

Evaluating CI/CD Security Best Practices in the GitHub Actions Documentation

Lukas Boschanski ✉ 🏠 

University of North Carolina at Charlotte, NC, USA

Marco Vieira ✉ 🏠 

University of North Carolina at Charlotte, NC, USA

Abstract

Background. GitHub Actions, the most widely used Continuous Integration and Continuous Deployment (CI/CD) platform, is frequently involved in large-scale software supply chain attacks. While prior work has focused on detecting vulnerable CI/CD pipelines, research has rarely considered preventive perspectives such as analyzing Security Best Practices (SBPs) in CI/CD documentation.

Aims. We investigate the completeness of the official GitHub Actions documentation from a security perspective and identify potential improvements to documenting secure CI/CD practices.

Method. We employ an Large Language Model (LLM)-based documentation mining pipeline to extract security advice items verbatim. We derive actionable CI/CD SBPs from the OWASP Top 10 CI/CD Security Risks framework and conduct a qualitative analysis by mapping the extracted advice items against these best practices to assess which best practices are explicitly covered.

Results. Across 639 documentation pages, we identify 459 security-related advice items, of which 288 are actionable, but only 50% of them map to actionable SBPs. These items primarily address insecure configurations and insufficient credential hygiene. Furthermore, only half of all security advice visual alerts have an adequate warning type.

Conclusions. The GitHub Actions documentation lacks a consistent methodology for incorporating SBPs. Moreover, coverage of established CI/CD security best practices is uneven across OWASP risk categories, with several categories receiving little to no actionable guidance.

2012 ACM Subject Classification Software and its engineering → Documentation

Keywords and phrases CI/CD security, Software documentation, Security best practices

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.15

Category Technical Track Paper

Supplementary Material *Software (Source Code)*: <https://github.com/bluuuk/gh-cicd-sbp-docs> [9]

1 Introduction

In the modern software supply chain, CI/CD pipelines have become essential “highways to production” [21], where changes to the Source Version Control (SVC) directly affect production releases and consequently reduce the window for security intervention. Due to its automated nature, CI/CD is prone to specific vulnerabilities, which the *OWASP Top 10 CI/CD Security Risks* highlights [21]. As such, CI/CD is an increasingly critical component of Software Supply Chain Security (SSCS) [42].

Recurring attacks on the most widely used CI/CD platform [6], *GitHub Actions (GHA)*, such as *tj-actions* [13], *singularity* [26], *Trivy* [10], and most recently *TanStack* [24], have exposed systemic software supply chain risks and prompted GitHub to propose a security roadmap aimed at reducing the attack surface [30]. These incidents have had widespread downstream impact, with *tj-actions* affecting over 200,000 repositories [13] and



© Lukas Boschanski and Marco Vieira;

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. 15; pp. 15:1–15:20



Leibniz International Proceedings in Informatics

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

Trivy disrupting approximately 10,000 CI/CD pipelines [10]. In GHA, *workflows* are YAML-based CI/CD pipelines that integrate first- and third-party *Actions*; throughout this paper, we collectively refer to both as *plugins*.

These attacks show substantial impact on the software ecosystem despite years of prior research on the *detection* of vulnerable workflows [20, 27, 8, 31, 23]. Preventive perspectives remain largely unexplored; a notable exception is Vassallo et al., who built a linter to discover configuration smells for GitLab Continuous Deployment (CD) pipelines [41]. Meanwhile, recent studies show that developers lack awareness and understanding of security practices for GHA [22] and face significant challenges when adopting platform security features to secure Open Source Software (OSS) repositories [3]. This points to a broader gap: little attention has been paid to whether CI/CD platforms such as GHA effectively *educate* developers to promote secure behavior in the first place.

The consequences of this gap are tangible: in the `tj-actions` attack, developers of the initially compromised `SpotBugs/sonar-findbugs` plugin combined the overprivileged pipeline trigger `pull_request_target` with a legacy *Personal Access Token* that had overly broad access, leading to credential exfiltration when creating a malicious pull request. GitHub warns developers through the documentation using visible alerts (e.g., Figure 1) or textual security advice about unsafe practices, yet such warnings are effective only if developers consult and correctly interpret them. However, the official plugin to check out repositories, `actions/checkout`, provides no indication that its use in conjunction with the trigger introduces a security risk.

We address this gap by assessing how established CI/CD SBPs are covered by the GHA documentation and how effectively they are communicated. This is motivated by two findings: 95% of GHA developers rely on official documentation for configuration guidance [22], and developers who consult official documentation are more likely to write secure code [1] – together suggesting that documentation gaps translate into insecure practices. Therefore, the work is guided by the following research questions:

- **RQ1** What are the characteristics of security advice items in the GHA documentation, and to what extent are they actionable?
- **RQ2** What actionable CI/CD SBPs can be identified from established industry guidance?
- **RQ3** How complete is the GHA documentation with respect to CI/CD SBPs, and how prominently are they communicated?

In contrast to related work, we systematically collect and evaluate SBPs across the full CI/CD-related documentation. Kubo et al. evaluate GHA repository security using a subset of *observable* SBPs only from the *secure use reference* page, which omits practices such as *immutable releases* [22], while Huang et al. examine only four CI/CD security issues, conflating vulnerability types with practices due to reliance on existing tools [19, 8]. Our empirical study employs an LLM-driven documentation-mining approach to extract security advice and assess how completely the GHA CI/CD documentation advocates SBPs. Technically, we tackle the largely overlooked problem of automatically extracting security advice from large, complex documentation, whereas prior security advice coding research has focused on classifying pre-segmented items [4, 37, 5]. The few-shot LLM-approach extracts documentation-grounded advice verbatim under semantic and syntactic constraints, validated against a manually labeled ground truth; we derive these extraction criteria by combining inductive documentation analysis with a deductive review of the SBP and security-advice literature (**RQ1**). To identify established CI/CD SBPs, we apply the *SAcoding* methodology [4] to the OWASP Top 10 CI/CD Security Risks [21], a consensus-based set

of recommendations from real-world attacks (**RQ2**). We then map the extracted advice to these SBPs and qualitatively analyze its presentation in the documentation (**RQ3**). Our primary contributions are as follows:

- An actionability assessment of the OWASP Top 10 CI/CD security risks via the SAcoding methodology, yielding a curated set of CI/CD security best practices.
- A set of semantic requirements and syntactic constraints for security advice in software documentation, derived from a manual analysis of the GHA documentation.
- A few-shot LLM-based extraction pipeline for security advice grounded in source documentation, validated against a manually labeled dataset.
- An empirical evaluation of the completeness of the GHA documentation with respect to established CI/CD security best practices, including an analysis of how prominently security advice is communicated through visual indicators.

The remainder of this paper is structured as follows. Section 2 discusses related work. Section 3 introduces security advice theory and derives extraction requirements. Section 4 identifies established CI/CD SBPs from the OWASP Top 10 CI/CD Security Risks. Section 5 presents the LLM-based extraction pipeline. Section 6 reports the empirical analysis of the GHA documentation. Section 7 discusses threats to validity, and Section 8 concludes.

2 Related Work

Security advice, whether in the form of best practices, guidelines, patterns, or recommendations, is commonly understood but subject to varying interpretations [16, 4]. Barrera et al. scrutinized the term *security best practices* from a semantic perspective and developed the SAcoding methodology to classify security advice items [4, 5], which was applied to *insider threats* [37] and the *Internet of Things* [4, 7].

Regarding security advice, developers relying on official documentation rather than *Stack Overflow* produce more secure code [1]. However, documentation often fails to reflect practical needs [2] or overwhelms developers with information [3]. User-friendly OSS documentation should contain SBPs and link to practical resources like sample projects [3]. Since developers tend to skim, integrating explicit security examples improves adoption [17]. Ultimately, high-quality documentation must be well-structured, readable, unambiguous, and correct [39], yet authors of security advice only update documentation after major incidents [29]. Kubo et al. evaluate developer adherence to 5 of the 16 SBPs from the documentation page *secure use reference* and conclude, in a mixed-methods study, that developers are unaware of these practices, lack understanding of them, and prioritize functionality over security [22]. Huang et al. observe, in a longitudinal study of four security issues, that two vulnerabilities remain while two misconfigurations decrease [19]. Large-scale measurements reveal threats caused by unverified plugins and delayed plugin updates within workflows [31].

CI/CD systems share common security problems such as token leakage [18, 23], unsafe plugin distribution and isolation [23, 20], and unrestricted secret access [23, 20], regardless of the platform provider. Research shows that GHA workflows are overprivileged [19, 20], an issue that Tystahl et al. address with a static analysis tool to determine least-privilege permission policies [40]. In addition, *code injection vulnerabilities* in workflows are still an ongoing problem [22, 19] although state-of-the-art taint analysis tools exist [27]. Recent efforts leverage LLMs to generate GHA workflows; however, these efforts struggle to generate valid workflows while placing limited emphasis on security [43]. Conversely, LLMs have demonstrated strong performance in extracting structured knowledge, such as finite-state machines, from large technical documents such as the *Matter* specification [25].

■ **Table 1** Scholarly definitions for security advice in relation to proposed quality metrics.

Author	Definition	Properties
Redmiles et al.	Implicit: Advice imperatives that feature an action [35]. In the context of security and privacy advice for end-users found in various types of documents and evaluated in terms of human perception.	• Perceived Actionability • Perceived Efficacy • Comprehensibility
Barrera et al.	Explicit: Define <i>actionable</i> security advice as advice items that pass Question 5. Additionally, they define a <i>practice</i> as “a specific means intended to achieve a given desired outcome”; it is a <i>best practice</i> when it is “considered to be better than, or at least as “good”, as the best of other widely-considered means to achieve that same outcome” [4, 5].	<i>SAcoding</i> decision-tree questions 1) Unambiguous language & focus 2) Security related 3) Outcome vs. Action 4) Specific technique/rule 5) Actionability 6) Resource feasibility 7)–8) Target audience 9)–11) Abstraction level
Reeder et al.	Implicit: General actionable recommendations for the broad public that security professionals would give to non-tech-savvy users [36].	• Effective to improve security • Actionable • Consistent • Concise
Quinlan et al.	Explicit: “A written instruction, provided by a trusted and professional source, with the explicit goal of enabling the recipient to be more secure once they execute the instruction.” In the context of security advice in news articles [32].	Implied metrics: • Trustworthiness • Commonality • Quality • Actionable by instructions

3 Security Advice in Software Documentation

This section establishes the foundation for security advice in software documentation. We first review existing definitions and derive a working definition for this paper (Section 3.1), then characterize the structure of the GHA documentation (Section 3.2), and finally present a manual analysis of a subset of GHA documents (Section 3.3) to derive the requirements and constraints used in the automated extraction pipeline (Section 3.4).

3.1 General Security Advice

Definitions for security advice vary across the literature, each emphasizing different characteristics and metrics depending on the target environment, as we summarize in Table 1. For the rest of this paper, we use the term *security advice* as a catch-all to include all related terminology [4]. As shown in Table 1, most scholarly work focuses on the end-user [34, 36, 32], while Barrera et al. propose a general-purpose definition that adapts to different audiences such as end-users, IT-specialists, and security experts [4].

A key property of security advice is *actionability*, which reflects whether the advice includes actionable steps the audience can follow to implement depending on the practitioner’s knowledge [34]. However, the steps may be indirect or *declarative*, referring to steps that the practitioner must infer from the security advice’s outcome or goal [4]. Such security advice is harder to comprehend [35, 4]. Other considerations for actionability concern implementation feasibility in terms of resources [4, 34]. For example, the first advice in Figure 2 implies input *sanitization* or *validation* while enumerating the input data sources, without stating it.

Each advice item provides a security outcome [4, 34, 36, 32], either stated explicitly (e.g., “Input sanitization prevents code injection”) or implied by the action (e.g., “Always sanitize user inputs”). Barrera et al. conclude a semantic **Action** $\Rightarrow$ **Outcome** structure [4]. Apart from advice semantics, practitioners’ perception of advice quality includes factors such as consistent writing (e.g., always using the affirmative [36]), clear and unambiguous language [4], high comprehensibility [34], and nearby examples [17]. In addition, the number of advice items should be kept as small as possible while remaining highly relevant [36].

Warning

Running untrusted code on the `pull_request_target` trigger may lead to security vulnerabilities. These vulnerabilities include cache poisoning and granting unintended access to write privileges or secrets. For more information, see [Secure use reference](#) in the GitHub Enterprise Cloud documentation, and [Preventing pwn requests](#) on the GitHub Security Lab website.

Figure 1 Visual alert of type `warning` taken from <https://docs.github.com/en/actions/reference/workflows-and-actions/events-that-trigger-workflows> on 12 May 2026.

Together, these properties – particularly actionability, security relevance, and the `Action` $\Rightarrow$ `Outcome` structure – directly inform the requirements we derive in Section 3.4 for extracting security advice from the GHA documentation.

3.2 Structure of the GHA Documentation

The GitHub documentation (as of 19 February 2026) consists of approximately 2853 pages organized hierarchically into 11 main topics, covering collaborative coding, GitHub features, API endpoints, CI/CD, and more. It is a public Git repository that accepts community contributions but is ultimately managed by GitHub. All documents are written in the lightweight markup language *Markdown* and converted to HTML when served via the web. Furthermore, GitHub establishes writing best practices that reflect high-quality documentation, such as readability, structure, clear and accessible language, and formatting [39].

GitHub mainly follows the *Diátaxis* documentation framework [14], organizing documents into four content-types by purpose and audience. *Conceptual* (corresponding to *Explanation*) documents introduce ideas that are further explored in *Tutorials*, *Quickstart* guides, and *How-tos*, while *Reference* documents provide additional technical details. In practice, some topics introduce additional or customized content-types beyond the standard categories.

CI/CD topics are *Actions*, *Packages*, and *Security and code quality*, which we refer to as the GHA documentation for the rest of the paper. *Security and code quality* has four further sub-topics, though these do not fully capture the domain’s breadth. Those topics account for about 22% of the total documentation, as shown in Table 2.

The *Actions* topic contains the highest number of *warning* alerts, as illustrated in Figure 1. *Markdown callout boxes*, e.g., `> \[!WARNING]` for a warning, indicate visual alerts. Table 2 lists the alert types in order of urgency; GitHub’s style guide associates *Warning* and *Caution* with high risk. Although *Caution* explicitly considers security risk, it is rarely used within the documentation, appearing only five times in the unrelated topics *Admin* (3), *Copilot* (1), and *Repositories* (1). The other alerts are informational, with *Note* being predominant.

Figure 2 shows an excerpt from a conceptual document from the *Actions* topic about script injections. Although the document promises to mitigate script injection risks, it only enumerates possible attack payloads in a single subsequent section. The *Secure use reference* is the only document that explains injection attack mitigations. It is also the most frequently referenced document, referenced 28 times within the GHA documentation and twice outside the selected topics, making it a hub for security advice.

This structural overview motivates the manual analysis: the scale, heterogeneity of content types, and use of alerts all influence what constitutes a security advice item and where such items are likely to appear, as evidenced by the *Secure use reference* emerging as the document with the highest concentration of security advice in our extraction (Section 6).

3.3 Manual Analysis of GHA Documents

From these 639 documents, we manually analyze 41 for security advice items to establish requirements that reflect scholarly consensus and showcase the challenges of extracting advice from documents. We ensure coverage of the GHA documentation by sampling documents across different content-types and topics. Prior work on security advice coding assumes advice is available in a digestible format, treating extraction as a decision problem rather than an extraction problem [4, 5, 37].

In an initial review, we identified 30 security advice items that (a) are clear and unambiguous, (b) describe or imply actionable steps, and (c) indicate a security-relevant outcome or goal. Such broad criteria are sufficient for a first analysis, allowing interpretive flexibility while respecting established security advice properties from Table 1. Furthermore, excluding advice items early affects the comparison against SBPs, as those practices might also lack the above properties, potentially understating the documentation’s coverage. In about 11 cases, security advice is entangled with specific product features (e.g., *Dependabot* or *Secret Scanning*), targeting that product’s ecosystem rather than offering agnostic guidance. Such features carry security outcomes when applied, for example: “Enable Dependabot alerts to be generated when a new vulnerable dependency is found in one of your repositories”. Hereafter, we use this case as our **running example**. In contrast, descriptions like “Custom patterns are user-defined patterns that you can use to identify secrets that are not detected by the default patterns supported by secret scanning” explain how to extend mechanisms without mandating action. While the former indicates urgency, the latter uses “you can” to signal that the feature is merely available rather than recommended.

Security mechanisms are also embedded within the documentation, e.g., “Artifact attestations enable you to increase the supply chain security of your builds by establishing where and how your software was built”, though less prominently than features. This example illustrates the fine line between description and advice, as it explains both what artifact attestation is and its security outcome. It is an instantiation of *declarative* advice where a practitioner may infer actions, such as generating a *Software Bill of Materials*. However, the text is presented as a feature description rather than a directive, lacking any signal verb or imperative to prompt the reader to act. This ambiguity between description and advice is a key challenge for automated extraction that our requirements in Section 3.4 address.

When creating workflows, custom actions, and composite actions, you should always consider whether your code might execute untrusted input from attackers. This can occur when an attacker adds malicious commands and scripts to a context. When your workflow runs, those strings might be interpreted as code which is then executed on the runner.

Attackers can add their own malicious content to the GitHub context, which should be treated as potentially untrusted input. These contexts typically end with `body`, `default_branch`, `email`, `head_ref`, `label`, `message`, `name`, `page_name`, `ref`, and `title`. For example: `github.event.issue.title`, or `github.event.pull_request.body`.

You should ensure that these values do not flow directly into workflows, actions, API calls, or anywhere else where they could be interpreted as executable code.

By adopting the same defensive programming posture you would use for any other privileged application code, you can help security harden your use of GitHub Actions. For information on some of the steps an attacker could take, see [Secure use reference](#).

In addition, there are other less obvious sources of potentially untrusted input, such as branch names and email addresses, which can be quite flexible in terms of their permitted content.

For example, `zzz";echo${IFS}"hello";#` would be a valid branch name and would be a possible attack vector for a target repository.

■ **Figure 2** Excerpt from the *Script injections* document from <https://docs.github.com/en/actions/concepts/security/script-injections> (11 March 2026). Each gray-colored text span represents a security advice item according to our requirements and constraints from Section 3.4.

■ **Table 2** Size of the GitHub documentation in files per topic and occurrences of the five alert types without *README.md* or *index.md* files from 11.02.2026.

Topic	Subtopic	Files	Avg. Words	Note	Tip	Important	Warning	Caution
actions		200	1385	296	4	2	56	0
packages		20	1371	42	1	0	3	0
code-sec.		419	1186	445	62	8	13	0
	secrets	16	953	28	1	0	0	0
	supply-chain	33	1096	42	3	0	1	0
	quality	4	616	7	1	0	0	0
	scale	40	785	47	7	2	6	0
Total CI/CD		639	1254	783	67	10	72	0
Total Documentation		2853	1660	2547	362	76	325	5

■ **Table 3** Number of practices *before* and *after* transforming practices into singular units and *SAcoding* results.

Category	Before	After	Actionable	Agreement
SEC-1	5	11	4	100.0%
SEC-2	9	19	11	100.0%
SEC-3	8	18	11	94.4%
SEC-4	6	11	7	100.0%
SEC-5	8	11	4	100.0%
SEC-6	8	15	8	73.3%
SEC-7	4	10	6	70.0%
SEC-8	4	9	2	55.6%
SEC-9	2	7	5	100.0%
SEC-10	4	6	3	100.0%
Total	58	117	61	89.7%

A strong security advice item is the visual alert in Figure 1. In contrast to other advice nudging the developer to use a feature, it shifts the focus to explicitly warning about it. All advice items were actionable, but to varying degrees. For example, the second advice in Figure 2 suggests adopting a “defensive programming posture” as an action to security-harden GHA. The outcome is clear, yet the action is unspecified because it does not further describe the posture and instead links to the *Secure use reference*. These observations directly motivate the requirements and constraints we derive in Section 3.4.

3.4 Security Advice Requirements and Constraints

We reformulate our initial requirements to address two objectives: (1) enabling automated extraction, and (2) collecting sufficiently many candidate advice items for subsequent in-depth analysis and quality assessment of the documentation.

Identifying security advice is subjective, as systematic coding of security advice via *SAcoding* yields varying levels of inter-coder agreement, with a mean Cohen’s κ of 0.67 (moderate) among three coders [4] and Krippendorff’s α of 0.92 (reliable) between two coders [37], depending on the documents analyzed [7] and coders’ experience [37, 4]. Yet advice reflects a broadly shared understanding, supporting flexible requirements that accommodate diverse perspectives, while still capturing items that most practitioners would recognize as security advice. Our requirements are:

- **Focused:** One distinct, specific mechanism or a tightly coupled set of mechanisms.
- **Clear:** Written in clear and unambiguous language.
- **Security Outcome:** States or signals a desired security goal or effect.
- **Prescriptive Actionability:** Imposes or recommends direct or indirect actions to be performed.

All requirements reflect the main properties of actionable security advice as proposed by Barrera et al., with one exception: we do not distinguish between *outcome* and *action*. Prior work operationalizes this by asking whether the advice emphasizes a desired end goal or the means to achieve it. However, empirical results show this distinction is difficult to apply consistently: coders frequently disagree when identifying outcome-oriented advice [5], as it

requires a more subjective judgment than other *SAcoding* attributes. Instead, we adopt the notion of *Security Outcome*, capturing whether a text segment expresses a security-relevant goal or effect, regardless of whether the means to achieve it are stated.

The *prescriptive* property addresses descriptive statements within actionability: a text segment that describes a feature without mandating its use for a security outcome, such as *custom patterns*, does not qualify as security advice, reducing relevant items from the first pass by 6. Regarding *clarity*, our sample contains no cases of ambiguous or unclear language, likely because GitHub enforces a style guide and the documentation is open-source with active community contributions. Ultimately, we reduce the number of relevant items from 30 to 20, with 30 documents containing zero advice items.

Now, we propose textual boundary constraints that are important for the extraction task:

- **Self-Containment:** The text must be understandable in isolation, remaining meaningful if encountered outside its original context (e.g., in a CI/CD pipeline guide). Contextual referents must be resolved.
- **Completeness:** One coherent and continuous block of text, not a text or sentence fragment.
- **Inclusion of Referential Material:** Examples and external links directly attached to the advice are included as part of it.

The first constraint is a consequence of the extraction task. We require context to understand the security advice so we can later map it to established SBPs. For example, the first security advice in Figure 2 refers to “these values” in its final paragraph, which requires the *GitHub context* from above. We therefore include it. Extractions are copied verbatim from the source to preserve the original wording and facilitate later quality assessment.

The completeness constraint imposes a continuous structure on security advice items. Our manual analysis revealed no instances of advice items that were split apart. Even though the first and last security advice items in Figure 2 both concern script injections, they address different input sources and thus constitute separate advice items. We argue that advice items spread across non-continuous text segments fail the *Completeness* constraint, as the reader must switch context while skimming the documentation.

Extracted advice items should include referential material, such as links to examples or external resources. Prior work shows that external references improve actionability [4]. Developers tend to rely on examples when skimming documentation [17]. For instance, the *Secure use reference* link in the second advice item in Figure 2 is a case where the referential material is integral to the advice: without it, the reader cannot follow up on the “defensive programming posture” that the advice recommends. However, examples alone do not constitute security advice, as they typically lack actionable steps and a security outcome.

Together, these requirements and constraints define the building blocks for the automated extraction described in Section 5 and answer **RQ1**. In addition, our **running example** satisfies those because it explicitly recommends to “enable Dependabot alerts” to implicitly achieve the security outcome of vulnerability awareness.

4 Identifying CI/CD Security Best Practices

In contrast to security advice, SBPs additionally encompass three properties [4]: *quality* reflects effectiveness compared to alternatives, *commonality* captures adoption in practice, and *stipulation* captures the degree of formal recommendation. For example, “branch protection rules on branches hosting production code” is a widely adopted SBP formally recommended by OWASP. We screen available guidelines, standards, and frameworks, as developers seek

guidelines that carry authority [2], and select the most suitable source for deriving CI/CD SBPs. From four candidates, we select the *OWASP Top 10 CI/CD Security Risks* and evaluate its actionability via the *SAcoding* methodology.

4.1 Selection of CI/CD Security Guidelines

For our initial screening, we targeted complete, current, free, English-language resources specifically detailing CI/CD security controls and SBPs for software developers. We excluded broader SSCS frameworks as industry practitioners already struggle to select relevant ones [38]. We restricted government resources to U.S. origins and excluded corporate or for-profit materials. To find platform-agnostic resources, we conducted a web search with the query `CI/CD AND Security AND (standard OR framework) and CI/CD security best practices`.

The *OWASP Top 10 CI/CD Risks* (2022) [21], categorized as a framework, offers the most extensive coverage with 58 identified practices. In comparison, *NIST SP 800-204D* (2024) [11], a formal standard, outlines 26 practices. The NSA and CISA-produced report *Defending CI/CD Environments* (2023) [28] contributes 19 practices, while the more recent *CI/CD Cybersecurity Guide* (2025) [12], also classified as a framework, presents 15 practices. Across these sources, we only count first-level bullet points or clearly defined sections representing individual practices. Notably, the *Center for Internet Security GitHub Benchmark* was excluded as it is not freely available due to non-commercial licensing restrictions.

NIST SP 800-204D and the *CI/CD Cybersecurity Guide* both build upon the Secure Software Development Framework (SSDF) as a foundational reference for secure software development practices. The former presents security measures, while the latter enumerates tooling for various situations. In contrast, the *Defending CI/CD Environments* report primarily addresses governmental entities and focuses on malicious threat actors to map their capabilities to the ATT&CK framework. Given these differences in scope, audience, and coverage, the *OWASP Top 10 CI/CD Security Risks* is the most suitable resource for our purposes, as we elaborate in Section 4.2.

4.2 OWASP Top 10 CI/CD Security Risks

The framework defines ten security risk categories, each described by adversary impact and applicable recommendations derived from real-world attacks on CI/CD systems (including those on GHA). It classifies itself as *documentation*, and its target audience comprises *builders*, *breakers*, and *defenders* [21]. We select this framework because it offers the broadest coverage of the four candidates, with 58 practices. Furthermore, it is referenced in the literature [27, 43] and incorporated in the *Defending CI/CD Environments* report. Its publication involved 17 industry security professionals, reflecting strong practitioner consensus.

The ten categories cover an interconnected attack lifecycle, motivating their retention as mapping targets in RQ3. Attackers typically gain initial access to the CI/CD ecosystem by exploiting *Insecure System Configurations* (SEC-7) or abusing authentication flaws via *Inadequate Identity and Access Management* (SEC-2). The attack surface further expands through *Ungoverned Usage of 3rd Party Services* (SEC-8), which often introduces overly permissive integrations such as GHA *plugins*. Alternatively, adversaries may gain an initial foothold via *Dependency Chain Abuse* (SEC-3), injecting malicious artifacts into build environments or developer workstations, mainly by propagating them via programming language package managers. *Poisoned Pipeline Execution* (SEC-4) risks arise from attacks on GHA that inject malicious code into SVC via forks, which the pipeline then consumes. This allows attackers to abuse *Insufficient Pipeline-Based Access Controls* (SEC-5) to move

laterally within or beyond the CI/CD environment, or to exploit *Insufficient Credential Hygiene* (SEC-6) to obtain secrets and move to other interconnected systems. Ultimately, *Insufficient Flow Control Mechanisms* (SEC-1) allow attackers to push unreviewed and malicious code or artifacts down the pipeline once inside. These malicious outputs successfully reach production because *Improper Artifact Integrity Validation* (SEC-9) fails to detect tampered resources. Throughout this attack lifecycle, defenders are often rendered blind by *Insufficient Logging and Visibility* (SEC-10) controls. Table 3 lists the number of best practices per security risk.

4.3 Actionability Assessment via SAcoding

We apply the *SAcoding* methodology adapted with the proposed recommendations by Barrera et al. [5] that was first used by Stewart et al. [37]. The coding uses a series of up to 11 binary questions (cf. Table 1) to traverse a decision tree. The leaf nodes of the tree correspond to “tags” that ultimately determine an advice item’s actionability. The first coder is a PhD student with a Master’s degree in Cybersecurity, and the second coder has 5 years of *DevOps* experience and a Master’s degree in Computer Science. Following the initial coding round, the coders, both working in English as an additional language, resolved all disagreements, resulting in 33 changes to coded items, primarily due to language-related issues in the OWASP framework and the coding questions.

As question Q_{1A} evaluates whether a security advice item is focused on a single topic, we first split top-level OWASP practices that contain multiple items into individual units. Table 3 summarizes the resulting increase from 58 to 117 practices, reflecting their dense structure. We split practices by dividing sentences at conjunctions, provided the split did not introduce a new practice. In 29 cases, we made changes to the sentence structure to resolve contextual references, e.g., *this*. The results support the validity of the strategy: both coders assigned the ambiguous language tag (M_{1B}) only once on the same item and did not assign the multiple-topics tag (M_{1A}) to any item. This step ensures that we do not discard any SBPs, thus prioritizing completeness over syntactic format issues.

Overall, 61 out of 117 practices (52.1%) are actionable, forming our set of actionable SBPs for **RQ2**. We retain the 56 non-actionable items in Table 3 for completeness in the mapping in Section 6, but their lack of prescriptive actionability makes alignment with the extracted GHA security advice unreliable. The overall inter-coder agreement of $\kappa = 89.7\%$ is almost perfect, consistent with prior *SAcoding* studies covered in Section 3.4. Agreement varies substantially across three categories: SEC-8 (55.6%), SEC-6 (73.3%), and SEC-10 (70.0%) yield the lowest agreement, as their practices blur the line between general policy, specific mechanism, and concrete rule.

This allows us to use the actionable SBPs as a validated reference when mapping extracted GHA security advice: an advice item that aligns with an actionable SBP inherits its actionability assessment. We use the same questions proposed by the modified version of Stewart et al. along with their tag selection to determine actionability [37]. The 61 actionable SBPs derived from the OWASP Top 10 CI/CD Security Risks serve as the mapping target in Section 6, where we assess how completely the GHA documentation covers them.

5 LLM-Based Security Advice Extraction

The overall goal of the automated extraction pipeline is to capture advice items that are highly likely to fit the definition of security advice. To this end, we propose a set of requirements and constraints in Section 3.4 to locate such advice, as the GHA documentation is written

from a general perspective and primarily focuses on product interaction. In this section, we explain how we mine and process the GHA documentation in Section 5.1, cover the metrics used for our task in Section 5.2, and describe the prompt engineering process in Section 5.3.

We reuse our sample of 41 documents and 20 security advice items as a test set to develop and fine-tune the LLM-based extraction pipeline. The 41 test documents are a subset of the 639 GHA documents; they are not held out from the final extraction run, as the goal of that run is full coverage rather than held-out evaluation. The test set is used exclusively for prompt development. From an evaluation standpoint, we prioritize high recall as the primary objective, followed by an acceptable match size that reflects our constraints. We use the open-source library *langextract*[15] that extracts structured information from text with precise or fuzzy source grounding to guarantee hallucination free extractions. A prompt and the required few-shot examples define the inputs for the extraction, which we derive in the subsequent subsections. Initial experiments with local models revealed significant resource constraints, most notably limited context windows and reduced inference throughput when processing the large volume and length of the documents. Therefore, we use models of the *Gemini* family, which support extended context windows of up to 1 million tokens. Hence, the generalizability of our prompt-engineering approach is limited to those models. A rough estimate shows that the longest GHA document with 18,477 words, one few-shot example, and the prompt consume at most $3 \cdot 18,477 \cdot 1.67 \leq 100,000$ tokens¹.

5.1 Data Preparation

Although the GitHub documentation is an open-source repository, it only contains templated Markdown files with placeholder variables. Therefore, we mine the documentation via a public GitHub API while using the repository’s folder and file structure, ensuring our dataset reflects what visitors of the official documentation see. We avoid using HTML representations, as they require extensive parsing for text extraction and are significantly more verbose than Markdown. Our mined corpus of Markdown files dates to *February 19, 2026*.

An analysis of our corpus reveals that Markdown files contain HTML elements. To reduce the impact on context windows and lower inference latency, we preprocess the documents by removing the visual HTML elements `<svg>` and `<path>` and comments. In addition, we redact Markdown code fences (````<lang> <text>````). Code fences without an explicitly specified language or those labeled as *text* are preserved, as they may contain relevant textual content. This follows the constraints in Section 3.4, as standalone examples are not considered security advice. The code block’s content is replaced with a unique marker, enabling post-hoc recovery of the original content. These steps reduce the overall file size by 50%.

5.2 Metrics

To evaluate the effectiveness of the predicted extractions against the ground truths, we use two sets of metrics. First, we compute *recall*, *precision*, and *F1* for extractions that overlap with at least one ground truth item. Second, we measure how well the size of each extraction matches the ground truth using *token-level* recall and precision [33].

An extraction is a continuous text span grounded in a source document, marked by start and end offsets at the character level. However, an LLM may fail to replicate a text span word by word; in such a case, *langextract* computes a fuzzy alignment using the longest common subsequence algorithm. Each extraction contains the fully extracted text span, which may differ slightly from the source, and its extraction status, i.e., *exact* or *fuzzy*.

¹ Google estimates 100 tokens per 60 words; see <https://ai.google.dev/gemini-api/docs/tokens>

Figure 4 shows all match types between ground-truth extractions (alphabetic) and predicted extractions (numeric). A fuzzy match example is ②, where ②* is outside the source document. Based on character offsets, we distinguish: **Subspan**, when the prediction is contained in the ground truth (e.g., ① in ④); **Superspan**, when the prediction contains the ground truth (e.g., ④ in ②); and **Exact**, when both coincide (e.g., ④ and ④). **Partial** matches are overlapping spans that are neither subspans nor superspans (e.g., ② and ④), while **Disjoint** predictions are false positives (FPs) (e.g., ③) and **Missed** cases are false negatives (FNs) (e.g., ⑤). Finally, a **Multimatch** occurs when a span overlaps multiple ground-truth items (e.g., ② and ④).

Let $\mathcal{P}$ be the set of predicted spans and $\mathcal{GT}$ the set of ground-truth spans for a given document. For each prediction $p \in \mathcal{P}$, we define the matching $\text{Type}_{\mathcal{GT}}(p)$ with the precedence of $\text{Exact} \succ \{\text{Subspan}, \text{Superspan}\} \succ \text{Partial} \succ \text{Disjoint}$. We then define true positives (TP), FPs, and FNs with $\mathbf{1}$ being the indicator function:

$$\begin{aligned} \text{TP} &= \sum_{p \in \mathcal{P}} \mathbf{1}(\text{Type}_{\mathcal{GT}}(p) \in \{\text{Subspan}, \text{Superspan}, \text{Partial}, \text{Exact}\}) \\ \text{FP} &= \sum_{p \in \mathcal{P}} \mathbf{1}(\text{Type}_{\mathcal{GT}}(p) = \text{Disjoint}) \quad \text{FN} = |\mathcal{GT}| - \text{TP} \end{aligned}$$

When we encounter a *multimatch* extraction, we count it only against the first matching ground truth item. Furthermore, we do not compute the number of true negatives (TN) because we cannot define the opposite of a security advice item with respect to extractions; therefore, we cannot compute accuracy but do compute precision, recall, and F1.

While binary metrics assess whether an item was detected at all, they do not capture how closely the predicted boundaries match the ground truth. For an extraction $p \in \mathcal{P}$ that matches $g \in \mathcal{GT}$, we compute the multisets of individual tokens, denoted as $T(p)$ and $T(g)$:

$$\text{Precision}_{\text{tok}}(p, g) = \frac{|T(p) \cap T(g)|}{|T(p)|} \quad \text{Recall}_{\text{tok}}(p, g) = \frac{|T(p) \cap T(g)|}{|T(g)|}$$

A high $\text{Precision}_{\text{tok}}(p, g)$ paired with a low $\text{Recall}_{\text{tok}}(p, g)$ implies the predicted extraction is shorter than the ground truth, and vice versa. Together, these metrics evaluate both detection rate and boundary precision of the extraction pipeline.

5.3 Prompt Engineering

We vary several parameters across five rounds: (1) model and thinking budget, (2) prompt persona, (3) explicit inclusion of the security advice requirements from Section 3.4, (4) targeted exclusion hints for edge cases identified via false-positive analysis, and (5) model settings and behavior. The one-shot example used throughout all experiments is Figure 2. Preliminary experiments highlight the difficulty of selecting few-shot examples for prompt engineering, as their addition or removal complicates attribution and increases fuzzy matches due to the expanded prompt length. Therefore, we fix one specific example to constrain the output format and iterate by adjusting the prompt. All experiments run with a temperature of zero [15] and a single extraction pass per document.

Table 4 shows all experiments alongside evaluated metrics. In the first experimental iteration, we examine model choice using all available Gemini variants with the identical prompt “Extract security advice verbatim” and constraints from Section 3.4, as Gemini models have no published parameter count. Results show that *Flash* variants produce substantially more FPs than *Pro*, while FNs remain consistently low across all models, indicating near-complete coverage of security advice. Furthermore, the match size appears near-optimal, as the high micro-averaged values $\overline{\text{Pre}}_{\text{tok}}$ and $\overline{\text{Rec}}_{\text{tok}}$ indicate.

Given the already high recall, we shift our focus to reducing FPs. We extend Experiment 1.1 and adopt a Markdown-formatted prompt incorporating a persona of a “cybersecurity documentation expert”. Experiment 2 demonstrates a reduction in FPs from 29 to 25. We investigate this by comparing FPs across both experiments to identify recurring cases, which reveals that the prompt does not adequately address *descriptive statements*, specifically: (1) *descriptions of tooling operations*, (2) *descriptions of threats without mitigation*, and (3) *policy statements*. Therefore, we add our requirements from Section 3.4 in Experiment 3.1 and, alternatively, extend the persona to exclude *prescriptive statements* in Experiment 3.2, both of which slightly improve precision. However, adding the identified issues as edge cases with *inclusion* and *exclusion* directions using generic examples, e.g., “How to install X” as an exclusion example, greatly reduces FPs to under 14.

Next, some extractions lack security relevance, such as “When you review a suggestion from Copilot Autofix, you must always consider the limitations of AI”, which is advice in terms of reliability. Based on Experiment 3.3, we compare two approaches: Experiment 4.1 enumerates *security-relevant* goals and terms, e.g., *security control*, to qualify as advice, while Experiment 4.2 establishes that advice is security-relevant if its primary association among multiple possible interpretations is security, yielding the best F1 score of 78.3. The rationale behind this approach emerged from a discussion between researchers with different backgrounds, e.g., code generation, who associated AI-related advice with functional correctness.

To mitigate the risk of overfitting and keep the prompt simple, we do not conduct further experiments. Comparing the best- and worst-performing models in terms of FPs, we achieve a relative reduction of $1 - \frac{8}{86} = 90\%$ while maintaining high recall.

Lastly, we study prompt-agnostic settings. While conducting reproducibility experiments, we observe fluctuations even with a fixed **seed**, which should result in highly deterministic outputs. Experiments 5.1.* show fluctuations in FPs and slight fluctuations in FNs. Hence, we require a manual evaluation of all extractions to assess whether they meet our requirements, which we conduct in Section 6. A potential explanation for the fluctuations may lie in rounding, scheduling, or live model updates, as all **gemini-3.1** models are still in preview. Experiment 5.2.1 uses two extraction passes with limited effect, as the implementation merges extractions via union, giving priority to the first pass. Conversely, using naive chunking, even though each document fits into the context window, increases FPs by nearly a factor of two.

■ **Table 4** Overview of the experiments and the metrics used. We compute Precision (Pre.) and Recall (Rec.) globally, while we evaluate the micro-average for Precision_{tok} and Recall_{tok}.

ID	Experiment	FP	Pre.	FN	Rec.	F1	Pre _{tok}	Rec _{tok}
1.1	Pro (Low thinking)	29	39.6	1	95.0	55.9	86.3	95.0
1.2	Pro (High thinking)	38	33.3	1	95.0	49.4	81.8	92.0
1.3	Flash (Minimal thinking)	86	17.3	2	90.0	29.0	79.0	88.5
1.4	Flash light (Minimal thinking)	73	18.9	3	85.0	30.9	87.0	82.2
2 via 1.1	Persona	25	43.2	1	95.0	59.4	87.8	85.5
3.1 via 2	Requirements	22	46.3	1	95.0	62.3	88.8	83.3
3.2 via 2	Prescriptive	23	43.9	2	90.0	59.0	87.5	89.6
3.3 via 3.1	Requirements + Edge Cases	13	56.7	3	85.0	68.0	91.3	84.6
3.4 via 3.1	Prescriptive + Edge Cases	12	60.0	2	90.0	72.0	87.1	79.6
4.1 via 3.3	Security relevance by terms	22	45.0	2	90.0	60.0	87.8	78.2
4.2 via 3.3	Primary Security Association	8	69.2	2	90.0	78.3	93.1	89.9
5.1.1 via 4.2	Repetition with fixed Seed	9	65.4	3	85.0	73.9	92.0	78.4
5.1.2 via 4.2	Repetition with fixed Seed	13	55.2	4	80.0	65.3	89.0	83.5
5.2.1 via 4.2	Two extraction passes	15	54.5	2	90.0	67.9	92.8	87.5
5.2.2 via 4.2	Chunking	33	34.0	3	85.0	48.6	96.0	88.1

All experiments after iteration 1 show no occurrences of multimatches. They also exhibit minimal fuzzy extraction issues; observed discrepancies are limited to minor syntactical artifacts, such as missing newline characters. Since start and end offsets are grounded for each extraction, these offsets can be used directly to recover *fuzzy-free* span boundaries.

5.4 Extraction Results

We apply the prompt from Experiment 4.2 to the full documentation corpus and extract 459 potential security advice items from 216 out of 639 total documents. Among these, 29 are fuzzy matches (6.3%) that exhibit the same characteristics observed during training, with one special case missing a Markdown header. The mean extraction length is 48 words, and 90% of extractions have fewer than 76 words. However, some extractions are long as they include extended contexts or Markdown structures such as tables and lists.

6 Analysis of Security Advice in the GHA Documentation

This section presents an in-depth qualitative and quantitative analysis of the large-scale extraction obtained from Section 5. In addition, we inspect two research-driven practices from the documentation’s viewpoint. In the following, SBPs always relate to OWASP, while security advice items are the actionable extractions. We denote the following multisets: $\mathcal{E}$ for all extractions, $\mathcal{A}$ for actionable advice items, and $\mathcal{O}$ for actionable advice items that map to one or more SBPs (hence $\mathcal{O} \subset \mathcal{A}$). $\mathcal{O}^+$ contains only those SBPs that are actionable as per Section 4.3. A hat ($\hat{\cdot}$) denotes that a set is free of syntactic duplicates.

Agglomerative clustering using the sorted token ratio with a threshold of 90% reveals that $|\hat{\mathcal{E}}| = 414$ out of $|\mathcal{E}| = 459$ extractions are unique. A notable source of duplicates is the repetition of 16 items across cloud provider platforms for OIDC access.

The first step is to identify actionable security advice, i.e., extractions that satisfy the requirements defined in Section 3.4 except for *focused*, as we want to assess the GHA documentation in full. The manual validation process yields $|\mathcal{A}| = 288$ instances, of which 257 are unique. As shown in Table 5, the majority of these advice items (157) fall under the topic *actions*. The *Secure use reference* contains the most security advice items (26). However, security advice is generally sparse within individual documents: among the 136 documents containing at least one advice item, 77 include only one and 33 contain two. On average, one advice item appears per 11,763 words. Most advice appears in the early portions of documents, approximately 10% to 60% of the document length, as illustrated in Figure 5. Nevertheless, developers must still scroll through substantial portions of the documentation to locate relevant guidance, underscoring the importance of well-structured headers that guide developers toward relevant content. Furthermore, Figure 3 reveals a contrasting trend: for the *code-security* topic, advice is concentrated in *tutorials* and *concepts*, whereas for the *actions* topic, it is primarily found in *reference* materials and *how-tos*. According to GitHub’s content organization framework [14], the four main content types are not aligned in their placement of security advice items.

We map each item of $\mathcal{A}$ to zero or more OWASP SBPs from Table 3 by best match with the guiding question “If we apply the security advice, does it (at least partially) operationalize the OWASP SBP?”. If the security advice revolves around a feature, we analyze the feature description for the match. Regarding our **running example**, we apply codes from the OWASP categories SEC-3, SEC-7, and SEC-10 because the detection of a *vulnerable dependency* covers multiple possible parts of system.

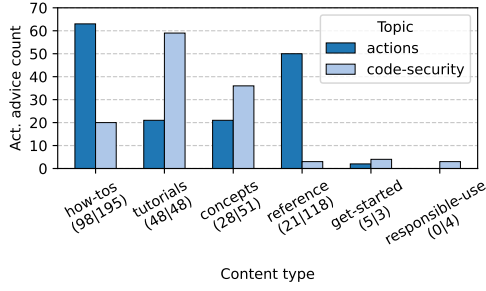


Figure 3 Counts of actionable advice items by content type and topic; ($\cdot/\cdot$) shows document counts for the respective topic.

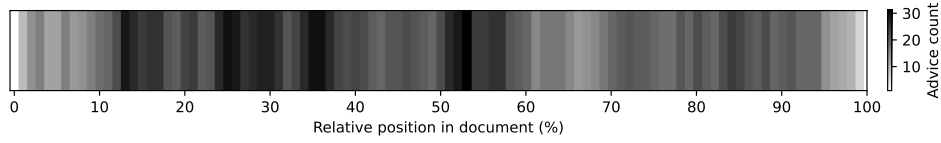


Figure 5 Interval-based heatmap of actionable advice occurrences across documents normalized by document length and accumulated into 100 bins over the relative document position.

Table 5 Distribution of advice types by topic.

Topic	Act.	Add.	Act. OWASP	SBP
code-sec.	125	47	24	54
actions	157	28	40	89
packages	6	3	0	3

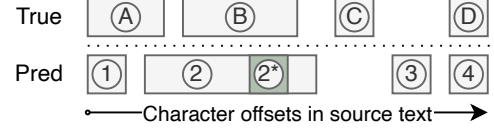
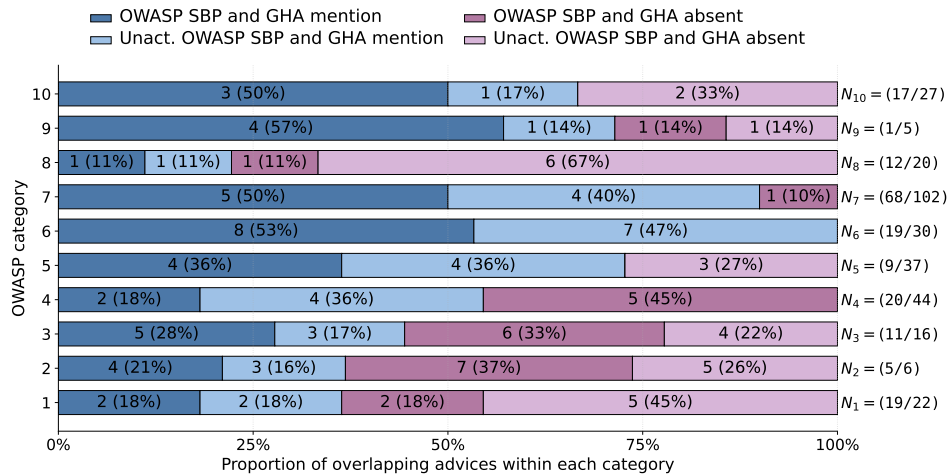


Figure 4 Types of relationships between predicted and ground-truth text spans in one source document.

We mark advice items as *additional* if no corresponding OWASP SBP exists, e.g., “Use code scanning to identify insecure code patterns”. Table 5 shows the overall number of actionable advice items (288), additional items (78), items that correspond to a non-actionable OWASP SBP ($|\mathcal{O}| = 64$), and items with at least one actionable OWASP SBP ($|\mathcal{O}^+| = 146$). It is expected that **code-security** is the primary source of additional items as it partly goes beyond CI/CD. Notably, only 50% of actionable advice items align with an actionable SBP, suggesting that a substantial portion of the documentation’s security guidance lacks a corresponding established best practice. Furthermore, 120 items have exactly one mapping while 90 have more than one. The most common SBPs of $\hat{\mathcal{O}}^+$ are *branch protection* (16), *stored or pushed secret detection* (15), and *prevention of secrets in logs and builds* (14).

To answer **RQ3**, we leverage the OWASP mapping to conduct a per-category analysis, examining the four intersections between actionable and non-actionable SBPs and mentioned or absent SBPs. Figure 6 summarizes all cases and lists the number of unique actionable and non-actionable SBPs covered in the documentation. GHA achieves high coverage of **SEC-6** and **SEC-7**, likely because many attacks exploit suboptimal system configurations (**SEC-7**, 102 mentioned SBPs) and steal credentials (**SEC-6**, 30 mentioned SBPs) [13, 26, 10, 24]. **SEC-4** also receives notable coverage (44 mentioned SBPs), as attacks typically exploit CI/CD pipelines via pull requests (*Poisoned Pipeline Execution*) [13, 26, 10, 24]. However, the GHA documentation fails to adequately inform developers about the usage of third-party services such as *plugins*. The OWASP framework does not fully cover this aspect: **SEC-4** mainly addresses governance and permission control of plugins but not their maintenance, e.g., updates and usage. The latter is conceptually addressed in **SEC-3** (Dependencies) and **SEC-7** (System Configuration) but does not apply to the scope of plugins.

Next, we analyze $\mathcal{A}$ to quantify the usage of visual alerts. The automated extraction initially only identifies 22 alerts, mainly due to missing *callout box* markers. Hence, we extend extractions by up to 15 characters before each extraction start offset, yielding 42 additional alerts, for a total of 64. The *actions* topic accounts for most alert usage (55



■ **Figure 6** Distribution of OWASP SBPs by actionable status and their corresponding mentioning status in the GHA documentation per OWASP category. N_i shows the unique number of actionable OWASP SBPs in proportion to the unique number of OWASP SBPs per category.

instances), with **WARNING** as the most frequent type (34 occurrences). However, alert usage remains suboptimal: 27 alerts are labeled **NOTE** and 2 **TIP**, which may not adequately convey the severity of the associated security guidance. Applying the same analysis to $\mathcal{O}$ shows that only 32 visual alerts correspond to SBPs, similarly reflecting inadequate alert categorization.

A common practice mentioned in research is **plugin pinning** to an immutable hash, especially for 3rd party plugins [20, 22, 8]. GitHub communicates this in 11 forms with varying degrees of security, trust, and functionality considerations. Only the *Secure use reference* mentions that this SBP mitigates the risk of propagation, as showcased by various attacks [13, 10]. However, throughout the full documentation, GitHub consistently uses workflow examples with pinned 3rd party plugins and strives to introduce a locking mechanism [30].

In contrast, security advice regarding **code injection** [27, 8] vulnerabilities appears in the same form seven times across the documentation; each instance links to the *secure use reference* for actual mitigations. Conversely, that reference contains no warning regarding the `pull_request_target` trigger. This trigger corresponds to only one piece of security advice in our dataset, namely Figure 2. Given that this trigger is often a prerequisite for a successful attack [13, 24, 26], it is notable that we encountered one extraction recommending the use of the trigger without providing additional security warnings.

7 Threats to Validity

In terms of **construct validity**, our approach builds upon existing definitions of security advice and established coding methodologies, which were validated through qualitative analyses but always assumed pre-segmented advice items. The *SAcoding* methodology involves inherently subjective judgments, and we assume that LLMs share similar biases when identifying security advice in documentation. Furthermore, our approach considers visual alerts as a strong proxy for prominence whereas other factors such as location, cross-linking and proximity to examples likely also matter. Selecting the OWASP Top 10 CI/CD Security Risks as the SBPs benchmark is a construct choice, as alternative frameworks may yield different coverage results.

Our experiments in Table 4 show unstable results, independent of zero temperature and a fixed seed, with consistently high recall but fluctuating FPs, threatening **internal validity**. As no additional validation set was used, we address this by conducting a manual review of all extracted items. The decision to use a proprietary model instead of an open-source alternative reflects a trade-off between model capability and economic considerations. Although proprietary models currently deliver superior performance, relying on them undermines reproducibility and long-term accessibility, since their availability may change or be discontinued. We accept this trade-off given the scale of the documentation corpus and our goal of assessing documentation quality rather than optimizing extraction performance. The impact is further mitigated by the task’s low frequency, as documentation evolves slowly, making reruns of the extraction pipeline rarely necessary.

For **external validity**, our analysis focuses on a single platform’s documentation; findings likely don’t generalize to other CI/CD platforms because of implementation details [20]. Nevertheless, GHA is the most widely used CI/CD platform [6], making it a representative and high-impact subject. A further limitation concerns developer information sources: Acar et al. show that developers frequently turn to alternative resources such as online forums or general web search results [2], which our study does not account for.

From the perspective of **conclusion validity**, both the manual review of all 459 extracted items subject to Section 3.4 and the mapping of advice items to OWASP SBPs by best match were performed by a single reviewer, introducing potential bias and subjectivity.

8 Conclusion

We show that the GHA documentation covers many actionable security advice items, but its coverage of CI/CD SBPs remains uneven across OWASP risks. These gaps indicate that the documentation should be treated as part of the CI/CD security surface, since missing or weakly communicated guidance can leave developers without the knowledge necessary to secure their repositories. However, we acknowledge that GitHub does not bear sole responsibility for developer issues but should strive to educate developers to the greatest extent possible, which our contribution supports by identifying areas for improvement. Even where guidance is complete, research suggests that the sheer volume of available security advice makes it difficult for developers to prioritize recommendations [35], so well-documented SBPs may still not translate into secure behavior.

Future work should propose a framework that enables documentation authors and SBP guideline authors to jointly translate common practices into documentation, with a focus on effective communication in terms of text structure and visual presentation. As a concrete step in this direction, we propose the following recommendations for GitHub:

- Extend the content best practices with structured rules that outline where and when to place SBPs, while also advocating for a uniform structure of security statements in terms of their semantics, e.g., as proposed by Barrera et al. with *Action* $\Rightarrow$ *Outcome* [4].
- Introduce a dedicated *security* visual alert to ease bookkeeping, or repurpose the unused *caution* alert, and unify existing ones.
- The current documentation structure places significant emphasis on the *secure-use* reference as a central hub. However, developers perceive this document as complex [22], reinforcing the need to distribute SBPs across the relevant documentation pages where they fit best, in line with the consensus of presenting them contextually when needed [36].

Data Availability Statement

Our corpus of all CI/CD-related documents mentioned in Table 2, coding results from Section 4.3, code and experimental data from Section 5, and evaluation notebooks for Section 6 are available at <https://github.com/bluuuk/gh-cicd-sbp-docs>. Our streamlit-based SAcoding tool is available at <https://github.com/bluuuk/SAcoder>.

References

- 1 Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, May 2016. ISSN: 2375-1207. doi:10/ghd865.
- 2 Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L. Mazurek, and Sascha Fahl. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *2017 IEEE Cybersecurity Development (SecDev)*, pages 22–26, September 2017. doi:10/hb4q6d.
- 3 Jessy Ayala, Yu-Jye Tung, and Joshua Garcia. A Mixed-Methods study of Open-Source software maintainers on vulnerability management and platform security features. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2105–2124, Seattle, WA, August 2025. USENIX Association. URL: <https://www.usenix.org/conference/usenixsecurity25/presentation/ayala>.
- 4 David Barrera, Christopher Bellman, and Paul Van Oorschot. Security Best Practices: A Critical Analysis Using IoT as a Case Study. *ACM Transactions on Privacy and Security*, 26(2):13:1–13:30, March 2023. doi:10/hb4q6f.
- 5 David Barrera, Christopher Bellman, and Paul C van Oorschot. A close look at a systematic method for analyzing sets of security advice. *Journal of Cybersecurity*, 9(1), January 2023. doi:10/hb4q6g.
- 6 Olga Bedrina. The State of CI/CD in 2025: Key Insights from the Latest JetBrains Survey | The TeamCity Blog, October 2025. URL: <https://blog.jetbrains.com/teamcity/2025/10/the-state-of-cicd/>.
- 7 Christopher Bellman and Paul C. van Oorschot. Systematic analysis and comparison of security advice as datasets. *Computers & Security*, 124:102989, January 2023. doi:10/hb4q6h.
- 8 Giacomo Benedetti, Luca Verderame, and Alessio Merlo. Automatic Security Assessment of GitHub Actions Workflows. In *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, pages 37–45, November 2022. doi:10/hb4q6j.
- 9 Lukas Boschanski and Marco Vieira. CI/CD Security Best Practices and GitHub Actions Documentation. Software (visited on 2026-08-24). URL: <https://github.com/bluuuk/gh-cicd-sbp-docs>, doi:10.4230/artifacts.27689.
- 10 Philipp Burckhardt. Trivy under attack again: Widespread github actions tag compromise exposes ci/cd secrets, March 2026. Accessed: 2026-04. URL: <https://socket.dev/blog/trivy-under-attack-again-github-actions-compromise>.
- 11 Ramaswamy Chandramouli, Frederick Kautz, and Santiago Torres-Arias. Strategies for the integration of software supply chain security in devsecops ci/cd pipelines. Technical Report 800-204D, National Institute of Standards and Technology, Gaithersburg, MD, February 2024. Approved 2024-01-31. doi:10.6028/NIST.SP.800-204D.
- 12 Continuous Delivery Foundation. CI/CD Cybersecurity Guide, 2025. URL: https://cybersecurity.cd.foundation/_print/docs/.
- 13 Omer Gil, Aviad Hahami, Asi Greenholts, and Yaron Avital. GitHub Actions Supply Chain Attack: A Targeted Attack on Coinbase Expanded to the Widespread tj-actions/changed-files Incident: Threat Assessment (Updated 4/2), March 2025. URL: <https://unit42.paloaltonetworks.com/github-actions-supply-chain-attack/>.

- 14 GitHub. Documentation done right: A developer's guide, 2024. URL: <https://github.blog/developer-skills/documentation-done-right-a-developers-guide/>.
- 15 Google. Langextract: A python library for extracting structured information from unstructured text using llms, 2026. Accessed: 2026-05-18. URL: <https://github.com/google/langextract>.
- 16 Peter L Gorski, Emanuel von Zezschwitz, Luigi Lo Iacono, and Matthew Smith. On providing systematized access to consolidated principles, guidelines and patterns for usable security research and development. *Journal of Cybersecurity*, 5(1), January 2019. doi:10/hb4q6q.
- 17 Peter Leo Gorski, Sebastian Möller, Stephan Wiefeling, and Luigi Lo Iacono. "I just looked for the solution!" On Integrating Security-Relevant Information in Non-Security API Documentation to Support Secure Coding Practices. *IEEE Transactions on Software Engineering*, 48(9):3467–3484, September 2022. doi:10/hb4q6r.
- 18 Yacong Gu, Lingyun Ying, Huajun Chai, Chu Qiao, Haixin Duan, and Xing Gao. Continuous Intrusion: Characterizing the Security of Continuous Integration Services. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1561–1577, May 2023. doi:10/hb4q6s.
- 19 Jiangnan Huang and Bin Lin. Revisiting Security Practices for Github Actions Workflows. In *2025 IEEE/ACM 33rd International Conference on Program Comprehension (ICPC)*, pages 73–77, 2025. doi:10/hb4q6t.
- 20 Igibek Koishybayev, Aleksandr Nahapetyan, Raima Zachariah, Siddharth Muralee, Bradley Reaves, Alexandros Kapravelos, and Aravind Machiry. Characterizing the Security of Github CI Workflows. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2747–2763, Boston, MA, August 2022. USENIX. URL: <https://www.usenix.org/conference/usenixsecurity22/presentation/koishybayev>.
- 21 Daniel Krivelevich, Omer Gil, and Ory Segal. OWASP Top 10 CI/CD Security Risks | OWASP Foundation. URL: <https://owasp.org/www-project-top-10-ci-cd-security-risks/>.
- 22 Yusuke Kubo, Fumihiro Kanei, Mitsuaki Akiyama, Takuro Wakai, and Tatsuya Mori. Action Required: A Mixed-Methods Study of Security Practices in GitHub Actions. In *NDSS*. The Internet Society, February 2026. doi:10/q64h.
- 23 Xiaofan Li, Yacong Gu, Chu Qiao, Zhenkai Zhang, Daiping Liu, Lingyun Ying, Haixin Duan, and Xing Gao. Toward Understanding the Security of Plugins in Continuous Integration Services. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 482–496. ACM, 2024. doi:10/hb4q6v.
- 24 Tanner Linsley. Postmortem: Tanstack npm supply-chain compromise, May 2026. URL: <https://tanstack.com/blog/npm-supply-chain-compromise-postmortem>.
- 25 Xiaoyue Ma, Lannan Luo, and Qiang Zeng. From One Thousand Pages of Specification to Unveiling Hidden Bugs: Large Language Model Assisted Fuzzing of Matter IoT Devices. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 4783–4800. USENIX, August 2024.
- 26 Rami McCarthy Merav Bar. singularity: supply chain attack leaks secrets on github: everything you need to know, August 2025. URL: <https://www.wiz.io/blog/singularity-supply-chain-attack>.
- 27 Siddharth Muralee, Igibek Koishybayev, Aleksandr Nahapetyan, Greg Tystahl, Brad Reaves, Antonio Bianchi, William Enck, Alexandros Kapravelos, and Aravind Machiry. ARGUS: A Framework for Staged Static Taint Analysis of GitHub Workflows and Actions. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6983–7000. USENIX, August 2023.
- 28 National Security Agency and Cybersecurity and Infrastructure Security Agency. Defending Continuous Integration/Continuous Delivery (CI/CD) Environments. Technical report, National Security Agency and Cybersecurity and Infrastructure Security Agency, June 2023. URL: https://media.defense.gov/2023/Jun/28/2003249466/-1/-1/0/CSI_DEFENDING_CI_CD_ENVIRONMENTS.PDF.
- 29 Lorenzo Neil, Harshini Sri Ramulu, Yasemin Acar, and Bradley Reaves. Who Comes Up with this Stuff? Interviewing Authors to Understand How They Produce Security Advice. In *Proceedings of the Nineteenth USENIX Conference on Usable Privacy and Security*, SOUPS '23, pages 283–299. USENIX, 2023. URL: <https://www.usenix.org/conference/soups2023/presentation/neil>.

- 30 Greg Ose. What's coming to our github actions 2026 security roadmap, March 2026. Accessed: 2026-04. URL: <https://github.blog/news-insights/product-news/whats-coming-to-our-github-actions-2026-security-roadmap/>.
- 31 Ziyue Pan, Wenbo Shen, Xingkai Wang, Yutian Yang, Rui Chang, Yao Liu, Chengwei Liu, Yang Liu, and Kui Ren. Ambush From All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines. *IEEE Transactions on Dependable and Secure Computing*, 21(1):403–418, 2024. doi:10/hb4q6x.
- 32 Mark Quinlan, Aaron Ceross, and Andrew Simpson. The Efficacy Potential of Cyber Security Advice as Presented in News Articles. *Interacting with Computers*, 37(1):30–48, January 2025. doi:10.1093/IWC/IWAE048.
- 33 Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. arXiv:1606.05250.
- 34 Elissa M. Redmiles, Amelia R. Malone, and Michelle L. Mazurek. I Think They're Trying to Tell Me Something: Advice Sources and Selection for Digital Security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 272–288, May 2016. doi:10/gnrpc8.
- 35 Elissa M. Redmiles, Noel Warford, Amritha Jayanti, Aravind Koneru, Sean Kross, Miraida Morales, Rock Stevens, and Michelle L. Mazurek. A Comprehensive Quality Evaluation of Security and Privacy Advice on the Web. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 89–108, 2020. URL: <https://www.usenix.org/conference/usenixsecurity20/presentation/redmiles>.
- 36 Robert W. Reeder, Iulia Ion, and Sunny Consolvo. 152 Simple Steps to Stay Safe Online: Security Advice for Non-Tech-Savvy Users. *IEEE Security & Privacy*, 15(5):55–64, 2017. doi:10/hb4q63.
- 37 Andrew Stewart and Christopher Hobbs. Systematic analysis of security advice on the topic of insider threats. *Computers & Security*, 154:104411, July 2025. doi:10/hb4q67.
- 38 Mindy Tran, Yasemin Acar, Michel Cucker, William Enck, Alexandros Kapravelos, Christian Kastner, and Laurie Williams. S3c2 summit 2202-09: Industry secure supply chain summit, 2023. doi:10.48550/arXiv.2307.15642.
- 39 Christoph Treude, Justin Middleton, and Thushari Atapattu. Beyond accuracy: Assessing software documentation quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*, pages 1509–1512. ACM, 2020. doi:10/hb4q68.
- 40 Greg Tystahl, Jonah Ghebremichael, Siddharth Muralee, Sourag Cherupattamoolayil, Antonio Bianchi, Aravind Machiry, Alexandros Kapravelos, and William Enck. Cosseter: Github actions permission reduction using demand-driven static analysis. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 2628–2645, 2026. doi:10.1109/SP63933.2026.00067.
- 41 Carmine Vassallo, Sebastian Proksch, Anna Jancso, Harald C. Gall, and Massimiliano Di Penta. Configuration smells in continuous delivery pipelines: A linter and a six-month study on GitLab. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020*, pages 327–337. ACM, 2020. doi:10/hb4q69.
- 42 Laurie Williams, Giacomo Benedetti, Sivana Hamer, Ranindya Paramitha, Imranur Rahman, Mahzabin Tamanna, Greg Tystahl, Nusrat Zahan, Patrick Morrison, Yasemin Acar, Michel Cukier, Christian Kästner, Alexandros Kapravelos, Dominik Wermke, and William Enck. Research Directions in Software Supply Chain Security. *ACM Transactions on Software Engineering and Methodology*, 34(5):146:1–146:38, 2025. doi:10/g9qww8.
- 43 Xinyu Zhang, Siddharth Muralee, Sourag Cherupattamoolayil, and Aravind Machiry. On the Effectiveness of Large Language Models for GitHub Workflows. In *Proceedings of the 19th International Conference on Availability, Reliability and Security, ARES '24*, pages 1–14. ACM, 2024. doi:10/hb4q7g.