Tracking Security Smell Diffusion Patterns in Ansible Playbooks Using Metadata

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. Infrastructure as Code (IaC) platforms lack mechanisms for detecting security smell diffusion, a challenge stemming from the absence of repository relationships. We present a similarity-based methodology combining content and structure metrics to identify repository clones. Validated against Ansible Galaxy repositories that have GitHub fork data, our approach achieved 99.6%–99.8% accuracy to detect forks. Analysis of Ansible Galaxy repositories across three popular technologies revealed 38.4%–54.1% share code overlap, creating vulnerability propagation pathways. Security analysis identified CWE-477 and CWE-546 as most prevalent, with CVE-2017-7550 (CVSS 9.8 - Critical) propagating from a popular repository version with 2.7 million downloads. Fork metadata absence causes users to download repositories with induced security smells at 100× higher rates than platforms with visible fork relationships. A survey of 24 IaC tools confirmed none provide cross-repository comparison capabilities, demonstrating a gap in repository relationship tracking within the IaC supply chain. Our work addresses this gap by providing a systematic approach to detect clones and track security diffusion in environments lacking fork metadata.

Keywords: Infrastructure as Code, Ansible, Vulnerability analysis, Security Smells, Repository analysis, Metadata, Clone, Fork Chain.

1 Introduction

Infrastructure as Code (IaC) has transformed how organizations manage computing infrastructure, enabling automated deployments through declarative, version controlled scripts. While IaC adoption accelerates across enterprises, the widespread practice of code reuse and repository cloning introduces security risks similar to those in traditional software development, but without established mechanisms for tracking code relationships and security smells. These risks manifest as security smells: code patterns, configurations, or practices in IaC scripts that indicate potential security weaknesses such as hard coded credentials, weak cryptographic keys, or misconfigured access controls. When repositories are cloned, security smells spread across codebases, a phenomenon we term security smell diffusion [23], which is the process by which security anti-patterns propagate from source repositories to derivative codebases through code

reuse. Without proper tracking mechanisms, this diffusion remains undetected, preventing security teams from identifying vulnerable code origins or tracing propagation paths. This paper addresses the critical gap in detecting cloned IaC repositories that facilitate untracked security smell diffusion.

1.1 Motivation

IaC repositories face unique security challenges compared with traditional software. While platforms like Ansible Galaxy[1] facilitate code sharing and reuse, they lack mechanisms to track repository relationships or detect derived repositories. In this context, we distinguish between repository forks[2] and derived repositories also known as code clones[3]. A fork represents an officially tracked derivative of a repository, with metadata linking it to its source. In contrast, a clone is any duplicated code, whether through copy-paste, unauthorized replication, or untracked derivation that lacks formal relationship metadata. While platforms like GitHub[4] track forks through explicit metadata, IaC platforms like Ansible Galaxy do not maintain such relationships, making all duplicated code effectively 'clones' from a detection perspective.

Our preliminary analysis of Ansible[5] repositories revealed hints of code duplication that sometimes included reused SSH keys and embedded credentials across multiple projects. Without fork metadata or clone detection capabilities, security smells in one repository can silently diffuse to multiple of derivative projects. The impact of compromised IaC scripts extends beyond traditional software weaknesses. Since IaC scripts directly control production infrastructure and often contain sensitive credentials, a single compromised playbook can grant attackers access to entire environments.

1.2 The Clone Detection Problem

Recent security incidents highlight the severity of supply chain attacks through repository cloning. In 2022, GitHub discovered 35,000 malicious repository clones containing backdoors [31], escalating to over 100,000 infected repositories by 2023 [5]. Microsoft reported that these attacks affected over one million devices by 2024 [18]. These events demonstrate the cascading nature of supply chain vulnerabilities: each infected repository serves as a new attack vector, multiplying the threat as downstream users unknowingly incorporate compromised code into their production systems. Additional incidents include the 'Stargazer Goblin' operation using 3,000 fake accounts to distribute malware [34], and various campaigns targeting developers through cloned repositories [36,20,19].

[1] Ansible Galaxy, https://galaxy.ansible.com/

[2] GitHub: About forks, https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/working-with-forks/about-forks

[3] GitHub: Cloning A Repository, https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository?tool=cli

[4] GitHub, https://github.com

[5] Ansible, https://www.ansible.com

While GitHub provides fork tracking to identify repository relationships, as demonstrated in our platform analysis (Section 3.1), current IaC repositories such as Ansible Galaxy do not support cross-repository comparison. This limitation prevents practitioners from determining security smell sources or measuring their spread across projects.

As a result of the absence of repository clone detection in IaC ecosystems, there are several factors that software developers need to contend with. First, developers cannot identify when they are using cloned repositories that may contain security smells. Second, security teams cannot track how security smells diffuse across infrastructure codebases, particularly when IaC-specific risks like unpinned module references [24] or hidden credential exfiltration [25] are involved. Third, maintainers cannot notify affected users when security smells are discovered in widely-cloned repositories.

1.3 Research Contributions

This paper presents a systematic approach to detect repository clones and track security smell diffusion patterns in the Ansible Galaxy ecosystem that lack fork metadata. Our contributions include:

- **IaC Platform and Tool Gap Analysis:** We surveyed 24 IaC security tools and 10 platforms to establish the current state of cross-repository analysis capabilities and metadata availability in the IaC ecosystem.
- **Detection and Diffusion Analysis:** We developed similarity-based algorithms to identify repository clones without fork metadata, then applied these to track security smell propagation patterns, establishing a classification system (propagated, induced, fixed) that reveals how security smells spread through code reuse in IaC ecosystems.
- **Empirical Analysis:** We conducted a study of Ansible Galaxy repositories across three technologies, analyzing both code reuse patterns and the users impacts of metadata absence on repository selection.

The remainder of this paper is structured as follows. Section 2 provides background on clone detection in traditional software and IaC. Section 3 surveys IaC platforms and existing smell detection tools, identifying gaps in cross-repository analysis. Section 4 presents our similarity-based methodology for detecting clones and tracking security smell diffusions. Section 5 reports empirical findings on repository cloning patterns and vulnerability propagation in Ansible Galaxy, along with implications and future directions. Section 6 concludes the paper.

2 Background

This section provides essential context for understanding code duplication issues in software engineering, particularly addressing the nature of repository and code clones, their detection methods, and their implications in infrastructure management contexts.

2.1 Clone Detection Techniques in Traditional Software

Clones [30] are duplicated code fragments sharing similarity through exact replication, syntactic variations, or functional equivalence, typically arising from copy-pasting, design patterns, API constraints, or developer habits. Clone detection is essential since duplicates affect maintenance, facilitate bug propagation, and influence system evolution. Over three decades, detection methods have evolved into five primary categories.

Early approaches focused on textual and lexical analysis. Johnson [7,8] pioneered text-based methods comparing code as line sequences through hashing, later enhanced by tools like *sif* [14] and *Duploc* [4] with visual comparison techniques. Baker [1] introduced token-based detection using parameterized suffix trees, advanced by *CCFinder* [9] and *CP-Miner* [13] for scalability. Mayrand *et al.* [16] developed metric-based approaches comparing numerical code representations, prioritizing speed over precision.

More sophisticated techniques emerged with structural analysis. *CloneDR* [2] introduced AST-based methods using sub-tree hashing to detect clones despite statement reordering, enhanced by *DECKARD* [6] with locality-sensitive hashing. Hybrid approaches like *NiCad* [28] combine normalization with sequence matching to balance scalability and accuracy. Roy *et al.* [29] synthesized these techniques, defining clone types and providing benchmark evaluations for all the above discussed tools that offer insights for addressing similar challenges in IaC environments.

2.2 Clone Detection in Infrastructure as Code

In containerized infrastructure, Tsuru *et al.* [33] analyzed 5,000 Dockerfiles using Type-2 clone detection that separated Docker syntax from shell scripts, achieving 95% precision through file-by-file analysis. Their domain-specific analyzer confirmed that Dockerfiles commonly exhibit clone patterns from template copying and configuration reuse. Similarly, Oumaziz *et al.* [22] conducted an empirical investigation revealing repeated sequences in Dockerfiles and identified three reuse management strategies: index-based clone detection, template-based generation, and internal tool development. These cloning patterns pose security risks in IaC environments, as Rahman *et al.* [3] demonstrated with insecure snippets propagating to 35 different infrastructure resources, where 99% of insecure patterns replicated across individual Puppet[6] scripts. Li *et al.* [13] further confirmed that vulnerabilities occur more frequently in cloned segments.

These studies demonstrate that IaC scripts built under code reuse policies increase attack surfaces and complicate remediation by replicating insecure configurations across multiple artifacts. While foundational patterns of duplication in container build files have been established, existing IaC clone detection research focuses on individual files rather than repositories as a whole. This file-level approach misses repository-wide cloning patterns and requires new definitions

[6] Puppet, https://puppet.com/

for repository clones rather than relying solely on Roy *et al.* [29] code clone types which are meant for traditional software. IaC repositories must be analyzed as complete units because they contain various resource and configuration file types which includes README files, Jinja templates, YAML configurations, and shell scripts that collectively define infrastructure. Security smell diffusion occurs across these interconnected files within entire codebases. Therefore, software configuration management requires specialized detection approaches that analyze complete repositories rather than isolated scripts.

2.3 About Infrastructure as Code and Metadata

Although most existing research has focused on software code and related security concerns, IaC scripts may also be susceptible to code cloning due to their structural and functional similarities with traditional software. This subsection outlines IaC fundamentals and its metadata structure. The IaC technology stack comprises three categories [35]: *infrastructure provisioning* (automates hardware resource allocation), *configuration management*(system setups and software management), and *image building* (generates standardized machine/container images).

To support these categories, IaC platforms organize scripts within structured repositories using platform-specific terminology: Ansible Galaxy uses 'roles', Chef Supermarket[7] uses 'cookbooks', while Puppet Forge[8] and Terraform Registry[9] use 'modules'. In Ansible Galaxy, each role [17] follows a standard format[10] with directories for handlers, vars, meta, templates, and files. These repositories contain rich metadata, which is typically hidden from users and the platforms retrieve before downloading code content, enabling analysis of repository relationships and characteristics. **Listing I** presents a sample of this metadata from Ansible Galaxy's API:

Listing I: Metadata of a Sample Ansible Galaxy Repository

```
1: { "id": 56789,"created": "2024-02-05","username": "devops_admin",
2:  "github_repo": "ansible-role-nginx",
3:  "github_user": "devops_admin",
4:  "github_branch": "master",
5:  "name": "nginx","summary_fields": {
6:    "dependencies": [{"id": 10,"name": "sec.hardening"},"..."],
7:    "namespace": {"id": 3050,"name": "devops_admin"},
8:    "provider_namespace": {"repository": {"name": "nginx"},
9:    "tags": ["webserver","nginx", "..."],
10:   "versions": [{"name":"1.1"}, "..."]}},"downloads": 342560}
```

[7] Chef Supermarket, https://supermarket.chef.io/
[8] Puppet Forge, https://forge.puppet.com/
[9] Terraform Registry, https://registry.terraform.io/
[10] Ansible: Best practices – directory layout, https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html#directory-layout

Metadata from Ansible Galaxy contains structured data including dependencies, version identifiers, namespace associations, download statistics, authorship, repository sources, and update histories. This information enables compatibility verification and maintenance while download metrics indicate adoption patterns. However, inadequate management of these artifacts may introduce security risks. It is possible for Ansible repositories to link directly to GitHub through metadata parameters '*github_ repo*', '*github_ branch*', and '*github_ user*' (as shown in lines **2-4** of Listing I). These parameters enable access to extended GitHub metadata including fork-chain details, commit histories, and version control, facilitating analysis of security smells diffusion across IaC ecosystems. Very few repositories, however, provide such linkage. With limited GitHub integration across Ansible repositories, practitioners lack fork metadata needed to trace code origins and security smells. This metadata gap raises a critical question: can existing IaC platforms and security analysis tools detect repository clones through other means? The following section examines current tool capabilities for identifying repository relationships.

3 IaC Codebases and State-of-the-Art Smell Detection Tools

As IaC adoption increases across organizations, practitioners rely on shared modules and configurations to accelerate deployment. This practice raises critical questions: Where do developers source IaC components? How do security smells diffuse through code reuse? What tools can detect these patterns?

To address these questions, we investigated the IaC landscape through two lenses. First, we mapped platforms, examining API accessibility and metadata availability. Second, we evaluated existing analysis tools to determine their capabilities for tracking code relationships and detecting cross-repository security smells. Our investigation revealed that while platforms provide rich metadata through APIs, current tools lack the capabilities to leverage this information for repository clone detection and security smell tracking.

3.1 Survey on IaC Codebases

We surveyed the IaC technology landscape to identify platforms with centralized code repositories suitable for large-scale analysis. Our investigation examined 10 major IaC platforms across three categories: infrastructure provisioning, configuration management, and image building. Among these platforms, 5 maintain dedicated codebases with searchable repositories for sharing reusable components.

While the remaining 5 (CloudFormation, Azure Resource Manager Templates, Google Cloud Deployment Manager, SaltStack, and Packer) rely on GitHub for code distribution without platform-specific codebases. Table 1 presents our findings, showing that all 5 platforms with dedicated repositories provide APIs

for programmatic access, exposing 15–25 metadata attributes per artifact including version information, dependencies, download statistics, and community trust relationships. Four platforms (Terraform Registry, Ansible Galaxy, Puppet Forge, and Chef Supermarket) integrate with GitHub as an external source, while Docker Hub operates without external integration. These findings demonstrate that the combination of structured metadata, API accessibility, and external GitHub integration enables analysis of repository clones across thousands of IaC artifacts, providing the infrastructure necessary for detecting duplication patterns and security vulnerabilities through code reuse which is the foundation for our security smell diffusion analysis.

Table 1. Survey of IaC Codebases

IaC Category	IaC Platform	IaC Codebase	API Access	Metadata Parameter	Fork Metadata	External Integration
Infrastructure Provisioning	Terraform	Terraform Registry	✓	19	×	✓
Configuration Management	Ansible	Ansible Galaxy	✓	20	×	✓
	Puppet	Puppet Forge	✓	25	×	✓
	Chef	Chef Supermarket	✓	15	×	✓
Image Building	Docker	Docker Hub	✓	25	×	×

Legend: × - Not Available, ✓ - Available

3.2 State-of-the-Art IaC Smell Detection Tools

Konala *et al.* [26] surveyed 24 state-of-the-art IaC smell detection tools spanning open-source, proprietary, academic research, and commercial solutions across three IaC categories: infrastructure provisioning, configuration management, and image building. These tools, developed between 2011 and 2023, were examined for their capabilities in detecting code and security smells in IaC technology stacks, revealing two critical limitations across all surveyed tools. First, as established by Konala *et al.* [11], none provide metadata analysis functionality. Second, our study found that despite accepting input as either single files or complete repositories, none of the surveyed 24 state-of-the-art tools support file-to-file or repository-to-repository comparison functionality for IaC scripts. This critical gap prevents analysis of security smell diffusion from original to cloned repositories, making it hard to track how vulnerabilities propagate through code reuse.

3.3 Summary

The IaC ecosystem survey reveals a disconnect between platform capabilities and analytical tools. Among 10 surveyed platforms, 5 maintain repositories with API access and 15-25 metadata attributes per artifact which are Terraform Registry, Ansible Galaxy, Puppet Forge, Chef Supermarket, and Docker Hub with 4 of them integrating GitHub for cross-platform analysis. Yet none of the 24 IaC analysis tools examined (spanning open-source, proprietary, and academic solutions from 2011-2023) support metadata analysis or cross-repository comparison. Previous research analyzes files individually rather than repositories

as complete units. IaC repositories contain interconnected components such as README files, configuration templates, variable definitions, and metadata files that collectively define infrastructure. In Ansible roles, tasks depend on variables, handlers respond to notifications, and templates reference both. Analyzing single files in isolation misses these interdependencies and the security implications of their interactions. This gap presents an opportunity to develop techniques that leverage metadata for repository clone detection and security smell tracking.

4 Methodology

In this section, we present our methodology in the form of two algorithms. Our methodology targets repository clones which are entire repositories duplicated with minimal modifications such as whitespace changes, README updates, comment variations, or formatting differences. For such minimally modified clones, simple content and structure metrics effectively capture similarities, as the changes are superficial rather than semantic. We focus on this approach because IaC codebases lack clone detection tools as established in subsection 3.1. Our first algorithm calculates similarity between repository pairs to identify cloning relationships, while the second tracks security smell diffusion using these similarity scores. More sophisticated detection involving semantic analysis for heavily modified repositories remains beyond our current scope.

4.1 Similarity Calculation

Algorithm 1 analyzes repository collections to determine relationships between them. Given a set of repositories ($\mathcal{R}$), it compares every unique pair ($\mathcal{P}$) once, avoiding redundant calculations. Each pair (r_i, r_j) is evaluated through three metrics ($S_{content}$, $S_{structure}$, $S_{overall}$) capturing different relationship aspects. The algorithm's time complexity is $O(n^2)$, where n is the number of repositories in the collection $\mathcal{R}$, as it must process $\binom{n}{2} = \frac{n(n-1)}{2}$ unique pairs. The space complexity is also $O(n^2)$ due to storing all pairwise similarity scores.

In IaC contexts, both content and structural analysis provide comprehensive similarity assessment, though structure plays a more vital role than in traditional software development. While traditional software allows flexible file organization, IaC frameworks enforce specific directory structures that directly map to infrastructure components. For instance, Ansible roles require precise placement of tasks, handlers, and variables in designated directories. Content similarity ($S_{content}$) reveals the degree of code sharing, identifying repositories with similar infrastructure configurations. Structure similarity ($S_{structure}$) becomes particularly significant as directory hierarchies in IaC represent deployment architectures and operational workflows, not merely code organization preferences. Combined, these metrics distinguish repositories sharing superficial structures from those containing substantially similar infrastructure definitions, with structural patterns often indicating whether repositories target the same deployment scenarios. Below we discuss how the different similarity scores are calculated.

Algorithm 1: Similarity Calculation

Input: Set of repositories $\mathcal{R} = \{r_1, r_2, ..., r_n\}$
Output: Similarity values $S_{ij} = \{S_{overall}\}$ for each pair (r_i, r_j)
Notation:
    r_i, r_j – individual repositories, $\mathcal{F}(r)$ – set of files in repository r
    $\mathcal{L}(r)$ – total lines in repository r, $\Delta(r_i, r_j)$ – git-diff statistics
Pairwise Comparison:
    Generate pairs: $\mathcal{P} \leftarrow \{(r_i, r_j) : r_i, r_j \in \mathcal{R}, i < j\}$

Similarity Computation:
for all $(r_i, r_j) \in \mathcal{P}$ **do**
    Compute similarity: $S_{ij} \leftarrow \text{ComputeSimilarity}(r_i, r_j)$
end for

ComputeSimilarity(r_i, r_j)**:**
Step 1 Content Similarity:
    Get files: $\mathcal{F}_i \leftarrow \text{GetFiles}(r_i)$, $\mathcal{F}_j \leftarrow \text{GetFiles}(r_j)$
    Count lines: $\mathcal{L}_i \leftarrow \sum_{f \in \mathcal{F}_i} \text{CountLines}(f)$,
                 $\mathcal{L}_j \leftarrow \sum_{f \in \mathcal{F}_j} \text{CountLines}(f)$
    Total lines: $\mathcal{L}_{total} \leftarrow \mathcal{L}_i + \mathcal{L}_j$
    Get diff: $\Delta(r_i, r_j) \leftarrow \text{GitDiff}(r_i, r_j)$
    Changed lines: $\mathcal{L}_{changed} \leftarrow \sum_{d \in \Delta}(d.additions + d.deletions)$
    $S_{content} \leftarrow 100 \times \max(0, 1 - \mathcal{L}_{changed}/\mathcal{L}_{total})$
Step 2 Structure Similarity:
    Get paths: $\mathcal{P}_i \leftarrow \text{RelativePaths}(\mathcal{F}_i)$, $\mathcal{P}_j \leftarrow \text{RelativePaths}(\mathcal{F}_j)$
    Jaccard similarity: $S_{structure} \leftarrow 100 \times |\mathcal{P}_i \cap \mathcal{P}_j|/|\mathcal{P}_i \cup \mathcal{P}_j|$
Step 3 Overall Similarity:
    return $S_{overall} = (S_{content} + S_{structure})/2$

Content similarity ($S_{content}$) examines code differences between repositories using `git-diff`[11], selected for its widespread adoption in handling large-scale version control systems worldwide [21] in detecting line-level changes across text-based files. `git-diff` options for used for repository comparison: `-no-index` enables comparison without git initialization; `-numstat` provides machine-readable statistics; `-ignore-all-space` and `-ignore-blank-lines` normalize formatting variations common in IaC files. The algorithm counts total lines ($\mathcal{L}_{total}$) across all text files in both repositories (r_i, r_j), identifies changes through git diff ($\Delta(r_i, r_j)$), and calculates similarity as the percentage of unchanged content. If repositories contain 1000 total lines and 200 lines differ between them, the content similarity equals 80%.

Structure similarity ($S_{structure}$) analyzes repository organization using the Jaccard coefficient [32] to compare file path sets. The Jaccard coefficient was

[11] git-diff. git project (2025), https://git-scm.com/docs/git-diff

chosen as it provides a normalized measure of set overlap that effectively captures structural correspondence between repositories regardless of their absolute sizes [15], making it ideal for comparing projects of varying scales. The algorithm extracts relative file paths ($\mathcal{P}_i$, $\mathcal{P}_j$) and measures overlap: if repositories share 30 paths among 50 unique paths total, structure similarity equals 60%.

Overall similarity ($S_{overall}$) combines content and structure measurements through arithmetic mean, reflecting equal contribution of code content and organizational structure to repository relationships.

4.2 Security Smell Diffusion Analysis

Algorithm 2 uses overall similarity scores to identify security smells and track their diffusion patterns. Since platforms like Ansible Galaxy lack explicit fork information, the algorithm infers cloning relationships based on similarity percentages, enabling security smell tracking in ecosystems without formal fork metadata. Our methodology applies a similarity threshold (τ) to filter repository pairs ($\mathcal{P}$), considering only those exceeding the threshold as potential repository clones ($\mathcal{Q}$). This ensures that only repositories with substantial code overlap are analyzed for security smell diffusion, avoiding false positives from coincidentally similar structures.

Algorithm 2: Security Smell Analysis

Input: Repository pairs with overall similarity scores $\mathcal{P} = \{(r_i, r_j, S_{ij})\}$,
threshold (τ)
Output: Security smell diffusion map $\mathcal{V}$
Cloned Repository Detection:
    $\mathcal{Q} \leftarrow \{(r_i, r_j, S_{ij}) \in \mathcal{P} : S_{ij}.overall \geq \tau\}$
Source Identification & Analysis:
for all $(r_i, r_j, S_{ij}) \in \mathcal{Q}$ **do**
    $t_i, t_j \leftarrow \text{GetCreationTime}(r_i), \text{GetCreationTime}(r_j)$
    $(source, clone) \leftarrow (r_i, r_j)$ if $t_i < t_j$ else (r_j, r_i)
    AnalyzeSecuritysmells($source, clone$)
end for
AnalyzeSecuritysmells($source, clone$)**:**
    $\mathcal{C}_{source} \leftarrow \text{MapCWEs}(\text{ScanSecuritySmells}(source))$
    $\mathcal{C}_{clone} \leftarrow \text{MapCWEs}(\text{ScanSecuritySmells}(clone))$
    $\mathcal{C}_{propagated} \leftarrow \mathcal{C}_{source} \cap \mathcal{C}_{clone}$
    $\mathcal{C}_{induced} \leftarrow \mathcal{C}_{clone} \setminus \mathcal{C}_{source}$
    $\mathcal{C}_{fixed} \leftarrow \mathcal{C}_{source} \setminus \mathcal{C}_{clone}$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(source, clone, \mathcal{C}_{propagated}, \mathcal{C}_{induced}, \mathcal{C}_{fixed})\}$

After identifying potential repository clones, the algorithm determines source-clone relationships using 'creation timestamps (t_i, t_j)' from repository metadata obtained from IaC platform API's. The earlier-created repository is designated as the source and the later one as the cloned repository, establishing the direction

of code flow and security smell diffusion. While temporal ordering may not capture all edge cases (such as simultaneous development), IaC platform-provided timestamps serve as a reliable proxy for source-clone relationships in the vast majority of non-malicious code reuse scenarios.

Both repositories undergo attribute-based static code analysis. We use the technique presented by Konala et al. [10], that employs regular expressions to identify security smells. These are then mapped to Common Weakness Enumerations (CWEs)[12]. This analysis enables comparison of security characteristics between source and clone repositories. The algorithm categorizes detected security smells through set operations on CWEs:

- **Propagated** ($\mathcal{C}_{propagated}$): Present in both source and clone repositories, indicating persistence through cloning
- **Induced** ($\mathcal{C}_{induced}$): Absent in source repository but present in repository clones, representing newly introduced security smells
- **Fixed** ($\mathcal{C}_{fixed}$): Present in source repository but absent in repository clones, suggesting security improvements

4.3 Dataset Overview

Data obtained from Ansible Galaxy served as the primary dataset for this study. While the presented methodology can be generalized to other code management platforms that provide metadata access, Ansible Galaxy was chosen for three reasons. First, its structured repository format enables straightforward extraction and analysis. Second, unlike platforms like GitHub that mix diverse technologies, Ansible Galaxy contains only Ansible-specific content, eliminating preprocessing requirements. Third, Ansible Galaxy provides access to a wide technology stack, and since configuration management inherently deals with software setup requiring extensive configuration parameters, it offers opportunities to observe security smell diffusion patterns across diverse software implementations.

For our analysis in this paper we selected three technologies representing different infrastructure categories: Elasticsearch (search engine, 188 repositories), Jenkins (CI/CD platform, 210 repositories), and MySQL (database, 491 repositories), totaling 889 unique repositories from Ansible Galaxy, excluding commit versions of the same repository in these counts. These technologies were chosen for their diverse repository counts, widespread adoption in production environments, and representation of critical infrastructure components where security vulnerabilities have an impact. Pairwise comparison generated unique pairs: 17,578 for Elasticsearch, 21,945 for Jenkins, and 120,295 for MySQL, providing sufficient data to assess cloning patterns while maintaining computational feasibility for initial validation.

4.4 Validation Using Known Fork Metadata

Since Ansible Galaxy and other IaC codebases lack repository fork information, we leveraged repositories of Ansible Galaxy which linked GitHub's fork

[12] Common Weakness Enumeration, https://cwe.mitre.org/index.html

chain data as ground truth to validate our methodology. From our survey (Section 3.1, Table 1) few IaC repository authors link their repositories to GitHub, which provides fork chain information enabling calculation of accuracy, precision, recall, and F1 score metrics. We analyzed metadata from the same three repository technologies to identify those linked to GitHub through the `github_repo`, `github_branch`, and `github_user` parameters. This analysis identified documented GitHub fork relationships: 43 pairs (37 unique repositories) from Elasticsearch, 34 pairs (30 unique repositories) from MySQL, and 32 pairs (33 unique repositories) from Jenkins. For each relationship, we retrieved the corresponding GitHub commit version repositories at fork creation time, enabling temporal alignment for similarity calculations. The known fork pairs exhibited high similarity scores across all technologies: Elasticsearch (mean 81.02%, median 89.60%), Jenkins (mean 86.16%, median 93.56%), and MySQL (mean 89.01%, median 95.22%). Based on these distributions exceeding 80%, we established our similarity threshold $\tau = 80\%$. To validate this threshold, we augmented each technology dataset with 10 randomly selected non-fork repositories from Ansible Galaxy, creating a test set containing both fork and non-fork pairs. Table 2 presents the validation results. In our context, True Positives (TP) represent actual forks correctly identified by our algorithm; False Negatives (FN) are actual forks misclassified as non-forks due to extensive modifications reducing their similarity below τ; False Positives (FP) would indicate non-fork pairs misidentified as forks, but none occurred since all randomly selected repository pairs fell below threshold τ; True Negatives (TN) represent non-fork pairs accurately classified, which are confirmed by their similarity scores remaining below τ. The algorithm achieved high accuracy across all three technologies (99.67%-99.88%) with 100% precision since no false positives occurred as all randomly selected non-fork pairs scored below the defined threshold τ. The false negatives (13 in Elasticsearch, 3 in Jenkins, 6 in MySQL) represent heavily modified forks whose similarity scores fell below τ. Though technically forks, they evolved substantially enough to constitute new repositories. Treating them as unique aligns with security considerations since heavily modified forks are unlikely to share the same security smells as their sources [27].

Table 2. Validation Results for Fork Detection Using Similarity Data

Technology	Confusion Matrix	Predicted Fork	Predicted Not Fork	Accuracy	Precision	Recall	F1 Score
Elasticsearch	Actual Fork	30 (TP)	13 (FN)	99.67%	100%	69.77%	82.19%
	Actual Not Fork	0 (FP)	3,873 (TN)				
Jenkins	Actual Fork	29 (TP)	3 (FN)	99.88%	100%	90.62%	95.08%
	Actual Not Fork	0 (FP)	2,524 (TN)				
MySQL	Actual Fork	28 (TP)	6 (FN)	99.77%	100%	82.35%	90.32%
	Actual Not Fork	0 (FP)	2,522 (TN)				

4.5 Comparison with Related Work

To contextualize our findings, we analyzed the 24 IaC smell detection tools (Refer to Section 3.2) and determined they detect security smells within individual repositories but lack capabilities to analyze metadata, identify repository clones,

or track security smell diffusion across IaC codebases. Direct comparison with traditional software code clone detection tools is not feasible as they operate on fundamentally different code structures. Traditional software differs from infrastructure configuration scripts, making such comparisons inappropriate for evaluating our methodology. The most relevant comparison point is Tsuru *et al.* [33], who analyzed 5,000 standalone Docker scripts and achieved 95% precision. However, their study reports only precision without providing accuracy, recall, or F1 scores, limiting comprehensive comparison. While their work falls under the IaC domain, it focuses specifically on image building rather than configuration management, making it an imperfect but closest available benchmark. This distinction is necessary as configuration management scripts exhibit different structural patterns and reuse patterns compared to container build files.

5 Findings and Discussion

This section presents the results of our methodology and examines security smell diffusion in Ansible Galaxy repositories. We analyze the complete Ansible Galaxy ecosystem, tracking how CWEs and CVEs[13] propagate through identified repository clones and their relationships.

5.1 Repository Clone Detection in Ansible Galaxy

After validating our methodology using known fork metadata, we applied it to analyze 25,425 Elasticsearch pairs, 28,680 Jenkins pairs, and 120,295 MySQL pairs. This analysis included repository commit versions of known forks and used the similarity threshold (τ) established during validation. Our analysis revealed extensive code reuse across all three technologies: 87 repositories (38.4%) out of 226 Elasticsearch repositories formed 154 similarity pairs; 130 repositories (54.1%) out of 240 Jenkins repositories formed 473 pairs; and 190 repositories (38.7%) out of 491 MySQL repositories formed 3,539 pairs. These findings indicate that 38.4%-54.1% of repositories on Ansible Galaxy share code overlap with at least one other repository. The prevalence of code reuse across all three technology categories and their repository commit versions demonstrates that repository cloning is a widespread practice in the Ansible Galaxy ecosystem, creating potential pathways for security smell diffusion across multiple repositories.

5.2 Security Smells Mapping To CWE

Our analysis of security smell diffusion involved mapping detected security smells to CWEs and studying their patterns in cloned repositories. Initial scanning revealed prevalent security smells: suspicious comments exposing sensitive information, hard-coded credentials and obsolete functions. We mapped these findings to three relevant CWEs commonly affecting IaC: CWE-546[14] manifests

[13] Common Vulnerabilities and Exposures, https://cve.mitre.org/
[14] CWE-546: Suspicious comments, https://cwe.mitre.org/data/definitions/546.html

through suspicious comments revealing sensitive system information, internal configurations, or debugging data; CWE-798[15] occurs when passwords, tokens, or API keys are embedded directly in code; and CWE-477[16] involves deprecated modules, plugins or outdated syntax containing known security smells. Table 3 presents the diffusion patterns for Ansible Galaxy repositories with GitHub fork metadata (Source-Fork Pairs) and Ansible Galaxy repositories identified as clones without fork metadata linkage (Source-Clone Pairs). The analysis reveals

Table 3. Diffusion Analysis Using Cumulative CWE Counts

Technology	Diffusion Patterns	Propagated			Induced			Fixed		
	CWE ID	546	798	477	546	798	477	546	798	477
Elasticsearch	Source	22	0	28	0	0	1	2	0	0
	Fork	22 ↕	0	28 ↕	1 ↑	0	2 ↑	0 ↓	0	0
	Source	15	6	31	6	0	0	4	0	9
	Clone	15 ↕	6 ↕	31 ↕	6 ↕	0	5 ↑	7 ↑	0	0 ↓
Jenkins	Source	1	0	15	0	0	0	10	0	13
	Fork	1 ↕	0	15 ↕	0	0	0	3 ↓	0	0 ↓
	Source	14	0	36	0	0	0	4	0	8
	Clone	14 ↕	0	36 ↕	0	0	2 ↑	0 ↓	0	0 ↓
MySQL	Source	0	0	2	0	0	0	0	0	2
	Fork	0	0	2 ↕	1 ↑	0	3 ↑	0	0	0 ↓
	Source	1	2	4	0	0	0	2	16	11
	Clone	1 ↕	2 ↕	4 ↕	2 ↑	7 ↑	16 ↑	0 ↓	0 ↓	0 ↓

Note: The values represent total CWE instances across repository pairs in specific diffusion pattern category. Arrow indicators show changes from source to fork/clone: ↕ same as source; ↑ increase; ↓ decrease in CWE counts.

distinct security smell patterns across technologies. CWE-477 (obsolete modules & plugins) and CWE-546 (suspicious comments) show the highest propagation rates across all three technologies, while CWE-798 (hardcoded credentials) appears minimally. The prevalence of CWE-477 aligns with Konala *et al.* [11] findings that Ansible Galaxy repositories have a mean release date of November 2018 and median of May 2018, representing approximately seven years without updates at the time of our study (July 2025). This temporal gap explains the widespread use of deprecated modules and plugins. Induced diffusion patterns, where repository clones introduce new security smells occur frequently. Elasticsearch repositories primarily exhibit CWE-477 and CWE-546, MySQL shows CWE-798 occurrences, and Jenkins demonstrates limited CWE-477 instances. Fixed diffusion patterns reveal a concerning trade-off in Elasticsearch and Jenkins: while developers remove obsolete functions (decreasing CWE-477), they introduce suspicious comments containing quick fixes and workarounds (in-

[15] CWE-798: Hard-coded credentials, https://cwe.mitre.org/data/definitions/798.html
[16] CWE-477: Obsolete Function, https://cwe.mitre.org/data/definitions/477.html

creasing CWE-546). This substitution suggests developers address deprecated functions by embedding temporary solutions in comments, inadvertently creating new information exposure security smells while attempting to modernize outdated code.

5.3 Author Practices and User Selection Analysis

The diffusion patterns reveal three observable author practices: repositories are cloned without modification from sources containing security smells, perpetuating existing smells; some repositories show security smell remediation during the cloning process; and new security smells appear in cloned repositories that were absent in the source. These induced security smells may result from various factors including coding errors, dependency changes, or configuration modifications during the cloning process. Such security smells pose threats because users may select repository clones believing them to be improved versions. This security gap in Ansible Galaxy becomes apparent: while repositories with GitHub's fork chain metadata enables tracking smell origins, Ansible Galaxy lacks such mechanisms despite containing identical security smells.

Table 4. Cumulative Repository Download Counts Across Diffusion Pattern Categories

Technology	Relationship	Propagated	Induced	Fixed
Elasticsearch	Source	4,865,481	797,204	5
	Fork	5,079	227	5,962
	Source	4,885,802	57*	$3,954,664^\star$
	Clone	106,263	119*	$82,670^\star$
Jenkins	Source	2,778,893	–	$5,316^\star$
	Fork	7,271	–	$1,105^\star$
	Source	2,829,994	37*	$2,761,135^\star$
	Clone	22,688	$2,063^\star$	$2,290^\star$
MySQL	Source	4,983,642	4,646,981	357
	Fork	8,528	190	1,232
	Source	5,446,593	$3,281^\star$	$2,354^\star$
	Clone	338,452	$5,019^\star$	782*

Note: The values show cumulative downloads for all repositories in each category. $\star$ - indicates anomalous patterns where vulnerable repositories receive higher downloads than expected.

The impact of fork metadata absence becomes quantifiable through download metrics obtained from Ansible Galaxy's repository metadata (Table 4), which reveal different user selection patterns between platforms with and without fork visibility. On repositories with linked GitHub fork metadata, where fork relationships are transparent, users demonstrate security awareness: inducing pattern forks constitute 0.03% of total downloads for Elasticsearch (227 out of 797,431 combined downloads) and 0.004% for MySQL (190 out of 4,647,171 combined downloads). This distribution may indicate security awareness or simply reflect

user preference for established source repositories when platform-based trust mechanisms such as stars and download counts are visible [12]. Conversely, Ansible Galaxy's absence of fork metadata creates concerning download patterns. Without visibility into repository origins, repository clones with induced security smells capture the majority of downloads: 67.61% for Elasticsearch (119 out of 176 total downloads), 98.24% for Jenkins (2,063 out of 2,100 total downloads), and 60.47% for MySQL (5,019 out of 8,300 total downloads) relative to their source repositories.

At first glance, data for the fixed diffusion pattern shown interesting results. The users of Ansible Galaxy repositories with fork metadata linked to GitHub demonstrate migration to security-improved repositories. It is however unclear, how users identify specific repositories that fix CWEs on GitHub. A closer look at the repositories shows this behaviour is potentially because fixed repositories receive more recent updates and appear higher in GitHub search results, rather than a security-conscious effort on the user's part. This behaviour seems amplified because of low download counts. The repositories without fork metadata in Ansible Galaxy exhibit the expected behaviour with continued downloads of vulnerable sources across all three technologies. The cascade effect is particularly concerning: new Ansible Galaxy users typically sort repositories by download count [12], unknowingly selecting popular (most downloaded) but vulnerable repositories, which further inflates their download numbers and perpetuates the security risk. The platform's interface in this case unknowingly leads users away from repositories that may have fixed certain vulnerabilities.

These findings demonstrate that by adding fork metadata information to Ansible Galaxy's user interface could improve security decision-making, enabling users to trace repository origins and identify potentially induced or improved versions before deployment.

5.4 Security Smells Mapping To Known CVE

To understand the severity of security smells that users unknowingly download, we mapped them to known CVEs to determine if diffusion patterns mirror CWE diffusion. Our analysis revealed three distinct known CVEs exhibiting diffusion patterns across the analyzed technologies.

CVE-2017-7550 (CVSS v3.x: 9.8 CRITICAL)[17] demonstrated extensive **propagated** diffusion in Jenkins, appearing in 3 source-fork pairs and 2 source-clone pairs. Most notably, it propagated from older versions of a popular Jenkins repository (2.7 million downloads) to multiple forks with 129 and 58 downloads. While the popular repository eventually released patches, fork authors never updated their repositories, leaving vulnerabilities active. This pattern also appeared in less popular repositories, propagating from a source (42 downloads) to clone repository (38 downloads). Conversely, one instance showed **fixed** diffusion where a source repository (17,514 downloads) contained the CVE but its

[17] CVE-2017-7550: Sensitive information exposure via ansible jenkins_plugin parameters, https://nvd.nist.gov/vuln/detail/CVE-2017-7550

repository clone (51 downloads) did not. However, the absence of fork metadata in Ansible Galaxy suppresses users from discovering this safer alternative.

CVE-2020-14365 (CVSS v3.x: 7.1 HIGH)[18] exhibited **propagated** diffusion in Jenkins, spreading from a source repository (2,631 downloads) to its clone (101 downloads), while CVE-2024-8775 (CVSS v3.x: 5.5 MEDIUM)[19] demonstrated **fixed** diffusion in Jenkins, where the vulnerability present in the source repository (132 downloads) was remediated in its clone (53 downloads). These findings demonstrate that code reuse propagates known CVEs similar to CWEs. Like traditional software, IaC is also impacted from vulnerability diffusions through repository cloning, highlighting the need for security screening in IaC-specific codebases. Once vulnerable repositories are identified, developers can be notified with remediation guidance, potentially using LLMs for automated fixes.

5.5 Limitations and Threats to Validity

Construct Validity: Our methodology requires metadata such as repository first release dates to detect relationships and map security smell diffusion patterns. Without version control history, we cannot establish the temporal relationships necessary to track security smell propagation, limiting our analysis to repositories with sufficient metadata and potentially excluding diffusion patterns that occur outside version-controlled environments. **Internal Validity:** Even with adequate metadata, the similarity threshold (τ), while validated on Ansible Galaxy repositories that have GitHub fork metadata, may not generalize across all IaC technologies or coding patterns. Different technologies may exhibit distinct code reuse patterns that require adjusted thresholds based on evidence and experimentation. **External Validity:** Beyond these methodological constraints, our evaluation focused on repositories of three technologies within Ansible Galaxy, which may not represent security smell diffusion in other IaC domains or platforms, as different configuration management tools may demonstrate alternative reuse behaviors not captured in our study.

5.6 Future Work

Future research should analyze repository evolution through commit history to track how security smells develop over time, distinguishing between gradual emergence and immediate injection. Expanding to all Ansible Galaxy technologies would reveal domain-specific vulnerability patterns such as database technologies may differ from web servers in their susceptibility to certain CWEs/CVEs. A comprehensive analysis of the entire Ansible Galaxy platform could reveal anomaly phenomena in repository relationships and security smell patterns. This knowledge would enable developers to implement targeted remediation

[18] CVE-2020-14365: Improper gpg signature verification in ansible engine dnf module, https://nvd.nist.gov/vuln/detail/CVE-2020-14365

[19] CVE-2024-8775: Exposure of sensitive information in ansible vault during playbook execution, https://nvd.nist.gov/vuln/detail/CVE-2024-8775

strategies. Applying our methodology to other metadata-rich IaC platforms such as Chef Supermarket, Puppet Forge, and Terraform Registry would determine whether observed diffusion patterns are platform-specific or ecosystem-wide phenomena.

6 Conclusion

This paper presented an approach to detect repository clones in IaC ecosystems lacking fork metadata. Our analysis of Ansible Galaxy repositories across three technologies revealed 38.4%–54.1% share code overlap, facilitating diffusion of security smells across 3 CWEs and 3 CVEs, including CVE-2017-7550 (CVSS 9.8 - Critical) which propagated from a version of a popular repository with 2.7 million downloads. This widespread cloning creates consequences: users download repository clones with induced security smells at 100× higher rates than they download Ansible Galaxy repositories linked to GitHub, where fork visibility enables informed decisions. Without metadata to trace code origins, this gap will widen as code reuse practices continue to grow across IaC repositories. Therefore, repository relationship tracking is essential for detecting vulnerability inheritance and securing the IaC ecosystem.

References

1. Baker, B.S.: On finding duplication and near-duplication in large software systems. In: Proc. of 2nd Working Conference on Reverse Engineering (WCRE). pp. 86–95 (1995)
2. Baxter, I.D., Yahin, A., Moura, L., Sant'Anna, M., Bier, L.: Clone detection using abstract syntax trees. In: Proc. of the International Conference on Software Maintenance (ICSM). pp. 368–377 (1998)
3. Dean, J., Tofade, O., Poudel, S., Rahman, A.: Propagation of insecure coding in configuration scripts. `https://publish.tntech.edu/index.php/PSRCI/article/view/688` (May 2020)
4. Ducasse, S., Rieger, M., Demeyer, S.: A language independent approach for detecting duplicated code. In: Proceedings of the 15th International Conference on Software Maintenance. pp. 109–118. ICSM '99, IEEE Computer Society, Oxford, UK (1999)
5. Giladi, M., David, G.: Over 100,000 infected repos found on GitHub. https://apiiro.com/blog/malicious-code-campaign-github-repo-confusion-attack/ (2024), apiiro Blog, 28 Feb 2024
6. Jiang, L., Misherghi, G., Su, Z., Glondu, S.: DECKARD: Scalable and accurate tree-based detection of code clones. In: Proc. of the 29th International Conference on Software Engineering (ICSE). pp. 96–105 (2007)
7. Johnson, J.: Identifying redundancy in source code using fingerprints. In: Proceedings of the 1993 Conference of the Centre for Advanced Studies on Collaborative Research. pp. 171–183. CASCON '93, IBM Press, Toronto, ON, Canada (1993)
8. Johnson, J.: Substring matching for clone detection and change tracking. In: Proceedings of the 10th International Conference on Software Maintenance. pp. 120–126. ICSM '94, IEEE Computer Society, Victoria, BC, Canada (1994)

9. Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering **28**(7), 654–670 (2002). https://doi.org/10.1109/TSE.2002.1019480
10. 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
11. Konala, P.R.R., Kumar, V., Bainbridge, D., Haseeb, J.: Metadata assisted supply-chain attack detection for Ansible. In: Katsikas, S., Shafiq, B. (eds.) Data and Applications Security and Privacy XXXIX. pp. 333–350. Springer Nature Switzerland, Cham (2025)
12. Larios Vargas, E., Aniche, M., Treude, C., Bruntink, M., Gousios, G.: Selecting third-party libraries: the practitioners' perspective. In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 245–256. ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3368089.3409711
13. Li, Z., Lu, S., Myagmar, S., Zhou, Y.: CP-Miner: Finding copy-paste and related bugs in large-scale software code. In: Proc. of the 6th Symposium on Operating Systems Design and Implementation (OSDI). pp. 289–302 (2004)
14. Manber, U.: Finding similar files in a large file system. In: Proceedings of the Winter 1994 Usenix Technical Conference. pp. 1–10. USENIX Winter '94, USENIX Association, San Francisco, CA, USA (1994)
15. Martinez-Gil, J.: Source code clone detection using unsupervised similarity measures. In: Bludau, P., Ramler, R., Winkler, D., Bergsmann, J. (eds.) Software Quality as a Foundation for Security. pp. 21–37. Springer Nature Switzerland, Cham (2024), https://doi.org/10.1007/978-3-031-56281-5_2
16. Mayrand, J., Leblanc, C., Merlo, E.: Experiment on the automatic detection of function clones in a software system using metrics. In: Proc. of the International Conference on Software Maintenance (ICSM). pp. 244–253 (1996)
17. Meijer, B., Hochstein, L., Moser, R.: Ansible: Up and Running. O'Reilly Media, Sebastopol, CA, 3rd edition edn. (2022)
18. Microsoft Threat Intelligence: Malvertising campaign leads to info stealers hosted on GitHub. https://www.microsoft.com/en-us/security/blog/2025/03/06/malvertising-campaign-leads-to-info-stealers-hosted-on-github/ (2025)
19. Munoz, A.: The octopus scanner malware: Attacking the open source supply chain. https://github.blog/security/vulnerability-research/the-octopus-scanner-malware-attacking-the-open-source-supply-chain/ (2020)
20. Nachshon, G.: Surprise: When Dependabot contributes malicious code. https://checkmarx.com/blog/surprise-when-dependabot-contributes-malicious-code/ (2023)
21. Nugroho, Y.S., Hata, H., Matsumoto, K.: How different are different diff algorithms in Git?: Use –histogram for code changes. Empirical Software Engineering **25**(1), 790–823 (January 2020). https://doi.org/10.1007/s10664-019-09772-z
22. Oumaziz, M.A., Falleri, J.R., Blanc, X., Bissyandé, T.F., Klein, J.: Handling duplicates in Dockerfiles families: Learning from experts. In: Proc. 35th IEEE Int. Conf. on Software Maintenance and Evolution (ICSME). pp. 524–535 (2019)
23. Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., Lucia, A.: On the diffuseness and the impact on maintainability of code smells: a large scale

empirical investigation. Empirical Software Engineering **23**, 1–34 (06 2018). https://doi.org/10.1007/s10664-017-9535-z
24. Proulx, F.: Erosion of trust: Unmasking supply chain vulnerabilities in the Terraform Registry. https://boostsecurity.io/blog/erosion-of-trust-unmasking-supply-chain-vulnerabilities-in-the-terraform-registry (2023), accessed: 2023
25. Raban, S.: The dark side of domain-specific languages: Uncovering new attack techniques in OPA and Terraform. https://www.tenable.com/blog/the-dark-side-of-domain-specific-languages-uncovering-new-attack-techniques-in-opa-and (2024)
26. Reddy Konala, P.R., Kumar, V., Bainbridge, D.: SoK: Static configuration analysis in infrastructure as code scripts. In: 2023 IEEE International Conference on Cyber Security and Resilience (CSR). pp. 281–288 (2023). https://doi.org/10.1109/CSR57506.2023.10224925
27. Reid, D., Jahanshahi, M., Mockus, A.: The extent of orphan vulnerabilities from code reuse in open source software. In: Proceedings of the 44th International Conference on Software Engineering. p. 2104–2115. ICSE '22, Association for Computing Machinery, New York, NY, USA (2022), https://doi.org/10.1145/3510003.3510216
28. Roy, C.K., Cordy, J.R.: NiCad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In: Proc. of the 16th IEEE International Conference on Program Comprehension (ICPC). pp. 172–181 (2008)
29. Roy, C.K., Cordy, J.R., Koschke, R.: Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming **74**(7), 470–495 (2009). https://doi.org/10.1016/j.scico.2009.02.007
30. Roy, C.K., Cordy, J.R.: A survey on software clone detection research. Tech. Rep. 2007-541, School of Computing, Queen's University at Kingston, Ontario, Canada (September 2007), https://research.cs.queensu.ca/TechReports/Reports/2007-541.pdf
31. Sharma, A.: 35,000 code repos not hacked—but clones flood GitHub to serve malware. https://www.bleepingcomputer.com/news/security/35-000-code-repos-not-hacked-but-clones-flood-github-to-serve-malware/ (Aug 2022), bleepingComputer, 3 Aug 2022
32. Tan, P.N., Steinbach, M., Karpatne, A., Kumar, V.: Introduction to Data Mining. Pearson, 2nd edn. (2019), chapter 2 covers Jaccard coefficient as a similarity measure
33. Tsuru, T., Sato, H., Matsumoto, S.: Type-2 code clone detection for Dockerfiles using syntax-aware normalization. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 219–229. IEEE (2021)
34. Vijayan, J.: 'stargazer goblin' amasses rogue GitHub accounts to spread malware. https://www.darkreading.com/application-security/stargazer-goblin-amasses-rogue-github-accounts-to-spread-malware (Jul 2024)
35. Wang, R.: Infrastructure as Code, Patterns and Practices: With examples in Python and Terraform. ITpro collection, Manning (2022)
36. Wixey, M., O'Donnell, A.: The strange tale of *ischhfd83*: When cybercriminals eat their own. https://news.sophos.com/en-us/2025/06/04/the-strange-tale-of-ischhfd83-when-cybercriminals-eat-their-own/ (2025), sophos X-Ops, 4 Jun 2025