# Cloud Analytics Sigma Rule Quickstart

The goal of this document is to cover information relating to Sigma rule creation, usage, and best practices.

## What is Sigma

The Sigma project consists of two main components, Sigma rules, and Sigma converters. Sigma rules are platform independent, and allow for the security community to quickly provide core queries for known vulnerability or attacker behavior. The generic rule can then be converted to any platform with a supported converter to provide platform-specific queries.

## Rules

The Sigma rule, a generic and open signature format for describing log events, also referred to as signatures, use YAML Ain't Markup Language (YAML) format, and must adhere to the YAML schema specification.

Sigma rules have a handful of required fields (title, logsource, detection, condition), with the majority of fields optional for flexibility. To date, the community has provided hundreds of rules, and some organizations such as Recorded Future and SOC Prime maintain custom rulesets for subscribers. It is worth noting that rules contributed to the Sigma community project must meet more stringent guidelines defined in the Sigma documentation.

### Creation

Getting started creating Sigma rules can be challenging. The recommended method for creating a new rule is to identify an existing rule that is somewhat close to the proposed new rule, for example starting with an existing rule which uses the same cloud platform or operating system.

The Sigma documentation on rule creation, as well as community resources from SOC Prime, BluSapphire, and blog posts, do a good job of covering the basics, but I have highlighted a few points of interest.

| Field | Notes | Mapping |
|---|---|---|
| title | Use title casing, less than 50 characters | |
| status | All new rules community start as `experimental` status, promoted after successful community usage. | |
| tags | Reference ATT&CK, CAR when relevant, for example `attack.t1059` or `car.2014-04-003` | |
| logsource.product | For cloud services, convention is `product` to represent the cloud provider, such as `azure` or `gcp`. | Identifies Cloud Service (e.g. azure, gcp, aws) |
| logsource.service[1] | For cloud services, convention is `service` to represent the specific log the alert will be found in. For example, `azureactivity` represents the `auditlogs` represents the Azure Audit Logs, while `gcp.audit` represents the the Google Audit Log source. | Maps to data source in target query language |
| detection | The detection section defines the query criteria required for the rule. | Maps to query criteria in target query language |

Tip: The *Visual Studio Code* `sigma` *plugin is useful during Sigma rule development. The plugin flags common issues in Sigma rule creation such as missing fields, and provides useful features such as generating UUIDs for a new rule.*

# Sigma Rule Usage

Sigma converters exist for many platforms. Conversions for Azure and GCP are discussed below.

## Converters

Sigma converters, such as Sigmac and pySigma, a conversion tool for converting generic Sigma rules to platform-specific queries. Sigmac is in the process of being deprecated, and the Sigma project recommends new development to target pySigma going forward. In addition to the command line tools, SOC Prime provides the Uncoder.io website as a web-based application for Sigma rule conversion.

### Azure

A few different targets exist for converting Sigma rules to Azure backends. For log querying using the Log Analytics toolset, the *Microsoft Sentinel Query* target on Uncoder.io provides conversion compatibility.

## GCP

At the moment, the Google Chronicle converter for targeting YARA-L rules is the primary backend supported for the GCP platform. Chronicle and YARA-L were out of scope of the project and not explored in detail. Future work on a GCP Big Query backend target for Sigma conversion would be beneficial to community use.

### Manual

The tooling mentioned above is useful, however it is also possible to manually convert Sigma rules with an understanding of the target query language. Below is an example of converting a GCP Sigma rule created for the project to Google Logs Explorer syntax.

| Sigma Field | Sigma Value |
|---|---|
| Rule: `gcp_ssh_key_added.yml` [2] | |
| `title` | `GCP SSH Key Added` |
| Metadata fields (such as `description`, `author`, `status`, `date`) ignored for conversion example | |
| `logsource.product` | `gcp` |
| `logsource.service` | `gcp.audit` |
| `detection.selection.gcp.audit.method_name` | `v1.compute.projects.setCommonIn:` |
| `detection.selection.gcp.audit.service_name` | `compute.googleapis.com` |
| `detection.selection.protoPayload.metadata|contains` | `addedMetadataKeys` |
| `detection.selection_ssh.protoPayload.metadata|contains` | `ssh-keys` |

Ok, now we have the appropriate Google Logs Explorer equivalent of each line for the Sigma rule. Since the rules are `AND` 'd together, we will add an `AND` statement to each element of the converted query[3], resulting in the following resulting query:

```
-- GCP SSH Key Added Sigma Rule
protoPayload.@type:"type.googleapis.com/google.cloud.audit.AuditLog" AND
protoPayload.methodName="v1.compute.projects.setCommonInstanceMetadata" AND
protoPayload.serviceName="compute.googleapis.com" AND
"addedMetadataKeys" AND "ssh-keys"
```

Below are screenshots from the Google Cloud console of the query and result.

### Manual Converted Query

## Logs Explorer

| Query | Recent (73) | Saved (0) | Suggested (0) | Library |
|---|---|---|---|---|

🕐 7/13/22, 9:10 AM – 7/15/22, 10:10 AM   🔍 "addedMetadataKeys" "ssh-keys"

```
1  -- GCP SSH Key Added Sigma Rule
2  protoPayload.@type:"type.googleapis.com/google.cloud.audit.AuditLog" AND
3  protoPayload.methodName="v1.compute.projects.setCommonInstanceMetadata" AND
4  protoPayload.serviceName="compute.googleapis.com" AND
5  "addedMetadataKeys" AND "ssh-keys"
```

**Manual Query Result**

∨ ⓘ    2022-07-14 09:54:55.264 EDT    compute.googleapis.com    ….projects.setCommonInstanceMetadata    projects/c
"v1.compute.projects.setCommonInstanceMetadata", principal_email:

```
▼ {
    insertId: "-            "
    logName: "projects/cloud-analytics-342815/logs/cloudaudit.googleapis.com%2Factivity"
  ▼ operation: {
      id: "operation-1657806886032-                              "
      last: true
      producer: "compute.googleapis.com"
    }
  ▼ protoPayload: {
      @type: "type.googleapis.com/google.cloud.audit.AuditLog"
    ▼ authenticationInfo: {
        principalEmail: "                    "
      }
    ▼ metadata: {
        @type: "type.googleapis.com/google.cloud.audit.GceProjectAuditMetadata"
      ▼ projectMetadataDelta: {
        ▼ addedMetadataKeys: [
            0: "ssh-keys"
          ]
        }
      }
      methodName: "v1.compute.projects.setCommonInstanceMetadata"
    ▼ request: {
        @type: "type.googleapis.com/compute.projects.setCommonInstanceMetadata"
      }
    ▶ requestMetadata: {2}
      resourceName: "projects/                        "
      serviceName: "compute.googleapis.com"
    }
    receiveTimestamp: "2022-07-14T13:54:55.448785424Z"
  ▶ resource: {2}
    severity: "NOTICE"
    timestamp: "2022-07-14T13:54:55.264381Z"
  }
```

# References

1. Log source definitions can be particularly confusing for Sigma rules related to cloud services. For example, in the case of GCP and Google Workspace, the first Sigma rules were added around August 2021 and used "gcp.audit" and "google_workspace.admin", respectively. After that, the Hawk (hawk.io) tool configuration started referencing those strings. ↩

2. GCP Sigma Rule: Detects addition of SSH key,gcp_ssh_key_added.yml ↩

3. See Google Cloud logging query documentation ↩