2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW) | 979-8-3503-2720-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/EuroSPW59978.2023.00007

A Preliminary Study of Privilege Life Cycle in Software Management Platform Automation Workflows

Giacomo Benedetti
University of Genoa
Genoa, Italy
giacomo.benedetti@dibris.unige.it

Luca Verderame
University of Genoa
Genoa, Italy
luca.verderame@unige.it

Alessio Merlo
CASD - Centre for High Defense Studies
Rome, Italy
alessio.merlo@casd.difesa.it

***Abstract*—This paper focuses on the role of privileges in automation workflows within modern software development practices, which heavily rely on DevOps principles. Automation workflows, which are sets of automated software management processes, have become essential to software development and are integrated into software management platforms such as GitHub, GitLab, and BitBucket. However, privileges are crucial in ensuring the security and integrity of the software development process. This paper aims to identify the phases in which privileges are involved in automation workflows and analyze how these platforms handle the privilege life cycle in automation workflows to provide a better understanding of their security implications. The security discussion highlighted in this analysis aims to stimulate solutions and further research.**

1. Introduction

Modern software development is based on the DevOps development approach [15]. DevOps relies on a range of tools to improve both the development phase (Dev), e.g., code management, and the operational phase (Ops), e.g., deployment, monitoring, and logging [33], [32]. These tools help to ensure that software is delivered quickly, reliably, and with minimal risk of errors or downtime. Automation tools (e.g., Jenkins [28]) have been instrumental in sustaining the principles of DevOps over the years.

However, software management platforms recently introduced automation workflows directly on their repositories [30]. An *automation workflow* is a set of steps that automate the management of software deployments, updates, and other operations in a streamlined and consistent manner. Automation workflows are designed to autonomously execute a series of actions (i.e., a task) when the developer invokes or responds to an event (i.e., a triggering condition).

Automation workflows on software management platforms are a crucial component of the DevOps process, as they help to automate and streamline the software delivery pipeline. Indeed, the possibility of applying automation without relying on external services pushed the community to thrive on this feature [14].

Thus, workflows rapidly became an essential aspect of modern software development practices affecting development throughout the development process, from code writing to release.

Privileges are crucial for automation workflows, as they allow ensuring the security and integrity of the software development process. Automation workflows involve a range of tasks, such as building, testing, and deploying software, that requires access to sensitive resources. Privileges determine a user or entity's actions on a particular resource. An entity is a non-human actor that can interact with the software management platform, e.g., a cloud application. Automation workflows can expose these resources without appropriate privileges to unauthorized access or modification, leading to serious security and compliance issues, as discussed in [26], [38]. In particular, recent work on GitHub Workflows [10] highlighted the importance of detecting privilege misconfigurations in their security assessment. Also, the authors evaluated more than 100k workflows discovering that 62% of them do not follow the GitHub security guidelines regarding permissions [20], thereby increasing the chance of attacks against the repository resources, e.g., the code hosted in the repository and the build process.

The industry leaders for software management platforms are GitHub [18], GitLab [21], and BitBucket [1]. These software management platforms share many similarities. However, their privileges and permissions management mechanisms differ in some aspects. Over the year, the documentation for these mechanisms received updates and patches. It grew stratified and sparse, making obtaining the proper knowledge of managing privileges challenging.

This paper aims to define *a rationalization of the privilege system used by automation workflows*. Defining the privilege life cycle in the automation workflows enabled us to categorize the fundamental steps involved in the configuration, authentication, and resolution of privileges during the life span of an automation workflow. We used the privilege life cycle to investigate and compare the different approaches to privilege management for automation workflows. Thanks to a clearer understanding of the privilege management involvement in automation workflows, we discuss the advantages and the potential pitfalls identified in the different approaches used on the software management platforms.

Structure of the paper Section 2 describes the different stages of the privilege life cycle and how privileges and permissions are involved in automation workflows. In Section 3, we analyze the privilege life cycle handling of the three most widely used software management platforms. In Section 4, we discuss the security implications of privilege life cycle studied implementations. Section 5 presents related work regarding automation workflows in

DOI 10.1109/EuroSPW59978.2023.00007

software management platforms. Finally, we conclude this work in Section 6.

2. Privilege Life Cycle in Automation Workflows

Software management platforms allow the creation, storage, and management of repositories. Software developers can store and manage a project's code and related assets in a repository. Repositories provide a structured way to organize and version control the code and assets, enabling collaboration between developers working on the same project. A repository can be created by a single user, namely the Owner, or by an Organization, i.e., a business-shared account. The repository owner allows other users to access the repository by assigning them specific privileges mapped as *roles*. According to the given role, other users can access privileged operations, e.g., writing on the repository or creating automation procedures called automation workflows.

An automation workflow is a procedure to automate tasks such as building, testing, and deploying software packages. Software management platforms enable developers to define workflows as a series of steps or tasks that will be executed on the repository. Because of its nature, an automation workflow can interact with contents hosted on the repository and elements external to the repository environment, e.g., container image repositories. All the operations affecting the repository are managed through the software management platform APIs. An automation workflow is usually described in one or more specification files written with human-readable data serialization languages (e.g., YAML [9]) or other programming languages like JavaScript and TypeScript.

Software management platforms offer various methods to configure privileges to automation workflow, e.g., with user roles, repository settings, or directly on automation workflow specification files. Privileges are expressed as permissions that can be granted to specific users or groups. Users' access to repository resources is regulated with permissions. Users with the necessary privileges can define permissions for other users and entities for automation workflow execution.

Permissions define both the capability for a user to trigger an automation workflow and which actions the workflow can execute. Indeed, a common approach is to run the workflow with the same privileges as the user that triggered the execution.

However, these privileges can be elevated or restricted depending on the other configuration methods available. For example, a user *A* with reading permission triggers the workflow *B*. However, a task in B requires the workflow to access the repository with *write* permission. The workflow will execute correctly if and only if the developer of B has included in the workflow specification file the *write* permission. If this is the case, *B* inherits the *read* permission from A and the *write* permission directly from its specification.

In our analysis, we modeled the involvement of privileges and permissions in the automation workflow life span with a life cycle composed of three different stages, i.e., *privilege configuration*, *automation workflow triggering*, and *privilege resolution*. Table 1 lists the stages and the operations happening in each stage. The rest of this section details the resulting life cycle.

Stages	Operations
(A) Privilege Configuration	*(A.1)* Roles configuration *(A.2)* Permissions definition
(B) Automation Workflow Triggering	*(B.1)* Trigger point stimulation *(B.2)* Authentication for triggering
(C) Privilege Resolution	*(C.1)* Permissions resolution

TABLE 1. STAGES OF PRIVILEGE LIFE CYCLE IN AUTOMATION WORKFLOWS.

2.1. Privilege Configuration

The first stage of the privilege life cycle deals with the configuration of privileges. An automation workflow executes successfully only if it has the set of permissions necessary to *(i)* invoke the API provided by the software management platform and *(ii)* access the repository resources.

Depending on the software management platform, permissions and privileges can be defined:

- *outside the automation workflow*. This approach relies on the concept of role. A role grants privileges associated with a set of permissions. Software management platforms allow dealing with roles at different levels of granularity, e.g., organization, groups of repositories, and single repositories.
- *inside the automation workflow*. In this case, developers can directly define permissions in the specification file that apply to the workflow or event limited to specific parts (e.g., at a job or step level).

In both ways, privileges granularity is an essential aspect in defining permissions. It depends on the specificity of resources; the more a resource is specific to a scope, the more the permission can be fine-grained.

During this stage, a developer should evaluate the expected automation workflow goals to understand which resources are needed by the workflow. A developer can obtain the required knowledge through the software management platform documentation. In detail, it needs to understand how the software management platform *(i)* manages privilege roles, *(ii)* handles permissions and resources, and *(iii)* provides methods to modify privileges of workflows.

2.2. Automation Workflow Triggering

The automation workflow triggering stage regards identifying the starting conditions of automation workflows. The starting conditions can be divided into *event-driven triggers* and *manual triggers*.

Event-driven triggers collect all the events affecting the repository and its configuration. Those events include, for example, push and pull requests, issues opening, and timed events. In this case, the workflow activation depends on the privileges and permissions required for a user or entity to fire one of its triggering events successfully. For example, an automation workflow triggered by a push event needs a user or entity with enough permissions to perform the push on the repository.

Manual triggers allow users and entities to start an automation workflow on-demand. In this case, the successful start of the automation workflow depends on the privileges of the user or entity to access the workflow. Software management platforms use tokens to authenticate users and entities on the repositories and to determine their privileges. The most common tokens are *user tokens* and *access tokens*.

User tokens establish the user's identity during the automation workflow execution and match the permissions granted to the user or entity. Once authenticated, the automation workflows inherit the permissions granted to the user.

Access tokens, on the other hand, are specific to individual repositories and are single-purpose. They grant the permissions that were assigned at the time of their creation.

The permissions contained in the tokens are inherited by the workflow unless overwritten by the permissions defined directly in the workflow specification file.

In this stage, the developer should check *(i)* the starting conditions the software management platform supports, *(ii)* the available means of authentications, and *(iii)* how the workflow inherits or overwrites the permissions of the token.

2.3. Privilege Resolution

The pipeline's user or entity is authenticated and authorized to start the workflow during the triggering stage. The running automation workflow then interacts with the platform to access the resources required to execute the workflow steps using the available APIs.

The platform evaluates the permissions of the authenticated user or entity to determine whether they have the necessary rights to access the requested resources. The resolution process is based on the policies adopted by the platform, which define the rules and criteria for granting or denying access to resources.

The policies implemented by the platform dictate the precedence order of permission levels. The evaluation process may involve multiple levels of permissions. The platform evaluates each permission level in the order defined by the policies to determine the level of access granted to the user or entity that triggered the pipeline.

In this stage, it is necessary to understand how the software management platform applies privilege resolution, considering precedence orders and granting access to resources.

3. Software Management Platforms and their approaches to privilege-granting

Software management platforms provide methods for managing the privilege life cycle in automation workflows. This section analyzes the privilege life cycle management for GitHub, Gitlab, and BitBucket software management platforms.

3.1. GitHub

GitHub provides *GitHub Actions* [12], [19] as automation technology on its platform. GitHub Actions was released in 2019 and gained attention because of its workflow definition and usage flexibility. A workflow is defined by a YAML file containing a set of operations called jobs. Each job will execute a sequence of tasks called steps. GitHub Actions also support the use and integration of external workflows, called reusable workflows, which allow developers to integrate third-party automation workflows to execute already defined jobs. Reusable workflows can be defined using YAML, JavaScript/TypeScript, or Docker.

(A.1) Roles configuration. GitHub distinguishes between personal repositories and organization-managed repositories. Personal repositories have only two roles: *(i)* the owner and *(ii)* collaborators. Organization-managed repositories, instead, have multiple roles that are (from the least privileged to the most privileged): *(i)* read, *(ii)* triage, *(iii)* write, *(iv)* maintain, and *(v)* admin (see Table 2). Additional roles can be created with the Enterprise plan offered by GitHub, allowing for a more granular permissions control for roles. Organizations can set base permissions for all organization members. Base permissions are automatically granted to users when added to the organization.

(A.2) Permissions definition. GitHub organizes repositories in scopes managing specific resources (e.g., file resources are in the *contents* scope), as reported in Table 3.

GitHub allows configuring workflow permissions over GitHub resource scopes according to two privilege levels, i.e., *restrictive* and *permissive*.

The former grants read permissions in the repository for the *contents*, and *packages* scopes only to the automation workflow. The permissive privilege level, instead, grants read and write permissions over all repository scopes.

More granular permissions can be declared in the workflow specification file. Scopes and corresponding permission values are declared in the workflow specification file through the `permissions` keyword.

Possible permission values for scopes are *read*, *write*, and *none*. The *read* value provides access to the resource. The *write* value enables the user to modify the resource. The *none* value disables all activities on the resource.

The permissions can be specified at workflow and job levels inside the workflow specification file, reducing the permission scope over the resource even more.

(B.1) Triggering points stimulation. GitHub Actions workflows support different types of triggers. Triggering events can be defined for an entire workflow inside the workflow specification file. In detail, there are 36 possible events [17] that can trigger a workflow. Filtering events depending on operations (e.g., creation of issues event) and branches (e.g., push on specific branch) is also possible. Manual triggering is enabled through a specific event, i.e., `workflow_dispatch`. Automation workflows triggered by the `workflow_dispatch` event starts when a user or an entity requests a specific API endpoint associated with the workflow. Workflow triggering can also be scheduled using the `crontab` syntax [16].

(B.2) Authentication for triggering. The user or entity needs to authenticate on GitHub to trigger an automation workflow to allow the platform to evaluate the possession of the required privileges to fire a starting event. This

GitLab		GitHub		BitBucket	
Role	**Description**	**Role**	**Description**	**Role**	**Description**
Owner	The Owner of a GitLab project or group has full access to all features and settings, including the ability to delete the project or group.	Owner	The Owner of a GitHub repository has full access to all features and settings, including the ability to transfer or delete the repository.	Admin	Users with this permission have full control over the repository, including the ability to create and delete branches, tags, and pull requests, and to modify repository settings.
Maintainer	Maintainers have similar access to Owners, but they cannot delete the project or group.	Collaborator	Collaborators have similar access to Owners, but they cannot transfer or delete the repository. This role is available only for organization level.	Create repository	Users with this permission can create new repositories within a project and become the Owner of those repositories. This role is available only for project level.
Developer	Developers have read and write access to the code, but they cannot modify project settings or invite others to the project.	Write	Users with write access can make changes to the repository, such as pushing commits and creating branches, but they cannot modify the repository's settings or add collaborators.	Write	Users with this permission can make changes to the repository, including creating and deleting branches and tags, and creating and merging pull requests.
		Triage	Users with triage access have the ability to manage issues and pull requests, such as assigning labels and milestones, but they cannot make changes to the repository's code or settings.		
Reporter	Reporters have read-only access to the project, and can view issues, merge requests, and code, but cannot make any changes.				
Guest	Guests have limited read-only access to the project, and can only view issues and merge requests that have been explicitly shared with them.	Read	Users with read access can view the repository's contents, but they cannot make any changes.	Read	Users with this permission can view the repository and its contents, including code, issues, and pull requests, but cannot make any changes.
Anonymous	Anonymous users have read-only access to the project, but do not need to be logged in to view it. This role is typically used for public projects that do not require authentication.				
				None	Users with this permission have no access to the repository.

TABLE 2. COMPARISON OF PRIVILEGE ROLES ON SOFTWARE MANAGEMENT PLATFORMS.

policy also applies to manual triggers because they are mapped as a specific event by the GitHub platform (i.e., `workflow_dispatch`).

The authentication to events is managed through two types of access tokens, i.e., the *user token* and the *fine-grained access token*. The former is associated with the user's role, and thus, it inherits the privileges related to her. The latter, instead, are tokens containing permissions defined during their creation.

(C.1) Permissions resolution. Once the workflow has been triggered, GitHub computes the privileges involved in the run. This activity considers all the permissions specified in the *privilege configuration* stage.

In detail, the workflow permissions are initially set to the base permissions declared in the repository settings (i.e., *restrictive* or *permissive* mode). Base permissions can be elevated or restricted by permissions defined in the automation workflow specification file. Hence, GitHub grants the finest-grained permissions defined in the *privilege configuration* stage. For example, a workflow needs to push changes to a branch in your repository. Still, the base permissions are set to *restrictive* mode, allowing the workflow only to read the repository content. In this case, it is necessary to elevate the permissions granted to the automation workflow during its execution. To do so, it is possible to declare such higher permissions directly in the automation workflow specification file.

3.2. GitLab

GitLab automation technology takes the name of *CI/CD pipelines* [24]. A CI/CD pipeline workflow comprises *stages* and *jobs*. Stages usually represent development pipeline phases (e.g., build, production, test). Jobs represent the action to be taken on the repository. Stages are executed sequentially, and their order is defined at the beginning of the workflow. When all jobs in the stage succeed, the workflow moves to the next stage.

A job is composed of sequential bash commands. A runner executes these commands using the repository root as the working directory.

GitLab can be installed on-premises on personal servers. For this reason, permissions can be granted at three levels of granularity: instance-wide (i.e., the GitLab instance on the server), group, or project level.

(A.1) Roles configuration. GitLab applies a role-based authorization policy. It defines five roles, from the least privileged to the full privileged: *(i)* Guest, *(ii)* Reporter, *(iii)* Developer, *(iv)* Maintainer, *(v)* Owner (see Table 2).

Scope	Description
actions	Grants permission to access the Actions API, including the ability to create and manage secrets. The read permission allows reading Actions metadata and logs, while the write permission allows starting and canceling workflows and approving and rejecting workflow runs.
checks	Grants permission to read and write check runs and check suites for a repository.
contents	Grants permission to read and write repository contents, including files and directories.
deployments	Grants permission to read and write repository deployments.
id-token	Grants permission to read and write ID tokens for a repository, which can be used to authenticate with GitHub APIs.
issues	Grants permission to read and write issues for a repository.
discussions	Grants permission to read and write discussions for a repository.
packages	Grants permission to read and write packages within a repository or organization.
pages	Grants permission to read and write GitHub Pages for a repository.
pull-requests	Grants permission to read and write pull requests for a repository.
repository-projects	Grants permission to read and write repository projects for a repository.
security-events	Grants permission to read and write security events for a repository.
statuses	Grants permission to read and write commit statuses for a repository.

TABLE 3. DESCRIPTIONS OF AVAILABLE SCOPES FOR PERMISSIONS IN GITHUB ACTIONS.

Moreover, GitLab has a special role, the *Administrator*. This is the sole role able to deal with the management of the GitLab platform instance.

(A.2) Permissions definition. Permissions can be defined through the GitLab settings. Roles enabled for permission granting depend on the granularity level. Instance-wide permissions and privileges can be managed only by the Owner role. The Owner and Maintainer roles deal with group permissions, while the Owner, Maintainer, and Developer roles can manage project permissions.

Project owners can configure more granular permissions for automation workflow execution using *Protected Branches* and *Protected Tags*. These settings allow project owners to restrict automation workflow execution for certain branches or tags to specific users or groups with defined permission levels. Moreover, it is possible to modify project-level permissions scope for automation workflows. In particular, the *public pipelines* and *pipeline visibility* options change the visibility of workflows for low-privilege roles.

GitLab does not allow managing permissions in the workflow definition file, as GitHub does. Then, automation workflows only rely on the user role granted during this phase.

(B.1) Triggering point stimulation. GitLab automation workflows are triggered by push and merge request[1] events by default. Also, it is possible to configure additional triggering events for CI/CD pipelines through dedicated webhooks [23]. Thanks to such a mechanism, GitLab provides a triggering-by-event approach similar to the one offered by GitHub. Webhooks can also be involved in the manual triggering of automation workflows. The webhook URL can be embedded in external services, integrating the automation workflow. Moreover, GitLab API provides specific endpoints for automation workflow triggering.

(B.2) Authentication for triggering. Users or entities triggering a workflow with an API call have to include an access token. The token is evaluated to authenticate the user or entity, and the workflow inherits permissions associated with the token during execution.

GitLab enables automation workflow triggering through *trigger tokens*. These are single-purpose tokens that can be used to trigger a workflow with specific permissions.

Using trigger tokens, users can initiate workflow execution without requiring specific roles or permissions. This feature helps integrate automation workflow with external services or scripts. It allows these services to trigger the workflow without requiring full access to the GitLab project.

(C.1) Permissions resolution. GitLab evaluates privileges based on the membership relation [22]. Hence, the memberships to specific groups and projects play a crucial role in defining which permissions the automation workflow has. The precedence of permissions can be summarized in the following four points:

- Access level precedence. If a user is granted access directly at a higher access level than her group membership, then the higher access level takes precedence.
- Higher access levels precedence. If a user has been granted multiple access levels explicitly, then the highest access level takes precedence.
- Inherited access levels precedence (over no access). If a user has access to a project or group through inheritance, such as being a group member with access, then that access level takes precedence over having no access.
- Narrower access levels precedence (over broader access levels): if a user has been granted access at different levels of granularity, such as access to a specific project versus access to all projects in a group, then the more specific access level takes precedence.

The permissions evaluation policy guarantees users the most permissive access level possible, considering their roles.

3.3. BitBucket

BitBucket Pipelines is a continuous integration and deployment service built into the BitBucket platform. It allows developers to automate their code's building, testing, and deployment. Pipelines use YAML configuration files to define workflows and support a range of programming languages and tools. It integrates with other Atlassian [6] products such as Jira [7] and can also be extended through third-party plugins [8]. Pipelines offer teams a flexible and scalable solution to streamline their development process and improve their productivity.

As for GitLab, Users can interact with three levels, so privileges are granted on three levels of granularity: global, project, and repository. The global level concerns the BitBucket platform instance, while projects are groups of repositories.

(A.1) Roles configuration. BitBucket provides roles at the global level, the project level, and the repository level.

1. equivalent to pull request in GitHub

Global-level privileges deal with management activities on the BitBucket instance, like creating projects and managing users and groups of users. These roles are, from the least privileged to the full privileged: *(i)* BitBucket User, *(ii)* Project Creator, *(iii)* Admin, and *(iv)* System Admin.

Concerning the project and repository levels, BitBucket allows the management of privileges directly associated with permissions. In detail, the permissions that can be associated with a user are, from the least privileged to the full privileged: *(i)* Read, *(ii)* write, and *(iii)* Admin (see Table 2).

(A.2) Permissions definition. Permissions can be defined through the BitBucket settings. This activity is restricted to the Admin role. An Admin user carries on the permissions granting activity for the scope of its role (i.e., a repository Admin manages permissions only for the repository, while a project Admin manages permissions for all the repositories in the project).

More granular permissions can be defined through the use of these features: *Deployment permission options* [5], *deployment variables* [4], and *branch permissions* [2].

The *Deployment permission options* feature provides two options. With *Admin restrictions*, repository owners can set privileges to trigger a deployment pipeline only for trusted users. With this feature, users need high-privilege roles to trigger critical pipelines that could accidentally or intentionally deploy malicious or buggy code into production. *Branch restrictions* enable controlling deployment to critical environments, like production. Repository owners specify the branches allowed to deploy to production. This feature prevents branches requiring fewer privileges to interact with them from deploying their content in critical environments.

Securing *deployment variables* prevents critical data from being stolen from execution logs during pipeline execution. Apart from the execution logs, the only place where deployment variables are available for displaying is in the repository settings. BitBucket imposes users to have at least the Admin role to access this repository section.

Using *branch permissions*, repository owners can set pipeline restrictions on specific branches. This feature prevents changes to the branch content and itself (e.g., deleting the branch). The permissions can be set on the branch at different granularity. Permissions can be set on branch types (e.g., development and production branches) or branch patterns (i.e., branches identified by regular expressions on their name).

Bitbucket Pipelines does not currently include access control as a configurable option in the automation workflow specification file.

(B.1) Triggering point stimulation. BitBucket Pipelines defines start conditions for automation workflows [3]. In detail, a pipeline can be triggered: *(i)* when a commit is pushed to any branch (default), *(ii)* when a commit is pushed to specific branches, *(iii)* when a pull request is created or updated and targeting specific branches, *(iv)* when a tag is created, *(v)* or manually by the user. *(i)* to *(iv)* are triggers by event.

BitBucket Pipelines does not allow the definition of other events for the automation workflows triggering. Automation workflows can be manually triggered when they present the `custom` property in their body. This property is comparable to the `workflow_dispatch` event on GitHub. Then, requesting a specific BitBucket API endpoint, the workflow is triggered.

(B.2) Authentication for triggering. BitBucket relies on user access tokens for authentication. The automation workflow runs with the permissions granted to the user and hence to the token. Because of the triggering events and the direct triggering policies, a user needs at least the permissions to write on the repository to initiate an automation workflow.

(C.1) Permissions resolution. In BitBucket Pipelines, permissions for automation workflow execution are determined by the permissions assigned to users and groups within the repository. If there is a conflict or overlap in permissions between different users or groups, BitBucket will prioritize the access level with the higher permission.

For example, suppose a user is a member of multiple user groups within the repository, and each group has different access levels. In that case, BitBucket will use the group's access level with the highest permission level to determine the user's access level during pipeline execution.

4. Discussion

Attacks targeting code repositories exploiting automation workflows are a real-world problem, as the scientific and industrial community pointed out e.g., in [10], [31], [34]. Although privilege management strategies cannot prevent these attacks as a whole, the role of permissions is mandatory for attack mitigation and prevention.

The introduction of role-based authentication and permissions by software management platforms aims to reduce the probability of triggering and exploiting vulnerabilities in automation workflows. Although, the task of granting privileges and roles to users is entirely transferred to the repository's owners and the workflows' developers. This is a critical task, and the time and competencies required to deal with repositories with many contributors should be more manageable.

All software management platforms support the specification of fine-grained privileges to mitigate privilege misuse. This allows developers to define the smallest number of resources involved in the privilege and the permissions to be granted.

Our analysis shows that the platform offering the finest granularity of permissions is GitHub. In particular, GitHub allows developers to define permission directly in the automation workflow specification file. This approach has two main implications. First, it breaks the separation between privileges management and automation workflow definition, shifting responsibilities to the workflow creator. Then, it provides a public mapping between resources and permission, enhancing the visibility of resources and permissions involved in the automation workflow. The sole use of roles for privilege management blurs the understanding of permissions involved in automation workflows.

Relying only on roles, a user role has to be elevated to guarantee the expected results from an automation workflow. This approach overcomes potential malfunction in the automation workflow. However, because of the coarse granularity of roles, the elevation of the role grants access to more resources than the ones necessary to execute the

automation workflow. Consequently, this overprivileged user can access a larger attack surface. GitLab and BitBucket apply this approach to their automation workflows.

Software management platforms implement repositories groups for privileges, roles, and users. This feature enables the transferability of user privileges from higher levels, i.e., the group, to lower levels, i.e., the single repository. This process adds a layer of complexity to the privilege resolution phase. Thus, additional awareness is required during the privilege configuration phase to avoid unexpected overprivileged users. Complex membership mechanisms, such as the one implemented by GitLab [22], make tracking the inheritance of privileges among repositories, groups, and projects difficult. This lack of visibility can make detecting and responding to security incidents challenging.

Moreover, as visible in Table 2, we notice that the distribution of roles on software management platforms is not uniform[2]. For example, BitBucket tends to have more highly privileged roles; GitHub, outside of organizations, has just two highly privileged roles. The lack of low and intermediate roles can lead to more highly privileged users causing security concerns.

Overall, from this study, we argue the need for a security-first approach to privilege management on software management platforms.

This lack is highlighted by the missing strategies to identify potential flaws from misconfigured privileges and permissions. Thus, all three software management platform documentation recommend achieving the principle of least privilege for automation workflow execution. However, no mechanism exists to help or force users to obtain such a principle. In particular, the privilege configuration and the privilege resolution phases could integrate procedures to verify and even enforce the principle of least privilege or at least control over permissions.

5. Related Work

Software management platforms and automation workflows gathered attention in the last years. In particular, GitHub with its GitHub Actions and GitLab with the CI/CD Pipelines. The advantage of automation workflows concerning their use for pull request management has been pointed out in [36], [37]. Works as [30], [13], [14], [35] studied the adoption, involvement, and evolution of automation workflows in software development. Automation workflows have also been analyzed referring to specific research fields [29], [11]. The study in [25] reports statistics on how the usage decrease of CI services, like Travis CI [27], coincides with the rise of GitHub Actions that, in eighteen months, became the dominant automation technology.

The variety of automation workflow applications and their growing adoption arose interest in the security community. In particular, the authors in [10] provided a methodology for the security assessment of GitHub Actions and conducted an in-the-wild assessment of GitHub Action workflows to demonstrate the presence of several security issues in public GitHub repositories. In [31], authors analyzed the security of GitHub Actions and other CI/CD automation pipelines by conducting an empirical assessment of automation workflows. Both works unveil the presence of potential misconfigurations in the privileges of automation workflows.

However, to the best of our knowledge, our work represents the first rationalization of privilege handling for automation workflows on the three major software management platforms.

2. We use a blank space when there is not a corresponding role on a different platform.

6. Conclusion

In this work, we defined the privilege life cycle in automation workflows. We investigate how software management platforms implement the steps of the life cycle. This analysis highlighted how the methods used by software management platforms are similar under specific aspects. For example, platforms are coherent in triggering automation workflows, relying on access tokens and APIs. Although significant differences have been identified. For example, the involvement of privileged roles in the automation workflows. From a security point of view, improvements are necessary to guarantee a stronger security position. In particular, the complexity and heterogeneity of privilege management mechanisms and the lack of proper documentation support increase the probability of security violations related to misconfiguration and over privilege.

Security should be more central during the privilege life cycle in automation workflows. In particular, the *privilege configuration* stage of the privilege life cycle should contain compliance verification and enforcement methods. This paper aims to encourage the discussion on privilege management on software management platforms, focusing on automation workflows. The significant advantages of automation workflows on software management platforms risk being dangerous without proper privilege management methodologies.

References

[1] Atlassian. Bitbucket. https://bitbucket.org. [Online; accessed 10-march-2023].

[2] Atlassian. Branch permissions. https://support.atlassian.com/bitbucket-cloud/docs/use-branch-permissions/. [Online; accessed 10-march-2023].

[3] Atlassian. Pipeline start conditions. https://support.atlassian.com/bitbucket-cloud/\-docs/\-pipeline-start-conditions/. [Online; accessed 10-march-2023].

[4] Atlassian. Variables and secrets: Bitbucket cloud. https://support.atlassian.com/bitbucket-cloud/docs/variables-and-secrets/. [Online; accessed 10-march-2023].

[5] Atlassian. Deployment permissions now available in bitbucket pipelines. https://bitbucket.org/blog/deployment-permissions-now-available-in-bitbucket-pipelines, Dec 2022.

[6] Atlassian, Inc. Atlassian. https://www.atlassian.com. [Online; accessed 10-march-2023].

[7] Atlassian, Inc. Jira. https://www.atlassian.com/software/jira. [Online; accessed 10-march-2023].

[8] Atlassian, Inc. Jira. https://bitbucket.org/product/features/pipelines/-integrations. [Online; accessed 10-march-2023].

[9] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml™) version 1.1. *Working Draft 2008*, 5:11, 2009.

[10] 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*, SCORED'22, page 37–45, New York, NY, USA, 2022. Association for Computing Machinery.

[11] Fabio Calefato, Filippo Lanubile, and Luigi Quaranta. A preliminary investigation of mlops practices in github. In *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '22, page 283–288, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Chaminda Chandrasekara and Pushpa Herath. *Introduction to GitHub Actions*, pages 1–8. Apress, Berkeley, CA, 2021.

[13] Tingting Chen, Yang Zhang, Shu Chen, Tao Wang, and Yiwen Wu. Let's supercharge the workflows: An empirical study of github actions. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 01–10, 2021.

[14] Alexandre Decan, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh. On the use of github actions in software development repositories. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 235–245, 2022.

[15] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. Devops. *IEEE Software*, 33(3):94–100, 2016.

[16] Free Software Foundation. Crontab. https://pubs.opengroup.org/onlinepubs/9699919799/utilities/\-crontab.html. [Online; accessed 10-march-2023].

[17] GitHub, Inc. Events that trigger workflows. https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows. [Online; accessed 10-march-2023].

[18] GitHub, Inc. Github. https://github.com. [Online; accessed 10-march-2023].

[19] GitHub, Inc. Github actions. https://docs.github.com/en/actions. [Online; accessed 10-march-2023].

[20] GitHub, Inc. Security hardening for github actions. https://docs.github.com/en/actions/security-guides/security-hardening-for-github-actions. [Online; accessed 10-march-2023].

[21] GitLab B.V. Gitlab. https://gitlab.com. [Online; accessed 10-march-2023].

[22] GitLab B.V. Gitlab - members of a project. https://docs.gitlab.com/ee/user/project/members/index.html. [Online; accessed 10-march-2023].

[23] GitLab B.V. Gitlab - webhooks. https://docs.gitlab.com/ee/user/project/integrations/webhooks.html. [Online; accessed 10-march-2023].

[24] GitLab B.V. Gitlab ci/cd pipelines. https://docs.gitlab.com/ee/ci/pipelines/. [Online; accessed 10-march-2023].

[25] Mehdi Golzadeh, Alexandre Decan, and Tom Mens. On the rise and fall of ci services in github. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 662–672, 2022.

[26] Qingyuan Gong, Jiayun Zhang, Yang Chen, Qi Li, Yu Xiao, Xin Wang, and Pan Hui. Detecting malicious accounts in online developer communities using deep learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 1251–1260, New York, NY, USA, 2019. Association for Computing Machinery.

[27] Idera, Inc. Travis ci. https://www.travis-ci.com/. [Online; accessed 10-march-2023].

[28] Paul Jenkins and Jean Cassou. *Jenkins*. Gerd Hatje, 1963.

[29] Albert Y. Kim, Valentine Herrmann, Ross Barreto, Brianna Calkins, Erika Gonzalez-Akre, Daniel J. Johnson, Jennifer A. Jordan, Lukas Magee, Ian R. McGregor, Nicolle Montero, Karl Novak, Teagan Rogers, Jessica Shue, and Kristina J. Anderson-Teixeira. Implementing github actions continuous integration to reduce error rates in ecological data collection. *Methods in Ecology and Evolution*, 13(11):2572–2585, 2022.

[30] Timothy Kinsman, Mairieli Wessel, Marco A. Gerosa, and Christoph Treude. How do software developers use github actions to automate their workflows? In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 420–431, 2021.

[31] 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 Association.

[32] Mathias Meyer. Continuous integration and its tools. *IEEE Software*, 31(3):14–16, 2014.

[33] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017.

[34] StepSecurity, Inc. Stepsecurity - securerepo. https://github.com/step-security/secure-repo#1-automatically-set-minimum-github_token-permissions. [Online; accessed 10-march-2023].

[35] Pablo Valenzuela-Toledo and Alexandre Bergel. Evolution of github action workflows. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 123–127, 2022.

[36] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. ESEC/FSE 2015, page 805–816, New York, NY, USA, 2015. Association for Computing Machinery.

[37] Mairieli Wessel, Joseph Vargovich, Marco A. Gerosa, and Christoph Treude. Github actions: The impact on the pull request process, 2022. https://arxiv.org/abs/2206.14118.

[38] Yiming Zhang, Yujie Fan, Shifu Hou, Yanfang Ye, Xusheng Xiao, Pan Li, Chuan Shi, Liang Zhao, and Shouhuai Xu. Cyber-guided deep neural network for malicious repository detection in github. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 458–465, 2020.