

This is the public version of the design doc.

Dependency-diff Visualization as a Scorecard GitHub Action

Visibility: Public

Status: Current

Authors: [aidenwang9867](#)

Reviewers: [laurentsimon](#), [azeemsshaikh38](#), [naveensrinivasan](#)

Last major revision: Aug 1, 2022

Motivation and Objective

Users now can use the [Dependency-diff API and CLI](#) to surface Scorecard results for their dependencies. We might also want to expand the scope of the API usage to a GitHub pull request CI workflow, by implementing the “dependency-diff” as an additional running mode of the [Scorecard GitHub Action](#).

With this design in place, the Scorecard Action will be further empowered to surface Scorecard checking results for those dependency-diffs (changes) between a base and an arriving head in a GitHub pull request (PR), visualizing them in a PR issue comment as a report (similar to the CodeCov one), and in a check run as annotations.

Background

In 2019, Googlers were active in over 70,000 repositories on GitHub, pushing commits and opening pull requests on over 40,000 repositories ([Source: Open Source by the Numbers at Google](#)). With commits & pull requests becoming the most common and popular way to contribute to open source codebases and repositories, it also brings higher risks for a code commit to introduce insecure dependencies and thus lead to malicious exploits by attackers to affect the repository itself and users who use it. Tools such as [GitHub Dependabot](#) and [Mend Renovate](#) are developed to help developers address this issue and find healthy dependencies for their open-source projects.

GitHub Dependabot

GitHub Dependabot is an automated dependency updating tool built into GitHub that can be configured to GitHub repository workflows and automatically keeps dependencies up to date, alleviating the pain by updating them automatically. Moreover, it helps reduce the risk of

exploiting dependency vulnerabilities to attack a repository. It [performs a scan](#) to find vulnerable dependencies and sends Dependabot alerts when (1) new vulnerabilities are added to the [GitHub Advisory Database](#), and (2) the dependency graph for a repository changes. For (1), the GitHub Advisory Database uses four threat intelligence sources to detect if a new vulnerability is added. For (2), the dependency graph change can be detected not only between the newly arrived commit and the previous one but also in a GitHub [pull request](#) (PR). This way, dependencies can be free from security vulnerabilities before they reach the codebase.

Mend Renovate

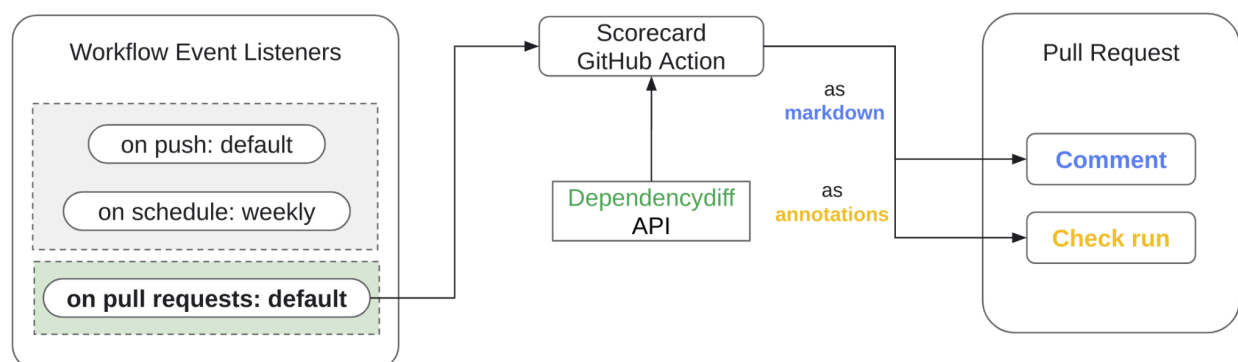
Renovate provides automated dependency updates with multi-platform and multi-language support. Compared to Dependabot, Renovate has an independent [Dependency Dashboard](#) to manage all history dependency changes. It creates one single PR for developers to solve all dependency updates at once, while Dependabot gives each update a separate PR. It can be set up as more versatile and flexible.

Scorecard Action

The [Scorecard Action](#) runs [ossf-scorecard](#) checks as a [GitHub code scanning tool](#) to harness [GitHub's alert management abilities](#). The current running modes of the Scorecard Action include (1) a scheduled weekly run and (2) a run triggered by a push to the default branch.

To remediate the risks of introducing tampered dependencies, as of June 2022, Scorecard has already proposed two dependency-related security checks, [Dependency-Update-Tool](#) and [Pinned-Dependencies](#), which check if the project uses tools to help update its dependencies and if the project declares and pin dependencies, respectively. However, being able to check for dependency changes in different branches and code commits is still a TODO for the Scorecard Action.

Framework



This figure shows the framework of this design. We are adding a new workflow event listener that can be triggered by either (1) creating a new pull request to the default branch or (2)

submitting a new code commit to an existing pull request to the default branch. Same as the other two Scorecard Actions' running modes, this is a workflow automation configured in the [GitHub Actions and the repository workflow](#). The difference is, that we use this new one as a pre-submit check run on those dependency changes.

Inputs

Workflow Environment Variable Inputs

Since the Dependency-diff Action workflow is triggered by a pull request, it will automatically fetch the three parameters `repoURI`, `base`, and `head` from the [GitHub workflow environment variables](#).

Input Parameters	GitHub Default Workflow Env
<code>repoURI</code>	<code>GITHUB_REPOSITORY</code>
<code>base</code>	<code>GITHUB_BASE_REF</code>
<code>head</code>	<code>GITHUB_HEAD_REF</code>

In the above table, the `GITHUB_REPOSITORY` variable stores the `repoURI` of the current GitHub repository, e.g. "ossf/allstar", `GITHUB_BASE_REF` stores the `base` branch name reference, e.g. "main", and `GITHUB_HEAD_REF` stores the `head` branch name reference, e.g. "dev". Here we give the API the branch name references as the base and head inputs, so it will handle and convert them into the latest commitSHAs of the specific branch.

Action Inputs

Furthermore, the Original Scorecard Actions' inputs are defined in [actions.yaml](#). For the Dependency-diff Action in a pull request, none of the original inputs are used and three additional ones are created:

name	description	required	default
<code>checks</code>	Checks to run.	false	<code>Maintained,Security-Policy,License,Code-Review,SAST</code>
<code>change_types</code>	Change types to check.	false	<code>added</code>
<code>pull_request_head_sha</code>	The commitSHA of the pull request head	false	<code>\${{github.event.pull_request.head.sha}}</code>

	reference.		sha}}
--	------------	--	-------

All three of the inputs are not required and have their default values. `checks` and `change_types` are inputs for the Dependency-diff API, and the `pull_request_head_sha` is a special requirement for the annotations API, which will be discussed later.

Outputs

The outputs of the Dependency-diff Action contain two separate parts: (1) visualization as a pull request comment and (2) visualization as check run annotations.

The reason we visualize the dependency-diff results in two different positions:

- Justifications for visualizing as a comment:
 - The comments in a pull request provide a straightforward and clear UI for users to take a quick look at the Scorecard checking results for their dependency changes.
 - The pull request comments support the markdown syntax, making it easier to create tags, links, bold fonts, etc. that are related to the dependencies.
- Justifications for visualizing as check run annotations.
 - Putting too much information in the comments will lead to long and redundant comments spamming the entire pull request.
 - Annotations are check run “notes” given in a specific changed file with a start line and an end line. See the CodeCov annotations example [here](#). Compared to a pull request comment, an annotation can contain more information and details, which is perfect to supplement those fields that are dropped in the comments.
 - The dependency ManifestPath returned by the GitHub Dependency Review API is a perfect input for an annotation file path.

Visualization as a pull request comment [\[example\]](#)

We visualize each dependency on a per-line basis in a [pull request issue comment](#) (a general comment). The [CreateComment](#), [EditComment](#), and [ListComments](#) API in the package [github.com/google/go-github@v45](#) is used for this purpose. Specifically, `CreateComment` is used to create a new comment if there is none, `ListComments` is used for listing the existing comments so that we can search through its returned list to find our previous comment to a pull request. If such a comment exists, `EditComment` will be used for updating that comment and generating a new dependency-diff visualization report there.

Change type tags

The default change types are “added” and “removed”. To make it more explicit in a pull request comment, we also create an “updated” tag for those dependencies updating from an old version. The markdown back quote “`” is used to create such a tag.

Sorting by the aggregate score

We first calculate the aggregate score for each dependency if it has a valid ScorecardCheckResult field, then use (ChangeType, AggregateScore) as the composite key to sort the dependency entries. At last, we'll see added dependencies shown in the front, then removed dependencies. Also, a dependency with a higher aggregate score will be shown in the front. The aggregate score of a package will also be shown in a Score tag, using the markdown back quote "`".

Generating the deps.dev package link

The Open Source Insights deps.dev provides a comprehensive overview of the packages in different ecosystems. For each dependency with a valid package ecosystem, package name, and package version, deps.dev provides the following REST API to query the existence of the corresponding deps.dev package page:

```
deps.dev/_/s/{ecosystem}/p/{name}/v/{version}
```

By sending standard HTTP requests to this API for each dependency package, we can fetch its response status and decide if there is a valid page for a package - if the response status is 200 OK, we will create a deps.dev link for the dependency leading to its deps.dev page. For example, we can request deps.dev/_/s/PyPI/p/tensorflow/v/2.7.0 and create the deps.dev/pypi/tensorflow/2.7.0 link for it if the response status is 200 OK.

Note the package name needs to be escaped using the url.PathEscape() API if it contains a path, especially for a lot of Go packages that use their URI paths to their source repositories as the package names, e.g. github.com/ossf/scorecard/v4/checks.

Visualization as check run annotations [\[example\]](#)

Dependency-diff annotations is generated on a per-check basis. For a single dependency-diff, there could be multiple annotation entries for each of its checks. This way, the check score, and check reason will be more explicit to users, and the check raw details will be put into the raw output section. Users might want to click the raw output button to see the raw details of a check, e.g., for the Security-Policy check, the security policy file path will be put into the raw details if such a file is detected.

Example raw outputs:

```
Info: security policy detected in current repo: SECURITY.md:1
```

By default, we will make every annotation a check notice. If the score of a single check is lower than 60% (6.0) of the maximum score (10), we will change that annotation to a check warning with a warning sign and showing the package name in yellow fonts, such that it is easier for a user to quickly identify the checks with lower scores. This might be helpful to find a potential insecure dependency.

Project Management

Time Schedule and Work estimates

Estimated time for developing the visualization in comment: 1.5 weeks


Estimated time for developing the visualization as annotations: 0.5 weeks.

Estimated time for adding the unit tests: 0.5 weeks.

E2e tests: TBD, can skip this and add in the future when this feature is ready to release.

Documentation plan

This one will be used as the design document for the Dependency-diff Visualization as a Scorecard GitHub Action feature.

 [Public] [Design Doc] Scorecard Dependency-diff API is another design document associated with this one.

Testing plan

Unit tests and e2e tests are needed after incorporating this module into the scorecard project. Specifically, we'll need unit tests to cover the visualization as a comment and visualization as annotations, making sure they won't crash.

Since this feature will remain as an experimental one once it is merged into the Scorecard Action repo, the e2e tests can be added later when we add the Scorecard REST API support and it is ready for release.

Further Discussions

This section contains other discussions and issues that have already been resolved (in gray fonts) or are currently pending solutions.