Mitigating Security Issues in GitHub Actions

Hassan Onsori Delicheh
hassan.onsoridelicheh@umons.ac.be
University of Mons
Mons, Belgium

Tom Mens
tom.mens@umons.ac.be
University of Mons
Mons, Belgium

ABSTRACT

Collaborative practices have revolutionised the software development process, enabling distributed teams to seamlessly work together. Social coding platforms have integrated CI/CD automation workflows, with GitHub Actions emerging as a prominent automation ecosystem for GitHub repositories. While automation brings efficiency, it also introduces security challenges, often related to software supply chain attacks and workflow misconfigurations. We outline the security issues associated with the software supply chain of GitHub Actions workflows, most notably their reusable Actions and their dependencies. We also explore the security risks associated with misconfigurations of repositories and workflows, such as poor permission management, command injection, and credential exposure. To mitigate these risks we suggest practical remediations, including dependency and security monitoring, pinning Actions, strict access control, verified creator practices, secret scanning tools, raising awareness, and training. In doing so, we provide valuable insights on the need to integrate security seamlessly into the automated collaborative software development processes. To enhance the security of workflow automation within GitHub repositories we encourage a proactive approach and advocate for the adoption of best practices.

KEYWORDS

GitHub Actions, collaborative software development, workflow automation, security risk, software supply chain

ACM Reference Format:
Hassan Onsori Delicheh and Tom Mens. 2024. Mitigating Security Issues in GitHub Actions. In *2024 ACM/IEEE 4th International Workshop on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and 2024 IEEE/ACM Second International Workshop on Software Vulnerability (EnCyCriS/SVM '24), April 15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3643662.3643961

1 INTRODUCTION

Collaborative software development practices have transformed modern development processes, enabling distributed teams to collaborate effortlessly in a multitude of tasks related to coding, debugging, testing, quality and security analysis, packaging, releasing and deploying software. Managing such a diverse range of activities requires the integrated usage of tools such as version control systems, software distribution managers, bug and issue trackers, and vulnerability and dependency analysers. To facilitate their integration, automation workflows or pipelines were introduced, making continuous integration, deployment, and delivery (CI/CD) a foundational aspect of collaborative DevOps practices. While CI/CD services such as Travis or Jenkins have been widely use for over a decade, a new generation has become tightly integrated into social coding platforms such as GitHub and GitLab, effectively revolutionising the landscape of development workflow automation [12].

GitHub is the largest social coding platform today, hosting millions of software repositories and serving over 100 million users as of November 2023. GitHub officially released GitHub Actions in November 2019, providing CI/CD support to enable GitHub repository maintainers to automate their workflows directly within the GitHub platform. The tight integration into GitHub offers significant advantages in terms of increased efficiency and productivity, but also creates new entry points for attackers, significantly expanding the attack surface and making repositories more susceptible to security breaches. With numerous collaborators and a diverse codebase, the risk is amplified further, emphasising the urgency to address this concern.

Unfortunately, security does not seem to be a primary concern for GitHub repository maintainers, especially when it comes to the adoption of GitHub Actions [3, 8]. Creating, building, testing, and deploying software are often carried out without ensuring security at every stage of the process. The failure to incorporate security into the workflows exposes various risks to the software project. Security should therefore be regarded as an important and integral part of the automation process, rather than being treated as a second-class activity. DevSecOps is emerging as an approach to incorporate efficient identification and resolution of security issues at the heart of DevOps, promoting collaboration among development, operation, and security teams [27, 31]. However, the effectiveness of this approach is still uncertain [24].

This position paper presents our views on security issues related to automation in GitHub repositories and recommends remediations based on our recent empirical insights in the use of GitHub Actions [8, 9, 12, 25, 26, 28]. We believe that these insights provide a valuable resource for researchers, workflow maintainers, repository contributors and tool developers seeking to comprehend, evaluate and mitigate security issues within the GitHub Actions workflow automation ecosystem.

2 ABOUT GITHUB ACTIONS

The GitHub Actions workflow automation seamlessly integrates into GitHub, empowering repository maintainers to automate a wide range of activities. Adhering to the configuration-as-code

EnCyCriS/SVM '24, April 15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0565-6/24/04…$15.00
https://doi.org/10.1145/3643662.3643961

paradigm, workflow code is articulated through YAML files stored in the `.github/workflows` directory of a repository.

Workflows can be triggered by various *events* such as a cron schedule, submitted pull requests, pushed commits, opened or closed issues, code reviews, and so on. Each workflow executes one or more *jobs* that run on a virtual machine called a *runner*, which can be hosted by GitHub with preconfigured environments, or self-hosted on the user's own servers for greater customisation and control. Jobs contain one or more *steps* that either specify shell commands to be executed on the virtual machine (using the `run:` syntax) or perform their designated task by executing a reusable Action component (using the `uses:` syntax). Workflows can depend on other reusable workflows to avoid duplication by identical workflow code across repositories.

Workflows frequently rely on reusable Action components that are made available on the GitHub Marketplace.[1] In January 2024, the Marketplace hosted over 21K reusable Actions. Such Actions are developed in some GitHub repository and require the presence of a YAML file action.yml stored at the root of the GitHub repository. This file contains essential metadata for executing the Action. Actions can be developed in three distinct ways:

JavaScript Actions enable running JavaScript code in a Node.js runtime environment. They prove invaluable for handling tasks demanding intricate logic or interactions with the GitHub API and external services. Additionally, leveraging JavaScript opens the door to tapping into an extensive array of JavaScript packages available through package managers like npm.

Docker Actions define tasks executed within a Docker container, providing enhanced flexibility and portability in workflow execution by allowing customisation of the environment to align with workflow requirements. The development of a Docker Action involves the creation of a *Dockerfile*, specifying the base image of the container, and the sequence of commands executed on top of this image. Container base images are accessible in container registries such as Docker Hub.

Composite Actions are written in the YAML workflow syntax, and allow to specify the code of one or more workflow steps. This allows to move complex or redundant workflow code into a composite Action, which can also depend on other Actions itself. Composite Actions were introduced by GitHub in August 2020, and their popularity has been growing ever since due to the reusability, customisability, and extensibility they enable [25].

Fig. 1 summarises the GitHub Actions ecosystem, illustrating how GitHub repositories can use workflows, which can themselves reuse other workflows or rely on reusable Actions. These Actions, based on their type, may depend on npm packages, Docker images, or other Actions [8, 32]. The many interdependencies in this ecosystem can lead to security risks in various ways. In the next sections, we outline how such security issues may arise and provide recommendations on how to mitigate these risks.

3 SUPPLY CHAIN ATTACK SURFACE

Reusable open-source components make up a substantial portion of modern software applications. Integrating such reusable components has become a common practice for software producers. This

[1] https://github.com/marketplace?type=actions

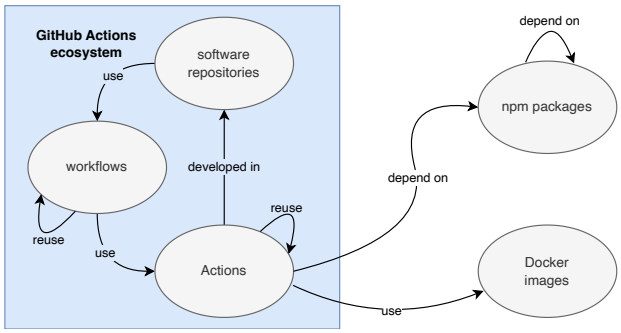

Figure 1: The GitHub Actions ecosystem [8, 32].

trend has given rise to software supply chains, where products consist of both internally controlled core components and externally sourced third-party components. While reuse of third-party components offers numerous advantages [11, 33], it can also result in intricate and interconnected dependency networks [7, 16]. These networks pose significant security risks, becoming intentional targets for attacks in which malicious actors infect vulnerable components, compromising the integrity of build and deployment pipelines. Depending on insecure third-party components has raised notable concerns [1, 4], with well-known incidents such as equifax [22] and Log4Shell [15] illustrating the importance of the problem.

GitHub Actions is not immune to such insecure software supply chains. Having become the predominant workflow automation service on GitHub [12], security concerns within its software supply chain could potentially impact millions of repositories making use of automated workflows. The heavy reliance ofsuch workflows on reusable Action components [9] poses a substantial risk, since malicious actors may intentionally create or modify Actions to compromise workflows and repositories relying on them.

Vulnerable Actions. *Reusable Actions* provide a robust method for streamlining software development workflows, covering tasks such as test execution, code linting, application deployment, and more. There is a potential risk of attackers exploiting and injecting malicious code into an Action release. Action developers may also unintentionally release Actions containing security weaknesses in their code. We used CodeQL to carry out a static analysis of security weaknesses in the latest releases of 8,107 JavaScript Actions [26]. We observed that over 54% of those Actions were affected by at least one security weakness type (CWE). A total of 9,700 weakness occurrences were identified across the 4,409 affected Actions. Seven out of the top ten most frequent security weakness types were associated with *Improper Input Validation* (CWE-20). This common weakness enumeration highlights the inadequacy of an application in accurately validating input data, making it potentially vulnerable to injection attacks. For example, regular expressions are commonly used for input data validation, but they can be prone to errors when attempting to match untrusted input without the use of anchors (i.e., $). Exploiting this vulnerability, malicious input can incorporate permitted patterns in unexpected locations, thereby bypassing security checks.

Vulnerable reusable workflows. *Reusable workflows* align with the DRY ("Dont' Repeat Yourself") principle, eliminating the need

to duplicate similar code across workflows within and across repositories. In a recent empirical investigation, however, we observed that only around 1% of nearly 70K workflows are reusing other workflows [9]. While workflow reuse is clearly beneficial for reducing code duplication, it can lead to an extra level of insecurity, because workflow maintainers need not only assess the security of their own workflows but also consider the reusable workflows they rely on. For instance, all workflows reusing a version below 2.7.5 of the run-terraform workflow (available in the public GitHub repository kartverket/github-workflows providing reusable workflows) are susceptible to a code injection of high severity.[2] An attacker could exploit this vulnerability by submitting a malicious pull request containing a payload, potentially resulting in the execution of arbitrary JavaScript code within the workflow's context.

Outdated Actions. Depending on outdated software components is acknowledged by the OWASP foundation as a top ten security risk.[3] More specifically, outdated dependencies to reusable libraries distributed through package managers (e.g., Maven, npm) increase the risk of becoming subject to vulnerabilities [5, 18, 20]. Reusable Actions undergo continuous updates, leading many automation workflows to rely on outdated versions of Actions. In a recent study we analysed a dataset comprising nearly one million workflows from over 22K+ repositories, finding evidence that it is common practice to reuse Actions in GitHub workflows, and this reuse tends to target a limited set of Actions [8]. We observed that Actions are frequently updated, and that a majority of workflows employ outdated Action releases, lagging behind the latest available release by at least 7 months. Moreover, these workflows missed opportunities for updates spanning at least 9 months. Numerous instances exist where Actions releases have encountered security issues that have been fixed in subsequent releases. Workflows that do not regularly update their reusable Actions could therefore become subject to vulnerabilities that attackers can take advantage of.

Exposure to vulnerable npm packages. JavaScript Actions are the most common type of Actions reused by workflows. Based on a dataset of nearly 31K Actions accessible through the ecosyste.ms API[4] on June 2023, we observed that the majority (53%) have been developed as JavaScript Actions [26], and that most of these Actions depend on npm packages. While such library reuse brings several benefits [11, 33], it also introduces security vulnerabilities [2, 6, 14, 19, 20, 30, 34]. We analysed to which degree vulnerabilities exist within the dependency network of JavaScript Actions, observing that more than 77% of JavaScript Actions releases relied on at least one npm package release with known vulnerabilities. We observed older versions of widely used packages that served as single points of failure because of specific vulnerabilities, posing a risk to a substantial number of JavaScript Actions [26]. We additionally observed that indirect dependencies are prone to high and critical vulnerabilities, requiring Action maintainers to investigate the full dependency tree to assess their security exposure [26]. This underscores the significance for security monitoring tools to take into account such indirect dependencies.

Using vulnerable Docker images. Docker Actions are the second most common type of Actions. They perform their functions by executing a Docker image, ensuring a more predictable and consistent behavior across various environments. This is beneficial for tasks that demand a particular operating system, tool version, or dependency. The Docker image can be retrieved from a registry (e.g., Docker Hub), or be dynamically built on the runner based on a Dockerfile. To set up a Docker Action, the action.yml file should either specify the URL pointing to a Docker image or the file path to a Dockerfile. In the case of relying on Docker images, Docker Hub is the most widely used registry for Docker Actions [25]. Yet, several studies have shown that images on Docker Hub often incorporate multiple packages with known vulnerabilities [21, 23, 29, 35]. This makes Docker Actions that depend on those images potentially vulnerable. In the case of Docker Actions relying on Dockerfiles, other security issues may arise. A Dockerfile configures the settings for the container intended to be used by the Action, including details such as the base image for creating the container and the specific commands to be executed by the container. Security issues may arise when using a Dockerfile within a Docker Action when fetching external resources, for example via the wget command, without verifying their checksum. In the context of software dependencies, a checksum provides a unique identifier for the file. When a file is downloaded, its corresponding checksum can be retrieved from a trusted source. Verifying this checksum is crucial because using the downloaded file without this verification poses risks to its integrity and authenticity, as these files might have been tampered with during transit. This opens the door to security loopholes, allowing malicious actors to exploit the download process and compromise the runner's security. Implementing checksum verification and considering digital signature verification when available help establish trust in external dependencies of Docker Actions, ensuring a more secure and reliable Docker image building process.

Composite Actions define steps that can be used across various workflows, promoting code reuse and minimising duplication. Additionally, they allow to customise specific workflows by combining shell commands and reusable Actions, allowing to incorporate additional functionality or steps. However, the use of composite Actions may further increase the attack surface. Any security issues within the dependency chain have an increased potential to make multiple workflows vulnerable. For instance, a security issue may arise when a composite Action depends on some vulnerable outdated Action version. This may make the composite Action vulnerable itself, and this vulnerability may continue to propagate through the supply chain, affecting all workflows that depend on this composite Action. In their turn, those workflows introduce vulnerabilities in the software repositories on which they are run, possibly even affecting the applications being built and deployed by those repositories.

4 MISCONFIGURATION ISSUES

Improperly configured GitHub repositories and workflows can pose significant security risks by unintentionally introducing vulnerabilities in several ways. Below we present some of these risks.

Permission management misconfiguration. Permissions misconfigurations have the potential to expose sensitive data or provide unauthorised access. When a workflow is executed, GitHub generates a short-lived token for interacting with the repository. This token can become a target for attackers who manage to execute

[2] https://nvd.nist.gov/vuln/detail/CVE-2022-39326
[3] https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components
[4] https://packages.ecosyste.ms/registries

code within a workflow, as it grants them equivalent permissions to the token itself. The tokens assigned to each workflow are configured by repository maintainers possessing read or write access to a defined set of scopes. If not explicitly defined in the workflow configuration file, the repository's default settings dictate the permissions. Realising the risk of too permissive tokens, in February 2023 GitHub changed the default *write-all* permissions across all available scopes to *read-only* permissions for the contents, packages, and metadata scopes. In a similar way, it is good practice to minimise permissions of each workflow. GitHub Actions provides support for *write-all* and *read-all* scopes, allowing overly excessive permissions to reusable Actions that can potentially lead to exploitable vulnerabilities. For instance, the *actions:write* permission enables the creation or modification of workflows. It is therefore crucial to explicitly restrict permissions for each job within a workflow, to ensure that the permissions align with the specific needs of any Action used in the job, and to isolate the use of an Action from the remainder of the workflow.

A related problem is that Actions lack a formal permission approval process. Furthermore, there is no centralised location, whether on GitHub or elsewhere, to identify the specific permissions required by an Action. This makes it quite challenging for workflow maintainers to adhere to the crucial principle of *least privilege*.

In 2022, Koishybayev et al. [17] conducted a thorough examination of 447,238 workflows across 213,854 GitHub repositories, revealing revealed that a whopping 99.8% of these workflows exhibit excessive privileges, providing read-write access to the repository rather than read-only access. An analysis[5] of the top 1,000 widely-used Actions on the GitHub Marketplace revealed that approximately 50% of them do not engage in any interactions with the repository, rendering the use of a GitHub token unnecessary for their intended functionality. Additionally, it was observed that a staggering 93% of 2,000 popular open-source projects relying on GitHub Actions had at least one overly permissive workflow.

Command injection. A workflow can be triggered by specific events within a repository. Each trigger comes with a GitHub context[6] that includes metadata about the triggering event, such as the initiating user, branch name, and other relevant contextual information. An extensive amount of such event context data could potentially be exploited by an attacker and should therefore be regarded as untrustworthy input (e.g., github.event.issue.body and github.event.comment.body). Similar to any program initiated by an external user, caution must be exercised with user-supplied data, treating it as potentially untrustworthy, ensuring that it does not inadvertently find its way into API calls where it could be treated as executable code. GitHub Actions incorporate a potent, language-independent feature in expression evaluation, which, if utilised within a *run:* execution, could be susceptible to command injections. Such risk of injection is not confined to shell commands, since the expression evaluator is not language-specific. For instance, the use of ${{ }} in JavaScript code could permit the injection of a syntactically valid construct.[7] The consequences of exploiting an injection vulnerability can be extremely severe. Attackers might, for instance, leverage this vulnerability to upload sensitive information to a website they control, or introduce new code to the repository [13], thereby creating a backdoor vulnerability or initiating a supply chain attack.

Credential exposure. Automation tools often require access to API keys or other credentials to interact with external services, such as deploying to cloud platforms, accessing databases, or integrating with third-party APIs. These credentials are commonly stored as secrets within GitHub repositories. Improper handling of these secrets in workflows or Actions can result in exposure, potentially granting unauthorised access to repositories [13]. This aligns with a top ten OWASP CI/CD security risk "Insufficient Credential Hygiene".[8] Inadequate encryption or configuration of secrets in Actions may expose them in logs or error messages during workflow execution. Misconfigurations in workflow files, especially if accidentally committed to the repository, can render secrets visible to anyone with repository access. Another scenario occurs during workflow execution when GitHub runner logs provide information about each step. If steps contain debugging statements or inadvertently log environment variables, secrets may be exposed. Additionally, workflows might use secrets in ways that accidentally expose them, such as when a script within a workflow unintentionally prints or echoes a secret, making it visible in the workflow logs.

5 RECOMMENDED REMEDIATIONS

We recommend several mitigation strategies to reduce the security risks induced by the use of GitHub Actions:

Use dependency and security monitoring. Enck and Williams [10] identified two main challenges in ensuring supply chain security: managing vulnerable dependencies and selecting secure reusable components. One could address these challenges by resorting to static or dynamic code analysis tools that identify security weaknesses in the code. `CodeQL` is an example of such a tool, integrated by GitHub, and supporting the analysis of security (and other) weaknesses in the code of reusable Actions. `actionlint`[9] is another example of such a tool, supporting static analysis of the YAML-based workflow code itself. Additionally, Software Composition Analysis (SCA) tools can be used for dependency and security monitoring, empowering repository maintainers to keep their dependencies current and mitigate the risk of known vulnerabilities. Examples of such tools are GitHub's Dependabot[10], Snyk[11], OWASP Dependency Check[12], trivy[13], and Checkov.[14] These tools play a crucial role in identifying and reporting vulnerabilities in the supply chain of reusable GitHub workflows and Actions by scanning their dependency networks and cross-referencing with advisory databases like the National Vulnerability Database (NVD) and the GitHub Advisory Database. This proactive approach enables maintainers to receive security alerts in their dependencies.

[5] https://www.paloaltonetworks.com/blog/prisma-cloud/github-actions-opt-out-permissions-model
[6] https://docs.github.com/en/actions/learn-github-actions/contexts#github-context
[7] https://github.blog/2023-08-09-four-tips-to-keep-your-github-actions-workflows-secure
[8] https://owasp.org/www-project-top-10-ci-cd-security-risks/CICD-SEC-06-Insufficient-Credential-Hygiene
[9] https://github.com/rhysd/actionlint
[10] https://docs.github.com/en/code-security/dependabot
[11] https://snyk.io
[12] https://owasp.org/www-project-dependency-check
[13] https://github.com/aquasecurity/trivy
[14] https://github.com/bridgecrewio/checkov

Action pinning. To reuse an Action in a workflow, the format owner/repo@ref needs to be used, where the ref key can either point to a branch (@main), a tag (@v1), or a commit hash (@8f4b7848644...). GitHub advocates using commit hashes to harden the security of reusing Actions.[15] By referencing a specific commit through its hash, confidence is established in using the exact version that underwent prior review and approval, This guarantees data integrity across the entire pipeline, thereby reducing the risk of supply chain attacks. This practice addresses a top ten OWASP CI/CD security risk "Improper Artifact Integrity Validation".[16] In the event of a compromised Action where malicious code is inserted into a newer release, workflows relying on secure previous releases, identified by their commit hashes, remain unaffected, providing an additional layer of protection against this type of supply chain attack. However, if Actions are not implemented carefully, they can easily undermine the practice of Action pinning. Docker images, used to run Actions pulled from a registry or built on the fly by the runner through a Dockerfile, are susceptible to issues when fetching external resources without verifying their checksum. This practice contradicts the expected security of Action pinning. Composite Actions enable the calling of other Actions, posing a risk if attackers modify the Actions that the pinned composite Action relies on. In such cases, the pinned composite Action remains vulnerable to code execution. JavaScript Actions are more robust to breaking the security induced by Action pinning, due to the absence of a runtime package installation process for JavaScript Actions. However, they can still fetch external resources at runtime. If the code downloads an external script without verifying its checksum, it opens the possibility for a new version to overwrite the script automatically, leading the Action to use the updated version. An analysis of the 1,000 most-starred Actions on GitHub Marketplace revealed that 32% of these Actions are *unpinnable* in the sense that, even if a workflow pins such Actions, the pinning is likely not offering the anticipated protection. An examination of 6,000 workflows in 2,000 highly-starred public open-source projects further disclosed that 67% of these projects had pinned *unpinnable* Actions.[17]

In short, while Action pinning ensures the immutability of the Action's code stored in its hosting repository, it fails to guarantee the immutability of the Action's dependencies and external resources. The entire dependency tree, encompassing container images, binaries, and other Actions, remains unprotected, potentially enabling the execution of malicious code.

Implement strict access control and permission settings. We recommend to implement the principle of *least privilege* in GitHub Actions workflows, although it can potentially disrupt established workflows. To address this challenge, the GitHub Security Lab has introduced the *GitHub Token Permissions Monitor and Advisor Actions*.[18] By integrating this Action into a workflow, it monitors the utilisation of the temporary GitHub repository token and provides suggestions on the minimal permissions needed for the workflow, based on the observed workflow activity.

Use Actions from verified creators. GitHub recommends Actions from the GitHub Marketplace with a "verified creator" badge, signifying that the Action was created by a team with a verified identity, hence adding a layer of trust to the source. Prioritising such Actions enhances the reliability and security of workflows, minimising potential risks associated with Actions maintained by untrusted developers. Despite this recommendation, an analysis of 447,238 workflows revealed that 97% of them used unverified reusable Actions [17].

Avoid using self-hosted runners in public repositories. During the setup of CI workflows, each workflow specifies its execution environment. GitHub offers a variety of runners (e.g., Ubuntu, Mac, and Windows) in the cloud, and their use creates a clean virtual machine on each occasion. An alternative exists to use self-hosted runners. It is crucial to highlight the importance of securing such runners, or to refrain from using self-hosted runners for public repositories to mitigate the potential threat posed by malicious pull requests. An important consequence of a compromised runner is the ability to merge arbitrary code into the main branch, which may go unnoticed until deployment or even production.[19]

Use secret scanning tools. Secret scanning tools are imperative to help enhance the security of GitHub repositories. Proper configuration and secure settings for project repositories and their workflows should be ensured. To safeguard sensitive information, encryption should be employed, and best practices in configuration management should be adhered to. A dedicated secret management feature is provided by GitHub, serving as a secure vault for storing and managing sensitive credentials. Refraining from hardcoding or exposing secrets directly within the code is crucial, as this practice poses a significant security risk. To bolster defense against potential vulnerabilities, we recommend to set up and use GitHub's secret scanning tool.[20] By doing so, exposed secrets can be proactively identified and addressed, leveraging the tool's capability to generate alerts and thereby improving the overall security posture.

Raise awareness and conduct training sessions. It is important to foster awareness concerning the far-reaching consequences of security issues, and to cultivate a profound understanding of its importance among practitioners. This involves conducting comprehensive training sessions to impart knowledge about best practices for identifying and resolving security issues during workflow automation. It is important to emphasise the significance of recognising potential threats and vulnerabilities early in the development process, thereby encouraging a proactive approach to security. Software maintainers need to be equipped with the necessary skills and tool sets to effectively address and mitigate security challenges. By prioritising both awareness-raising initiatives and continuous training, practitioners can establish a robust security culture ingrained in every aspect of their development practices.

6 CONCLUSION

In this position paper we shed light on security issues associated with the automation of GitHub Actions workflows in GitHub repositories. A first major concern is the significantly increased supply chain attack surface that GitHub Actions brings along. GitHub

[15] https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions
[16] https://owasp.org/www-project-top-10-ci-cd-security-risks
[17] https://www.paloaltonetworks.com/blog/prisma-cloud/unpinnable-actions-github-security
[18] https://github.com/GitHubSecurityLab/actions-permissions
[19] https://www.praetorian.com/blog/self-hosted-github-runners-are-backdoors
[20] https://docs.github.com/en/code-security/secret-scanning

repositories can rely on workflows, that can rely on reusable workflows or reusable Actions, as well as depend on third-party components such as npm packages and Docker images. Each of these dependencies can give rise to potential vulnerabilities, and the interconnected nature of this complex supply chain makes workflows susceptible to intentional attacks by malicious actors.

A second area of concern is the misconfiguration of GitHub repositories and their automated workflows, encompassing issues such as excessive permissions, susceptibility to command injection, and credential exposure. We therefore stress the need for meticulous configuration to prevent unintended vulnerabilities, especially in terms of permission scopes and the handling of sensitive information like access tokens and API keys.

To mitigate such security risks related to GitHub's workflow automation, we recommend a set of remediations. These include the continuous use of dependency and security monitoring, pinning Actions to specific commits, implementing strict access control and permission settings, prioritising Actions with verified creators, employing secret scanning tools, and promoting awareness and training among practitioners. In essence, we emphasise the urgency of integrating security considerations seamlessly into the automated workflows. We advocate a proactive and comprehensive approach to identify, address, and mitigate security issues throughout the continuous collaborate software development process. By adopting the proposed remediations and fostering a security-conscious culture, GitHub repository maintainers can fortify their defenses against potential threats, thereby enhancing the overall security posture of their software repositories and associated automation workflows.

ACKNOWLEDGMENTS

This research is supported by the Fonds de la Recherche Scientifique - FNRS under grant numbers T.0149.22 and J.0147.24.

REFERENCES

[1] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *Symp. Security and Privacy*. IEEE, 289–305. https://doi.org/10.1109/SP.2016.25

[2] M. Alfadel, D. E. Costa, and E. Shihab. 2021. Empirical Analysis of Security Vulnerabilities in Python Packages. In *Int'l Conf. Software Analysis, Evolution and Reengineering*. https://doi.org/10.1109/saner50967.2021.00048

[3] F. Angermeir, M. Voggenreiter, F. Moyón, and D. Méndez. 2021. Enterprise-Driven Open Source Software: A Case Study on Security Automation. In *Int'l Conf. Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 278–287. https://doi.org/10.1109/ICSE-SEIP52600.2021.00037

[4] M. Chen, F. Fischer, N. Meng, X. Wang, and J. Grossklags. 2019. How Reliable is the Crowdsourced Knowledge of Security Implementation?. In *Int'l Conf. Software Engineering*. 536–547. https://doi.org/10.1109/ICSE.2019.00065

[5] J. Cox, E. Bouwers, M. van Eekelen, and J. Visser. 2015. Measuring Dependency Freshness in Software Systems. In *Int'l Conf. Software Engineering*. IEEE, 109–118. https://doi.org/10.1109/ICSE.2015.140

[6] A. Decan, T. Mens, and E. Constantinou. 2018. On the impact of security vulnerabilities in the npm package dependency network. In *Int'l Conf. Mining Software Repositories*. 181–191. https://doi.org/10.1145/3196398.3196401

[7] A. Decan, T. Mens, and P. Grosjean. 2019. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empir. Softw. Eng.* 24, 1 (2019), 381–416. https://doi.org/10.1007/s10664-017-9589-y

[8] A. Decan, T. Mens, and H. Onsori Delicheh. 2023. On the outdatedness of workflows in the GitHub Actions ecosystem. *J. Syst. Softw.* 206 (2023). https://doi.org/10.1016/j.jss.2023.111827

[9] A. Decan, T. Mens, P. Rostami Mazrae, and M. Golzadeh. 2022. On the Use of GitHub Actions in Software Development Repositories. In *Int'l Conf. Software Maintenance and Evolution*. IEEE. https://doi.org/10.1109/ICSME55016.2022.00029

[10] W. Enck and L. Williams. 2022. Top Five Challenges in Software Supply Chain Security: Observations From 30 Industry and Government Organizations. *IEEE Security and Privacy* 20, 2 (2022), 96–100. https://doi.org/10.1109/MSEC.2022.3142338

[11] W. B. Frakes and K. C. Kang. 2005. Software reuse research: status and future. *Trans. Softw. Eng.* 31 (2005), 529–536. https://doi.org/10.1109/TSE.2005.85

[12] M. Golzadeh, A. Decan, and T. Mens. 2021. On the rise and fall of CI services in GitHub. In *Int'l Conf. Software Analysis, Evolution and Reengineering*. IEEE. https://doi.org/10.1109/SANER53432.2022.00084

[13] Y. Gu, L. Ying, H. Chai, C. Qiao, H. Duan, and X. Gao. 2023. Continuous Intrusion: Characterizing the Security of Continuous Integration Services. In *Symp. Security and Privacy*. IEEE, 1561–1577. https://doi.org/10.1109/SP46215.2023.10179471

[14] J. Hejderup, M. Beller, K. Triantafyllou, and G. Gousios. 2022. Präzi: from package-based to call-based dependency networks. *Empir. Softw. Eng.* 27, 5 (2022), 102. https://doi.org/10.1007/s10664-021-10071-9

[15] R. Hiesgen, M. Nawrocki, T. C. Schmidt, and M. Wählisch. 2022. The Race to the Vulnerable: Measuring the Log4j Shell Incident. *ArXiv* abs/2205.02544 (2022). https://doi.org/10.48550/arXiv.2205.02544

[16] R. Kikas, G. Gousios, M. Dumas, and D. Pfahl. 2017. Structure and Evolution of Package Dependency Networks. In *Int'l Conf. Mining Software Repositories*. 102–112. https://doi.org/10.1109/MSR.2017.55

[17] I. Koishybayev, A. Nahapetyan, R. Zachariah, S. Muralee, B. Reaves, A. Kapravelos, and A. Machiry. 2022. Characterizing the Security of Github CI Workflows. In *USENIX Security Symposium*.

[18] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue. 2018. Do developers update their library dependencies? *Empir. Softw. Eng.* 23, 1 (2018), 384–417. https://doi.org/10.1007/s10664-017-9521-5

[19] T. Lauinger, A. Chaabane, and C. B. Wilson. 2018. Thou shalt not depend on me. *Comm. ACM* 61, 6 (2018), 41–47. https://doi.org/10.1145/3190562

[20] C. Liu, S. Chen, L. Fan, B. Chen, Y. Liu, and X. Peng. 2022. Demystifying the Vulnerability Propagation and Its Evolution via Dependency Trees in the NPM Ecosystem. In *Int'l Conf. Software Engineering*. 672–684. https://doi.org/10.1145/3510003.3510142

[21] P. Liu, S. Ji, L. Fu, K. Lu, X. Zhang, W.-H. Lee, T. Lu, W. Chen, and R. A. Beyah. 2020. Understanding the Security Risks of Docker Hub. In *European Symp. Research in Computer Security*. https://doi.org/10.1007/978-3-030-58951-6_13

[22] J. Luszcz. 2018. Apache Struts 2: how technical and development gaps caused the Equifax breach. *Network Security* 1 (2018), 5–8. https://doi.org/10.1016/S1353-4858(18)30005-9

[23] A. Mills, J. Jonathan, and P. Legg. 2023. Longitudinal Risk-based Security Assessment of Docker Software Container Images. *Computers & Security* 135 (2023). https://doi.org/10.1016/j.cose.2023.103478

[24] H. Myrbakken and R. Colomo Palacios. 2017. DevSecOps: A Multivocal Literature Review. In *Int'l Conf. Software Process Improvement and Capability Determination*. https://doi.org/10.1007/978-3-319-67383-7_2

[25] H. Onsori Delicheh, A. Decan, and T. Mens. 2023. A Preliminary Study of GitHub Actions Dependencies. In *Post-proceedings of the 15th Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE)*,, Vol. 3483. CEUR Workshop Proc., 66–77.

[26] H. Onsori Delicheh, A. Decan, and T. Mens. 2024. Quantifying Security Issues in Reusable JavaScript Actions in GitHub Workflows. In *Int'l Confn Mining Software Repositories*.

[27] R. N. Rajapakse, M. Zahedi, M. A. Babar, and H. Shen. 2022. Challenges and solutions when adopting DevSecOps: A systematic review. *Inf. Softw. Technol.* 141 (2022). https://doi.org/10.1016/j.infsof.2021.106700

[28] P. Rostami Mazrae, T. Mens, M. Golzadeh, and A. Decan. 2023. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. *Empir. Softw. Eng.* 28, 2 (2023), 52. https://doi.org/10.1007/s10664-022-10285-5

[29] R. Shu, X. Gu, and W. Enck. 2017. A Study of Security Vulnerabilities on Docker Hub. In *Conf. Data and Application Security and Privacy (CODASPY)*. ACM, 269–280. https://doi.org/10.1145/3029806.3029832

[30] H. H. Thompson. 2003. Why Security Testing Is Hard. *IEEE Security and Privacy* 1, 4 (2003), 83–86. https://doi.org/10.1109/MSECP.2003.1219078

[31] N. Tomas, J. Li, and H. Huang. 2019. An Empirical Study on Culture, Automation, Measurement, and Sharing of DevSecOps. In *Int'l Conf. Cyber Security and Protection of Digital Services*. IEEE.

[32] M. Wessel, T. Mens, A. Decan, and P. Rostami Mazrae. 2023. The GitHub Development Workflow Automation Ecosystems. In *Software Ecosystems: Tooling and Analytics*. Springer, 183–214. https://doi.org/10.1007/978-3-031-36060-2_8

[33] L. Williams. 2022. Trusting Trust: Humans in the Software Supply Chain Loop. *IEEE Security and Privacy* 20, 5 (2022), 7–10. https://doi.org/10.1109/MSEC.2022.3173123

[34] A. Zerouali, T. Mens, A. Decan, and C. De Roover. 2022. On the impact of security vulnerabilities in the npm and RubyGems dependency networks. *Empir. Softw. Eng.* 27, 5 (2022), 1–45. https://doi.org/10.1007/s10664-022-10154-1

[35] A. Zerouali, T. Mens, A. Decan, J. M. Gonzalez-Barahona, and G. Robles. 2021. A multi-dimensional analysis of technical lag in Debian-based Docker images. *Empir. Softw. Eng.* 26 (2021). https://doi.org/10.1007/s10664-020-09908-6