Pipelines Under Pressure: An Empirical Study of Security Misconfigurations of GitHub Workflows

Edoardo Riggio[0009−0000−5649−3412] and Cesare Pautasso[0000−0002−2748−9665]

Software Institute, Università della Svizzera italiana (USI), Lugano, Switzerland
edoardo.riggio@usi.ch c.pautasso@ieee.org

Abstract. Continuous Integration and Continuous Deployment (CI/CD) pipelines have grown in popularity in recent years and are essential in streamlining the process of development and deployment of high quality software. However, developers often overlook security concerns in CI/CD pipelines, opening the door to many vulnerabilities. This paper presents an empirical investigation of nine security misconfigurations sourced from a comprehensive review of security guidelines, developer blogs, GitHub documentation, and prior research. To study the presence, co-occurrence and yearly trends of these security misconfigurations within current CI/CD practices, we analyzed a large dataset containing the most recent version of over 200 000 GitHub workflow specification files, taken from open source repositories. To aid us in this study, we developed *Soteria*, a static analysis tool equipped with custom detectors that can systematically identify security misconfigurations. Given that less than 1% of the analyzed workflows do not include any misconfiguration, our detection tool makes a contribution to raise awareness about widespread security misconfigurations. Our findings challenge conventional practices and motivate the need for an urgent shift in how security principles are systematically applied in the development and operation of CI/CD pipelines with more robust and finer-grained security controls.

Keywords: DevOps · Security · Misconfigurations · Workflows · CI/CD Build Pipeline · GitHub Actions

1 Introduction

In recent years, DevSecOps [1] has emerged as a transformative approach that fosters a culture of collaboration between developers, operations teams, and other stakeholders involved in the software lifecycle. DevSecOps leverages automation, replicability, continuous integration (CI), and continuous deployment (CD) to streamline and enhance the efficiency of software delivery, all while maintaining high-quality and high-security standards. The increased adoption of CI/CD practices [2] has brought security concerns to the forefront [3]. For example, a CI/CD pipeline could be misconfigured to run injected malicious code in privileged environments, which in turn could lead to losses in both the integrity and confidentiality of the software being packaged for release.

Many software repository hosting platforms offer CI/CD functionalities: developers can easily describe custom-made build pipelines as workflows. As developers focus on quickly setting up their CI/CD pipelines, they sometime fall into traps which are reflected by the smells [4] and security misconfigurations introduced in their workflows. As a consequence, there have been many incidents reported due to attacks exploiting vulnerable pipelines [5]. This has led to a growing body of literature collecting security guidelines and sharing experience on how to mitigate and control known vulnerabilities [6–11].

In this empirical study we statically analyze an existing large dataset of GitHub workflow specifications mined from open source repositories [12] looking for nine types of misconfigurations that lead to potential security vulnerabilities in the CI/CD pipeline. Our goal is to answer the following research question:

What are the most frequent and trending types of security misconfigurations found in GitHub CI/CD workflows?

Approach To introduce detectors for 9 types of misconfigurations in GitHub CI/CD workflows that could lead to security vulnerabilities [6], we used a two-step methodology. First, we surveyed the grey literature (GitHub documentation, security guidelines, DevSecOps expert blogs) looking for supply chain attacks caused by misconfigured GitHub workflows. After collecting 9 security misconfiguration types, we searched the academic literature for empirical studies and existing detection and correction tools, finding related work on six. To efficiently detect the miconfigurations, we have built *Soteria*, a static analyzer for GitHub workflow files, available as a VSCode extension[1]. Its source code can be found in the replication package[2] and on GitHub[3].

Main Findings Our findings highlight how security best practices are widely neglected. The most frequent type of misconfiguration affecting 94.1% of the workflows concerns the use of regular version pins when invoking external actions, which opens the door to supply chain attacks. While half of the misconfiguration types have been detected in less than 2% of the workflows, only 1 641 (0.81%) workflows did not present any misconfiguration at all. We also report an inverse relationship between severity and frequency, where critical misconfigurations can be found, but only rarely.

2 Related Work

Ongoing and recent research contributes to DevSecOps by detecting and mitigating vulnerabilities in the workflow before they can be exploited [6,13–22]. The position paper by Delicheh and Mens [6] inspired our work by pointing out many security risks associated with workflow misconfigurations. Also Paule et al. [19] identified the pressing need to detect vulnerabilities in CI/CD pipelines. Pecka

[1] https://marketplace.visualstudio.com/items?itemName=aegis-forge.soteria
[2] https://figshare.com/s/fdaeb40937fbe0af8a13
[3] https://github.com/aegis-forge/soteria

et al. [13, 14] studied vulnerabilities in CI/CD pipelines that use K8 clusters; Benedetti et al. [15] developed the GitHub Action Security Tester (GHAST) tool that performs some automatic security checks on GitHub workflow files; Kushwaha et al. [16] analyzed some misconfigurations in CI/CD pipelines in the context of financial systems; Pan et al. [17] performed attack surface analysis on GitHub workflow files; Li et al. [18] built a tool to detect malicious crypto-mining jobs in GitHub CI/CD workflow files; The ARGUS [21] tool uses static taint analysis to identify code injection vulnerabilities in GitHub Actions. Koishybayev et al. [20] show a detailed comparison of the security properties of GitHub CI/CD with other platforms as well as a large-scale empirical study on over 447 238 workflows used in 213 854 repositories focused on a subset of the misconfigurations we collected in this paper.

3 Security Misconfigurations

Table 1 lists the nine security misconfigurations studied in this paper, together with their short definitions, severity, and literature sources. We marked with a $\star$ the ones which have never been mentioned in previous empirical studies. The security misconfigurations have been divided into five different categories based on OWASP Top 10 CI/CD Security Risks [23].

To collect the security misconfigurations, we have also surveyed CI/CD blogs maintained by cybersecurity companies [7–9], open source security-focused organizations [24], or by GitHub itself [10, 11]. In addition, there are also some security researchers that write articles about specific attacks performed on software systems by exploiting CI/CD workflows [25, 26].

Blog posts [7, 24] and GitHub workflows documentation [11] were the main sources from which we extracted most of the security misconfigurations used in this study. We also consulted research done by security researchers [25, 26] to identify three less-known types of security misconfigurations which have not yet been covered by existing empirical studies [15, 20, 21]. After collecting the misconfigurations, we searched the academic literature for empirical studies and existing detection and correction tools, finding related work on 6 misconfigurations. The used sources all: 1) explicitly discuss the misconfiguration as a security risk; 2) show how to exploit the misconfiguration with proof of concepts or actual attacks; 3) present how to mitigate the misconfiguration impact.

3.1 Dependency Chain Abuse (CICD-SEC-3)

In Dependency Chain Abuse, the software supply chain on which the workflow relies on is compromised. This could enable malicious actors to push tainted version of packages which are then downloaded and executed by workflows with the following misconfiguration:

no-hash-version-pin Using regular version pins (such as `@main`, `@v1`, `@v2.1.1`) when calling external GitHub Actions opens the door to supply chain attacks in the pipeline. Being regular version pins mutable, malicious actors could gain

Table 1. Security Misconfigurations: Definitions, Severity and Sources

Group	ID	Misconfiguration	Severity	Literature
CICD-SEC-3	1	`no-hash-version-pin`	Low	[15, 17, 20, 22, 27, 28]
CICD-SEC-4	2	`unconditional-injection`	High	[6, 10, 15, 21]
	3	`conditional-injection`	Medium	[6, 10, 15, 21]
	4⋆	`pwn-request`	High	[7, 29, 30]
CICD-SEC-5	5	`coarse-permission`	Medium	[6, 15, 20, 22, 31]
	6	`global-secret`	Medium	[6, 15, 20, 32]
CICD-SEC-7	7	`self-hosted-runner`	Medium	[8, 9, 20, 25]
CICD-SEC-9	8⋆	`caching-in-release`	Critical	[26, 33, 34]
	9⋆	`unsafe-artifact-download`	Critical	[35, 36]

access to the repository of the external third-party Action, change its source code, and publish a new release under the same major, minor, or branch tag (via tag-renaming attacks [24]). To avoid this vulnerability, one would have to use a hash version pin when referring to external GitHub Actions [7, 11, 35] – i.e. `actions/checkout@1a...53`.

3.2 Poisoned Pipeline Execution (CICD-SEC-4)

With Poisoned Pipeline Execution, an attacker may manipulate in some way the pipeline process without having direct access to its workflow definition suffering from one of the following misconfigurations:

unconditional-injection In workflow files, developers can refer to environment variables present in GitHub's context [10], for example, to read the name of the issue that triggered the pipeline, the name of the commit, etc. As we can see on line 9 in Figure 1, a step runs a bash script. The bash script will run what seems like an innocuous `echo` command. The problem is that the `issue.title` variable is not under the control of the maintainer of the repository, and can be set by anyone. Attackers may create an issue to the repository with the following title: `title" && ls / && echo "`. This will cause GitHub workflow's runner to replace `${{ ... }}` in the `run` section with the tainted issue title. The runner will execute the commands `echo "title" && ls / && echo ""`, thus also running a `ls` command on the root directory of the runner [6, 10, 15, 21]. This becomes especially dangerous when it is paired with a trigger on the creation or modification of an issue. To avoid this, maintainers should save the data in environment variables, so that it will be treated as a string.

conditional-injection This misconfiguration combines the previous misconfiguration with a condition – the `if` section of the workflow – which controls when to run the job containing the problematic GitHub context variable [6, 10, 15, 21].

```yaml
on:
  issues:
    types: [opened]
jobs:
  print_issue_title:
    runs-on: ubuntu-latest
    name: Print issue title
    steps:
      - run: echo "${{ github.event.issue.title }}"
```

Fig. 1. An example of workflow vulnerable to a code injection attack.

pwn-request When a workflow uses the `pull_request_target` trigger, it runs in the context of the target repository – the one under the control of the repository maintainer. Thus, if the workflow also checks out the forked repository, it opens the workflow to possible remote code execution [7, 29, 30]. To avoid this, never checkout the repository when using a `pull_request_target` trigger.

3.3 Insufficient Pipeline-Based Access Controls (CICD-SEC-5)

Insufficient Pipeline-Based Access Controls (PBAC) represents security misconfigurations that deal with permission-granting and access to sensitive data such as secrets.

coarse-permission A permission is said to be coarse when the principle of least privilege is not followed – thus ending up with a workflow or job that is too permissive. For example, setting the permissions to be `read-all`, `write-all`, or not setting them at all, are all instances of `coarse-permission` [7]. It is good practice to always set all the permissions to `none` at a workflow level (with the `permissions: {}` syntax) so that at job level only the strictly necessary permissions are granted.

global-secret Secrets are environment variables containing very sensitive information such as keys, tokens, and passwords. Such variables can be defined at any point in the pipeline. If defined at workflow- or job-levels, there is a risk that the secret is exposed to steps or containers that do not need them. This could allow potentially tainted external GitHub Actions to gain access to such secrets [7].

3.4 Insecure System Configuration (CICD-SEC-7)

Insecure System Configuration concerns flaws in the hardening of external systems used by the pipeline. Such vulnerable systems can facilitate an attacker's process to expand its foothold in the workflow environment.

self-hosted-runner Workflow jobs can be executed by a GitHub runner or a self-hosted runner [33]. While the former is by default configured to be ephemeral

and isolated, the latter must be secured by the workflow developers themselves. Using non-ephemeral self-hosted runners can open the door to runner poisoning attacks. Moreover, if permissions are not set properly, a user can fork the repository and trigger workflow runs without the owner's approval – which could happen especially when a "contributor" badge is assigned to the user after making their first contribution to the repository. After running the payload on the runner, an attacker could get a web shell on the target runner, thus performing a successful runner poisoning attack [8,9,25]. For this reason, self-hosted runners should be avoided in public repositories.

3.5 Improper Artifact Integrity Validation (CICD-SEC-9)

The following misconfigurations allow attackers to modify the artifacts generated by the pipeline by inserting some malicious payload.

caching-in-release Caching in the CI/CD pipeline [33] can be done in GitHub through the `actions/cache` Action. This Action, if not used correctly, can lead to caching sensitive data or, in worst cases, to cache poisoning attacks [26,34,37]. To conduct a cache poisoning attack, an attacker would need to exfiltrate both the `CacheServerUrl` and the `AccessToken` from a workflow – which can be done through a poisoned pipeline execution (CICD-SEC-4) security misconfiguration or through a compromised dependency. Next, the attacker creates the payload and inserts it in the cache. To do so, the attacker could restore or anticipate the cache keys, or could start blasting the cache such that the oldest cache entries are evicted and the poisoned ones are inserted [26]. In addition, the use of caching in release workflows is especially dangerous. As stated in the SLSA (Supply-chain Levels for Software Artifacts) L3 requirements, "It MUST NOT be possible for one build to inject false entries into a build cache used by another build, also known as 'cache poisoning'. In other words, the output of the build MUST be identical whether or not the cache is used." [38]

unsafe-artifact-download There is a specific third-party Action used for downloading artifacts that, if not configured correctly, can lead to file overriding. The Action is `dawidd6/action-download-artifact`, which has three parameters `path`, `commit`, and `run_id`. The `path` parameter must be passed to avoid files to be overwritten by tainted ones, while both the `commit` and `run_id` parameters are used to verify that the artifact comes from a known and trusted source. If used without setting these parameters, the Action will extract the downloaded artifact into the default directory, overriding any file with the same name. This means that files to be executed can be replaced with malicious ones [35, 36].

4 Dataset

The dataset used in this work was taken from Cardoen et al. [12], in particular the latest version of the dataset from October 2024 [39]. In their work, they extracted 2 367 030 workflow files from 43 342 different GitHub repositories thanks

Table 2. Structural Characteristics in Workflows and Repositories

	Workflows				Repositories				
Element	**Mean**	**Median**	**Min**	**Max**	**Mean**	**Median**	**Min**	**Max**	**Total**
Repositories	—	—	—	—	—	—	—	—	40 584
Workflows	—	—	—	—	4.9	3	1	288	200 758
Jobs	1.4	1	0	250	7.0	4	1	750	285 818
Steps	6.8	3	0	681	33.5	16	1	2 992	1 363 134
Containers	0.05	0	0	111	0.24	0	0	201	9 787
Permissions	1.5	0	0	367	1.5	0	0	367	61 070
Env. Variables	12.2	2	0	13 531	12.2	2	0	13 531	495 151
Ext. Workflows	0.08	0	0	175	0.4	0	0	13 531	16 582
Actions	3.6	3	0	681	17.7	9	0	13 531	718 611

to *gigawork*, which iterates through the `.github/workflows` directory for every given GitHub repository, and extracts all the pushed versions of those workflow files. In addition to a ZIP file containing all of the actual workflow files, the authors provided some CSV files containing provenance metadata.

From this huge dataset, we have selected the most recent version of each workflow as we assume them to be the most improved ones and we are not interested (in this study) in performing a historical comparison of different versions of the same workflows. To carry out our study, we performed some data cleaning steps on the CSV file containing the workflows' metadata. In the data cleaning process, we used the `committed_date` column to retrieve the most recent workflow version per history. After retrieving the most recent, we used the `valid_workflow` column to discard all those workflows that were not valid.

From the CSV file provided by Cardoen et al., we created our own CSV file containing all the data and metadata of the workflows. In particular, we have taken the `repository`, `commit_hash`, `file_path`, and `committed_date` columns. In addition to these columns, we've added our own, namely `workflow_id` and `content`. In the `content` column, we have a base64 encoding of the actual content of the workflow, which was taken from the TAR.GZ archive provided by the authors of the dataset. This encoding was done so as to have the necessary information – i.e. data, metadata, and provenance – all in the same place.

After all the data cleaning and preprocessing steps mentioned above – which included discarding 469 non-existent workflows – we are left with 200 759 unique workflows from 40 584 repositories, which have been analyzed in this study. Table 2 shows statistics on the main structural elements of the dataset's workflows and repositories.

5 Methodology

To detect the security misconfigurations present in GitHub workflows we developed *Soteria*. This tool statically analyzes Y(A)ML files containing GitHub

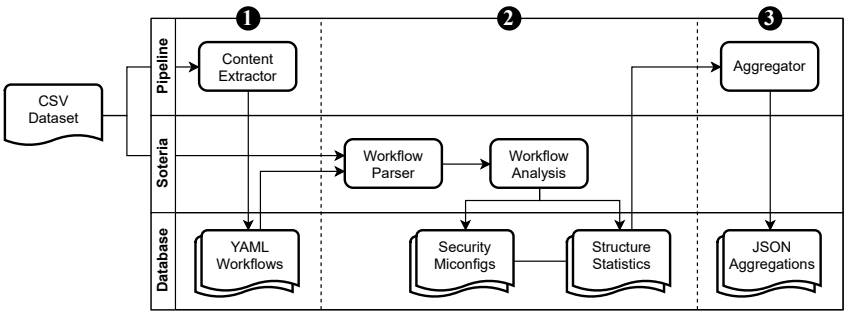

Fig. 2. Analysis pipeline

workflows based on detectors which can be customized and defined by developers. It also measures and computes statistics on the structure of the workflows and the misconfigurations detected within a repository.

In this study, we used the analysis pipeline described in Figure 2. The YAML files extracted from the dataset ❶ are fed to *Soteria* to both detect the security misconfigurations present in the workflows, and to compute workflows' structure statistics ❷. To detect the security misconfigurations, *Soteria* uses pattern matching rules derived from or even provided by the authors of the literature sources identifying the misconfigurations. The tool returns structural metrics together with a collection of detected security misconfigurations. The collections returned by *Soteria* are aggregated ❸ by workflow and by repository and classified by severity. For replicability and transparency purposes, all intermediate data returned by each step of the pipeline is saved to a database. Each file (with the exception of the "YAML Workflows" collection) is structured in a JSON format and contain the frequencies and actual occurrences (i.e. the line of code) of either misconfigurations or structures.

6 Results: Trends and Frequencies

After running the detection and analysis pipeline described in Section 5, we obtained the results described in Table 3, presenting statistics on the number of misconfigurations detected in each workflow and in each repository. We also aggregated the results by severity and reported the very low number and percentage of workflows and repositories in which no misconfiguration was detected.

The top two misconfigurations affecting 93% of workflows and found in almost all repositories are `no-hash-version-pin` and `coarse-permission`. All other misconfigurations have a median of 0, indicating that the majority of workflows (and repositories) do not include them. Finally, the `caching-in-release` statistics refer to the best-case scenario, where we consider our heuristics to detect a release workflow to work 100% of the time.

Table 3. Statistics on the number of detected misconfigurations, and number of affected workflows and repositories

	Workflows				Repositories				
ID	**Mean**	**Max**	**Affected**		**Mean**	**Max**	**Affected**		**Total**
1	3.4	681	188 782	(94.0%)	16.8	2992	40 375	(99.5%)	681 682
low	3.4	681	188 782	(94.0%)	16.8	2992	40 375	(99.5%)	681 682
3	0.0	1	43	(0.0%)	0.0	4	37	(0.1%)	43
5	0.93	7	186 711	(93.0%)	4.61	286	39 836	(98.2%)	186 996
6	0.6	285	48 440	(24.1%)	2.98	1700	18 464	(45.5%)	120 860
7	0.03	113	3 423	(1.7%)	0.15	227	869	(2.1%)	6 029
medium	1.56	286	190 742	(95.0%)	7.74	1988	40 006	(98.6%)	313 928
2	0.0	1	14	(0.0%)	0.0	1	14	(0.0%)	14
4	0.0	1	502	(0.3%)	0.01	107	295	(0.7%)	502
high	0.0	1	516	0.3%	0.01	107	309	(0.8%)	516
8	0.01	1	2 209	(1.1%)	0.05	33	1 436	(3.5%)	2 209
9	0.0	1	387	(0.2%)	0.01	9	266	(0.7%)	387
critical	0.01	2	2 590	(1.3%)	0.06	33	1 670	(4.1%)	2 596
None	—	—	1 641	(0.81%)	—	—	35	(0.08%)	—

6.1 Misconfigurations Trends

GitHub Actions was announced in 2018 and released in 2019, as reflected in the workflow commit dates, spread between the 4$^{\text{th}}$ of August 2019 to the 10$^{\text{th}}$ of October 2024. Table 4 shows the yearly number of occurrences of each misconfiguration – also grouped by severity. We did not include 2019 since there are only 5 months of data, and only the first 10 months of 2024 are covered.

By looking at Table 4, we can see that the absolute number of misconfigurations tend to vary over the years. However, if we also include the normalized values (the percentage in parenthesis), we can actually see that these values stay more or less the same throughout the 5 years. As reported in the last line of the table, the number of workflows constantly increases over the years. In the case of the single misconfigurations, the most prevalent ones are `no-hash-version-pin` (ID 1), `coarse-permission` (ID 5), and `global-secret` (ID 6).

6.2 Misconfigurations Co-Occurrence

Figure 3 shows how many distinct security misconfigurations types co-occur in a workflow and a repository. From this figure, we can evince that most of the times (in 69% of workflows and 51.7% of repositories) there is a co-occurrence of two different misconfigurations. The pairwise Jaccard similarity scores (Intersection over Union, IoU) for each co-occurring pairs of misconfigurations can be found in Figure 4. In this figure, the lower triangular matrix represents the Jaccard scores

Table 4. Yearly count of detected misconfigurations and misconfigured workflows

ID	2020		2021		2022		2023		2024	
1	114 697	(69.5%)	151 511	(69.2%)	156 008	(67.8%)	154 087	(67.6%)	91 355	(67.2%)
2	1	(0.001%)	4	(0.002%)	2	(0.001%)	4	(0.002%)	3	(0.002%)
3	2	(0.001%)	10	(0.005%)	10	(0.004%)	12	(0.005%)	9	(0.007%)
4	22	(0.01%)	90	(0.04%)	212	(0.09%)	118	(0.05%)	60	(0.04%)
5	32 550	(19.7%)	41 791	(19.1%)	42 656	(18.5%)	40 841	(17.9%)	24 038	(17.6%)
6	17 003	(10.3%)	23 770	(10.8%)	28 771	(12.5%)	29 795	(13.0%)	19 026	(14.0%)
7	290	(0.1%)	904	(0.4%)	1477	(0.6%)	2214	(0.9%)	1130	(0.8%)
8	370	(0.2%)	597	(0.2%)	560	(0.2%)	451	(0.1%)	205	(0.1%)
9	13	(0.008%)	86	(0.03%)	114	(0.05%)	127	(0.05%)	47	(0.03%)
low	114 697	(69.5%)	151 511	(69.2%)	156 008	(67.8%)	154 087	(67.6%)	91 355	(67.2%)
medium	49 845	(30.2%)	66 474	(30.3%)	72 914	(31.7%)	72 862	(32.0%)	44 203	(32.5%)
high	23	(0.01%)	94	(0.04%)	214	(0.09%)	122	(0.05%)	63	(0.05%)
critical	383	(0.2%)	683	(0.3%)	674	(0.2%)	578	(0.2%)	252	(0.1%)
Total	164 934		218 762		229 806		227 642		135 864	
Workflows	32 563	(99%)	42 149	(99%)	45 541	(99%)	47 039	(99%)	28 346	(97%)
None	13		34		249		754		591	
Repositories	12 452	(100%)	11 022	(99%)	6 955	(99%)	5 283	(99%)	1 993	(99%)

for the workflow-grouped data, while the upper triangular matrix represents the Jaccard scores for the repository-grouped data. There is a very frequent co-occurrence of `no-hash-version-pin` and `coarse-permission` both in workflow-grouped data (IoU = 0.89) and in repository-grouped data (IoU = 0.98).

6.3 Actions and Version Tag Types

Our analysis pipeline identified 494 265 first-party Actions, 207 577 third-party Actions (Fig. 5). In both, the *major version* tagging method is the most used – respectively 91.4% and 60.7% of the cases. Only 2.4% of first-party Actions and 6.3% of third-party Actions use the full commit hash. Based on the latest security advisories[4], 4 897 actions are vulnerable. Of these, 875 actions (18%) were referenced from 597 workflows using a version for which a security advisory was published before the commit date of the workflow. This is not the case for the remaining 2 569 workflows referencing 4 022 actions.

[4] https://docs.github.com/en/code-security/security-advisories/working-with-global-security-advisories-from-the-github-advisory-database/about-the-github-advisory-database

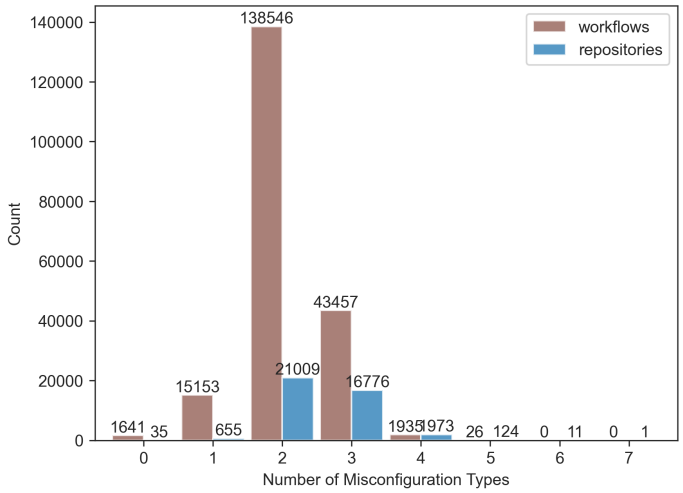

Fig. 3. Co-occurrence of multiple misconfiguration types in the same workflow (in brown) and in the same repository (in blue)

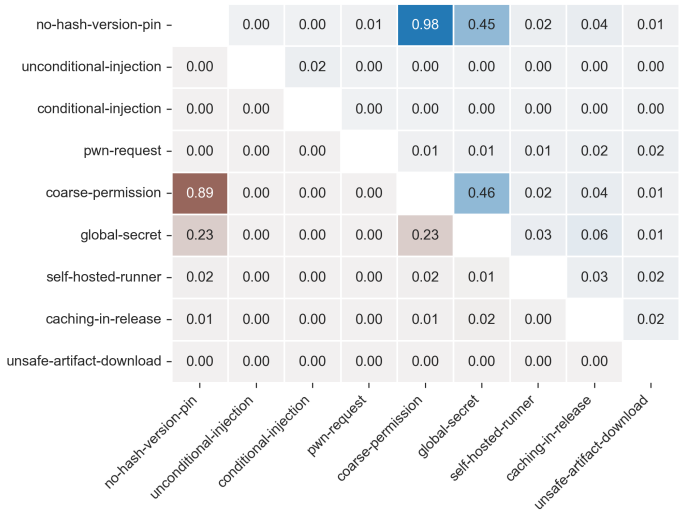

Fig. 4. Jaccard similarity for pairs of misconfiguration types detected in the same workflow (lower-triangular) or repository (upper-triangular)

7 Discussion

The empirical analysis of the most recent version of more than 200 000 GitHub CI/CD workflow specifications carried out in this study shows how security misconfigurations are nearly ubiquitous. Only less than 1% of workflows are free from the security misconfigurations *Soteria* can detect. The pervasive presence of such misconfigurations highlights a shortfall in the adoption of security best

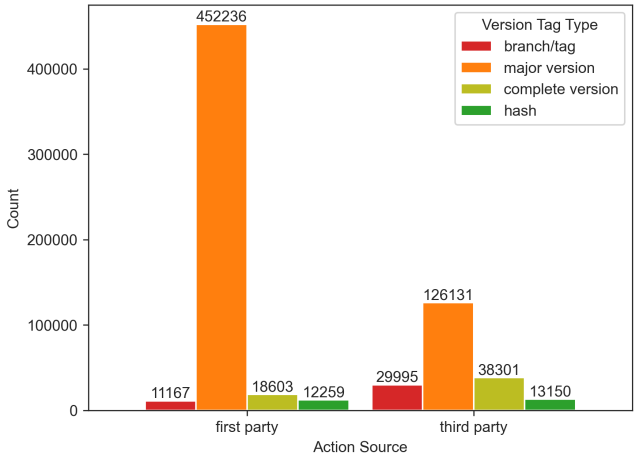

Fig. 5. Number of Action invocations by version tag type

practices, raising concerns about the integrity of the open source software supply chain based on GitHub.

The trend analysis of the nine security misconfigurations reveals that while the low severity `no-hash-version-pin` represents between 67% and 70% of all detected misconfigurations every year, the more critical ones such as `pwn-request` fluctuate between 0.013% in 2020 to a small peak of 0.092% in 2022, or become more prevalent – like `global-secret` growing from 10% to 14% in 2024.

In general, there is an inverse relationship between the prevalence and the severity of a misconfiguration: on average, every year we classified 68.32% low and 31.37% medium severity misconfigurations, leaving only 0.3% of highly severe or critical ones. The overall stability of the trend suggests that some developers still need to raise their awareness and remediation efforts as they still rely on build pipelines that are inadequate for the current threat landscape.

Our analysis also indicates that the most common security misconfigurations – `no-hash-version-pin` and `coarse-permission` – are present in more than 93% of the 200 758 workflows in our dataset. This may be due to developers being unaware that version pins are mutable and thus relying on them for external actions exposes the pipeline to potential supply chain attacks. Also, the use of overly permissive token settings violates the principle of least privilege. These results confirm similar "disturbing observations" published in previous studies [15]. If we consider insufficient access control, 10 826 workflows (62%) with no permissions declared – thus using the default permissions defined by `GITHUB_TOKEN` – have been detected in [15], while a very small number of workflows overriding default permission setting was counted at 900 (0.2%) in [20]. In our dataset, this number grows to 14 053 (7%). Regarding self-hosted runners, we detected 3 423 workflows in 227 public repositories affected by this misconfiguration. A lower bound of this metric was set at 565 public repositories in [20]. Immutable references to third-party actions are used only in 6% of the work-

flows in our dataset across 209 (0.5%) repositories. This is comparable to the observation by [20], where only 1.7% (3 248) of repositories do not incur in the `no_hash_version_pin` misconfiguration. Using ARGUS [21], Muralee et al. used taint analysis to discover code injection vulnerabilities in 4 307 workflows (0.16% of their sample), while our approach detected 57 (0.03%) workflows.

Overall, we present a large-scale, reproducible, exploratory empirical study of 9 security misconfigurations which can be detected using the same tool. Existing studies focus on the detection of individual (or a few) security misconfigurations, mostly on smaller datasets (which have not always been published). Our findings not only cover detecting the presence but also measure the frequency and the co-occurrence. The high correlation between the total number of detected misconfigurations of the two most widespread ones is an example of a surprising observation, which can lead to hypothesis formation and more in-depth studies to provide explanations, which could range from fundamental design issues of the automation platform to developer errors.

8 Threats to Validity

Dataset Limitations — Our dataset of 200 758 workflows provides a large sample size. However, only the most recent versions of the workflows have been selected to be analyzed. By doing so, we are omitting a possible understanding of the historical progression – or regression – of workflow configurations over time. In addition to this, despite the filtering done by the original authors of the dataset, some toy projects might still be present. We also performed the same analysis on an older dataset of 580 187 workflows from [20], also present in the replication package. Due to space restrictions, we could not include the detailed results, which show a similar relative frequency and ranking of the misconfigurations.

Construct Validity — Both types and definitions of the misconfigurations could also represent a threat to validity. The misconfigurations used in this study were chosen based on common CI/CD security guidelines and known vulnerabilities. However, there is a risk that the decision both of the types and the definition of the misconfiguration itself could be limited, and others could exist.

External Validity — This study focuses only on workflows written for GitHub's CI/CD engine, which might not represent also the workflows written for other CI/CD engines – such as GitLab's – or enterprise-level engines. In addition, the workflows in our dataset have been taken only from publicly available GitHub repositories, thus these misconfigurations might not apply to CI/CD workflow files used in private repositories. For example, it is considered safer to use self-hosted runners in private repositories than in public ones [11].

9 Conclusion

This paper presents an extensive empirical investigation into the detection of security misconfigurations within a large collection of GitHub workflow specifi-

cations. Aided by *Soteria* we managed to uncover nine different types of misconfigurations, some of which affecting almost all sampled workflow specifications. These misconfiguration types have been gathered by surveying previous literature, including blogs from top cybersecurity firms and researchers.

From our analysis, we were able to find the most recurring security misconfigurations, and report their trends across a five-year time span. We found that the most common security misconfiguration are the ones that use mutable version pins, overly permissive token settings, and secrets at a global scope. We also observed that these nine security misconfigurations often co-occur, broadening the attack surface.

Our main contributions lie in providing a detailed and large scale empirical analysis of these nine security misconfigurations, three of which have not been yet empirically studied. By developing a static analysis tool we automatically detected and characterized these misconfigurations at scale. These results should raise awareness on the current weak state of security practices in DevSecOps. Educators can use this result to raise awareness so that practitioners avoid introducing such security misconfigurations, thus preventing supply chain attacks. In our view, developers should prioritize the removal of critical misconfigurations, especially those affecting workflows used to build releases. Our study also indicates that a complete removal of all misconfigurations would require a large effort as they affect most repositories. The study results justify investing in linters for GitHub workflows, which should not only integrate security misconfiguration detectors but also provide auto-correction recommendations. Designers of future CI/CD automation tools should ensure that workflow languages are improved to reduce exposure and prevent introducing such misconfigurations.

The development of *Soteria* demonstrates the feasibility of using static analysis for automatic detection of many different security misconfigurations in CI/CD workflow specifications. Looking forward, we would like to perform a more detailed research to refine and extend the detection techniques towards the definition of security smells and anti-patterns, possibly extending the analysis to other CI/CD platforms.

References

1. K. Carter, "Francois Raynaud on DevSecOps," *IEEE Software*, vol. 34, no. 5, pp. 93–96, 2017.
2. J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson, 2010.
3. R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen, "Challenges and solutions when adopting DevSecOps: A systematic review," *Information and software technology*, vol. 141, p. 106700, 2022.
4. C. Vassallo, S. Proksch, A. Jancso, H. C. Gall, and M. Di Penta, "Configuration smells in continuous delivery pipelines: a linter and a six-month study on GitLab," in *Proc. 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, Nov. 2020, pp. 327–337.

5. T. H.-C. Hsu, *Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps.* Packt Publishing Ltd, 2018.
6. H. O. Delicheh and T. Mens, "Mitigating Security Issues in GitHub Actions," in *Proc. 4th Int'l Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2nd Int'l Workshop on Software Vulnerability (SVM).* ACM/IEEE, Aug. 2024, pp. 6–11.
7. T. Segura. GitHub Actions Security Best Practices. [Online]. Available: https://blog.gitguardian.com/github-actions-security-cheat-sheet/
8. H. Vincent. GitHub Actions exploitation: self hosted runners. [Online]. Available: https://www.synacktiv.com/en/publications/github-actions-exploitation-self-hosted-runners
9. R. Blit. GitHub, PyTorch and More Organizations Found Vulnerable to Self-Hosted Runner Attacks. [Online]. Available: https://www.legitsecurity.com/blog/github-pytorch-and-more-organizations-found-vulnerable-to-self-hosted-runner-attacks
10. J. Lobacevski, "Keeping your GitHub Actions and workflows secure Part 2: Untrusted input." [Online]. Available: https://securitylab.github.com/resources/github-actions-untrusted-input/
11. GitHub, "Security hardening for GitHub Actions." [Online]. Available: https://docs.github.com/en/actions/security-for-github-actions/security-guides/security-hardening-for-github-actions
12. G. Cardoen, T. Mens, and A. Decan, "A dataset of GitHub Actions workflow histories," in *Proc. 21st Int'l Conference on Mining Software Repositories (MSR).* ACM, Apr. 2024, pp. 677–681.
13. N. Pecka, L. b. Othmane, and A. Valani, "Making Secure Software Insecure without Changing Its Code: The Possibilities and Impacts of Attacks on the DevOps Pipeline," Jan. 2022. [Online]. Available: http://arxiv.org/abs/2201.12879
14. N. Pecka, L. Ben Othmane, and A. Valani, "Privilege Escalation Attack Scenarios on the DevOps Pipeline Within a Kubernetes Environment," in *Proc. Int'l Conference on Software and System Processes and Int'l Conference on Global Software Engineering (ICSSP).* ACM, May 2022, pp. 45–49.
15. G. Benedetti, L. Verderame, and A. Merlo, "Automatic Security Assessment of GitHub Actions Workflows," in *Proc. Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED).* ACM, Nov. 2022, pp. 37–45.
16. M. K. Kushwaha, P. David, and G. Suseela, "Automation and DevSecOps: Streamlining Security Measures in Financial System," in *Proc. Int'l Conference on Electronics, Computing and Communication Technologies (CONECCT)*, 2024, pp. 1–6.
17. Z. Pan, W. Shen, X. Wang, Y. Yang, R. Chang, Y. Liu, C. Liu, Y. Liu, and K. Ren, "Ambush From All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 1, pp. 403–418, Jan. 2024.
18. Z. Li, W. Liu, H. Chen, X. Wang, X. Liao, L. Xing, M. Zha, H. Jin, and D. Zou, "Robbery on DevOps: Understanding and Mitigating Illicit Cryptomining on Continuous Integration Service Platforms," in *2022 IEEE Symposium on Security and Privacy (SP)*, May 2022, pp. 2397–2412.
19. C. Paule, T. F. Düllmann, and A. Van Hoorn, "Vulnerabilities in Continuous Delivery Pipelines? A Case Study," in *2019 IEEE Int'l Conference on Software Architecture Companion (ICSA-C)*, Mar. 2019, pp. 102–108.
20. I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry, "Characterizing the Security of Github CI Workflows," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2747–2763.

21. S. Muralee, I. Koishybayev, A. Nahapetyan, G. Tystahl, B. Reaves, A. Bianchi, W. Enck, A. Kapravelos, and A. Machiry, "ARGUS: A Framework for Staged Static Taint Analysis of GitHub Workflows and Actions," in *Proc. 32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6983–7000.
22. A. Khatami, C. Willekens, and A. Zaidman, "Catching Smells in the Act: A GitHub Actions Workflow Investigation," in *2024 IEEE Int'l Conference on Source Code Analysis and Manipulation (SCAM)*, Oct. 2024, pp. 47–58.
23. K. Daniel and G. Omer. OWASP Top 10 CI/CD Security Risks. [Online]. Available: https://owasp.org/www-project-top-10-ci-cd-security-risks
24. OpenSSF, "Mitigating Attack Vectors in GitHub Workflows." [Online]. Available: https://openssf.org/blog/2024/08/12/mitigating-attack-vectors-in-github-workflows/
25. A. Khan. One Supply Chain Attack to Rule Them All – Poisoning GitHub's Runner Images. [Online]. Available: https://adnanthekhan.com/2023/12/20/one-supply-chain-attack-to-rule-them-all/
26. ——, "The Monsters in Your Build Cache – GitHub Actions Cache Poisoning." [Online]. Available: https://adnanthekhan.com/2024/05/06/the-monsters-in-your-build-cache-github-actions-cache-poisoning
27. J. Renaux, "Use GitHub actions at your own risk." [Online]. Available: https://julienrenaux.fr/2019/12/20/github-actions-security-risk/
28. A. Decan, T. Mens, P. R. Mazrae, and M. Golzadeh, "On the Use of GitHub Actions in Software Development Repositories," in *2022 IEEE Int'l Conference on Software Maintenance and Evolution (ICSME)*, Oct. 2022, pp. 235–245.
29. J. Lobacevski. Producing artifacts. [Online]. Available: https://securitylab.github.com/resources/github-actions-preventing-pwn-requests/
30. A. Khan. Long Live the Pwn Request. [Online]. Available: https://www.praetorian.com/blog/pwn-request-hacking-microsoft-github-repositories-and-more/
31. M. Heap. The ultimate guide to GitHub Actions authentication. [Online]. Available: https://michaelheap.com/ultimate-guide-github-actions-authentication/
32. Mackenzie. Biggest security takeaway of 2020 - Don't leak secrets on GitHub. [Online]. Available: https://dev.to/advocatemack/biggest-security-takeaway-of-2020-don-t-leak-secrets-on-github-44k2
33. S. McIntosh, "Mining Our Way Back to Incremental Builds for DevOps Pipelines," in *Proc. of the 21st Int'l Conference on Mining Software Repositories (MSR)*. ACM, 2024, pp. 48–49.
34. B. Brudo, "GitHub Cache Poisoning." [Online]. Available: https://scribesecurity.com/blog/github-cache-poisoning
35. D. Noam. GitHub Actions That Open the Door to CI/CD Pipeline Attacks. [Online]. Available: https://www.legitsecurity.com/blog/github-actions-that-open-the-door-to-cicd-pipeline-attacks
36. ——. Novel Pipeline Vulnerability Discovered; Rust Found Vulnerable. [Online]. Available: https://www.legitsecurity.com/blog/artifact-poisoning-vulnerability-discovered-in-rust
37. A. Khan. Cacheract: The Monster in your Build Cache. [Online]. Available: https://adnanthekhan.com/2024/12/21/cacheract-the-monster-in-your-build-cache/
38. J. Lobacevski. Producing artifacts. [Online]. Available: https://slsa.dev/spec/v1.0/requirements#isolated
39. G. Cardoen, "A dataset of GitHub Actions workflow histories," Oct. 2024, version Number: 2024-10-25. [Online]. Available: https://doi.org/10.5281/zenodo.13985548