LEGIT SECURITY

State of

GitHub Actions Security

Insecure GitHub Actions could allow attackers to compromise organizations. How big is this threat? The Legit research team found out.

Research led by Noam Dotan, Legit Security
Senior Researcher and AI Lead

Contents

EXECUTIVE SUMMARY

Open-source software is the driver of modern technology, and GitHub is the platform that drives open source.

As of January 2023, GitHub reported having over 100 million developers and more than 420 million repositories, including at least 28 million public repositories.

GitHub Actions is GitHub's built-in CI/CD platform, and it is the de facto standard for building open-source software. A GitHub Actions workflow is a specific pipeline that the CI/CD platform needs to run.

GitHub Actions security is an important aspect of open-source security. Insecure GitHub Actions could allow attackers to compromise open-source and initiate supply chain attacks or use them as an initial attack vector into organizations that use GitHub.

Considering this threat:

Organizations should only use custom Actions after thoroughly vetting them, and use GitHub's built-in features to enforce GitHub Actions best practices and policies, including:

- Creating a list of allowed third-party Actions
- Configuring default permission for workflow tokens to be the least privileged
- Determining where GitHub Actions can approve pull requests
- Controlling which users can execute workflows

How big is this threat?

To find out, the Legit research team recently analyzed:

2,500,000

GitHub Actions workflow files

Belonging to:

553,000

Organizations and personal users

State of GitHub Actions Security at a Glance

Most GitHub Actions workflows are insecure in some way — they are overly privileged, have risky dependencies, etc. Legit research reveals that even projects from enterprises like Google and Apache are flawed. In addition, the GitHub Actions marketplace security posture is concerning. Most of the Actions there are not verified, maintained by one developer, and have low security scores based on OpenSSF Scorecard.

Security of GitHub Actions Workflows

Legit found the following vulnerabilities in GitHub Actions workflows:

1

Interpolation of Untrusted Input

Workflows with this vulnerability:

7,000+

An attacker could add malicious content to interfere with the workflow execution

2

Execution of Untrusted Code

Workflows with this vulnerability:

2,500+

3

Using Untrustworthy Artifacts

Workflows with this vulnerability:

3,000+

Security of Building Blocks of GitHub Actions Workflows

Legit examined the security of the building blocks of GitHub Actions and found the following:

	Building Blocks	
1	Trigger	Workflows that are using triggers vulnerable to attacks like remote code execution: **3,902**
2	Jobs and Steps	References used by jobs and steps that are not following the best practice of dependency pinning:* **98.4%**
3	Runners	Public repositories using risky hosted runners: **44,803**
4	Permissions	Workflows that don't limit token permissions: **86%** *Without these limits, the token could be used to perform malicious actions or take over the repository.*

* **A famous example of the danger of using third-party software without dependency pinning is the CodeCov breach where attackers were able to modify the CodeCov CI script to exfiltrate the CI runner environment variable.**

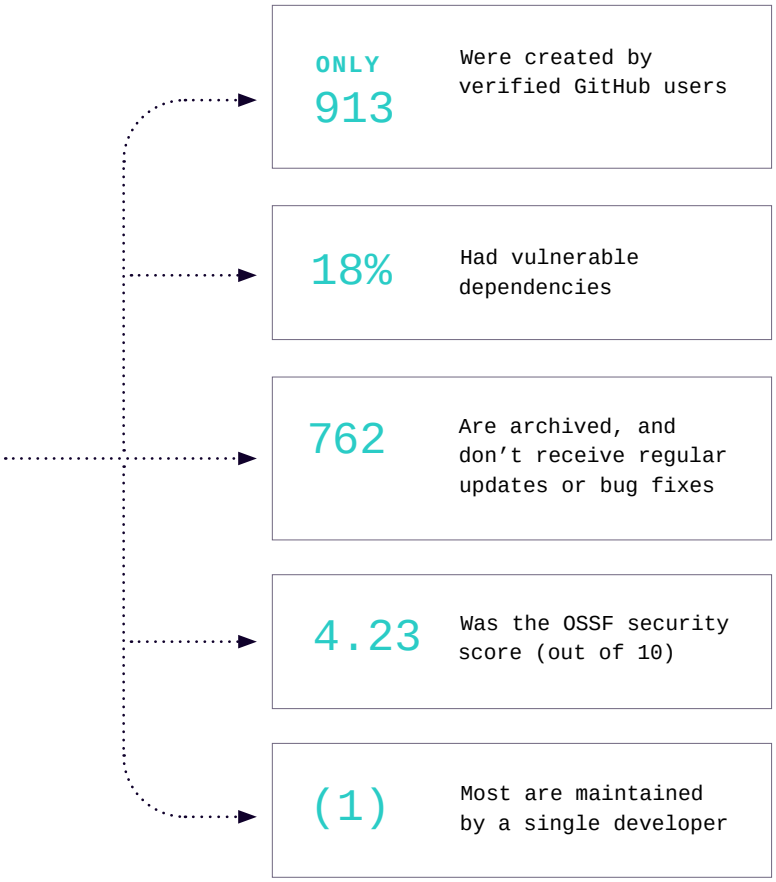
Security of Custom GitHub Actions

Legit found the following vulnerabilities in custom GitHub Actions:

OUT OF THE

19,113

Custom GitHub Actions in the marketplace…



GitHub is an extremely popular platform that enables developers to work together on development projects and see each other's changes in real-time. GitHub Actions add automation to the software development lifecycle on GitHub via event-driven triggers. These triggers are specified events that can range from creating a pull request to building a new branch in a repository.

The most active GitHub Actions organizations are shown in the table below.

Organization	Workflows
conda-forge	33,079
Deepin-community	5,566
NAV IT (The Norwegian Labour and Welfare Directorate)	4,418
moderneinc	4,016
microsoft	3,567

Legit's recent research explores multiple aspects of GitHub Actions security, including:

- How developers write GitHub Actions workflows and whether they adhere to best practices
- The GitHub Actions marketplace (where developers upload Actions for public use) and the security posture of the Actions published there
- Recommendations on best practices and how to avoid the risks outlined throughout this research

Use this research to better understand:

- GitHub Actions and how they work
- The GitHub Actions attack surface
- Risks and mitigations when writing GitHub Actions
- Risks when using GitHub Custom Actions

Vulnerabilities Found in GitHub Actions Workflows

As of today, there are four* main types of application-level GitHub Actions vulnerabilities.

Legit uses the term application level since these vulnerabilities are not in the platform but rather in the code the workflow authors write (as SQL injection to SQL). These vulnerabilities are the following:

1. **Interpolation of untrusted input**
2. **Execution of untrusted code**
3. **Using untrusted artifacts**
4. **Environment variables injection**

* There is an additional risk that arises when referencing a vulnerable third-party action.

Exploitation of these vulnerabilities could allow an attacker to control an organization's GitHub Actions (CI/CD).

Due to the highly privileged nature of CI/CD platforms, an attacker could then move laterally into the organization's cloud environment and other assets.

1 Interpolation of Untrusted Input

This vulnerability occurs when there is untrusted input in the preprocessing part of the workflow. The Action runner processes the workflow in two stages, first it replaces “compile time” variables with their actual value (interpolation):

```
- run: echo ${{github.event.issue.title}}
```

Then it will execute the final script `‘echo title value’`.
An attacker with access to the interpolated value can replace it with malicious content to interfere with the workflow execution.

Workflows we found with this vulnerability:

7,000+

For more information:

Keeping Your GitHub Actions and Workflows Secure Part 2: Untrusted Input

Recommendation

Make sure you do not interpolate untrusted data in your workflows. When needed, process user data with environment variables.

2 Execution of Untrusted Code

This vulnerability occurs when a privileged workflow downloads and executes source code originating from an untrusted source, primarily when processing pull requests from a fork or when downloading third-party source code.

```
name: workflow
on: pull_request_target

jobs:
  job:
    runs-on: ubuntu-latest
    steps:
      - name: Setup Action
        uses: actions/checkout@v2
        with:
          ref: ${{github.event.pull_request.head.ref}}
          repository: ${{github.event.pull_request.head.repo.full_name}}
      - run: make build
```

Workflows we found with this vulnerability:

2,500+

For more information:

Vulnerable GitHub Actions Workflows Part 1: Privilege Escalation Inside Your CI/CD Pipeline

Recommendation

Do not execute code from forks (or any other external source) within a privileged workflow.

3

Using Untrusted Artifacts

This vulnerability occurs when a workflow downloads an artifact from an untrusted source and then uses it in an insecure way, such as executing its payload or interpolating its content.

As we discovered in our latest GitHub Actions research, 'Artifacts Poisoning in GitHub Actions — Rust Found Vulnerable', even organizations that focus on security fail to use artifacts correctly.

```yaml
on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Download artifact
      uses: dawidd6/action-download-artifact@v2
      with:
          workflow: main.yml
          path: gcc-build
          repo: external-repo/repo-name
          name: executable
    - run: ./executable
```

Workflows we found with this vulnerability:

3,000+

Recommendation

Avoid using artifacts from different workflows; if you must, treat them as suspicious and do not trust any input they contain.

4

Environment Variables Injection

This vulnerability occurs when a workflow configures environment variables based on untrusted input. Configuring environment variables in GitHub Actions is based on a special mechanism called `GITHUB_ENV`. To set the variable workflow, the author needs to write its value into `GITHUB_ENV` in the following format:

```
echo "ENV_NAME=$ENV_VALUE" >> $GITHUB_ENV
```

This mechanism is vulnerable to environment variables injection when the attacker can control the variable value. In the above case, if `ENV_VALUE` is controlled by the attacker, it can be used to inject additional environment variables.

```
ENV_VALUE=VALUE\nNEW_ENV=VALUE
```

This will translate into setting up two variables:

- `ENV_VALUE`
- `NEW_ENV`

Attackers abuse this behavior to gain remote code execution on the target workflow by setting special environment variables such as `LD_RELOAD` and `NODE_OPTIONS`.

For more information:

Google & Apache Found Vulnerable to GitHub Environment Injection

Recommendation

Do not use untrusted input to set environment variables. With this input, an attacker could configure environment variables that can lead to pipeline takeover.

Third-Party Risk

This vulnerability occurs when a workflow references a vulnerable third-party Action. Since a typical workflow consists of multiple third-party Actions chained together, there is a big risk of importing a malicious/buggy Action.

Unfortunately, this area is not explored enough, and the CVE list of GitHub Actions consists only of 16 entries.

Custom Actions developers, like any open source developer, are not obligated to publish a CVE for a vulnerability found in their code, and sometimes they explicitly refuse to do so. The Legit research team experienced this first-hand after finding a risky custom Action and asking the maintainer to fix it; he refused to do so, leaving many repositories vulnerable.

For more information:

Vulnerable GitHub Actions Workflows Part 2: Actions That Open the Door to CI/CD Pipeline Attacks

Recommendation

Audit the third-party Actions you're using and verify that they adhere to security best practices.

EXAMPLES OF VULNERABLE WORKFLOWS

Examples of vulnerable workflows can be found in Legit's GitHub Actions vulnerabilities blog series.

- Vulnerable GitHub Actions Workflows Part 1: Privilege Escalation Inside Your CI/CD Pipeline
- Vulnerable GitHub Actions Workflows Part 2: Actions That Open the Door to CI/CD Pipeline Attacks
- Google & Apache Found Vulnerable to GitHub Environment Injection
- Novel Pipeline Vulnerability Discovered; Rust Found Vulnerable

Risk Mitigation for GitHub Actions Workflows

Workflow Hardening

Application-level vulnerabilities are hard to avoid; however, workflow authors should use GitHub Actions hardening features to mitigate risk in case of compromise:

Always scope the token permissions to the bare minimum required.

Pin the workflow dependency to a specific commit you tested and validated.

Do not allow pull-request workflows to run without maintainer approval.

Secure Workflow Coding

Do not execute untrusted code in privileged workflows (such as code coming from fork).

Avoid interpolating user input in “run” and “scripts” steps as it could lead to command injection; use environment variables instead.

Do not trust artifacts from different workflows as they can be altered by attackers and lead to workflow takeover.

Security of Custom GitHub Actions

Custom GitHub Actions are those developed by the community to enhance GitHub Actions capabilities.

Verified Actions

Verified Actions are Actions whose creator is verified by GitHub. Verification is a signal that the creator is credible, and Legit recommends using Actions from verified sources whenever possible.

Custom GitHub Actions in the marketplace:

19,113

Those that were created by verified GitHub users:

913

Security Scorecard for Custom Actions

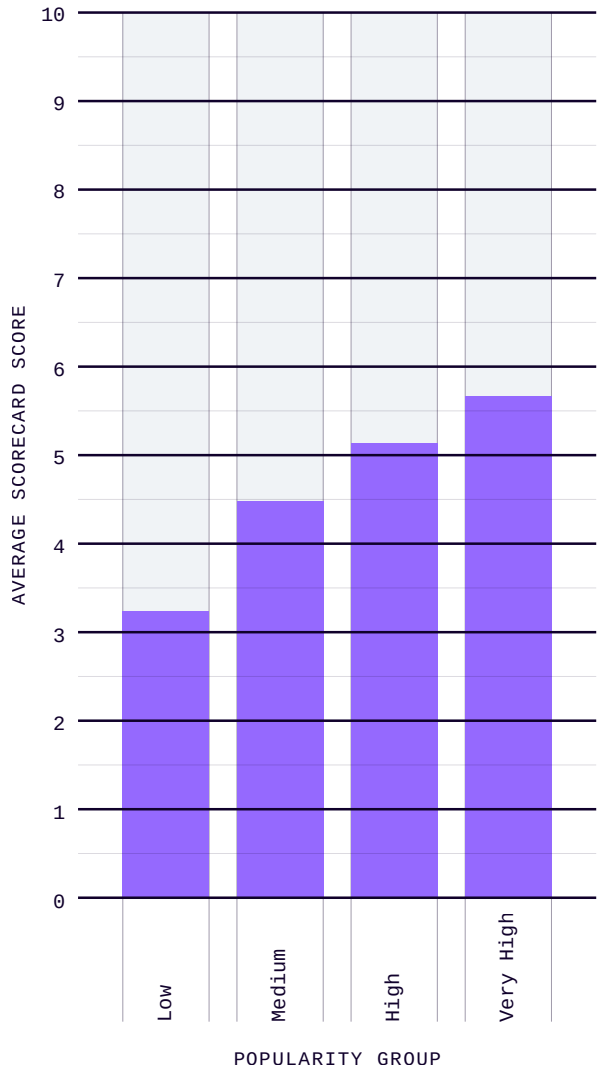
Legit used the scorecard project from OSSF to assess the security posture of each marketplace Action.

GitHub says of the scorecard:

"Scorecard is an automated tool that assesses a number of important heuristics ("checks") associated with software security and assigns each check a score of 0–10. You can use these scores to understand specific areas to improve in order to strengthen the security posture of your project. You can also assess the risks that dependencies introduce, and make informed decisions about accepting these risks, evaluating alternative solutions, or working with the maintainers to make improvements."

The graph below clearly illustrates that popular custom Actions exhibit consistently higher security scores on average, providing users with an additional compelling reason to opt for well-known and widely used custom Actions.

Scorecard Breakdown by Action Popularity



Violation	Count
CII-Best-Practices	18,629
Fuzzing	18,620
Security-Policy	17,485
SAST	17,086
Maintained	15,565
Code-Review	15,001
Branch-Protection	14,260
Dependency-Update-Tool	12,968
Pinned-Dependencies	11,794
Token-Permissions	11,139
CI-Tests	6,517
Contributors	6,470
License	4,132
Vulnerabilities	2,766
Signed-Releases	392
Dangerous-Workflow	163
Binary-Artifacts	51

Among the entire sample, the scorecard score:

4.23

Highlighting the insecurity of custom GitHub Actions

Unhandled Vulnerable Dependencies

Custom Actions with unhandled vulnerable dependencies are more likely to contain security issues and bugs. It also reflects on the Action author's willingness to resolve security issues.

To determine if an Action has a vulnerable dependency, the Legit research team checked whether the repository contains Dependabot pull requests. Dependabot is an SCA scanner built into the GitHub platform that automatically creates fixing pull requests for vulnerable dependencies. For each Action, Legit counted the number of open Dependabot pull requests.

Note: Legit can't distinguish between Actions with no vulnerable dependencies and Actions that didn't enable Dependabot.

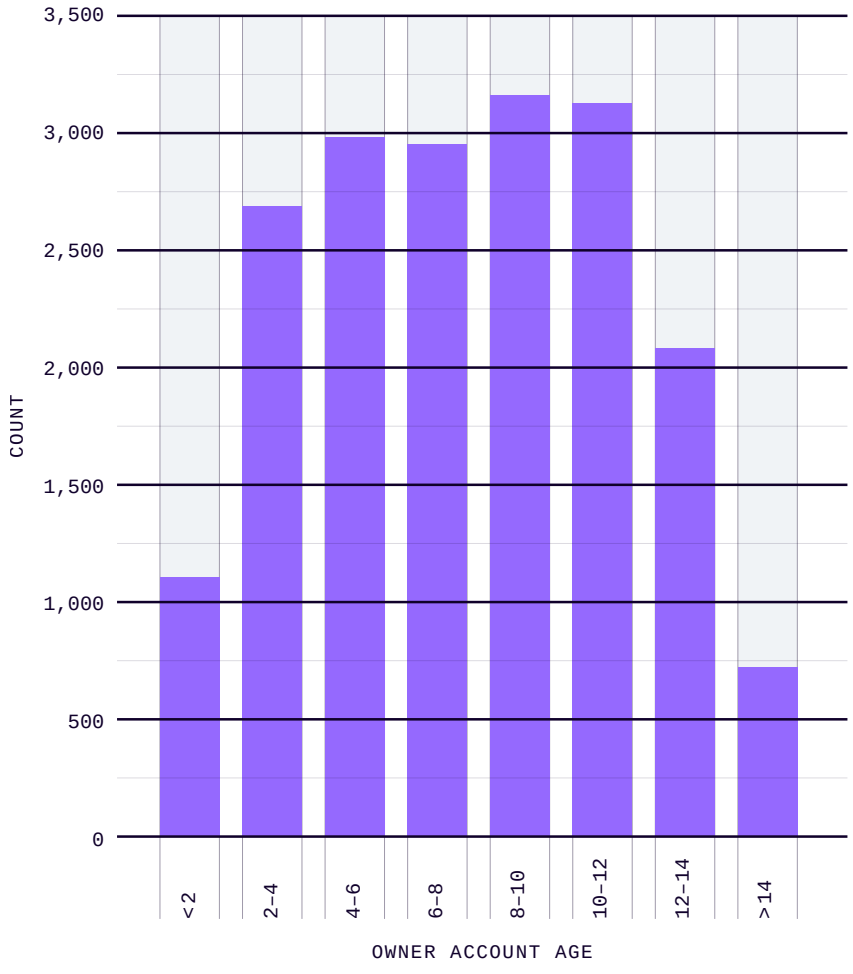
Open Dependabot Pull Requests	Number of Actions
> 20	24
1–10	3,139
10–20	366

Owner Account Age

Account age highlights the author's experience, expertise, and dedication to the platform. Indicating a track record of contributions and engagement within the open-source community, account age can establish trust among users.

Legit research found that most custom Actions are written by developers with less than one year of experience.

Experience of Custom GitHub Actions Developers (in Months)

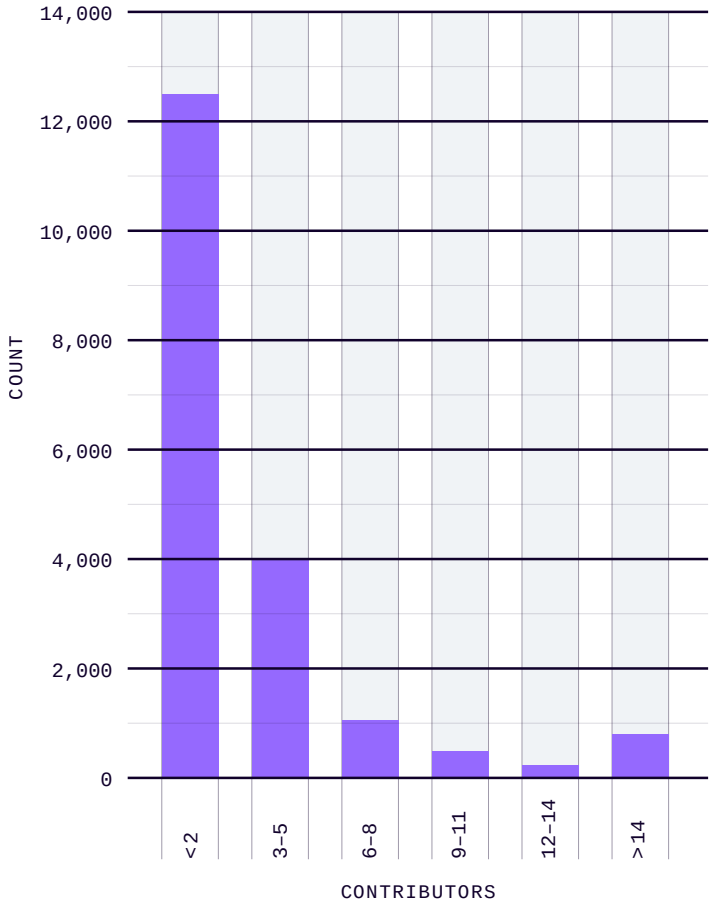


Contributors

The number of contributors to a custom Action strongly reflects its credibility. A higher number of contributors indicates broader community involvement and active development. This collaborative effort leads to increased security, testing, and overall quality.

Unfortunately, most of the GitHub Actions in the marketplace are maintained by a single developer. This could be the case because many Custom Actions have small scopes and don't require a lot of effort to create and maintain.

Number of Contributors to Custom GitHub Actions



Archived

Marketplace Actions that are archived: 762

Using archived Actions presents a couple of problems. First, archived Actions do not receive regular updates or bug fixes, potentially leading to compatibility issues or security vulnerabilities. Second, the lack of ongoing maintenance and support for archived Actions makes it challenging to address any issues raised by GitHub users.

CONCLUSION

Custom Actions, although useful, pose a significant risk for workflow authors. To limit the risk, adhere to strict standards when using them. Make sure that you import Actions from verified owners, and those that are active, maintained, and have high security scores.

Security of the Building Blocks of GitHub Actions Workflows

All GitHub Actions automations are handled via workflows.

Workflows' main building blocks are:

1

Triggers

Control when the workflow should run

2

Jobs

A collection of steps to execute with a shared context

Steps

The most basic building block — an execution

3

Runners

Where the job should run (on which machine)

4

Permissions

What permissions the workflow has

1 Triggers

Triggers control when a workflow should run. GitHub Actions provide 36 event types that trigger workflows.

The table to the right shows the most used triggers.

Trigger	Count
push	1,584,209
pull_request	1,077,723
workflow_dispatch	506,937
schedule	249,011
release	92,110

Risky Triggers

Some triggers inherently present more risk than others, for instance, `pull_request_target` and `workflow_run`. We found thousands of occurrences of both these triggers, listed in the table to the right.

Trigger	Occurrences
pull_request_target	42,036
workflow_run	15,988

Trigger: pull_request_target

Although useful, `pull_request_target` in particular opens the door to a wide range of security issues, including RCE.

This workflow checks out the `pull_request` code and then builds it (in a privileged context). Since the code is from an untrusted source (fork), it leads to RCE in a privileged context.

Number of pull_request_target workflows we found vulnerable to attacks like remote code execution:

2,341

```yaml
name: workflow
on: pull_request_target

jobs:
  job:
    runs-on: ubuntu-latest
    steps:
      - name: Setup Action
        uses: actions/checkout@v2
        with:
          ref: ${{github.event.pull_request.head.ref}}
          repository: ${{github.event.pull_request.head.repo.full_name}}
      - run: make build
```

Trigger: workflow_run

In this example of `workflow_run`:

```
on: workflow
  workflow_run: pull_request_target
    workflows: [Run Tests]
    types:
      - completed
```

The workflow runs when `'Run Tests'` workflow completes.

A '`workflow_run`' triggered workflow acts as a triggered workflow that is executed when another workflow completes, in a privileged context (has access to secrets and tokens). Normally, it is used to chain workflows. In public repositories, it is used to process untrusted pull requests from forks. The fork pull request executes the "dangerous parts" and then triggers the `workflow_run` workflow to finish the processing.

Number of workflow_run workflows we found vulnerable to attacks like remote code execution:

1,561

Recommendation

When possible, avoid using dangerous triggers. If you can't avoid them, use the following mitigation strategies.

- Do not run the workflow automatically. Require a maintainer approval before executing (using labels or environments).
- Set the workflow token permissions to the bare minimum.
- Do not trust any input from the originating workflow; treat any data as potentially malicious.

2 Security of Jobs and Steps

Jobs and steps are the building blocks of a GitHub Actions workflow. A job is composed of several steps, while steps perform the actual work.

This build job is composed of two steps, one that downloads the repository source code and another that performs the actual build.

Steps can reference prebuilt workflows from third-party entities, such as open source builders and vendors. When a step uses the "`uses: repository_name@ref`" clause, the Actions engine will download '`repository_name`' at the specified '`ref`' and execute it.

```yaml
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Setup Action
        uses: actions/checkout@v2
      - run: make build
```

There are several options to reference an action:

Reference to *a commit*

```yaml
- uses: actions/checkout@8f4b7f84864484a7bf31766abe9204da3cbe65b3
```

Pro

The most secure way to reference a third-party Action, it guarantees the third-party Action version specified.

Con

The Actions are not updated automatically.

Reference to *the main branch*

```yaml
- uses: actions/download-artifact@main
```

This is insecure, as the workflow will use any code from the third-party Action without control.

Reference to *a tag*

```yaml
- uses: actions/checkout@v1
```

Pro

Using a published release of the Action, it receives updates only if the workflow author intends it to.

Con

The Action can change without the dependent workflow's knowledge.

For more explanation about the risk of mutable tags:

What Are Immutable Tags and Can They Protect You From Supply Chain Attacks?

Reference to *a non-default branch*

```
- uses: actions/download-artifact@develop
```

This is the most insecure way to reference a third-party Action, and imposes the same risk as “reference to the main branch,” with the additional risk that unlike the main branch, other branches usually don’t have security guardrails that validate their quality, such as code review and code scanning.

Reference Type	Count
tag	6,762,757
main	433,628
commit	185,790
branch	68,900

Reference to *a docker image*

```
- uses: docker://fastai/fastpages-jekyll
```

A less common way to use custom Actions is to use prebuilt docker images. This method is not prevalent in the GitHub Actions ecosystem and imposes risk on the image consumers because they can’t access the third-party Action source code (without reverse engineering the image).

Docker Reference Type	Count
non-versioned	8,958
latest	7,024
tag	6,758
hash	381

CONCLUSION

98.4%

Of the references used by jobs and steps are not following the best practice of dependency pinning, which specifies which package or library an Action can rely on.

Dependency pinning in GitHub Actions (and in general) is crucial for ensuring workflows are stable and reproduceable. By pinning dependencies, you specify the exact versions of the external packages or libraries your Actions rely on. This practice guards against unexpected changes or updates that could potentially introduce breaking changes, compatibility issues, or security vulnerabilities.

3 Security of Runners

The GitHub Actions runner is the engine that executes the workflow definition. It both communicates with the GitHub server to obtain credentials and configurations and reads the user workflow file and executes it according to the definitions. The runner code is open-source and hosted on GitHub.

There are two ways to use runners:

1. **Managed Runners**
   GitHub owned virtual machines that anyone can use instantly.
2. **Hosted Runners**
   Self-hosted runner deployed by the user to meet individual requirements such as security, performance, or memory.

Self-hosted runners pose significant security risks (see blog on self-hosted runner security), especially if untrusted workflows run on them. Malicious programs may run on the machine, and it's possible for the runner sandbox to be escaped, exposing access to the machine's network environment. Additionally, unwanted or dangerous data can be persisted on the machine.

This is especially critical for public repositories since they may allow the organization's users to execute workflows on the runners used by the public repositories.

Public repositories we found using hosted runners:

44,803

Recommendation

We recommend only using self-hosted runners with private repositories. Forks of your public repository could potentially run dangerous code on your self-hosted runner machine by creating a pull request that executes the code in a workflow.

GitHub-hosted runners, on the other hand, are clean, isolated virtual machines that are destroyed at the end of the job execution and do not pose the same risks.

4 Security of Workflow Permissions

Each workflow run is assigned a GitHub personal access token scoped to the current repository that enables it to communicate with a GitHub API. By default, this token is highly privileged and has access to the following scopes and corresponding APIs:

```
permissions:
  actions: read|write|none
    checks: read|write|none
    contents: read|write|none
    deployments: read|write|none
    id-token: read|write|none
    issues: read|write|none
    discussions: read|write|none
    packages: read|write|none
    pages: read|write|none
    pull-requests: read|write|none
    repository-projects: read|write|none
    security-events: read|write|none
    statuses: read|write|none
```

If a workflow is compromised, the token can be used to perform malicious actions and even take over the repository.

Therefore, it is important that workflow authors use the "`permission`" key to specify the scope their workflow requires.

Scoped Permissions	Count
Yes	344,690
No	2,079,309

Workflows that limit token permissions:

ONLY 14%

Without these limits, the token could be used to perform malicious actions or take over the repository.

Recommendations for Using GitHub Actions Securely

The risk companies face from insecure GitHub Actions use is significant. GitHub Actions provide the key to a company's most critical infrastructure. They are connected both to an organization's source code and to their deployment environment, meaning that once exploited by attackers, the organization is completely in the attacker's hands.

Awareness and Training

Organizations should prioritize educating their development and operations teams about the security risks associated with GitHub Actions.

This education includes:

- The proper handling of secrets
- The risks of code injection
- The best practices for using third-party Actions

Training sessions should cover the identification and remediation of common vulnerabilities, with a strong emphasis on security-first coding practices and the regular auditing of workflows to detect any unauthorized changes. Moreover, continuous education programs should be implemented to keep all team members updated on the latest security threats and mitigation techniques.

Organization-Wide Best Practice Enforcement

Implementing organization-wide best practice configuration enforcement is crucial for maintaining the security of GitHub Actions workflows. Organizations should use GitHub's built-in features for controlling GitHub Actions behavior to enforce best practices and organization policies.

These policies include:

- Create a list of allowed third-party Actions.
- Configure default permission for workflow tokens to be the least privileged.
- Control which users can execute workflows in the organization.
- Control which repositories are allowed to use self-hosted runners.
- Establish the forking policy in the organization.
- Determine where GitHub Actions should be allowed to approve pull requests.

Tooling

Organizations should leverage security tools that integrate seamlessly with GitHub to provide continuous scanning for GitHub Actions vulnerabilities, misconfigurations, vulnerable dependencies, secrets detection, and more.

Configure these tools to automatically scan the code for each code or configuration change, ensuring immediate feedback on potential security issues.

How Legit Can Help

The Legit platform contains a myriad of security policies that capture and help mitigate the security issues mentioned in this report.

For each policy, the Legit platform provides a detailed description of the risk, information on where the risk is located, who introduced it, and detailed remediation steps.

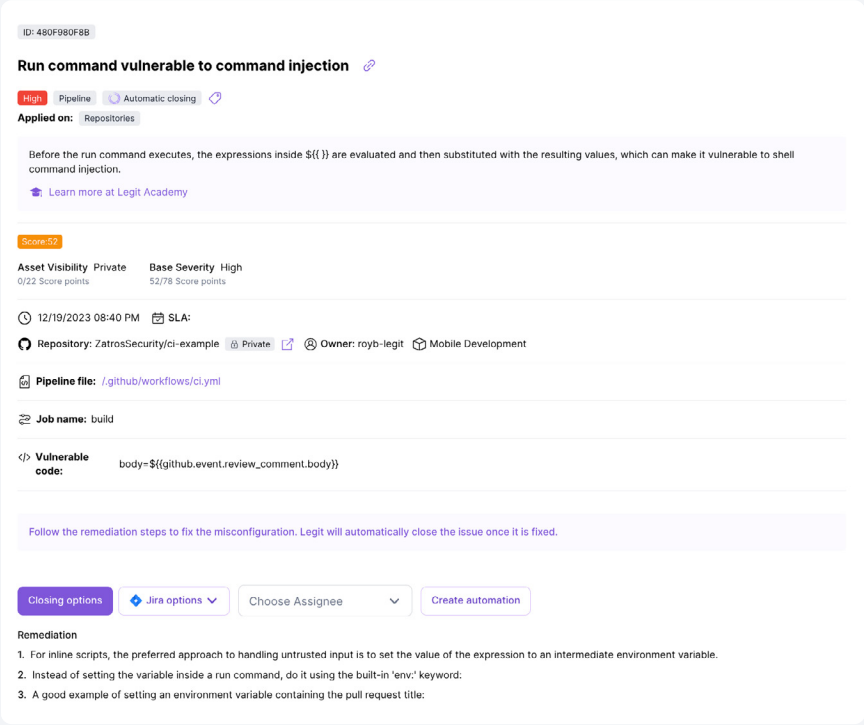
SEE EXAMPLES IN APPENDIX

APPENDIX

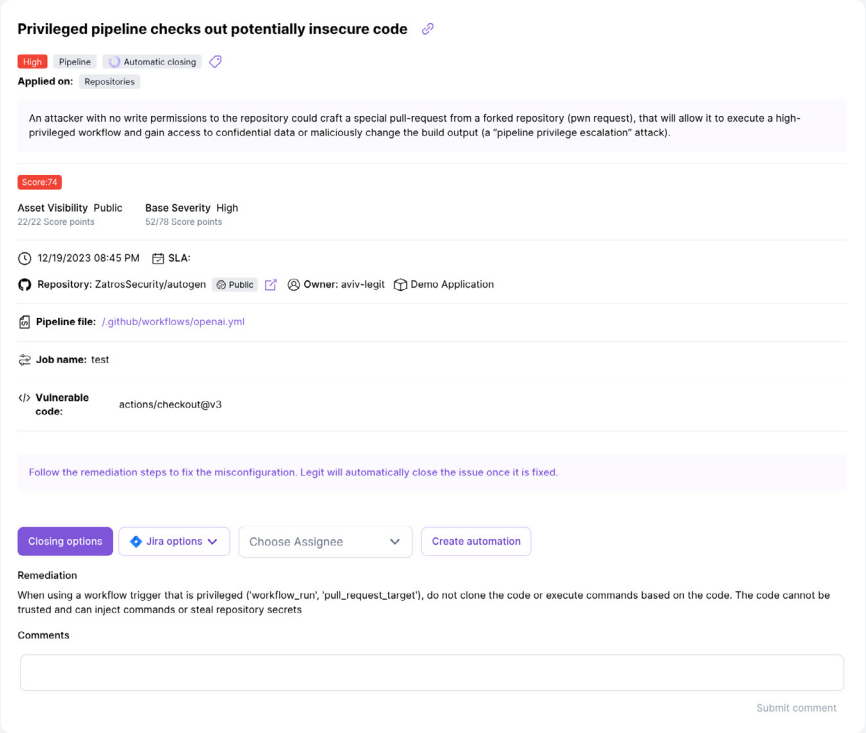
Examples of Legit GitHub Actions Security Policies

Vulnerable Workflows

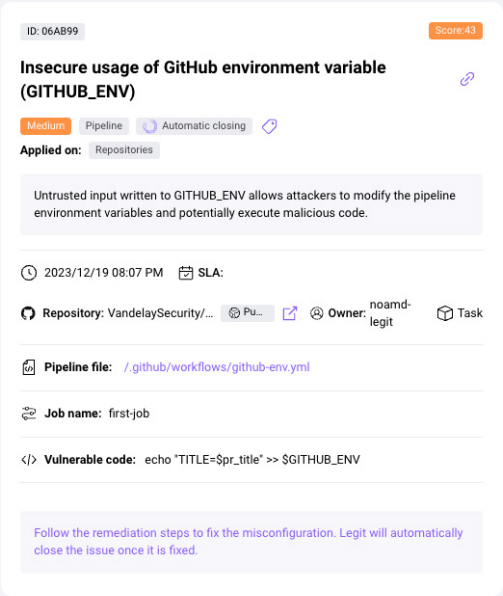
`Interpolation of untrusted input`



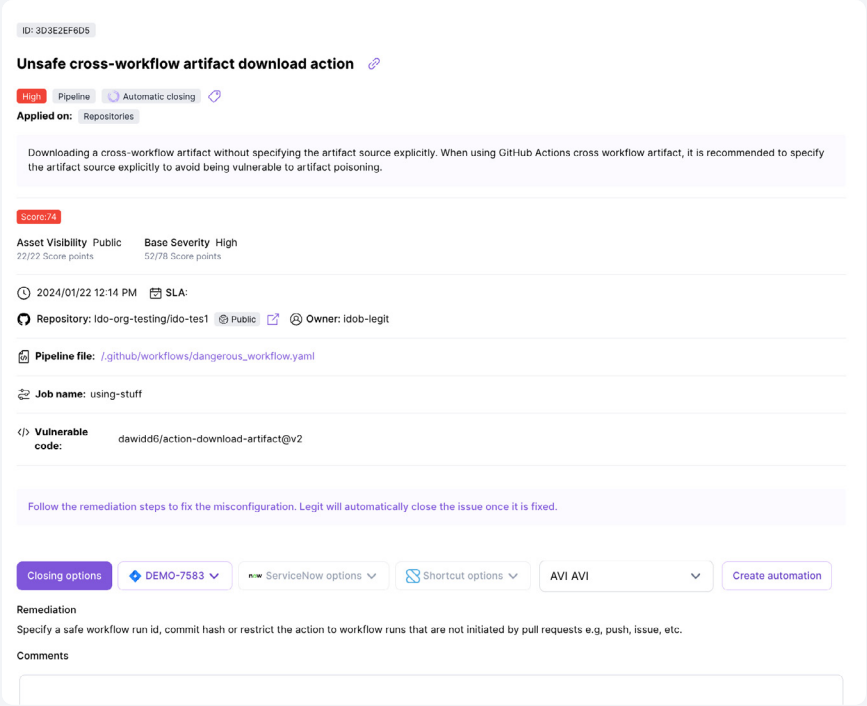
Execution of untrusted code



Insecure use of GITHUB_ENV

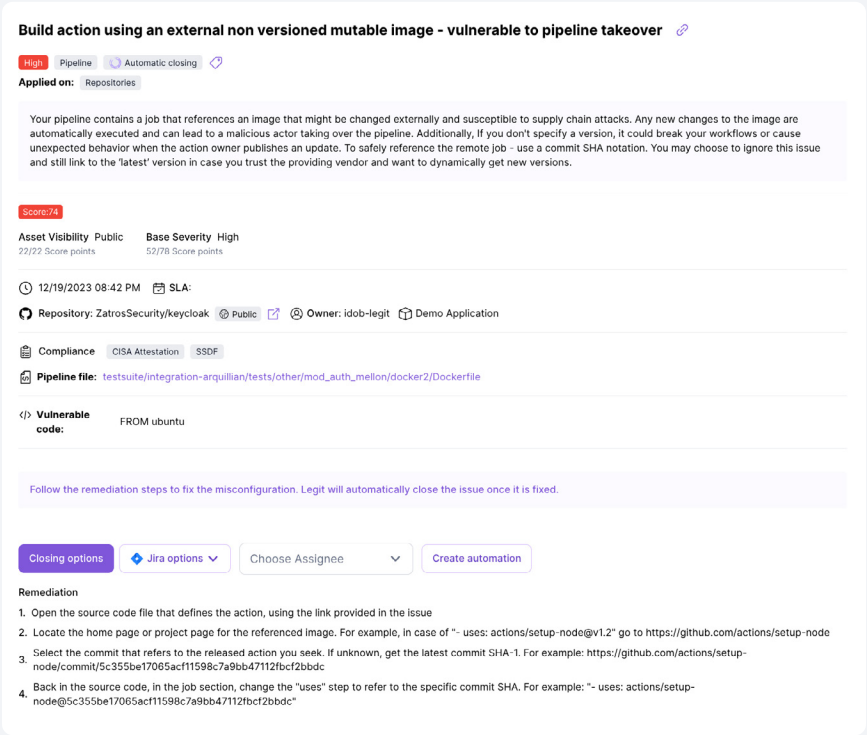


Unsafe cross-workflow artifact download

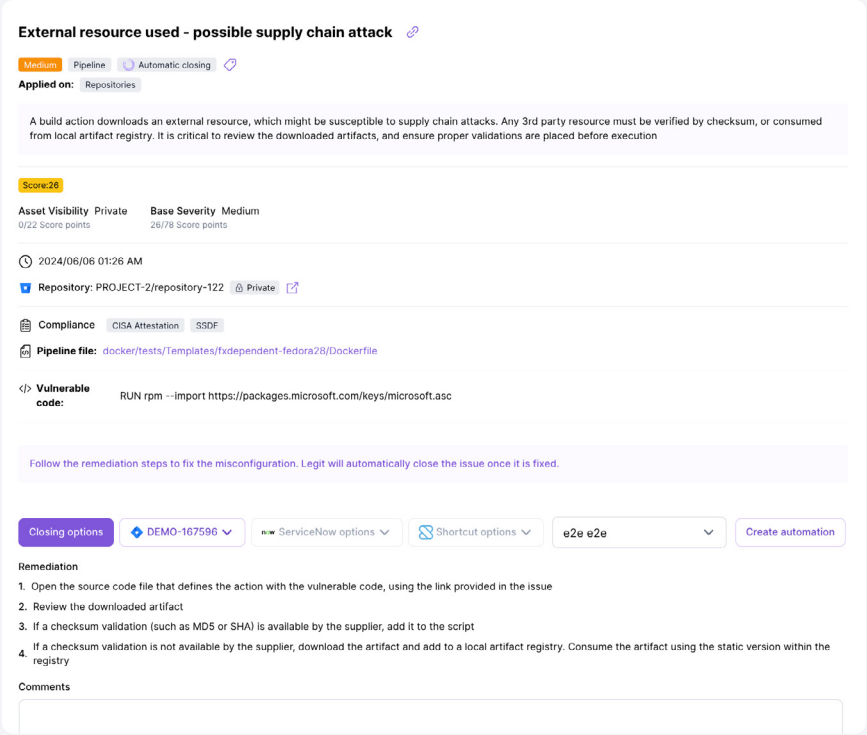


Third-Party Risk

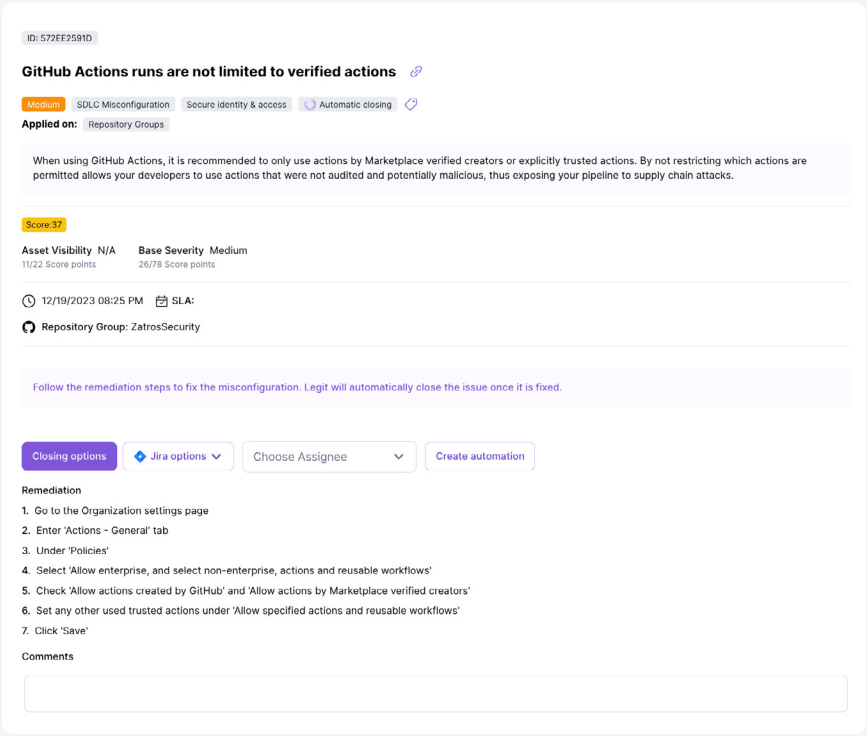
Build action using an external non-versioned mutable image – vulnerable to pipeline takeover



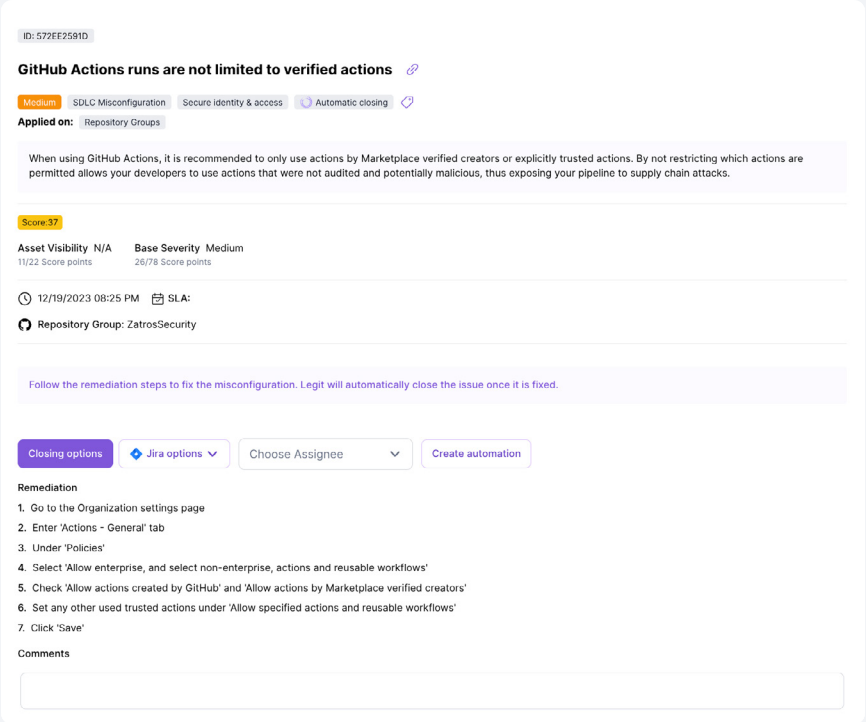
External unverified resource used



GitHub Actions is not restricted to selected repositories

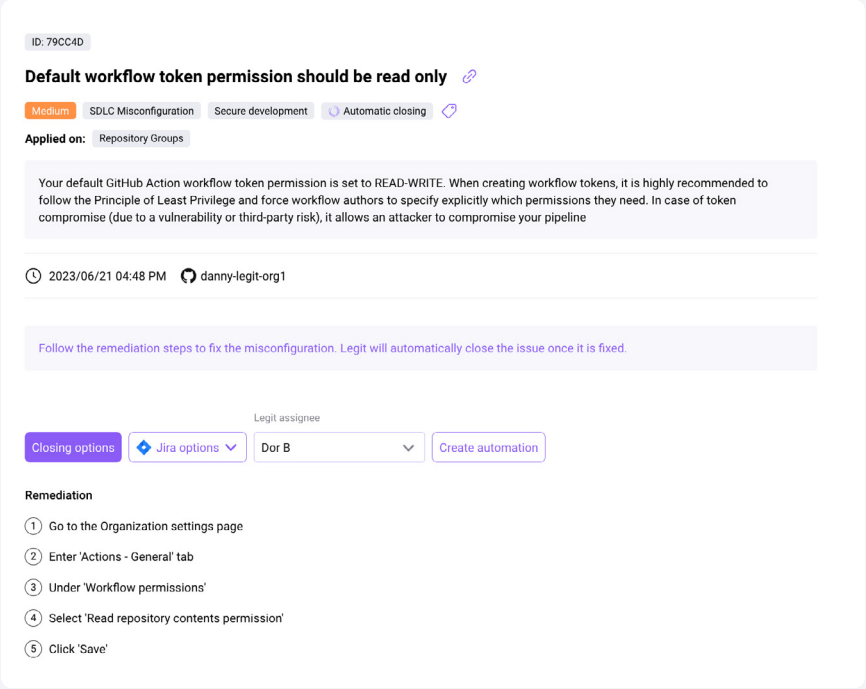


GitHub Actions runs are not limited to verified actions



Workflow Permissions

Default workflow token permission should be read only



Legit is a new way to manage your application security posture for security, product and compliance teams.

With Legit, enterprises get a cleaner, easier way to manage and scale application security, and address risks from code to cloud. Built for the modern SDLC, Legit tackles the toughest problems facing security teams, including GenAI usage, proliferation of secrets and an uncontrolled dev environment. Fast to implement and easy to use, Legit lets security teams protect their software factory from end to end, gives developers guardrails that let them do their best work safely, and delivers metrics that prove the success of the security program. This new approach means teams can control risk across the business — and prove it.

BOOK A DEMO TODAY

LEGIT SECURITY