Allowlist-based vulnerability scanning with Container Analysis

This tutorial describes how to build and use Kritis Signer with Container Analysis to check for vulnerabilities in container images built with Cloud Build. Kritis Signer checks identified vulnerabilities against a vulnerability policy. The Cloud Build build step fails if an identified vulnerability violates the vulnerability policy.

This document provides sample Cloud Build build configuration files with build steps that demonstrate how to nerabilities. Binary Authorization users can incorporate these build steps into a build pipeline that signs an ation (/binary-authorization/docs/key-concepts#attestations). To learn how to create an attestation-signing e, see Cloud Build integration (/binary-authorization/docs/cloud-build).

Overview

When securing your container-based software supply chain, it can be critically important to prevent a vulnerable container image from being deployed. Container Analysis provides vulnerability scanning (/container-registry/docs/vulnerability-scanning) services for container images. Kritis Signer is an open-source custom builder (/cloud-build/docs/cloud-builders) for Cloud Build that runs as a build step and uses Container Analysis to identify vulnerabilities in a container image. It then checks those vulnerabilities against a vulnerability policy. If any identified vulnerabilities violate the Kritis Signer vulnerability policy, the build fails. In the context of Binary Authorization, you can add a Kritis Signer build step to a container image build pipeline so that a policy violation would fail the build before an attestation is created.

In this tutorial you:

1. Clone the Kritis (https://github.com/grafeas/kritis) repository.
2. Build the Kritis Signer custom builder that scans for vulnerabilities with Container Analysis.
3. View a vulnerability policy.
4. Submit sample builds to Cloud Build that build a container image and check it for vulnerabilities.

- The first sample container image build, `cloudbuild-bad.yaml`, is a failure case. In this case, Kritis Signer finds vulnerabilities in container image that violates the vulnerability policy. The build step fails, terminating the build pipeline.
- The second sample container image build, `cloudbuild-good.yaml`, is a success case. In this case, Kritis Signer should not find a vulnerability in container image that violates your vulnerability policy. The build step should succeed.

Products used in this tutorial

This tutorial uses the following Google Cloud and open source products:

- Cloud Build (/cloud-build) is a Google Cloud product that executes your builds on Google Cloud infrastructure. Cloud Build executes your build as a series of build steps, where each build step is run in a Docker container. In this tutorial, the Kritis Signer is built into a Cloud Build custom builder that can be used thereafter in your build pipelines. Cloud Build is also used to build the example success and failure cases, demonstrating how to include vulnerability scanning into your build pipelines.
- Kritis (https://github.com/grafeas/kritis) is an open source solution for securing your software supply chain for Kubernetes applications. **Kritis Signer** is a Cloud Build custom builder that checks vulnerabilities identified by Container Analysis against a vulnerability policy.
- Container Registry (/container-registry) is a Google Cloud private container image registry that runs on Google Cloud. In this tutorial, the Kritis Signer custom builder and the sample container images are stored in Container Registry.
- Container Analysis (/container-registry/docs/container-analysis) is a Google Cloud product that provides vulnerability scanning and metadata storage for software artifacts. The service performs vulnerability scans on built software artifacts, such as the images in Container Registry, then stores the resulting metadata and makes it available for consumption through an API. In this tutorial, Kritis Signer calls Container Analysis to identify vulnerabilities in container images.

Set up

In this section, you perform a one-time setup of the system. This includes the following steps:

1. Set up your environment.
2. Enable APIs.
3. Set permissions on the service account Cloud Build uses to read from Container Analysis.
4. Clone the Kritis repo.
5. Build the Kritis Signer Cloud Build custom builder.
6. Edit a vulnerability policy.

In the next section, you execute sample build pipelines with Cloud Build.

Set up your environment

First, set ***PROJECT_ID*** to your Google Cloud project. This project is used for the entire tutorial.

```
t PROJECT_ID=PROJECT-ID
```

Where ***PROJECT-ID*** is the ID of the Google Cloud project.

```
t PROJECT_NUMBER=$(gcloud projects list --filter="${PROJECT_ID}" --format="value(PRO
```

Enable APIs

This tutorial uses the following Google Cloud products. Costs may be incurred.

- Container Registry - pricing (/container-registry/pricing)
- Container Analysis - pricing (/container-registry/pricing)
- Cloud Build - pricing (/cloud-build/pricing)

```
d --project=$PROJECT_ID services enable cloudbuild.googleapis.com
d --project=$PROJECT_ID services enable containerregistry.googleapis.com
d --project=$PROJECT_ID services enable containeranalysis.googleapis.com
d --project=$PROJECT_ID services enable containerscanning.googleapis.com
```

Set up permissions

Run the following command to give the Cloud Build service account read access to vulnerability information from Container Analysis.

```
d projects add-iam-policy-binding $PROJECT_ID --member serviceAccount:$PROJECT_NUMBE
```

Create the Kritis Signer custom builder

Run the following commands to get the code and configuration files used in this tutorial.

1. Clone the Kritis (https://github.com/grafeas/kritis) repo

   ```
   git clone --branch policy-check-v1.0.0 https://github.com/grafeas/kritis.git
   ```

   This repository contains the following:

   - Open source for **Kritis**.
   - **Build configuration files** used by Cloud Build to build the Kritis Signer custom builder.
   - An example **vulnerability policy**.
   - Example **Cloud Build build configuration files** that demonstrate Kritis Signer vulnerability scanning in a build pipeline.

2. Navigate into the `/kritis` directory:

```
cd kritis
```

3. Submit the Kritis Signer custom builder to your project

   This one-time step builds a custom builder that is used in your vulnerability pipelines.

   ```
   gcloud --project=$PROJECT_ID builds submit . --config deploy/kritis-signer/clou
   ```

Edit the vulnerability signing policy

This section describes the vulnerability policy Kritis Signer enforces. Edit this policy to determine allowable severity levels and list specific allowable vulnerabilities.

To view the vulnerability policy:

```
amples/policy-check/policy.yaml
```

It should look like this:

```
rsion: kritis.grafeas.io/v1beta1
 VulnzSigningPolicy
ata:
e: my-vsp

geVulnerabilityRequirements:
aximumFixableSeverity: MEDIUM
aximumUnfixableSeverity: MEDIUM
llowlistCVEs:
 projects/goog-vulnz/notes/CVE-2020-10543
 projects/goog-vulnz/notes/CVE-2020-10878
 projects/goog-vulnz/notes/CVE-2020-14155
```

Where:

- **allowlistCVEs** is list of vulnerabilities. Note that each entry must be an exact match of the note name.
- **maximumUnfixableSeverity** and **maximumFixableSeverity** each of these may be one of:
  - CRITICAL
  - HIGH
  - MEDIUM
  - LOW
  - BLOCK_ALL - Fails if any vulnerability is identified
  - ALLOW_ALL - Always succeeds

Test the Kritis Signer custom builder

In this section, you submit example builds to Cloud Build.

The build steps described in these builds can be used in a more complex build pipeline. When used with Binary rization, for example, the build pipeline could check for vulnerabilities and then create and sign an attestation. Build integration (/binary-authorization/docs/cloud-build) for more details on creating an attestation in Cloud

In the following steps, two container image builds are submitted to Cloud Build:

- Failure case: The first build creates a container image that contains a vulnerability that triggers a vulnerability policy violation. This build should fail.
- Success case: The second build creates a container image that contains vulnerabilities that should be allowed by the vulnerability policy. This build should succeed.

You may see different results if new vulnerabilities are identified by Container Analysis in the future. To ensure ss-case build passes, all identified vulnerabilities should be added to the `allowlistCVEs` section of the vulner (#edit_the_vulnerability_signing_policy).

Both example build configuration files contain the following steps:

1. The `build` step builds a Docker container image.
2. The `push` step pushes the newly built container image to Container Registry (/container-registry).
3. The `vulnsign` step analyzes the container image by:
    a. Waiting for Container Analysis to return vulnerability findings on the newly built container image.
    b. Checking the findings against the vulnerability policy.
    c. Succeeding or failing based on policy compliance.

Submit the failure-case sample build

This example builds a container image based on Debian 9. At present, the container has a vulnerability that is not allowed by the vulnerability policy described above. The Kritis Signer custom builder will fail the `vulnsign` build step.

To view the contents of the build file, enter the following command:

```
amples/policy-check/cloudbuild-bad.yaml
```

Run the build pipeline

```
d  --project=$PROJECT_ID builds submit --config=samples/policy-check/cloudbuild-bad.
```

You should see something like the following:

```
: (gcloud.builds.submit) build [BUILD-ID] completed with status "FAILURE"
```

Note the [BUILD-ID] value and set the $BUILD_ID environment variable for the next step.

```
t BUILD_ID=BUILD-ID
```

Alternatively, you can find the build ID of your last build by running the following command:

```
d builds list --limit 1
```

The build ID is the first returned field. Alter the `--limit` value to see more builds, or remove the flag altogether to view all builds.

Verify the result

```
l cat gs://${PROJECT_NUMBER}.cloudbuild-logs.googleusercontent.com/log-${BUILD_ID}.t
```

Submit the success-case sample build

To submit the success-case sample build to Cloud Build, do the following:

```
d --project=$PROJECT_ID builds submit --config=samples/policy-check/cloudbuild-good.
```

Again, make sure to note the build ID and store it in $BUILD_ID environment variable:

```
t BUILD_ID=BUILD-ID
```

Finally, to verify the result, enter the following command:

```
d --project=$PROJECT_ID builds describe $BUILD_ID | grep status
```

What's next

- See Cloud Build integration (/binary-authorization/docs/cloud-build) to learn how to use Cloud Build to sign a Binary Authorization attestation.
- Learn more about creating a custom Cloud Build step for signing and uploading Binary Authorization attestations on GitHub (https://github.com/GoogleCloudPlatform/cloud-builders-community/tree/master/binauthz-attestation)
- Learn more about Binary Authorization (/binary-authorization/docs/overview).
- Learn more about Container Analysis (/container-registry/docs/container-analysis) and its vulnerability scanning (/container-registry/docs/vulnerability-scanning) capability.
- Learn about Cloud Build community and custom cloud builders (/cloud-build/docs/configuring-builds/use-community-and-custom-builders)

Last updated 2020-08-22 UTC.