How to use the Ingestion API

How-to guide

Requirements

- You should have a basic understanding of how the Chronicle APIs are structured, these are documented here
- Understanding how REST APIs work will be helpful
- A basic understanding of Python is expected
- Authentication will be via OAuth 2.0 so will require a Service Account credential JSON file, this will be provided by your Chronicle representative

Overview

This document covers the basics on how to use the Chronicle Ingestion API, we will cover sending entries which are both unstructured and pre-formatted in the Unified Data Model. The examples included will be in Python but these could easily be converted to work in any language which has a suitable web client library.

Chronicle's Ingestion API is documented here. Note that the API URLs used in these examples are for the US region, but there are more specific regional endpoints which should be used in Europe and South-East Asia regions.

Authenticating with the Ingestion API

Like all the Chronicle REST APIs, the ingestion API uses OAuth 2.0 as its authentication mechanism, with a Service Account JSON credential being the typical method to authenticate. The below example shows us using Python to authenticate with the Chronicle ingestion API.

```
from google.oauth2 import service_account
from google.auth.transport import requests

SCOPES = ['https://www.googleapis.com/auth/malachite-ingestion']
key_path='/path/to/credentials.json'

credentials = service_account.Credentials.from_service_account_file(key_path,
scopes=SCOPES)
http_client = requests.AuthorizedSession(credentials)
```

The **http_client** object is now authorized for the Ingestion API scope and is ready to start being used. The rest of the examples in this document will pick up where this step left off, using this **http_client** object. We can test this connection by using the following code to list the log types available on the API:

```
import json

log_types = http_client.request(
 method="GET",
 url="https://malachiteingestion-pa.googleapis.com/v2/logtypes",
)
print(json.dumps(log_types.json(),indent=2))

# Output:
{
  "logTypes": [
    {
      "logType": "ABNORMAL_SECURITY",
      "description": "Abnormal Security"
    },
    ...
    {
      "logType": "ZSCALER_WEBPROXY",
      "description": "Zscaler"
    }
  ]
}
```

The returned data shows that the authentication was successful.

Sending Unstructured Log Entries to the Ingestion API

Unstructured log entries is any log which requires parsing once it is received by the Chronicle API. Examples of this could be syslog or CEF messages, or JSON or XML logs from an application. In this section we will show how to take these logs and present them in the format required by the Chronicle Ingestion API, and finally send them to the API.

Getting our log data

In this example we will send two log entries to the Chronicle Ingestion API, these will be in a basic JSON format, and will be stored in a Python list. Often we can build such lists either by making an API call to a 3rd Party system and storing the results, or by reading in lines from a file. The log entries are shown below:

```
log_entries = [
  '{"variableA": "a","variableB":199}',
  '{"variableA": "b","variableB":205}',
]
```

Wrapping the log data to send to the ingestion API

The data must now be wrapped in the correct request body, before sending to the Chronicle Ingestion API. Basically this involves tagging the body with a log type so that Chronicle knows how to parse the data when it receives it, and by batching the log entries together. This is shown below:

```
customer_id = '3eaa8274-d1e7-11ec-9d64-0242ac120002'
log_type = 'ZSCALER_WEBPROXY'

entries = [{"logText": t} for t in log_entries]
body = {
  "customerId": customer_id,
  "logType": log_type,
  "entries": entries,
}

response = http_client.request(method="POST",
url="https://malachiteingestion-pa.googleapis.com/v2/unstructuredlogentries:batchCreate",
json=body)
response.raise_for_status()

if response.ok:
  print("Success")

# Output:
Success
```

Note that the maximum message size for the Ingestion API is 1MB, so when we are sending large amounts of data we will need to be aware of the size of our payload, and batch our log entries accordingly.

We will also need the customer ID for our tenant, this can be provided by Chronicle support as needed.

In this example we used the log type of **ZSCALER_WEBPROXY**, the list of log types can be fetched programmatically as shown in the authentication example, or are also listed (as Ingestion Label) at this link.

Sending UDM Events to the Ingestion API

In this example we will take pre-formatted events in the Unified Data Model (UDM) format, and push them to the ingestion API. This will bypass the parsing pipeline, and lets us more simply control and deliver events to Chronicle in a native format.

We are going to use the following event:

```
events = [
  {
    "metadata": {
      "eventTimestamp": "2022-04-14T15:30:18.142265Z",
      "eventType": "USER_LOGIN"
    },
    "additional": {
      "id": "9876-54321"
    },
    "principal": {
      "hostname": "userhost",
      "ip": ["10.0.0.75"],
      "mac": ["28:80:23:0b:c9:b2"]
    },
    "target": {
      "hostname": "systemhost",
      "user": {
        "userid": "employee"
      },
      "ip": ["10.0.0.77"],
      "mac": ["28:80:23:0b:c9:b3"]
    },
    "securityResult": [
      {
        "action": ["ALLOW"]
      }
    ],
    "extensions": {
      "auth": {
        "type": "MACHINE",
        "mechanism": ["USERNAME_PASSWORD"]
      }
    }
  }
]
```

Sending the data to the Ingestion API is very similar to when doing this with unstructured events, the below code will send the events:

```
customer_id = '3eaa8274-d1e7-11ec-9d64-0242ac120002'

body = {
   "customerId": customer_id,
   "events": events
}

response = http_client.request(
 method="POST",
 url="https://malachiteingestion-pa.googleapis.com/v2/udmevents:batchCreate",
 json=body
)
response.raise_for_status()

if response.ok:
 print("Success")
```

The key to sending events in UDM format like this is in understanding the schema of the Unified Data Model, this is defined on the Chronicle documentation site here. Pre-formatting data in this way takes more work for the customer than letting Chronicle parse the data, but also reduces reliance on the parsing process, which in certain cases can be advantageous.