Keeping your GitHub Actions and workflows secure Part 1: Preventing pwn requests

By jarlob

Published: 2021-08-03 · Archived: 2026-04-05 16:54:11 UTC

This post is the first in a series of posts about GitHub Actions security. Part 2, Part 3, Part 4

Secure your workflows with CodeQL: You can enable CodeQL for GitHub Actions to identify and fix the patterns described in this post.

In this article, we'll discuss some common security malpractices for GitHub Actions and workflows, and how to best avoid them. Our examples are based on real-world GitHub workflow implementation vulnerabilities the GitHub Security Lab has reported to maintainers.

GitHub workflows can be triggered through a wide variety of repository events. This includes events related to incoming pull requests (PR). There exists a potentially dangerous misuse of the `pull_request_target` workflow trigger that may lead to malicious PR authors (i.e. attackers) being able to obtain repository write permissions or stealing repository secrets.

***TL;DR**: Combining `pull_request_target` workflow trigger with an explicit checkout of an untrusted PR is a dangerous practice that may lead to repository compromise.*

Any automated processing of PRs from an external fork is potentially dangerous and such PRs should be treated like untrusted input. It is common CI/CD practice to ensure that when a new PR is submitted that it does not break the build for your project, that no functionality regressions are introduced, and that tests are passing. But when operating on untrusted PRs, such automated behavior can leave your repository exposed to abuse if you're not careful.

Since, by definition, a PR supplies code to any build or test logic in place for your project, attackers can achieve arbitrary code execution in a workflow runner operating on a malicious PR in a variety of ways. They may submit malicious changes to the existing build scripts like `make` or `powershell` files or redefine the build script in the `package.json` file. They can simply write their payload as a new test that will be run with others. They can achieve code execution even before the actual build happens. Npm packages for example may have custom `preinstall` and `postinstall` scripts, so running `npm install` would already trigger any malicious code if the attackers added a new package reference. Any modern build orchestration is complex enough to have multiple code injection points.

This is why you should never checkout and build PRs from untrusted sources on a local machine without carefully examining the code for the PR.

Due to the dangers inherent to automatic processing of PRs from forks, GitHub's standard `pull_request` workflow trigger by default prevents write permissions and secrets access to the target repository. However, in

some scenarios such access is needed to properly process the PR. To this end the `pull_request_target` workflow trigger was introduced.

Like how the introduction of Cross-Origin Resource Sharing (CORS) in the browser security model allowed a web site developer to relax the default Same Origin Policy (SOP), the introduction of `pull_request_target` trigger allowed a workflow writer to relax some restrictions to a target repository and must be used carefully. The main differences between the two triggers are:

1. Workflows triggered via `pull_request_target` have write permission to the target repository. They also have access to target repository secrets. The same is true for workflows triggered on `pull_request` from a branch in the same repository, but not from external forks. The reasoning behind the latter is that it is safe to share the repository secrets if the user creating the PR has write permission to the target repository already.
2. `pull_request_target` runs in the context of the target repository of the PR, rather than in the merge commit. This means the standard checkout action uses the target repository to prevent accidental usage of the user supplied code.

These safeguards enable granting the `pull_request_target` additional permissions. The reason to introduce the `pull_request_target` trigger was to enable workflows to label PRs (e.g. `needs review`) or to comment on the PR. The intent is to use the trigger for PRs that do not require dangerous processing, say building or running the content of the PR.

Together with the `pull_request_target`, a new trigger `workflow_run` was introduced to enable scenarios that require building the untrusted code and also need write permissions to update the PR with e.g. code coverage results or other test results. To do this in a secure manner, the untrusted code must be handled via the `pull_request` trigger so that it is isolated in an unprivileged environment. The workflow processing the PR should then store any results like code coverage or failed/passed tests in artifacts and exit. The following workflow then starts on `workflow_run` where it is granted write permission to the target repository and access to repository secrets, so that it can download the artifacts and make any necessary modifications to the repository or interact with third party services that require repository secrets (e.g. API tokens).

Below is an example of the intended usage in which the results of an unprivileged `pull_request` workflow are combined with a privileged workflow to leave a comment in response to a received PR:

ReceivePR.yml

```
 name: Receive PR

# read-only repo token
# no access to secrets
on:
  pull_request:

jobs:
  build:
```

```
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      # imitation of a build process
      - name: Build
        run: /bin/bash ./build.sh

      - name: Save PR number
        run: |
          mkdir -p ./pr
          echo ${{ github.event.number }} > ./pr/NR
      - uses: actions/upload-artifact@v2
        with:
          name: pr
          path: pr/
```

CommentPR.yml

```
 name: Comment on the pull request

# read-write repo token
# access to secrets
on:
  workflow_run:
    workflows: ["Receive PR"]
    types:
      - completed

jobs:
  upload:
    runs-on: ubuntu-latest
    if: >
      github.event.workflow_run.event == 'pull_request' &&
      github.event.workflow_run.conclusion == 'success'
    steps:
      - name: 'Download artifact'
        uses: actions/github-script@v3.1.0
        with:
          script: |
            var artifacts = await github.actions.listWorkflowRunArtifacts({
               owner: context.repo.owner,
               repo: context.repo.repo,
               run_id: ${{github.event.workflow_run.id }},
            });
```

```
            var matchArtifact = artifacts.data.artifacts.filter((artifact) => {
              return artifact.name == "pr"
            })[0];
            var download = await github.actions.downloadArtifact({
               owner: context.repo.owner,
               repo: context.repo.repo,
               artifact_id: matchArtifact.id,
               archive_format: 'zip',
            });
            var fs = require('fs');
            fs.writeFileSync('${{github.workspace}}/pr.zip', Buffer.from(download.data));
      - run: unzip pr.zip

      - name: 'Comment on PR'
        uses: actions/github-script@v3
        with:
          github-token: ${{ secrets.GITHUB_TOKEN }}
          script: |
            var fs = require('fs');
            var issue_number = Number(fs.readFileSync('./NR'));
            await github.issues.createComment({
              owner: context.repo.owner,
              repo: context.repo.repo,
              issue_number: issue_number,
              body: 'Everything is OK. Thank you for the PR!'
            });
```

This example saves the PR number as a workflow artifact, but can be easily extended to pass code coverage messages or similar PR artifacts in the same way.

Note that:

1. The example above comments on the PR with the help of `github-script` instead of a more direct PR driven action. This is because not all actions can be used from `workflow_run` if they expect to find all their needed information in the context object. The `workflow_run` context is different from the `pull_request` context and it doesn't contain, for example, the PR number. Such actions need to be updated to accept optional explicit input parameters to provide what is missing in the `workflow_run` context.
2. Incoming data from artifacts is potentially untrusted. When used in a safe manner, like reading PR numbers or reading a code coverage text to comment on the PR, it is safe to use such untrusted data in the privileged workflow context. However if the artifacts were, for example, binaries built from an untrusted PR, it would be a security vulnerability to run them in the privileged `workflow_run` workflow context. Artifacts resulting from untrusted PR data are themselves untrusted and should be treated as such when handled in privileged contexts.

As you can see, the usage of two workflows and passing around workflow artifacts introduces some overhead. If your workflow scenario simply requires commenting on the PR, but does not require a check out of the modified code, using `pull_request_target` is a logical shortcut.

Unfortunately some repository workflows take this a step further and use `pull_request_target` with an explicit PR checkout, for example:

```
 # INSECURE. Provided as an example only.
on:
  pull_request_target

jobs:
  build:
    name: Build and test
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v2
      with:
        ref: ${{ github.event.pull_request.head.sha }}

    - uses: actions/setup-node@v1
    - run: |
        npm install
        npm build

    - uses: completely/fakeaction@v2
      with:
        arg1: ${{ secrets.supersecret }}

    - uses: fakerepo/comment-on-pr@v1
      with:
        message: |
          Thank you!
```

The potentially untrusted code is being run during `npm install` or `npm build` as the build scripts and referenced packages are controlled by the author of the PR.

Having said that, mixing `pull_request_target` with an explicit PR checkout is not always vulnerable. The workflow may, for example:

- Reformat and commit the code
- Checkout both base and head repositories and generate a diff
- Run grep on the checked out source.

Generally speaking, when the PR contents are treated as passive data, i.e. not in a position of influence over the build/testing process, it is safe. But the repository owners must be extra careful not to trigger any script that may

operate on PR controlled contents like in the case of `npm install`.

To remediate the issue, repository owners could:

- Avoid using `pull_request_target` if the workflow doesn't need write repository permissions and doesn't use any repository secrets. They can simply use the `pull_request` trigger instead.
- Assign repository privileges only where needed explicitly through `pull_request` and `workflow_run` as in our previous example.
- Add a condition to the `pull_request_target` to run only if a certain label is assigned the PR, like `safe to test` that indicates the PR has been vetted by someone with write privileges to the target repository. Note that this kind of label based verification is still prone to a race condition in which the attacker may push new changes after the workflow was approved (labeled), but has not started yet. As such this approach should only be used as a temporary solution, until a proper fix from the options above is applied. Since external users do not have the permission to assign labels, this effectively requires repository owners to manually review changes first and is also prone to human error.

```
 # Only as a temporary fix.
on:
  pull_request_target:
    types: [labeled]

jobs:
  build:
    name: Build and test
    runs-on: ubuntu-latest
    if: contains(github.event.pull_request.labels.*.name, 'safe to test')
```

Note that there is an important "gotcha" to any remediation put in place for a vulnerable workflow. All PRs that were opened before a fix was made to the vulnerable workflow will use the version of the workflow as it existed at the time the PR was opened. That means that if there is a pending PR, any updates to the PR may still abuse the vulnerable workflow. It is advisable to either close or rebase such PRs if untrusted commits may be added to them after a vulnerable workflow is fixed.

You may ask yourself: if the `pull_request_target` workflow only checks out and builds the PR, i.e. runs untrusted code but doesn't reference any secrets, is it still vulnerable?

Yes it is, because a workflow triggered on `pull_request_target` still has the read/write repository token in memory that is potentially available to any running program. If the workflow uses `actions/checkout` and does not pass the optional parameter `persist-credentials` as false, it makes it even worse. The default for the parameter is `true`. It means that in any subsequent steps any running code can simply read the stored repository token from the disk. If you don't need a repository write access or secrets, just stick to the `pull_request` trigger.

We have also noticed a pattern that has a vulnerable intent of use, however due to misunderstanding of `pull_request_target` ends up being a broken, but not vulnerable, workflow. In these cases, repositories

switched to using the `pull_request_target` trigger, but use the default values for the `actions/checkout` action, for example:

```
 # The workflow is broken. DO NOT use it in production.
on: [push, pull_request_target]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Build and test
        run: /bin/bash ./build.sh && /bin/bash ./runtests.sh
      - name: Report
        if: failure() && github.event.action != 'push'
        with: fancy/commenter@v1
          message: |
            Some checks have failed.
```

This doesn't do what the authors most likely intended. In case of a PR, it builds the latest changeset from the target repository. The workflow is broken in the sense that it does not actually build the PR, but luckily is not vulnerable, since there is no explicit checkout of the actual PR contents or unsafe handling of those contents.

Conclusion

In this post we've examined some of the common issues when processing untrusted PR input to your GitHub workflows. It is important to understand the various privilege levels that `pull_request`, `pull_request_target`, and `workflow_run` afford the code processing the incoming PR. Be careful when using `pull_request_target` and only use it when you actually need the privileged context of the target repo available in your workflow, especially when combined with explicit handling of the contents of an untrusted PR.

This post is the first in a series of posts about GitHub Actions security. Read the next post

---