Overview - Security model

Whenever a PR job is run on CircleCI, we want to build `angular.io` and host the build artifacts on a publicly accessible server so that collaborators (developers, designers, authors, etc) can preview the changes without having to checkout and build the app locally.

This document discusses the security considerations associated with moving build artifacts as part of the CI process and serving them publicly.

Security objectives

- **Prevent hosting arbitrary content on our servers.** Since there is no restriction on who can submit a PR, we cannot allow arbitrary, untrusted PRs' build artifacts to be hosted.
- **Prevent overwriting other people's hosted build artifacts.** There needs to be a mechanism in place to ensure that the hosted content does indeed correspond to the PR indicated by its URL.
- **Prevent arbitrary access on the server.** Since the PR author has full access over the build artifacts that would be hosted, we must ensure that the build artifacts will not have arbitrary access to the server or expose sensitive info.

Issues / Caveats

- Because the PR author can change the scripts run on CI, any security mechanisms must be immune to such changes.
- For security reasons, encrypted CircleCI variables are not available to PRs, so we can't rely on them to implement security.

Implemented approach

In a nutshell

The implemented approach can be broken up to the following sub-tasks:

1. Receive notification from CircleCI of a completed build.
2. Verify that the build is valid and can have a preview.
3. Download the build artifact.
4. Fetch the PR's metadata, including author and labels.
5. Check whether the PR can be automatically verified as "trusted" (based on its author or labels).
6. If necessary, update the corresponding PR's verification status.
7. Deploy the artifacts to the corresponding PR's directory.

8. Prevent overwriting previously deployed artifacts (which ensures that the guarantees established during deployment will remain valid until the artifacts are removed).
9. Prevent hosted preview files from accessing anything outside their directory.

Implementation details

This section describes how each of the aforementioned sub-tasks is accomplished:

1. **Receive notification from CircleCI of a completed build**

CircleCI is configured to trigger a webhook on our preview-server whenever a build completes. The payload contains the number of the build that completed.

2. **Verify that the build is valid and can have a preview.**

   We cannot trust that the data in the webhook trigger is authentic, so we only extract the build number and then run a direct query against the CircleCI API to get hold of the real data for the given build number.

   We perform a number of preliminary checks:

   - Was the webhook triggered by the designated CircleCI job (currently `aio_preview`)?
   - Was the build successful?
   - Are the associated GitHub organization and repository what we expect (e.g. `angular/angular`)?
   - Has the PR touched any files that might affect the angular.io app (currently the `aio/` or `packages/` directories, ignoring spec files)?

   If any of the preliminary checks fails, the process is aborted and not preview is generated.

3. **Download the build artifact.**

   Next we make another call to the CircleCI API to get a list of the URLs for artifacts of that build. If there is one that matches the configured artifact path, we download the contents of the build artifact and store it in a local folder. This download has a maximum size limit to prevent PRs from producing artifacts that are so large they would cause the preview server to crash.

4. **Fetch the PR's metadata, including author and labels**.

   Once we have securely downloaded the artifact for a build, we retrieve the PR's metadata - including the author's username and the labels - using the GitHub API. To avoid rate-limit restrictions, we use a Personal Access Token (issued by @mary-poppins).

5. **Check whether the PR can be automatically verified as "trusted"**.

"Trusted" means that we are confident that the build artifacts are suitable for being deployed and publicly accessible on the preview server. There are two ways to check that:

1. We can verify that the PR has a pre-determined label, which marks it as "safe for preview". Such a label can only have been added by a maintainer (with the necessary rights) and designates that they have manually verified the PR contents.
2. We can verify (again using the GitHub API) the author's membership in one of the trusted GitHub teams. For this operation, we need a Personal Access Token with the `read:org` scope issued by a user that can "see" the specified GitHub organization. Here too, we use the token by @mary-poppins.

6. **If necessary update the corresponding PR's verification status**.

   Once we have determined whether the PR is considered "trusted", we update its "visibility" (i.e. whether it is publicly accessible or not), based on the new verification status. For example, if a PR was initially considered "not trusted" but the check triggered by a new build determined otherwise, the PR (and all the previously downloaded previews) are made public. It works the same way if a PR has gone from "trusted" to "not trusted".

7. **Deploy the artifacts to the corresponding PR's directory.**

   With the preceding steps, we have verified that the build artifacts are valid. Additionally, we have determined whether the PR can be trusted to have its previews publicly accessible or whether further verification is necessary.

   The artifacts will be stored to the PR's directory, but will not be publicly accessible unless the PR has been verified. Essentially, as long as sub-tasks 2, 3, 4 and 5 can be securely accomplished, it is possible to "project" the trust we have in a team's members through the PR to the build artifacts.

8. **Prevent overwriting previously deployed artifacts**.

   In order to enforce this restriction (and ensure that the deployed artifacts' validity is preserved throughout their "lifetime"), the server that handles the artifacts (currently a Node.js Express server) rejects builds that have already been handled. *Note: A PR can contain multiple builds; one for each SHA that was built on CircleCI.*

9. **Prevent hosted preview files from accessing anything outside their directory.**

   Nginx (which is used to serve the hosted preview) has been configured to not follow symlinks outside of the directory where the preview files are stored.

Assumptions / Things to keep in mind

- Other than the initial webhook trigger, which provides a build number, all requests for data come from the preview-server making requests to well defined API endpoints (e.g. CircleCI and Github). This means that any secret access keys need only be stored on the preview-server and not on any of the CI build infrastructure (e.g. CircleCI).
- Each trusted PR author has full control over the content that is hosted as a preview for their PRs. Part of the security model relies on the trustworthiness of these authors.
- Adding the specified label on a PR to mark it as trusted, gives the author full control over the content that is hosted for the specific PR preview (e.g. by pushing more commits to it). The user adding the label is responsible for ensuring that this control is not abused and that the PR is either closed (one way of another) or the access is revoked.