About dependency review

In this article

Dependency review lets you catch insecure dependencies before you introduce them to your environment, and provides information on license, dependents, and age of dependencies.

Dependency review is enabled on public repositories. Dependency review is also available in private repositories owned by organizations that use GitHub Enterprise Cloud and have a license for GitHub Advanced Security. For more information, see "About GitHub Advanced Security."

About dependency review

Dependency review helps you understand dependency changes and the security impact of these changes at every pull request. It provides an easily understandable visualization of dependency changes with a rich diff on the "Files Changed" tab of a pull request. Dependency review informs you of:

- Which dependencies were added, removed, or updated, along with the release dates.
- How many projects use these components.
- Vulnerability data for these dependencies.

If a pull request targets your repository's default branch and contains changes to package manifests or lock files, you can display a dependency review to see what has changed. The dependency review includes details of changes to indirect dependencies in lock files, and it tells you if any of the added or updated dependencies contain known vulnerabilities.

Sometimes you might just want to update the version of one dependency in a manifest and generate a pull request. However, if the updated version of this direct dependency also has updated dependencies, your pull request may have more changes than you expected. The dependency review for each manifest and lock file provides an easy way to see what has changed, and whether any of the new dependency versions contain known vulnerabilities.

By checking the dependency reviews in a pull request, and changing any dependencies that are flagged as vulnerable, you can avoid vulnerabilities being added to your project. For more information about how dependency review works, see "Reviewing dependency changes in a pull request."

For more information about configuring dependency review, see "Configuring dependency review."

Dependabot alerts will find vulnerabilities that are already in your dependencies, but it's

much better to avoid introducing potential problems than to fix problems at a later date. For more information about Dependabot alerts, see "About Dependabot alerts."

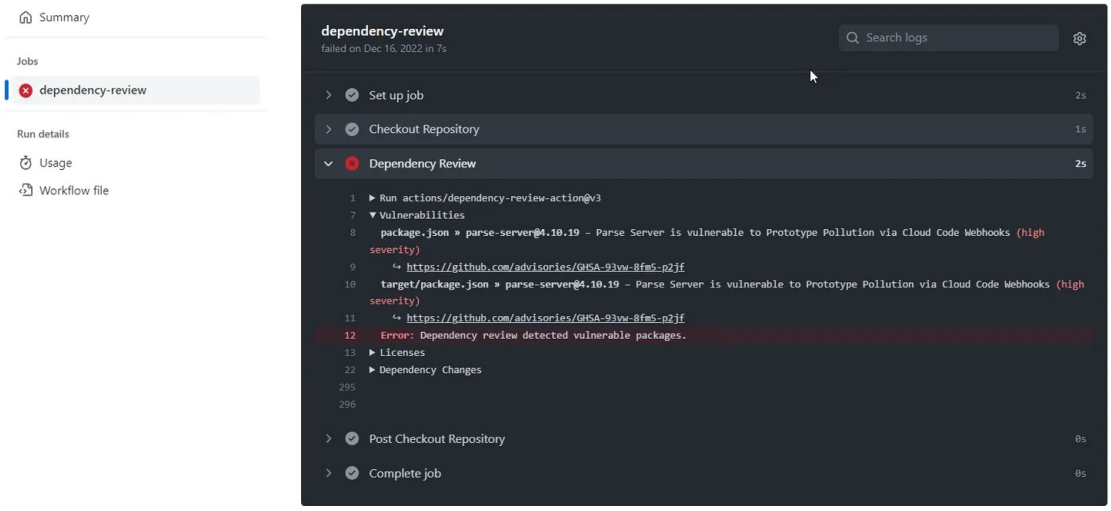
Dependency review supports the same languages and package management ecosystems as the dependency graph. For more information, see "About the dependency graph."

For more information on supply chain features available on GitHub, see "About supply chain security."

Dependency review enforcement

The action is available for all public repositories, as well as private repositories that have GitHub Advanced Security enabled.

You can use the dependency review action in your repository to enforce dependency reviews on your pull requests. The action scans for vulnerable versions of dependencies introduced by package version changes in pull requests, and warns you about the associated security vulnerabilities. This gives you better visibility of what's changing in a pull request, and helps prevent vulnerabilities being added to your repository. For more information, see `dependency-review-action`.



By default, the dependency review action check will fail if it discovers any vulnerable packages. A failed check blocks a pull request from being merged when the repository owner requires the dependency review check to pass. For more information, see "About protected branches."

The action uses the dependency review REST API to get the diff of dependency changes between the base commit and head commit. You can use the dependency review API to get the diff of dependency changes, including vulnerability data, between any two commits on a repository. For more information, see "Dependency Graph." The action also considers dependencies submitted via the dependency submission API. For more information about the dependency submission API, see "Using the Dependency submission API."

> **Note:** The dependency review API and the dependency submission API work together. This means that the dependency review API will include dependencies submitted via the dependency submission API. This feature is currently in public beta and subject to change.

You can configure the dependency review action to better suit your needs. For example, you can specify the severity level that will make the action fail, or set an allow or deny list for licenses to scan. For more information, see "Configuring dependency review."

Best practices for using the dependency review API and the dependency submission API together

The dependency review API and the dependency review action both work by comparing dependency changes in a pull request with the state of your dependencies in the head commit of your target branch, which is usually your default branch.

If your repository only depends on statically defined dependencies in one of GitHub's supported ecosystems, the dependency review API and the dependency review action work consistently.

However, you may want your dependencies to be scanned during a build and then uploaded to the dependency submission API. In this case, there are some best practices you should follow to ensure that you don't introduce a race condition when running the processes for the dependency review API and the dependency submission API, since it could result in missing data.

The best practices you should take will depend on whether you use GitHub Actions to access the dependency submission API and the dependency review API, or whether you use direct API access.

Using GitHub Actions to access the dependency submission API and the dependency review API

If you use GitHub Actions to access the dependency submission API or the dependency review API:

- Make sure you run all of your dependency submission actions in the same GitHub Actions workflow as your dependency review action. This will give you control over the order of execution, and it will ensure that dependency review will always work.
- If you do choose to run the dependency review action separately, for instance, as a required workflow, you should:
  - Set `retry-on-snapshot-warnings` to `true`.
  - Set `retry-on-snapshot-warnings-timeout` to slightly exceed the typical run time (in seconds) of your longest-running dependency submission action.

Using direct API access to the dependency submission API and the dependency review API

If you don't use GitHub Actions, and your code relies on direct access to the dependency submission API and the dependency review API:

- Make sure you run the code that calls the dependency submission API first, and then run the code that calls the dependency review API afterwards.
- If you do choose to run the code for the dependency submission API and the dependency review API in parallel, you should implement a retry logic and note the following:
  - When there are snapshots missing for either side of the comparison, you will see an explanation for that in the `x-github-dependency-graph-snapshot-warnings` header (as a base64-encoded string). Therefore, if the header is non-empty, you should consider retrying.
  - Implement a retry logic with exponential backoff retries.
  - Implement a reasonable number of retries to account for the typical runtime of your dependency submission code.