SCC Tools: **Connector** User Guide

Security Cloud Command Center Tools

Contents

The **Connector App** ingests security findings stored in a GCS bucket populated by a partner. The findings ingestion process is triggered when a new file completes the upload process to the bucket.

what happens next depends on the operation mode the **Connector App** is currently configured.

The connector app has two modes:

- *prod*: all files added to the configured GCS bucket are read and their findings are ingested and send to Cloud SCC.
- *demo*: When a file is added to the configured GCS bucket its full location on GCS is stored in Datastore. On the last file added to the bucket is stored, new files overwrite the info on datastore. A message can be posted to pub/sub topic to flush this cash and force processing of the file.

The mode can be changed posting a message to the **configuration** topic with a payload.

```
{"mode": "prod|demo"}
```

For example, using gcloud cli:

```
gcloud pubsub topics publish projects/${connector_project_id}/topics/configuration \
 --message "{\"mode\": \"prod\"}"
```

Or

```
gcloud pubsub topics publish projects/${connector_project_id}/topics/configuration \
 --message "{\"mode\": \"demo\"}"
```

It can also be done on the Google Cloud Console going to: “Pub/Sub” > “Topics” > click on the “configuration” topic name > “PUBLISH MESSAGE”

When on “demo” mode, to force execution of the last file uploaded to the bucket you need to post an empty message to the “flushbuffer” topic:

```
gcloud pubsub topics publish projects/${connector_project_id}/topics/flushbuffer \
 --message "{}"
```

A findings file must be utf-8 without BOM encoded and it must be a valid JSON or CSV/TSV files.

The findings file processing is done using a YAML configuration file, this file maps a known set of attributes and properties from the provided partner findings file.

Attributes are a set of fields that are common to all findings from all partners:

- **Id** the unique identifier of the finding;
- **Source ID** the partner identity, currently must be one of:
  - GOOGLE_ANOMALY_DETECTION
  - CLOUDFLARE
  - CROWDSTRIKE
  - DOME9
  - FORSETI
  - PALO_ALTO_NETWORKS
  - QUALYS
  - REDLOCK
- **Category** The security finding category according to the Partner classification
- **Asset Ids** A list of Google cloud asset ids, usually in the form of (any of these)
  - organization/<organization_id>
  - <project_id>/<asset_type>/<id>
- **Event Time** The date/time of the finding identification
- **URL** An URL with additional information on the finding on the partner original system.

Properties are partner specific information that will be stored as a set of <Key,Value> pairs, like for example: **severity**, **Solution**, **remediation** or **summary**.

Properties values are stored as strings, so if the key in the findings json file points to a nested json object, the connector app will **Stringify** the json object so that it can be sent to SCC and no information lost.

The YAML file used to guide the ingestion the findings has the following structure:

- A metadata section, with information regarding:
  - The **type** of the file:
    - json: for json files with a single json object or a jsonArray of objects
    - csv: for fixed position files. In this case another field, **delimiter** , is used to identify the separator: "," or TAB character;
  - The **org_name** with the organization name
  - The **root_element** with values:
    - null if the finding object is flat or
    - the key on the json file for the json object that contains the information to be used.
  - **deep_levels** used to map an inner json object related to the root_source to search for the data to be parsed
  - **mapped_ips** a key value custom map that can be used by the Connector app to link partner info to the corresponding info on the Google Cloud side, like for example an external IP linked to and GCE instance asset id
- A fixed value section, **fixed_fields**, with forced values to be ingested. Mostly used now for **sourceId** and **URL**.
- The actual mapped values section, **api_to_fields**, with fields to be read from the findings file, which includes the attributes not yet mapped and all the partner properties, on a subsection called **properties**

This is a sample YAML file for GOOGLE_ANOMALY_DETECTION findings

```yaml
type: json
org_name: organizations/<Your Organization ID>
root_element: null
deep_levels: !!seq [ assetIds ]
fixed_fields:
  sourceId: GOOGLE_ANOMALY_DETECTION
api_to_fields:
  id:
    transform: concat_organization_id
    path: id
  category: category
  assetIds:
    transform: to_array_asset_ids
    path: assetIds
  eventTime:
    transform: time_to_millis
    unit: 1000
    path: eventTime
  properties:
    action: properties.action
    serviceAccount: properties.serviceAccount
    storageBucket: properties.storageBucket
    product: properties.product
    summary: customSummary
```

Two important observations:

1) If a field is not mapped on the YAML file it will not be ingested.
2) Any field can have a **transform** which is a way to preprocess the field value before it is ingested. Currently we have transform's for converting date/time fields, to concatenate the organization id to the fiding id, to convert a single field value to a json array on the object that will be sent to SCC and a transform that uses the **mapped_ips** meta field to a from -> to transformation based on the original field value.

The directory **./connector/dm/mapper_samples** contains the default example for each one of the current partners.

If you update YAML file you can redeploy the cloud function to see the changes:

Simulate cloud function update

```
(cd setup; \
 pipenv run python3 update_cloud_function.py \
    --project_id ${connector_project_id} \
    --bucket_name ${connector_cf_bucket} \
    --cloud_function <CONNECTOR_CF_NAME> \
    --sa_file ${scc_sa_file} \
    --simulation)
```

CONNECTOR_CF_NAME is one of

- configuration
- flushbuffer
- forwardfilelink
- translation

So if you changed the translation function it would be **-cf translation**

Run cloud function update

```
(cd setup; \
 pipenv run python3 update_cloud_function.py \
    --project_id ${connector_project_id} \
    --bucket_name ${connector_cf_bucket} \
    --cloud_function <CONNECTOR_CF_NAME> \
    --sa_file ${scc_sa_file} \
    --no-simulation)
```