

Tools for Detecting Security Issues in Infrastructure-As-Code: A Focused Literature Review

Mohamed Haady Tientore ✉ 

Université d'Orléans, INSA CVL, LIFO UR 4022, Orléans, France

Frédéric Loulergue ✉ 

Université d'Orléans, INSA CVL, LIFO UR 4022, Orléans, France

Abstract

Infrastructure-as-Code (IaC) artefacts such as Puppet manifests, Ansible playbooks, Kubernetes manifests, Helm charts, Terraform configurations, and CloudFormation templates are increasingly used to manage critical infrastructures. Security defects in these artefacts may therefore be propagated rapidly through automated deployment pipelines. This paper reports emerging results from a focused literature review of tools for detecting security issues in IaC artefacts. The current review protocol covers the ACM Digital Library and IEEE Xplore and includes 23 primary studies. We identify the main families of tools captured by this protocol, classify their detection approaches, and analyse how they are evaluated. The review corpus includes rule-based linters, deeper static analyses, graph- and model-based approaches, comparative scanner studies, deployment-oriented analyses, and recent machine-learning and LLM techniques.

2012 ACM Subject Classification Security and privacy → Software security engineering; Software and its engineering → Software verification and validation

Keywords and phrases Infrastructure-as-Code, security tooling, misconfiguration detection, static analysis, empirical software engineering, focused literature review

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.55

Category Emerging Results, Vision & Reflection Track Paper

Supplementary Material *Dataset:* <https://doi.org/10.5281/zenodo.21820732>

Funding partly by For-CoaLa, a project funded by the French National Research Agency (ANR-24-CE25-3158)

1 Introduction

Infrastructure-as-Code (IaC) is the practice of managing and provisioning infrastructure through machine-readable artefacts. In such an approach, infrastructure descriptions are versioned, reviewed, tested, and maintained similarly to software source code. IaC artefacts include Puppet manifests, Ansible playbooks, and Chef recipes for configuration management; Terraform and CloudFormation templates for cloud provisioning; and Kubernetes manifests and Helm charts for container orchestration.

The increasing adoption of IaC is motivated by automation, reproducibility, and scalability. However, these same characteristics also amplify the consequences of defects. A security weakness may be propagated quickly through deployment pipelines and replicated over many machines, clusters, or cloud resources. In practice, such weaknesses include insecure defaults, over-permissive access-control rules, exposed services, embedded secrets, and unsafe scripting constructs. The security of IaC artefacts is therefore an important concern.

Over the last decade, a growing body of research has addressed this concern. The literature includes rule-based detectors, static analyses, graph-based and model-based approaches, and more recently machine-learning and LLM based techniques. These studies target different IaC



© Mohamed Haady Tientore and Frédéric Loulergue;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 55; pp. 55:1–55:14



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

ecosystems and different classes of security issues. However, the landscape is fragmented. It is therefore difficult to obtain a structured view of the existing tools, the technical approaches they rely on, and the evaluations reported for them.

Previous secondary studies provide a broad map of IaC research [15], survey static analysis for both security and reliability defects [3], or examine artificial intelligence across the full IaC lifecycle [13]. Our review is narrower: it focuses on security *detection* tools across analysis families and, in particular, on the empirical evidence used to evaluate them. This focus helps researchers identify comparable evaluation designs and helps practitioners judge what kind of evidence supports a tool family in a given IaC setting.

We present emerging results from a focused literature review of tools for detecting security issues in IaC artefacts. The goal is to establish a structured first map of the reported research landscape captured by the current review protocol. This first iteration covers the ACM Digital Library and IEEE Xplore, and it is intended to be extended with additional digital libraries and backward and forward snowballing. This scope is focused on detection. Topics such as automated repair, prevention, or runtime enforcement are adjacent and relevant, but require a broader study. More specifically, the contributions are:

- an overview of the tools and methods proposed for detecting security issues in IaC;
- a taxonomy of these studies according to the targeted artefacts, the security issues considered, the core detection mechanisms, and the evaluation methods;
- a discussion of the reported evaluations, with implications for empirical software engineering research on IaC security tooling.

In the remainder of the paper, Section 2 describes the review methodology. Section 3 presents the results with respect to the research questions. Section 4 introduces the proposed taxonomy. Section 5 discusses challenges. Section 6 summarizes threats to validity. Section 7 concludes and gives future research directions.

2 Research Methodology

The SLR carried out by the authors of this work adhered to established literature guidelines [14]. It consisted of three main steps: (i) *planning*, which involves protocol creation with research questions, search strategy, selection criteria, and data extraction and synthesis methods; (ii) *execution*, which entails the identification, selection, and analysis of studies; and (iii) *reporting*, which aggregates the information extracted from relevant studies in relation to the research questions, synthesizing the findings to draw conclusions.

Research Goal. The research goal is to establish a focused overview of the tools and methods for detecting security issues in IaC.

Research Questions. To guide this study, we formulated three research questions (RQs) aimed at identifying primary studies on security tools and methods for IaC.

RQ1. What tools and methods have been proposed to detect security issues in Infrastructure-as-Code artefacts?

RQ1 Rationale: IaC security tooling has evolved rapidly over the past decade, with a growing number of tools targeting different technologies, threat models, and stages of the development lifecycle. However, no consolidated inventory of these tools and methods exists to date. Answering RQ1 provides practitioners and researchers with a structured overview of the current tooling landscape for IaC security detection, enabling informed decisions about tool adoption and identifying areas where tooling is still lacking.

RQ2. What technical approaches are used by existing IaC security tools?

RQ2 Rationale: Understanding the technical approaches of IaC security tools is essential to assess their capabilities and limitations. Different approaches offer varying trade-offs in terms of coverage, precision, and applicability. Answering RQ2 provides a deeper understanding of how these tools work, facilitating comparisons between approaches and helping researchers identify promising directions for future work.

RQ3. What empirical evidence supports the effectiveness of existing IaC security tools?

RQ3 Rationale: Proposing a tool is not sufficient to establish its practical value. Experimental validation is necessary to assess whether a tool effectively detects security issues in real-world IaC scripts. However, the rigour and reproducibility of evaluations vary significantly across the literature. Answering RQ3 provides an overview of how existing tools have been evaluated, and to what extent their effectiveness has been empirically demonstrated.

Search strategy

We carried out an automated search process over two electronic publication databases to retrieve relevant studies to answer the posed RQs: ACM Digital Library and IEEE Xplore. They are among the most popular publication databases in Computer Science and Engineering, and have good coverage.

Based on the defined RQs, two main groups of terms were identified. The first group covers security-related concepts, while the second targets IaC technologies and frameworks. The following search string was built considering synonyms and alternative terms to increase coverage of the literature. Security-related terms were searched in the document title only, ensuring that security is the primary focus of retrieved studies.

IaC related terms were searched in both the title and abstract, in order to capture studies where the IaC context is described in the abstract without being explicitly mentioned in the title. The named technologies operationalize three families in the review goal – configuration management, cloud provisioning, and container orchestration – and complement rather than replace the generic “Infrastructure as Code” and “IaC” terms. Restricting security terms to titles favours precision and feasible manual screening. The search was limited to publications from 2016 to 2026, covering the last decade:

```
(Title:(Security OR DevSecOps OR Secure OR Detection OR Scanning OR Check
OR Identification) ) AND
(Title:("Infrastructure as Code" OR IaC OR Ansible OR Puppet OR Chef OR Terraform
OR Kubernetes OR Pulumi OR CloudFormation OR SaltStack) OR
Abstract:("Infrastructure as Code" OR IaC OR Ansible OR Puppet OR Chef OR Terraform
OR Kubernetes OR Pulumi OR CloudFormation OR SaltStack) )
```

This search design intentionally favours focus and traceability over maximal recall. We restricted the automated search to the ACM Digital Library and IEEE Xplore because these databases provide strong coverage of software engineering and security venues while keeping a first review iteration manageable for full manual screening. We also relied on title and abstract metadata for automated retrieval, which is common in systematic searches but may miss studies whose security contribution is expressed only through body text, keywords, or alternative terminology. We therefore treat the present study as a focused first-pass review of IaC security *detection* tooling, rather than as an exhaustive census of all work touching IaC security. A natural next step is to extend the protocol with additional libraries and citation snowballing to assess the stability of the trends reported here under broader coverage.

Eligibility Criteria

We included primary studies published in peer-reviewed venues that propose, implement, evaluate, or systematically compare a concrete tool, prototype, or technically specified method for detecting security issues in Infrastructure-as-Code artefacts. To be eligible, a study had to focus either on the analysis of IaC artefacts themselves or on the security effects of configurations that remain explicitly traceable to IaC level artefacts and policies. We also retained studies that use a concrete security checker to assess policy violations in generated IaC artefacts when this contributes direct empirical evidence on IaC security tooling. In all cases, the study had to provide sufficient technical detail to allow classification of the underlying detection approach. Only full-text studies written in English were considered. Empirical evaluation was not required as an inclusion criterion, because excluding unevaluated studies would have biased the corpus against technically relevant tools and methods needed to answer RQ1 and RQ2. Instead, the presence, type, and strength of empirical evidence were extracted and summarized separately to address RQ3.

We excluded studies that did not address the detection of security issues in IaC artefacts or in their directly traceable deployment effects, studies concerned only with non-security properties of IaC, and studies that discussed security at the cloud or infrastructure level without an explicit link back to IaC artefacts or configuration policies. We also excluded papers that lacked sufficient technical detail to classify the detection approach, papers published outside the review window, and duplicate reports of the same study or tool, in which case only the most complete version was retained.

For borderline studies, we considered detection as a technically specified method that produces a security finding from an IaC artefact, an explicit configuration policy, or a deployment effect traceable to such an artefact. Repair- or prevention-centred work was excluded unless it exposed a separable detection component. Compliance and generated-IaC studies were retained only when they applied a concrete checker, while deployment-oriented studies were retained only when the reported violations remained traceable to IaC configurations.

For transparency, the operational eligibility criteria are summarized below.

Inclusion Criteria.

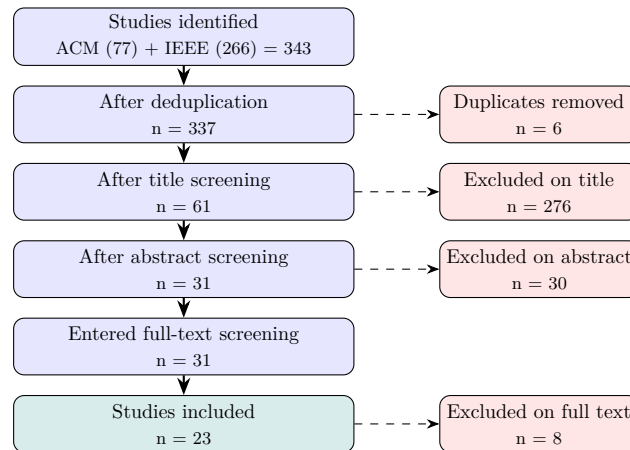
- **IC1:** The study is a primary peer-reviewed study published between January 2016 and April 2026.
- **IC2:** The study focuses on detecting security issues in IaC artefacts or deployments.
- **IC3:** The study proposes, implements, evaluates, or systematically compares a concrete detection tool, prototype, or technically specified method.
- **IC4:** The study provides sufficient technical detail to classify the detection approach.
- **IC5:** The study is presented in a full text available in English.

Exclusion Criteria.

- **EC1:** The study is not a primary peer-reviewed study.
- **EC2:** The study does not address the detection of security issues in IaC.
- **EC3:** The study addresses only non-security properties of IaC or only general cloud, infrastructure, or runtime security without an explicit link back to IaC artefacts.
- **EC4:** The study lacks sufficient technical detail to classify the detection approach.
- **EC5:** The study was published outside the review window.
- **EC6:** The full text is unavailable, or the study is not written in English.
- **EC7:** The study is a duplicate report of the same tool or study.

Selection process

The selection process is divided into five phases: (i) exclusion of duplicates, (ii) selection based on title, (iii) selection based on abstract, (iv) entry into full-text screening, and (v) final full-text inclusion or exclusion. The automated search retrieved 343 results (77 from ACM Digital Library and 266 from IEEE Xplore). After removing 6 duplicates, 337 studies remained. In Phase 2, 276 studies were excluded based on title screening, leaving 61 studies. In Phase 3, 30 additional studies were excluded based on abstract reading, leaving 31 studies for full-text screening. Finally, full-text reading retained 23 studies in the review corpus and excluded 8 studies. Figure 1 depicts the complete selection process, from the initial set of retrieved studies to the final set of relevant primary studies.



■ **Figure 1** Process for selecting the relevant primary studies.

Data extraction and synthesis

For each included study, we extracted bibliographic data, the tool or method, target artefacts, technical approach, reported empirical results, evaluation basis, and a scope note. These fields directly support RQ1–RQ3 and are available in the replication package. For synthesis, the extracted descriptions were coded into non-exclusive categories for ecosystem, security target, primary reasoning mechanism, and evaluation design. A study may therefore contribute to several evaluation categories, for example by combining a mined corpus, a baseline comparison, and practitioner feedback. We kept study role (new detector, refinement, comparison, or pipeline) distinct from detection mechanism, and used the resulting study-to-category mapping to derive the taxonomy and the counts reported below.

3 Results and Discussion

3.1 RQ1. What tools and methods have been proposed to detect security issues in Infrastructure-as-Code artefacts?

The reviewed studies cover several families of IaC security detectors. A first family targets configuration-management languages such as Puppet, Ansible, and Chef. In this part of the literature, security issues are often expressed as *security smells* or insecure coding patterns. Representative examples include SLIC for Puppet [17], SLAC for Ansible and Chef [18],

GLITCH as a cross-language smell detector [21], GASEL for Ansible [12], TaintPup for Puppet [16], and SecureCode for risky embedded scripts in Ansible playbooks [4]. Recent work in the same ecosystem also studies the detection of secret-sensitive Ansible parameters with machine-learning and language-model based classifiers, which extends the earlier smell-oriented literature toward more targeted semantic classification tasks [11].

A second family targets container-orchestration artefacts in the Kubernetes ecosystem, mainly Kubernetes manifests and Helm charts. Some studies propose dedicated detectors such as SLI-KUBE [19]. Others study the behaviour of existing analysers or compare several tools on Kubernetes and Helm artefacts, either directly on manifests and charts or indirectly through deployed configuration effects [23, 7, 8, 22, 6]. These comparative and deployment oriented studies bring concrete tools such as KubeLint, Kubesec, Kubescape, Trivy, kube-bench, Terrascan, and Checkov into the corpus, even when the paper contribution lies more in the evaluation setting than in the proposal of a new detector. The literature in this family also includes work that reasons about deployment topology or structural conformance in order to identify security exposures [1, 10].

A third family concerns cloud-provisioning artefacts such as Terraform and CloudFormation. The reviewed studies include threat-model reconstruction from Terraform configurations [26], comparisons of Terraform-oriented scanners [24], an evaluation of security policy compliance in LLM-generated IaC [5], and a recent workflow combining LLM based agents for vulnerability detection and remediation in CloudFormation templates [25]. A smaller set of studies targets cross-ecosystem validation workflows, for example design-time IaC scanning combined with runtime anomaly detection through the IaC Scan Runner/LOMOS combination [2] and LLM-augmented policy validation across Kubernetes, IAM, and Terraform [9]. Finally, a few recent papers revisit detection across several IaC ecosystems with machine-learning or LLM based approaches, e.g. SecLLM for Ansible, Puppet, and Chef [27]. Taken together, these studies show that the area is no longer limited to language-specific linters, even if ecosystem-specific tools still dominate the corpus.

Overall, the literature shows that IaC security detection is supported by a variety of tools and methods. However, the coverage of ecosystems remains unbalanced. Puppet and Ansible are well represented in the smell-detection literature, Kubernetes and Helm are increasingly studied through both direct and deployment-oriented analyses, while CloudFormation and other industrial ecosystems remain much less covered.

3.2 RQ2. What technical approaches are used by existing IaC security tools?

Rule-based static analysis is the most common approach in the reviewed literature. In this family, detectors rely on handcrafted rules, smells, or best-practice checks over parsed IaC artefacts [17, 18, 19, 20, 23]. These approaches are attractive because they are usually lightweight, interpretable, and easy to integrate into development pipelines. They also underpin a large part of the comparative literature, where tools such as Checkov, Tfsec, Tflint, KubeLint, Kubesec, or Kubescape are evaluated as representatives of current practice rather than as research prototypes in their own right.

Several studies go beyond simple rule matching by introducing richer static analyses. GASEL uses program dependence graphs and graph queries to analyse Ansible scripts [12]. TaintPup reasons about the propagation of weaknesses in Puppet manifests through taint-style data-dependence analysis [16]. SLI-KUBE tracks information flow across Kubernetes configurations through def-use style reasoning [19], while SecureCode analyses embedded scripting fragments inside IaC code using structured intermediate trees [4]. These approaches aim at capturing interactions that are difficult to express with local syntactic rules only.

Another family of approaches relies on intermediate representations or structural abstractions. GLITCH translates several IaC languages into a common representation in order to support reusable cross-language detectors [21]. Other studies build graphs, architectural models, or threat abstractions before applying security reasoning [1, 10, 26]. In this group, the technical core is not merely parsing but the construction of a model on which further reasoning is performed, for example topological graphs enriched with ATT&CK-inspired features, architecture-conformance models, or data-flow diagrams consumed by threat-modelling engines. Such approaches are particularly useful when the relevant security property depends on interactions among several resources or components.

The most recent studies introduce learning-based techniques. SecLLM uses large language models for smell detection across configuration languages [27]. Opdebeeck et al. predict secret-sensitive Ansible parameters with machine-learning and language models [11]. Toprani and Madiseti combine retrieval-augmented generation with specialized agents for CloudFormation vulnerability detection and remediation [25]. Mittal and Venkatesan augment pipeline-level policy validation with an LLM ensemble [9]. These methods add contextual interpretation, cross-language reuse, and explanatory or remediation output beyond fixed syntactic rules. By contrast, Hase et al. use the deterministic Checkov policy checker to measure violations in IaC generated by a LLM [5]: the LLM produces the input but is not the detector. The evidence does not establish general superiority of learning-based detection because evaluations use different tasks, datasets, baselines, and model settings.

The corpus also includes hybrid or deployment-oriented approaches. IaC Scan Runner combines design-time scanning across several IaC languages with runtime anomaly detection in logs [2]. Other studies evaluate dynamic application security testing (DAST) tools or workload scanners against configuration violations observed after deployment [22, 6]. In these papers, the observation point moves from the artefact alone to the effects of that artefact once deployed, typically through cluster queries, workload scans, or compliance checks over running resources. These papers broaden the technical landscape by connecting artefact-level properties to their operational manifestations.

The answer to RQ2 is therefore that the field is technically diverse, but still dominated by interpretable static analyses. Rule-based checks are lightweight, reproducible, and easy to integrate, but cover only encoded policies and may miss contextual or indirect flows. Data-flow and graph-based analyses capture richer relations at the cost of greater ecosystem-specific engineering. LLM-based methods offer flexible semantic reasoning but add nondeterminism, model drift, and cost. They also remain less uniformly evaluated than deterministic static analyses.

3.3 RQ3. What empirical evidence supports the effectiveness of existing IaC security tools?

Most reviewed studies provide some form of empirical evaluation, but the evaluation settings differ sharply. Some papers evaluate their approaches on mined open-source corpora, report standard detection metrics, and compare against baselines or prior tools [21, 12, 16, 27, 11]. Others complement quantitative results with practitioner feedback, accepted bug reports, or qualitative analyses [17, 18, 20, 19]. Large mined-repository evaluations dominate Puppet, Ansible, and Kubernetes studies, whereas recent LLM-based and pipeline-level papers often rely on smaller synthetic or case-study settings. As a result, strong metric values in one paper are not directly commensurable with strong metric values in another.

■ **Table 1** Quantitative synthesis of reported evaluation evidence ($N = 23$). Dataset bases are exhaustive; other categories are non-exclusive.

Dimension	Category	n
Dataset	Large mined repository corpus	12
Dataset	Controlled benchmark, case, or deployment	8
Dataset	Synthetic or small template set	3
Comparison	Explicit baseline or oracle	17
External validation	Practitioner/maintainer feedback or bug reports	10
Metric	Detection quality (e.g., precision, recall, F1, FP/FN)	16
Metric	Runtime, resource use, latency, or cost	11

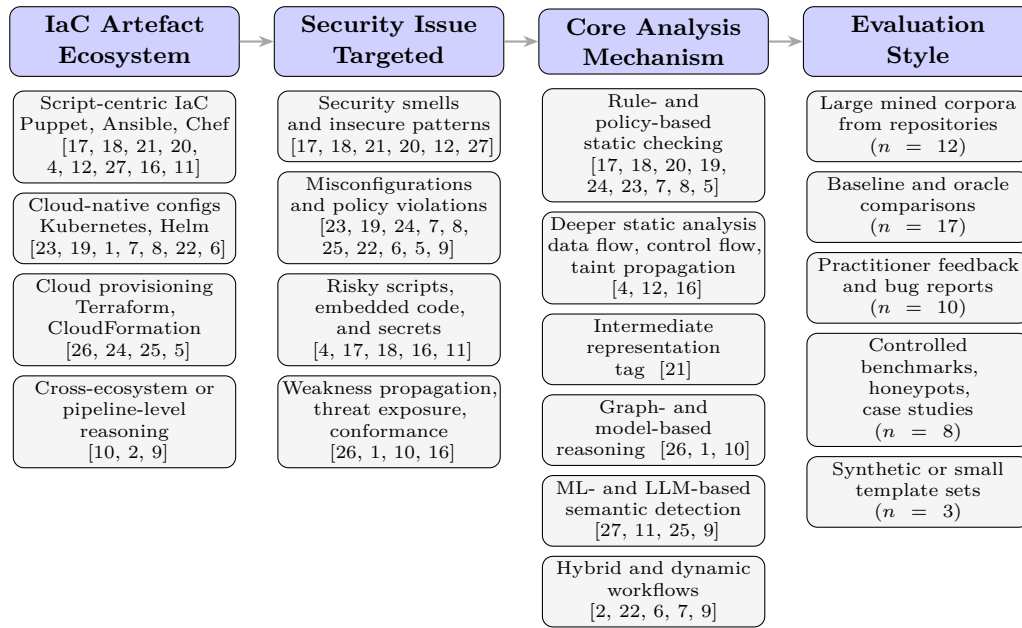
Table 1 makes this heterogeneity explicit. Baseline or oracle evidence is reported by 17 studies, but practitioner or maintainer evidence occurs in 10 and runtime, resource, latency, or cost evidence in 11. Accuracy-oriented metrics are common, but their denominators and ground truths differ. Consequently, the table summarizes evaluation designs rather than pooling reported precision or recall values. The study-level audit, mapping, and operational definitions are included in the replication package.

Comparability is particularly limited across ecosystems and evaluation settings. In studies of Kubernetes and Helm analysers, tools targeting similar artefacts may implement different policies and produce different outputs [8, 7]. Deployment studies introduce a further distinction by observing IaC security through running systems rather than through direct artefact inspection [22, 6]. A similar issue arises in studies of Terraform and generated IaC: one paper may benchmark scanners on intentionally vulnerable infrastructures, whereas another uses them as measurement instruments to evaluate LLM-generated configurations [24, 5]. Cross-study comparisons may therefore reflect differences in artefacts, policies, and observation points as much as differences in detector effectiveness.

The realism of the evaluations also varies. Some studies rely on large repositories of publicly available IaC artefacts [21, 18, 19, 16]. Others rely on smaller case studies, benchmark suites, intentionally vulnerable artefacts, deployment-specific observations, or synthetic templates [7, 24, 25, 2, 22, 6, 5, 9]. The benchmarked settings themselves also differ: some studies use curated honeypot or challenge environments such as TerraGoat, others use real Helm-chart repositories, and others use synthetic IaC cases or LLM-generated templates. Such evaluations are useful for controlled experimentation, but they do not necessarily reflect the complexity of industrial deployment pipelines or organization specific policies.

The main conclusion for RQ3 is twofold. First, the area is empirically active: most papers do not stop at proposing a detector, but also attempt to evaluate it. Second, the reported evaluations remain uneven in scope, rigour, and realism. This limits cumulative knowledge building and makes it difficult to compare approaches fairly across ecosystems and detector families.

For practitioners, the current evidence supports deterministic rule-based scanners as a first-line CI check, with data-flow or graph analyses added when security depends on indirect resource interactions. Deployment checks complement rather than replace source analysis. LLM-based detectors should presently augment a deterministic baseline and human review rather than act as the sole deployment gate.



■ **Figure 2** Graphical taxonomy of the reviewed literature along four complementary dimensions.

4 Proposed Taxonomy

Based on the results of Section 3, we propose a four-dimensional taxonomy. The goal is to classify the studies according to the targeted IaC artefacts, the security issues they consider, the technical core of the detection approach, and the way it is evaluated. Categories are non-exclusive when a study combines mechanisms or evaluation designs. Study role – for example, proposing a detector or comparing existing tools – is recorded separately.

Figure 2 summarizes the taxonomy. The first two dimensions characterize what is analysed and which types of security issues are targeted. The last two dimensions characterize how detection is performed and how the resulting tools or methods are evaluated. This organization makes it possible to keep deployment-oriented and runtime-observation studies inside the same review map: such studies differ not only by where they observe the issue, but also by the mechanism they use and by the kind of evaluation they report.

Dimension 1: IaC artefact ecosystem

The first dimension distinguishes the artefact families targeted by the reviewed studies. A first family concerns script-centric IaC, mainly Puppet, Ansible, and Chef [17, 18, 21, 12, 16, 27, 11]. A second family concerns container orchestration such as Kubernetes manifests and Helm charts, including studies that observe their security effects after deployment [19, 23, 1, 8, 7, 22, 6]. A third family addresses cloud-provisioning, mainly Terraform and CloudFormation [26, 24, 25, 5]. Finally, a small group reasons across ecosystems rather than on a single IaC language [10, 2, 9].

Dimension 2: Security issue targeted by the detector

The second dimension concerns the type of security issue under consideration. A large group of studies targets security smells and insecure coding patterns [17, 18, 21, 12, 27]. Another group targets security misconfigurations or policy violations in manifests, charts, cloud templates, generated IaC, or deployed configurations traceable to IaC artefacts [19, 23, 8, 24, 25, 22, 6, 5, 9]. Other studies focus on risky embedded scripts, secrets, weakness propagation, or architectural and threat-exposure analyses [4, 11, 16, 1, 10, 26]. These issue classes are not distributed uniformly across ecosystems: smell-oriented work remains concentrated in configuration-management languages, whereas misconfiguration and policy-compliance studies dominate Kubernetes, Helm, Terraform, and the newer LLM-related papers.

Dimension 3: Core analysis mechanism

The third dimension records the reasoning mechanism rather than the study’s contribution type. It distinguishes rule- and policy-based static checking [17, 18, 19, 5], deeper static analysis based on control flow, data flow, or taint propagation [12, 16], graph- and model-based reasoning [1, 10, 26], ML- and LLM-based semantic detection [27, 11, 25, 9], and hybrid or dynamic workflows [2, 22, 6]. Intermediate representations are coded as a supplementary implementation tag because they support, rather than replace, a reasoning mechanism [21]. Comparative papers are classified by the mechanisms of the analysers they evaluate and separately identified as comparison studies.

Dimension 4: Evaluation

The fourth dimension captures how a tool or method is evaluated. Some papers use large corpora mined from open-source repositories and report classical detection metrics [21, 18, 19, 16]. Others emphasize baseline or oracle comparisons, practitioner feedback, or accepted bug reports [17, 20]. A further group relies on controlled benchmarks, honeypots, case studies, deployment-oriented observations, or synthetic templates [7, 24, 25, 2, 22, 6, 5, 9]. This dimension matters because technically similar detectors may rely on very different evaluation settings. This dimension therefore cautions against comparing studies solely through their reported metrics: e.g. even evaluations of Kubernetes security tools may differ substantially in dataset construction, oracle definition, observation point, and validation procedure.

Cross-dimensional synthesis

This taxonomy shows that the reviewed literature is diverse, but not uniformly so. The field remains concentrated on a few ecosystems and on a few dominant evaluation styles. The cloud-native part of the corpus is methodologically the most diverse because it spans manifest-level checking, topology- or conformance-oriented reasoning, and deployment-oriented observation. By contrast, other ecosystems are represented by fewer technical families. The taxonomy also shows that the recent diversification of detection mechanisms is not yet matched by equally mature empirical support across all families of approaches.

5 Challenges and Perspectives

The evaluation heterogeneity identified in RQ3 makes the development of a shared benchmark a central challenge for IaC security detection. A useful benchmark should include versioned, appropriately licensed artefacts and combine vulnerable and secure examples from real-world

projects and controlled cases. For each labelled issue, it should identify the location, the relevant policy or weakness class, and whether the issue is observable in source artefacts or only after deployment. It should also control for duplicates and data leakage. These properties would support reproducible evaluations.

A second, closely related challenge is establishing a minimum reporting checklist. Studies should report dataset provenance and selection criteria, exact tool and ruleset versions, the observation level, oracle construction, baselines, per-class metrics, analysis failures, procedures for validating false positives and false negatives, runtime and cost, and links to publicly available replication artefacts. The Kubernetes and Helm comparisons show why this is necessary: tools targeting similar artefacts may implement different policy semantics, check different resources, and produce outputs that are not directly comparable [8, 7].

A third challenge is evaluation realism. Public repositories, benchmark datasets, and intentionally vulnerable artefacts are useful for controlled experimentation, but they do not always reflect the operational constraints of industrial IaC pipelines, organization-specific policies, or multi-tool development workflows. A promising direction is therefore to complement public datasets with industrial case studies, longitudinal observations, or deployment-oriented evaluations. The deployment and CI/CD studies illustrate both the value of these settings and the current lack of common evaluation designs [22, 6, 2, 9].

A fourth challenge concerns ML- and LLM-based detectors [27, 11, 25, 9]. Their evaluation should, at minimum, include an appropriate deterministic baseline where one exists, record exact model versions and prompts, repeat runs to quantify variability, report latency and cost, and independently validate a sample of findings. Releasing prompts and retrieval sources would support reproducibility, while evaluations across prompt variants and over time would help quantify prompt sensitivity and model drift. Until such evidence accumulates, hybrid use with deterministic checks is more defensible than treating an LLM as a sole security gate.

6 Threats to Validity

This review is subject to several threats to validity. First, validity is limited by the operational definition of IaC security detection, since the boundary between detection, prevention, policy enforcement, and compliance checking is not always sharp. To reduce this ambiguity, the review restricts inclusion to studies whose primary contribution concerns detection. Second, selection bias may arise from the restriction to English-language studies with accessible full text, as well as from the chosen publication window. These decisions improve feasibility and reproducibility, but they may exclude relevant studies.

A further threat to completeness comes from the search configuration itself. The automated search was restricted to ACM Digital Library and IEEEExplore, and the retrieval query relied on title and abstract metadata rather than full-text search. This design keeps the review protocol focused and auditable, but it may miss relevant studies that are indexed in other databases, that use different terminology, or that describe their security contribution outside the queried fields. For this reason, the findings should be interpreted as a focused review of the currently retrieved corpus, not as a claim that every relevant IaC security detection study has been captured. Third, publication bias remains a concern because peer-reviewed literature tends to over-represent positive or mature results, while underrepresenting negative findings, industrial tools, and unpublished systems used in practice. Fourth, classification bias may affect the categorization of technical approaches and reported evaluations, especially when papers use inconsistent terminology or provide limited methodological detail.

Finally, external validity is limited because the review reflects the state of reported research rather than actual industrial adoption or operational effectiveness of IaC security tools. Consequently, the findings should be interpreted as an overview of the reported research landscape, rather than as a complete representation of all tools used in practice.

7 Conclusion

This paper presented emerging results from a focused literature review of tools for detecting security issues in IaC artefacts. The reviewed studies show that the area is active and technically diverse. The literature includes rule-based detectors, deeper static analyses, graph- and model-based approaches, and more recent ML and LLM based techniques. It also spans several IaC ecosystems, especially Puppet, Ansible, Kubernetes, Helm, and Terraform.

Despite this technical breadth, heterogeneous datasets, oracles, and evaluation settings limit direct comparison and cumulative evidence. Shared benchmarks and consistent reporting are therefore practical priorities for strengthening the evidence base.

These findings should be interpreted in light of the review’s focused scope: they characterize the corpus of IaC security *detection* studies retrieved through our protocol, rather than all IaC security research.

Future work includes extending the search protocol to additional digital libraries and complementing automated search with backward and forward snowballing. The present review can serve as a basis for separate or explicitly broadened reviews on adjacent topics such as repair, prevention, or runtime enforcement.

Data Availability Statement

A public archival artefact is available at: <https://zenodo.org/10.5281/zenodo.21820732>. The repository contains the ACM and IEEE BibTeX exports, screening decisions for the title, abstract, and full-text phases, phase-count summaries, and metadata and extraction tables for the 23 included studies. It also contains the full-text audit and derived category mapping used for the quantitative evaluation synthesis.

References

- 1 Agathe Blaise and Filippo Rebecchi. Stay at the Helm: Secure Kubernetes deployments via graph generation and attack reconstruction. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 59–69. IEEE, 2022. doi:10.1109/CLOUD55607.2022.00022.
- 2 Matija Cankar, Nenad Petrovic, Joao Pita Costa, Ales Cernivec, Jan Antic, Tomaz Martincic, and Dejan Stepec. Security in DevSecOps: Applying tools and machine learning to verification and monitoring steps. In *ACM/SPEC International Conference on Performance Engineering Companion (ICPE Companion)*, pages 201–205. ACM, 2023. doi:10.1145/3578245.3584943.
- 3 Michele Chiari, Michele De Pascalis, and Matteo Pradella. Static analysis of infrastructure as code: A survey. In *19th IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2022. doi:10.1109/ICSA-C54293.2022.00049.
- 4 Ting Dai, Alexei Karve, Grzegorz Koper, and Sai Zeng. Automatically detecting risky scripts in infrastructure code. In *11th ACM Symposium on Cloud Computing (SoCC)*, pages 358–371. ACM, 2020. doi:10.1145/3419111.3421303.
- 5 Ryo Hase, Ye Wang, Toshiaki Koike-Akino, Jing Liu, Kieran Parsons, and Jumpei Hato. Evaluating security policy compliance in infrastructure as code generated by large language models. In *International Symposium on Digital Forensics and Security (ISDFS)*, pages 1–6. IEEE, 2026. doi:10.1109/ISDFS69419.2026.11458930.

- 6 Kacper Kamieniarz and Wojciech Mazurczyk. A comparative study on the security of Kubernetes deployments. In *International Wireless Communications and Mobile Computing (IWCMC)*, pages 718–723. IEEE, 2024. doi:10.1109/IWCMC61514.2024.10592468.
- 7 Ioanna Angeliki Kapetanidou, Alexandros Nizamis, and Konstantinos Votis. An evaluation of commonly used Kubernetes security scanning tools. In *2nd International Workshop on MetaOS for the Cloud-Edge-IoT Continuum (MECC)*, pages 20–25. ACM, 2025. doi:10.1145/3721889.3721924.
- 8 Francesco Minna, Agathe Blaise, Katja Tuma, and Fabio Massacci. Automated analysis of security policy violations in Helm charts. *IEEE Trans. Dependable Secure Comput.*, 23(2):2519–2533, 2026. doi:10.1109/TDSC.2025.3628213.
- 9 Akshay Mittal and Vivek Venkatesan. Practical integration of large language models into enterprise CI/CD pipelines for security policy validation: An industry-focused evaluation. In *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 197–203. IEEE, 2025. doi:10.1109/SOSE67019.2025.00027.
- 10 Evangelos Ntontos, Uwe Zdun, Ghareeb Falazi, Uwe Breitenbücher, and Frank Leymann. Assessing architecture conformance to security-related practices in infrastructure as code based deployments. In *IEEE International Conference on Services Computing (SCC)*, pages 123–133. IEEE, 2022. doi:10.1109/SCC55611.2022.00029.
- 11 Ruben Opdebeeck, Valeria Pontillo, Camilo Velázquez-Rodríguez, Wolfgang De Meuter, and Coen De Roover. Smelling secrets: Leveraging machine learning and language models for sensitive parameter detection in Ansible security analysis. In *IEEE International Conference on Source Code Analysis and Manipulation (SCAM)*, pages 66–77. IEEE, 2025. doi:10.1109/SCAM67354.2025.00014.
- 12 Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. Control and data flow in security smell detection for infrastructure as code: Is it worth the effort? In *IEEE/ACM International Conference on Mining Software Repositories (MSR)*, pages 534–545. IEEE, 2023. doi:10.1109/MSR59073.2023.00079.
- 13 Claus Pahl, Övgün Can Sezen, and Florian Hofer. Artificial intelligence for infrastructure-as-code—a systematic literature review. *Electronics*, 15(4), 2026. doi:10.3390/ELECTRONICS15040755.
- 14 Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Inf. Softw. Technol.*, 64:1–18, 2015. doi:10.1016/J.INFSOF.2015.03.007.
- 15 Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. A systematic mapping study of infrastructure as code research. *Inf. Softw. Technol.*, 108:65–77, 2019. doi:10.1016/J.INFSOF.2018.12.004.
- 16 Akond Rahman and Chris Parnin. Detecting and characterizing propagation of security weaknesses in Puppet-based infrastructure management. *IEEE Trans. Softw. Eng.*, 49(6):3536–3553, 2023. doi:10.1109/TSE.2023.3265962.
- 17 Akond Rahman, Chris Parnin, and Laurie Williams. The seven sins: Security smells in infrastructure as code scripts. In *41st International Conference on Software Engineering (ICSE)*, pages 164–175. IEEE, 2019. doi:10.1109/ICSE.2019.00033.
- 18 Akond Rahman, Md Rayhanur Rahman, Chris Parnin, and Laurie Williams. Security smells in Ansible and Chef scripts: A replication study. *ACM Trans. Softw. Eng. Methodol.*, 30(1), 2021. doi:10.1145/3408897.
- 19 Akond Rahman, Shazibul Islam Shamim, Dibyendu Brinto Bose, and Rahul Pandita. Security misconfigurations in open source Kubernetes manifests: An empirical study. *ACM Trans. Softw. Eng. Methodol.*, 32(4), 2023. doi:10.1145/3579639.
- 20 Sofia Reis, Rui Abreu, Marcelo d’Amorim, and Daniel Fortunato. Leveraging practitioners’ feedback to improve a security linter. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 2023. doi:10.1145/3551349.3560419.

- 21 Nuno Saavedra and João F. Ferreira. GLITCH: Automated polyglot security smell detection in infrastructure as code. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 2023. doi:10.1145/3551349.3556945.
- 22 Shazibul Islam Shamim, Hanyang Hu, and Akond Rahman. Dynamic application security testing for Kubernetes deployment: An experience report from industry. In *ACM International Conference on the Foundations of Software Engineering Companion (FSE Companion)*, pages 514–519. ACM, 2025. doi:10.1145/3696630.3728573.
- 23 Shazibul Islam Shamim, Hanyang Hu, and Akond Rahman. On prescription or off prescription? an empirical study of community-prescribed security configurations for Kubernetes. In *47th IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 2432–2444. IEEE, 2025. doi:10.1109/ICSE55347.2025.00170.
- 24 Rashika Singh, Abel Yeboah-Ofori, Saurabh Kumar, and Aishat Ganiyu. Fortifying cloud DevSecOps security using Terraform infrastructure as code analysis tools. In *International Conference on Electrical and Computer Engineering Researches (ICECER)*, pages 1–6. IEEE, 2024. doi:10.1109/ICECER62944.2024.10920371.
- 25 Dheer Toprani and Vijay K. Madiseti. LLM agentic workflow for automated vulnerability detection and remediation in infrastructure-as-code. *IEEE Access*, 13:69175–69181, 2025. doi:10.1109/ACCESS.2025.3560911.
- 26 Anh-Duy Tran, Laurens Sion, Koen Yskout, and Wouter Joosen. TerrARA: Automated security threat modeling for infrastructure as code. In *15th ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 269–280. ACM, 2025. doi:10.1145/3714393.3726494.
- 27 Gabriele De Vito, Fabio Palomba, and Filomena Ferrucci. SecLLM: Enhancing security smell detection in IaC with large language models. *IEEE Access*, 13:204480–204498, 2025. doi:10.1109/ACCESS.2025.3637505.