ tfsec[4]	2019	Infrastructure Provisioning	Opensource	Single File, Repository Level	No

⋆ - Tools that are developed and maintained by private organizations, suggesting a high probability of them being proprietary.

2. Do the examined tools analyze metadata parameters retrieved from the IaC codebase's APIs? (To examine how these tools process metadata.)

Our analysis as shown in Table 1, for static code analysis tools for IaC, reveals that 3 out of 10 examine both single file and repository levels, while 4

out of 10 focus exclusively on repository-level analysis and 3 out of 10 only perform single file analysis. Regarding availability, 7 out of 10 of surveyed tools are open-source solutions, 2 out of 10 are proprietary, and 1 out of 10 offer hybrid licensing models. Of these tools, 3 out of 10 are commercially developed by private organizations, while 7 out of 10 are developed within research environments. This distinction matters because commercially developed tools often prioritize for production ready environments whereas research-based tools typically emphasize experimentation. Despite this diversity in approaches, a critical finding is that none of the examined tools incorporate metadata analysis, significantly limiting their ability to detect supply chain attacks.

The examination of supply chain attacks in the context of both traditional software and IaC, along with the analysis of metadata and the capabilities of smell detection tools, offers insights into the current state of research in this domain, as discussed in Section 2 and summarized in Table 1. However, several gaps remain. The key gaps identified are:

- **Lack of methods for IaC:** Current methods for detecting supply chain attacks mainly target traditional software code and related codebases, as outlined in Section 2. In contrast, research on IaC security is still in its infancy and needs further development [22].
- **Limited functionality of IaC tools:** Current static vulnerability assessment tools for IaC detect various smells but lack the capability to process and analyze metadata at all levels, leaving them vulnerable to supply chain attacks.

3.4 Summary

Cyber attackers are taking advantage of vulnerabilities is supply chain of software ecosystems, as evidenced by events at Colonial Pipeline [33], JBS [34], and PyPI [41]. Prior work has investigated open-source dependencies, package managers, and security methods to detect code that breaches security [18, 15, 35]. IaC, which automates infrastructure provisioning and configuration, may also be exposed to such attacks because it shares properties with software code. Current IaC smell detection tools analyze files and repositories but do not examine metadata, which diminishes their ability to guard against attacks that seek to exploit this. Further investigation is necessary to address this by integrating metadata analysis into IaC vulnerability assessment.

4 METHODOLOGY

To overcome the limitations identified in existing tools, it is essential to broaden the analytical scope beyond individual files and repositories to encompass their interdependencies as well. To achieve this, a methodology was devised that incorporates metadata analysis and dependency traversal to identify susceptible repositories.

The proposed methodology begins with selecting a repository for analysis. Upon selection, the repository's metadata information is retrieved and its dependency data is extracted. Using the extracted dependency information, associated dependency repositories are identified, and their metadata is also retrieved. This process is repeated iteratively, traversing the dependency chain until no additional dependencies remain.

Algorithm 1 Supply-Chain Attack Detection Methodology

Input: Primary repository R
Output: Identified vulnerabilities $\mathcal{V}_{\text{total}}$ mapped to CWE [1] and known CVE [32]
Notation:
    r – repository being analyzed (primary or dependency)
    $\mathcal{M}(r)$ – metadata associated with repository r
    $\mathcal{D}(r)$ – set of direct dependencies of repository r
    $\mathcal{A}(r)$ – analysis performed on repository r
    $\mathcal{V}(r)$ – vulnerabilities identified in repository r
    $\mathcal{T}(R)$ – transitive closure of all dependencies
Dependency Discovery:
Initialize: $\mathcal{T}(R) \leftarrow \emptyset$
repeat
  for each repository $r \in \{R\} \cup \mathcal{T}(R)$ **do**
    Retrieve metadata: $\mathcal{M}(r)$
    Extract direct dependencies: $\mathcal{D}(r)$
    Update transitive dependency set: $\mathcal{T}(R) \leftarrow \mathcal{T}(R) \cup \mathcal{D}(r)$
  end for
until $\mathcal{D}(r) = \emptyset, \forall r \in \mathcal{T}(R)$
Static Vulnerability Analysis:
for each repository $r \in \{R\} \cup \mathcal{T}(R)$ **do**
  Perform static analysis: $\mathcal{A}(r) \rightarrow \mathcal{V}(r)$
end for
Vulnerability Aggregation:
$\mathcal{V}_{\text{total}} \leftarrow \bigcup_{r \in \{R\} \cup \mathcal{T}(R)} \mathcal{V}(r)$
return $\mathcal{V}_{\text{total}}$

Once all the repositories in a dependency chain have been identified, static code analysis is performed on each repository. This analysis evaluates both the source code and metadata across the entire dependency chain. By including dependency repositories and their metadata in the analysis, this approach enables the identification of supply chain attacks, such as poisoned dependencies, that could compromise the security of the overall system. This methodology, as presented in Algorithm 1, provides a systematic approach for addressing the limitations of existing tools within the IaC ecosystems.

For each primary repository R, Algorithm 1 outputs a dependency tree $\mathcal{T}(R)$. For each repository r in $\{R\} \cup \mathcal{T}(R)$, the Algorithm outputs a set of vulnerabilities $\mathcal{V}(r)$. The set of vulnerabilities $\mathcal{V}(r)$ is obtained by performing static code analysis ($\mathcal{A}(r)$), as outlined by Konala et al. [23], focusing on code security attributes such as deprecated keywords, hard-coded credentials, improper file permissions,

logging and monitoring failures, and insecure default configurations to evaluate individual repositories.

4.1 Dataset Overview

Data obtained from Ansible Galaxy [2] served as the primary dataset for this study. Although the presented methodology can be generalizable to other code management platforms that provide metadata access, Ansible Galaxy was specifically chosen due to its structured repository format, containing distinct Ansible roles and collections. This facilitates straightforward extraction and subsequent analysis without the necessity for further filtering. Conversely, platforms such as GitHub encompass repositories with diverse code types and functionalities, thus necessitating additional preprocessing steps to isolate relevant Ansible-specific content. Consequently, Ansible Galaxy was selected to simplify the implementation of the methodology, yet its applicability is not constrained exclusively to this platform. Furthermore, the selection was motivated by the diversity of repositories available in Ansible Galaxy, along with its built-in quality mechanisms.

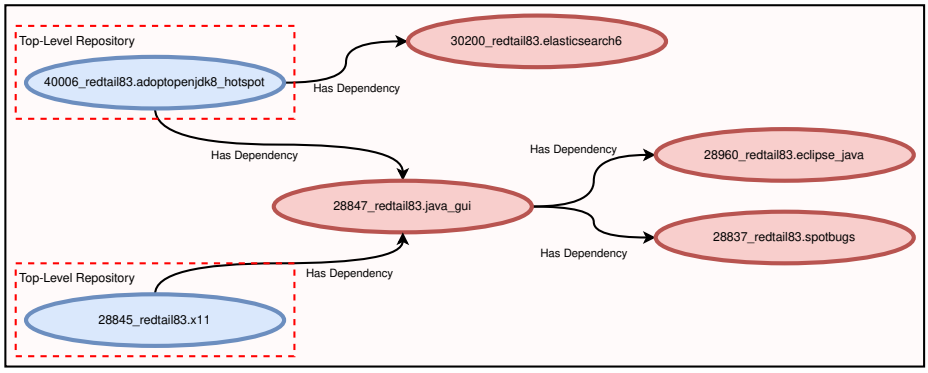

Fig. 2. Example of dependency chains from the dataset

Of the 31,076 repositories that were examined, 27,801 showed no dependencies, while 3,275 had identifiable dependencies. It must be noted that many repositories contain code directly copied from sources. In such cases dependencies and therefore their vulnerabilities cannot be identified in a straightforward manner. Consequently, only repositories from the latter category were considered for detailed analysis. From these, 268 repositories and their associated dependency chains [37] were selected based on metadata completeness. Completeness, in this context, refers to the thorough population of essential metadata fields, including repository descriptions, dependency information, versioning history, author details, and other relevant data, without omissions. Ensuring metadata

completeness is crucial for performing a reliable dependency chain analysis, as required by our proposed methodology.

Figure 2, illustrates an example of dependency chains derived from our dataset. Collectively, the selected 268 dependency chains involved 482 repositories, providing the foundation for subsequent analysis. The primary objective of this analysis was to pinpoint top-level repositories and investigate if they were vulnerable due to security smells not in their code but in the code of their dependencies. Here, top-level repositories are defined as those operating exclusively as downstream consumers without themselves serving as dependencies.

5 FINDINGS AND DISCUSSION

The analysis of the Ansible Galaxy dataset, following Algorithm 1, revealed security smells within IaC dependency chains. A total of 45 dependency chains, involving 40 repositories, were found to contain vulnerabilities that could enable supply chain attacks both currently and in the future. Subsequently, the study focused on a detailed examination of these vulnerable dependencies, specifically aiming to:

- Map identified security smells to the CWE framework.
- Map identified security smells with known CVEs.

The following subsections provide an discussion of these analyses.

5.1 Security Smells Mapping To CWE

Our analysis revealed that security smells were primarily linked to the use of deprecated keywords and modules. The distribution of these security smells indicated that out of the 40 repositories that were studied, 95% of the repositories contained deprecated components, which can be associated with the Use of Obsolete Function security weakness (CWE-477) [28]. This reliance on outdated repositories utilizing unmaintained versions of Ansible suggests that developers have not migrated their scripts to the latest versions.

Furthermore, additional security concerns were identified, with hard-coded database credentials present in 1 repository representing 2.5% of the repositories under consideration, corresponding to CWE-798 (Use of Hard-Coded Credentials) [29]. Similarly, improper file permissions were observed in 2.5% of the repositories, aligning with CWE-280 (Improper Handling of Insufficient Permissions or Privileges) [27]. These findings indicate potential security risks within IaC scripts, emphasizing the need for improved security practices in dependency management and script maintenance.

Temporal Analysis of Repository Releases Fig. 3 presents an analysis of the latest release dates for repositories within the dependency chain, providing further support for observations related to the Use of Obsolete Function CWE-477[28]. The histogram analysis indicates a mean release date of November 2018

and a median release date of May 2018, highlighting a significant gap of approximately seven years at the time of our study (January 2025). This prolonged period without updates increases the risks associated with relying on outdated dependencies. Furthermore, to substantiate this claim, an examination of Ansible releases [11] reveals that the number of deprecated keywords increases with each new version, further emphasizing the challenges posed by outdated infrastructure.

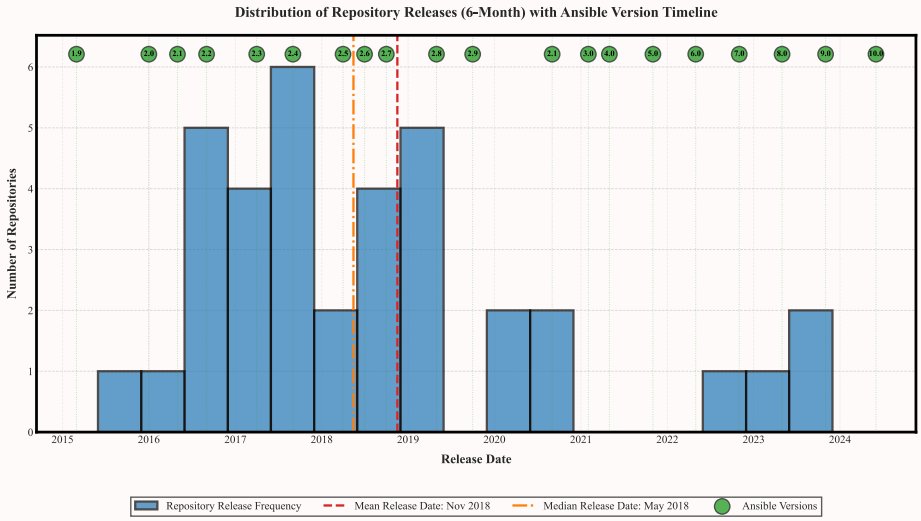

Fig. 3. Timeline Histogram of Repository Releases Over 6 Months with Ansible Versions

5.2 Author Activity Analysis

In addition to the code-based analysis, repositories and their authors were examined based on contribution records, specifically the last commit information derived from repository metadata. Repositories were classified into 5 categories based on whether contributions occurred within a six-month timeframe, corresponding to the typical update cycle for Ansible. Authors were categorized based on their involvement in single or multiple repositories and further distinguished as either individual contributors or organizational entities. Moreover, the analysis provides a count of repositories that are potentially susceptible to present or future supply chain attacks for each category if vulnerable dependency repositories are utilized, as outlined in Table 2.

While many vulnerable repositories within the dependency chain are no longer actively maintained, their authors often continue contributing to other

Table 2. Repository & Author Contribution Analysis

Category	Repository Activity	Author Activity	Author Status	Vulnerable Repositories	Cumulative Downloads
C1	inactive	active	single	21	941,327
C2	inactive	active	org	9	34,370
C3	inactive	inactive	single	5	227
C4	active	active	single	3	7,100,309
C5	inactive	inactive	org	2	72

projects (e.g., Categories C1, C2). Given their continued activity, it is conceivable that they may recognize the associated risks and take proactive measures to mitigate them. However, security risks become significant when a repository remains unmaintained for an extended period, particularly when its author is inactive (Category C3). This risk is exacerbated when an unmaintained repository is linked to an inactive author from an organizational entity (Category C5), increasing the likelihood of RepoJacking supply chain attacks [21] involving IaC scripts. Notably, our analysis identified a repository containing hard-coded database credentials, classified as CWE-798 (Use of Hard-Coded Credentials). Such vulnerabilities could be exploited by malicious actors to carry out RepoJacking attacks, compromising the security of dependent systems.

To gain deeper insights into the extent of vulnerabilities in Ansible Galaxy, Table 3 aggregates the data from the detailed classification by summarizing the total number of affected repositories across the defined categories from Table 2. With 117 repositories identified as potentially vulnerable to compromise, this table provides an overall view of the impact, highlighting the distribution of vulnerabilities within the ecosystem. Categories C1 and C4 account for the majority of the affected repositories, suggesting that even projects with active contributions can be at risk due to vulnerable dependencies. Conversely, the counts in Categories C3 and C5 are particularly concerning given their association with inactive repositories and authors, which may impede the timely application of security patches. Together, these findings underscore the critical need for continuous monitoring and proactive mitigation strategies to address vulnerabilities in IaC environments and safeguard against supply chain attacks.

5.3 Security Smells Mapping To Known CVE

Building on the repository and author contribution analysis, the mapping of publicly known CVEs [32] associated with Ansible revealed three affected repositories. These repositories represent a range of activity statuses and author types, highlighting diverse security risks within the IaC ecosystem.

The first repository, maintained by an active individual author, is a widely used project with a high download count (4,389,601), reflecting extensive adoption within the community. Despite its active maintenance, it is associated with

Table 3. Number of Affected Repositories

Category	Total Affected Repositories
C1	47
C2	19
C3	6
C4	43
C5	2

CVE-2024-8775 [31], which has a CVSS v3.1 base score of 5.5, indicating a moderate security risk. This suggests that even actively maintained repositories may harbor vulnerabilities that can affect downstream projects.

The second repository, although maintained by an active individual author, has been inactive for some time and has a relatively low download count (1,567). It is also affected by CVE-2024-8775, illustrating how vulnerabilities can persist across different repositories and dependency chains, regardless of their popularity or level of maintenance. The inactivity further increases the security risk, as the absence of updates reduces the likelihood of timely patching.

The third repository, managed by an organizational author, is inactive and has a minimal download count (41). However, it is associated with CVE-2017-7550 [30], a critical vulnerability with a CVSS v3.1 base score of 9.8. The presence of such a high-severity CVE in an unmaintained repository raises significant concerns about potential exploitation, especially in the context of supply chain attacks.

This analysis aligns with prior findings that inactive repositories managed by organizations represent a notable security risk. These repositories can serve as attack vectors, allowing vulnerabilities in dependencies to facilitate lateral movement within systems. The identification of CVEs in both actively and inactively maintained repositories emphasizes the need for continuous monitoring and proactive security measures to mitigate threats in IaC environments.

5.4 Limitations & Future Work

Current practices for detecting vulnerabilities in dependency chains rely predominantly on static analysis methods that examine code without executing it. Such methods may not capture vulnerabilities that manifest when code segments from separate repositories interact during execution. While static analysis may overlook these dynamic interactions, their combination during runtime could lead to behavior that compromises system security.

Additionally, we acknowledge a limitation in the temporal analysis discussed in subsection 5.1. This analysis, which involves calculating mean and median release dates, is affected by the dominance of older releases, resulting in a

graph that disproportionately reflects repositories with earlier dates. Establishing whether recent repositories exhibit improved security standards would require additional supporting data. However, determining this falls outside the scope of the current study, which primarily focuses on metadata and supply-chain attack detection.

Future work should focus on developing and integrating dynamic analysis methods that assess code during execution, especially for infrastructure as code. Expanding the investigation to include alternative approaches beyond static analysis could help capture vulnerabilities that are otherwise hidden, thereby reducing the chance of supply chain attacks.

6 CONCLUSION

This study analyzed the vulnerability of IaC to supply chain attacks, highlighting limitations in existing static smell detection tools, which lack the capability to incorporate metadata analysis. By applying a methodology that integrates metadata and dependency repository analysis, security smells were identified within Ansible dependency chains, including reliance on deprecated dependencies (CWE-477 [28]), hard-coded credentials (CWE-798 [29]), and improper file permissions (CWE-280 [27]). Additionally, the analysis identified vulnerabilities associated with CVE-2024-8775 [31] and CVE-2017-7550 [30] in three repositories.

The findings indicate the necessity for developing tools that integrate metadata analysis with security assessments to address supply chain risks in IaC environments. Future research should extend the methodology by incorporating dynamic analysis and evaluating security smells across multiple IaC platforms to establish security guardrails within IaC ecosystems against supply chain attacks.

ACKNOWLEDGEMENT

We appreciate the valuable feedback provided by the reviewers. The authors would also like to acknowledge funding support from the New Zealand Ministry of Business, Innovation, and Employment (MBIE) for project UOWX1911, “Artificial Intelligence for Human-Centric Security.

References

1. Common weakness enumeration, https://cwe.mitre.org/index.html
2. Ansible Community: Ansible Galaxy (2023), https://galaxy.ansible.com/
3. Aqua: Cloudsploit, cloudsploit.com
4. aquasecurity: tfsec, https://github.com/aquasecurity/tfsec
5. Borovits, N., Kumara, I., Krishnan, P., Palma, S.D., Di Nucci, D., Palomba, F., Tamburri, D.A., van den Heuvel, W.J.: Deepiac: Deep learning-based linguistic anti-pattern detection in iac. In: Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation. p. 712. MaLTeSQuE 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3416505.3423564, https://doi.org/10.1145/3416505.3423564
6. BridgeCrew: Checkov, https://www.checkov.io/
7. Brogi, A., Canciani, A., Soldani, J.: Modelling and analysing cloud application management. In: Dustdar, S., Leymann, F., Villari, M. (eds.) Service Oriented and Cloud Computing. pp. 19–33. Springer International Publishing, Cham (2015)
8. Brogi, A., Di Tommaso, A., Soldani, J.: Sommelier: A tool for validating tosca application topologies. In: Pires, L.F., Hammoudi, S., Selic, B. (eds.) Model-Driven Engineering and Software Development. pp. 1–22. Springer International Publishing, Cham (2018)
9. Checkmarx: Kics, https://www.kics.io/
10. Chef: Cookstyle (2023), https://github.com/chef/cookstyle
11. Community, A.: Releases and Maintenance. Red Hat, Inc. (2025), https://docs.ansible.com/ansible/latest/reference_appendices/release_and_maintenance.html
12. Cunin, A.: 'yamllint' documentation: A linter for YAML files (2023), https://yamllint.readthedocs.io
13. Dai, T., Karve, A., Koper, G., Zeng, S.: Automatically detecting risky scripts in infrastructure code. In: Proceedings of the 11th ACM Symposium on Cloud Computing. p. 358371. SoCC '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3419111.3421303
14. Dalla Palma, S., Di Nucci, D., Palomba, F., Tamburri, D.A.: Within-project defect prediction of infrastructure-as-code using product and process metrics. IEEE Transactions on Software Engineering **48**(6), 2086–2104 (2022). https://doi.org/10.1109/TSE.2021.3051492
15. Duan, R., Alrawi, O., Kasturi, R.P., Elder, R., Saltaformaggio, B., Lee, W.: Towards measuring supply chain attacks on package managers for interpreted languages. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) (2021). https://doi.org/10.14722/ndss.2021.23055, https://www.ndss-symposium.org/wp-content/uploads/ndss2021_1B-1_23055_paper.pdf
16. GitHub Inc.: The state of the octoverse. https://octoverse.github.com/2019/ (2020), accessed: 2025-01-07
17. Hassan, M.M., Rahman, A.: As code testing: Characterizing test quality in open source ansible development. In: 2022 IEEE Conference on Software Testing, Verification and Validation (ICST). pp. 208–219 (2022). https://doi.org/10.1109/ICST53961.2022.00031
18. Hossain Faruk, M.J., Tasnim, M., Shahriar, H., Valero, M., Rahman, A., Wu, F.: Investigating novel approaches to defend software supply chain attacks. In: 2022 IEEE International Symposium on Software Reliability Engi-

neering Workshops (ISSREW). pp. 283–288 (2022). https://doi.org/10.1109/ISSREW55968.2022.00081
19. Jayaraman, K., Bjørner, N., Outhred, G., Kaufman, C.: Automated analysis and debugging of network connectivity policies. Microsoft Research pp. 1–11 (2014)
20. Kasturi, R.P., Fuller, J., Sun, Y., Chabklo, O., Rodriguez, A., Park, J., Saltaformaggio, B.: Mistrust plugins you must: A Large-Scale study of malicious plugins in WordPress marketplaces. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 161–178. USENIX Association, Boston, MA (Aug 2022), https://www.usenix.org/conference/usenixsecurity22/presentation/kasturi
21. Kim, M., Jung, W., Lee, S., Kwon, T., Kim, E.T.: Code repository vulnerability focusing on repojacking. In: 2023 14th International Conference on Information and Communication Technology Convergence (ICTC). pp. 1880–1884 (2023). https://doi.org/10.1109/ICTC58733.2023.10392354
22. Konala, P.R.R., Kumar, V., Bainbridge, D.: Sok: Static configuration analysis in infrastructure as code scripts. In: IEEE International Conference on Cyber Security and Resilience, CSR 2023, Venice, Italy, July 31 - Aug. 2, 2023. pp. 281–288. IEEE (2023). https://doi.org/10.1109/CSR57506.2023.10224925, https://doi.org/10.1109/CSR57506.2023.10224925
23. Konala, P.R.R., Kumar, V., Bainbridge, D., Haseeb, J.: A framework for measuring the quality of infrastructure-as-code scripts (2025), https://arxiv.org/abs/2502.03127, submitted to arXiv
24. Kumara, I., Vasileiou, Z., Meditskos, G., Tamburri, D., Heuvel, W.J., Karakostas, A., Vrochidis, S., Kompatsiaris, I.: Towards semantic detection of smells in cloud infrastructure code. pp. 63–67 (06 2020). https://doi.org/10.1145/3405962.3405979
25. Lepiller, J., Piskac, R., Schäf, M., Santolucito, M.: Analyzing infrastructure as code to prevent intra-update sniping vulnerabilities. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 105–123. Springer International Publishing, Cham (2021)
26. Meijer, B., Hochstein, L., Moser, R.: Ansible: Up and Running. O'Reilly Media, Sebastopol, CA, 3rd edition edn. (2022)
27. Mitre: About CWE-280: Improper Handling of Insufficient Permissions or Privileges (2018), https://cwe.mitre.org/data/definitions/280.html
28. Mitre: About CWE-477: Use of Obsolete Function (2018), https://cwe.mitre.org/data/definitions/477.html
29. Mitre: About cwe-798: Use of hard-coded credentials (2018), https://cwe.mitre.org/data/definitions/798.html
30. MITRE Corporation: CVE-2017-7550: Ansible Jenkins Plugin Module Exposes Passwords in Remote Host Logs (2017), https://nvd.nist.gov/vuln/detail/CVE-2017-7550
31. MITRE Corporation: CVE-2024-8775: Exposure of Sensitive Information in Ansible Vault Files Due to Improper Logging (2024), https://nvd.nist.gov/vuln/detail/CVE-2024-8775
32. MITRE Corporation: Common Vulnerabilities and Exposures (CVE) (2025), https://cve.mitre.org/
33. Nahta, P.: Securing the digital supply chain: Challenges, innovations, and best practices in cybersecurity (2025). https://doi.org/10.4018/979-8-3693-8357-5.ch008
34. Ofori-Yeboah, A., Addo-Quaye, R., Oseni, W., Amorin, P., Agangmikre, C.: Cyber supply chain security: A cost benefit analysis using net present value. In: 2021 International Conference on Cyber Security and Internet of Things (ICSIoT). pp. 49–54 (2021). https://doi.org/10.1109/ICSIoT55070.2021.00018

35. Ohm, M., Stuke, C.: Sok: Practical detection of software supply chain attacks. In: Proceedings of the 18th International Conference on Availability, Reliability and Security. ARES '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3600160.3600162, https://doi.org/10.1145/3600160.3600162
36. Open Policy Agent: Open Policy Agent (OPA) (2025), https://www.openpolicyagent.org/
37. Palo Alto Networks: What is dependency chain abuse? (2023), https://www.paloaltonetworks.com/cyberpedia/dependency-chain-abuse-cicd-sec3
38. Rahman, A., Farhana, E., Parnin, C., Williams, L.: 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 (2020). https://doi.org/10.1145/3377811.3380409
39. Rahman, A., Parnin, C., Williams, L.: The seven sins: Security smells in infrastructure as code scripts. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). pp. 164–175 (2019). https://doi.org/10.1109/ICSE.2019.00033
40. Rahman, A., Rahman, M.R., Parnin, C., Williams, L.: Security smells in ansible and chef scripts: A replication study. ACM Trans. Softw. Eng. Methodol. **30**(1) (1 2021). https://doi.org/10.1145/3408897, https://doi.org/10.1145/3408897
41. Research, J.S.: Revival hijack: Pypi hijack technique exploited, 22k+ packages at risk. https://jfrog.com/blog/revival-hijack-pypi-hijack-technique-exploited-22k-packages-at-risk/ (2024)
42. Saavedra, N., Ferreira, J.F.: Glitch: Automated polyglot security smell detection in infrastructure as code (2022). https://doi.org/10.48550/ARXIV.2205.14371, https://arxiv.org/abs/2205.14371
43. Schwarz, J., Steffens, A., Lichter, H.: Code smells in infrastructure as code. In: 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC). pp. 220–228 (2018). https://doi.org/10.1109/QUATIC.2018.00040
44. Semgrep: Semgrep, semgrep.dev
45. Shambaugh, R., Weiss, A., Guha, A.: Rehearsal: A configuration verification tool for puppet. In: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. p. 416430. PLDI '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2908080.2908083, https://doi.org/10.1145/2908080.2908083
46. Sharma, T., Fragkoulis, M., Spinellis, D.: Does your configuration code smell? In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). pp. 189–200 (2016)
47. SNYK: Snykiac, https://snyk.io/product/infrastructure-as-code-security/
48. Sonar: Sonarqube, https://www.sonarsource.com/products/sonarqube/
49. Thames, W., contributors: ansible-lint - checks playbooks for practices and behavior that could potentially be improved (2023), https://ansible-lint.readthedocs.io
50. Wang, R.: Infrastructure as Code, Patterns and Practices: With examples in Python and Terraform. ITpro collection, Manning (2022)
51. Zhu, H.: Postmortem for malicious packages published on july 12th, 2018 (2018), https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes/