FNC Python Client Library

FortiNDR Cloud 1.0.5

FORTINET DOCUMENT LIBRARY
https://docs.fortinet.com

FORTINET VIDEO LIBRARY
https://video.fortinet.com

FORTINET BLOG
https://blog.fortinet.com

CUSTOMER SERVICE & SUPPORT
https://support.fortinet.com

FORTINET TRAINING & CERTIFICATION PROGRAM
https://www.fortinet.com/training-certification

FORTINET TRAINING INSTITUTE
https://training.fortinet.com

FORTIGUARD LABS
https://www.fortiguard.com

END USER LICENSE AGREEMENT
https://www.fortinet.com/doc/legal/EULA.pdf

FEEDBACK
Email: techdoc@fortinet.com

December 17, 2025
FortiNDR Cloud 1.0.5 FNC Python Client Library
78-100-1035862-20251217

TABLE OF CONTENTS

Change Log

Date	Change Description
2024-12-17	Initial release v1.0.5

Overview

Fortinet FortiNDR Cloud is a cloud-native network detection and response solution built for the rapid detection of threat activity, investigation of suspicious behavior, proactive hunting for potential risks, and directing a fast and effective response to active threats. It ingests and stores threat intelligence from a wide variety of sources. FortiNDR Cloud provides a *Python Client Library* that allows you to retrieve some of this threat intelligence data using API and Metastream clients.

This document provides information and recommendations about how to use our Python Client Library to import the information gathered by FortiNDR Cloud into the Security Operation Center's (SOC) main platform of choice so it can be further processed and analyzed.

How Python Client Library works

The FortiNDR Cloud Service exposes several fully RESTful APIs that allows users to interact with the FortiNDR Cloud backend. Leveraging these APIs, it is possible to incorporate the data collected and analyzed by FortiNDR Cloud into the third-party Security solution system.

In addition to this, MetaStream provides raw events, observations, and Suricata alerts from FortiNDR Cloud in an AWS S3 bucket provisioned by Fortinet.

The Python Client Library leverages the exposed RESTful APIs to allow users to incorporate the network telemetry data collected and analyzed by FortiNDR Cloud into the third-party Security solution system. This Client Library also allows users to connect to AWS and retrieve selected data from the Metastream AWS S3 Bucket. This information can then be imported into the Security Operation Center's (SOC) main platform of choice or utilized as required.

Requirements

The use of this library provided by FortiNDR Cloud Service requires:

API Token[EMB1]	An API Token is required to use the API Client. This token is passed with any request to the FortiNDR Cloud REST APIs for authentication purposes. For more information about how to create API-only users and tokens, see the API Getting Started Guide.
API Domain	The API Domain is required to use the API Client. This domain is used by the client to direct any API request. To know the appropriate domain to be use, see the API Getting Started Guide.

AWS S3 Credentials	The AWS's credentials are required to use the Metastream Client. The credentials are used to authenticate and connect with the AWS S3 bucket. See, Create MetaStream credentials on page 8.
AWS boto3 library	The Metastream Client, provided with the Python Client Library, uses the AWS boto3 library to interact with AWS S3. Therefore, it needs to be available. The provided link shows how to install it if it is not available: https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html.

Considerations

- Only the last seven days of information is stored in AWS S3 Buckets. The events that can be retrieved by the Metastream Client are limited to those seven days.
- There are multiple types of events stored in AWS S3 Buckets by the FortiNDR Cloud Service. All of the events are accessible and can be retrieved from AWS S3 Buckets. However, it is not recommended to import all the events since this would be a very large amount of information.
- The Metastream Client allows users to interact with the AWS S3 Bucket and import selected data from it. Only *Suricata* and *Observations* are supported by the latest version (v1.0.3) of this library.

How to retrieve data from FortiNDR Cloud Services

FortiNDR Cloud Service collects an array of information that can be imported using this library.

- **Detections**:
  An alert mechanism that notifies you of events impacting your device and network. Detections allow you to quickly identify and respond to suspicious or known malicious activity in your network.
- **Events**:
  FortiNDR Cloud network sensors perform deep packet inspection of all observed network traffic and extract key protocol metadata. This metadata is enriched and organized into records called Events. In addition, FortiNDR Cloud observations and Suricata detections appear as events.
- **Entities**:
  FortiNDR Cloud entities are unique identifiers on the network. Entities are extracted from the event data and cataloged in their own data store. Contextual information is then added to the entities when applicable.
- **Suricata**:
  A match for a single Suricata signature with details.
- **Observation**:
  Observations are a result of multi-stage data pipelines that go across multiple events, do historical data lookups, draw correlations, compute several statistics and in some cases use machine learning algorithms to classify and predict outcomes.

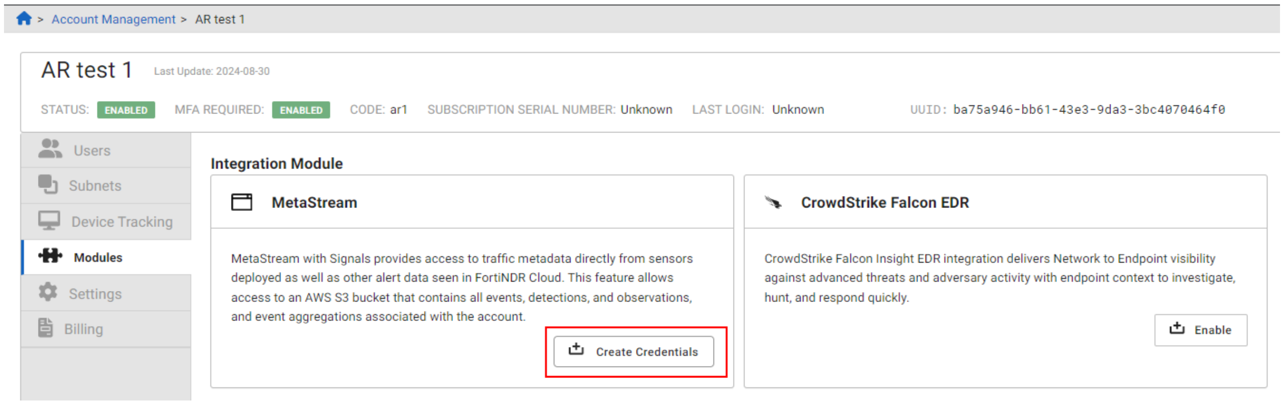
Any of the above information can be retrieved using this Client Library. The next section provides more details, instructions and recommendations regarding how to build and use the client library to retrieve each event type in order to import them.

Create MetaStream credentials

You must be an Admin user to see and use the MetaStream feature. MetaStream is enabled for all accounts by default. You can use the MetaSteam tile in the *Account Management* page to create, delete, recreate or retrieve a credential.

To create a credential:

1. Go to *Account Management* and click an account.
2. Click the *Modules* tab.
3. In the *MetaStream* tile, click *Create Credentials*. A message appears confirming the credentials were created.



MetaStream with Signals provides access to traffic metadata directly from sensors deployed as well as other alert data seen in FortiNDR Cloud. This feature allows access to an AWS S3 bucket that contains all events, detections, and observations, and event aggregations associated with the account.

CrowdStrike Falcon Insight EDR integration delivers Network to Endpoint visibility against advanced threats and adversary activity with endpoint context to investigate, hunt, and respond quickly.

To delete credentials:

1. Go to *Account Management* and click an account.
2. Click the *Modules* tab.
3. In the *MetaStream* tile, click *Delete Credentials*. A confirmation message appears.
4. Click *Confirm*. A message appears confirming the credentials were deleted.

How to build FortiNDR Cloud Client Library

The FortiNDR Cloud Client Library is written in Python version 3. Its source code can be found at FNC-Python-Library. In this section, we will review the functionalities of the FNC-Python-Library and how it can be used.

The Client Library can be built in two different formats, as a *pip file* or as *whl file*. This section shows how the library can be built and installed for any of these formats.

Build and install as a pip package

To build, run the following command:

```
python3 setup.py sdist
```

To install, run the following command:

```
pip install dist/com.fortinet.fndrc.integrations.python_client-x.x.x.*.tar.gz
```

To install to specific directory, run the following command:

```
pip install --target <directory> dist/com.fortinet.fndrc.integrations.python_client-x.x.x.*.tar.gz
```

Build and install as a wheel package

To build, run the following command:

```
python3 setup.py bdist_wheel
```

To install, run the following command:

```
pip install dist/com.fortinet.fndrc.integrations.python_client-x.x.x-py3-none-any.whl
```

To install to specific directory, run the following command:

```
pip install --target <directory> dist/com.fortinet.fndrc.integrations.python_client-x.x.x-py3-none-
      any.whl
```

Getting the client

The `FncClient` class that can be imported from `fnc.fnc_client`, is used to create the specific client instance that is required. There are two different types of clients provided by the FortiNDR Cloud Client Library.

- **FncApiClient**: This client leverages the REST APIs exposed by the FortiNDR Cloud Services to retrieve and manage detections, events and entities.
- **FncMetastreamClient**: This client allows access to the AWS S3 buckets to retrieve suricata and observation events observed within the FortiNDR Cloud Services.

FncApiClient

To create the `FncApiClient` instance we call the `get_api_client` class method that receive 5 arguments:

Argument	Type	Required	Default	Description
`name`	string	false	FNC_Py_Client-v1.0.0	This value is used to create the User-Agent id that will identify any request made by this client in the Server's API logs. It is also used to identify the client logs if no specific logger is passed.
`domain`	string	false	icebrg.io	Domain to where any REST request will be sent. For additional information, see the API Getting Started Guide.
`api_token`	string	true		API Token required to authenticate any request coming from this client. For additional information, see the API Getting Started Guide.
`rest_client`	FncRestClient	false		The Client API provide a default FncRestClient implementation that will be used if no rest_client is passed. To use a different one, its implementation needs to extend the FncRestClient class.
`logger`	FncLoger	false	default implementation of the FncLogger class	To provide different logging mechanism, the FncLogger class need to be extended and the its methods implemented appropriately. The specific handler needs to be enabled to send logs to file or console.

`FncClientError` is returned if no `api_token` is provided or if it is invalid. Also, if there is an active `FncApiClient`, created for the same domain and API Token that was provided, it will not be created again. Instead, a reference to the existing instance will be provided. Otherwise, a new client will be created replacing the active one.

Example

```
from fnc.api import FncApiClient
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient

client_name = ''
api_token = ''
domain = ''
log_level = None
client: FncApiClient = None

try:
    client = FncClient.get_api_client(
        name=client_name,
        domain=domain,
        api_token=api_token
    )
except FncClientError as e:
    client.get_logger().error(e)

client.get_logger().set_level(level=log_level)
```

FncMetastreamClient

To create the `FncMetastreamClient` instance we call the `get_metastream_client` class method that receive 6 arguments:

Argument	Type	Required	Default	Description
`name`	string	false	FNC_Py_Client-v1.0.0	This value is used to create the User-Agent id that will identify any request made by this client in the Server's API logs. It is also used to identify the client logs if no specific logger is passed.
`account_code`	string	true		The customer account code for which the raw events need to be retrieved.
`secret_key`	string	true		AWS secret access key required for authentication.

Argument	Type	Required	Default	Description
`access_key`	string	true		AWS key required for authentication.
`bucket`	string	false	'fortindr-cloud-metastream'	Bucket from where to retrieve the events
`logger`	FncLoger	false	default implementation of the FncLogger class	To provide different logging mechanism, the FncLogger class need to be extended and its methods implemented appropriately. The specific handler needs to be enabled to send logs to file or console.

`FncClientError` is returned if either `sercret_key`, `access_key` or `account_code` is missing. Also, if there is an active `FncMetastreamClient`, created for the same `sercret_key`, `access_key` and `account_code` that was provided, it will not be created again. Instead, a reference to the existing instance will be provided. Otherwise, a new client will be created replacing the active one.

Example:

```
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient
from fnc.metastream import FncMetastreamClient

client_name = ''
access_key = ''
secret_key = ''
account_code = ''
bucket_name = ''
log_level = None
client: FncMetastreamClient = None

try:
    client = FncClient.get_metastream_client(
        name=client_name,
        access_key=access_key,
        secret_key=secret_key,
        account_code=account_code,
        bucket_name=bucket_name
    )

except FncClientError as e:
    client.get_logger().error(e)

client.get_logger().set_level(level=log_level)
```

Logging

There are many different security platforms for which a FortiNDR Cloud Service connector could be useful. All those security software might have their own logging mechanism. For this reason, the Python Client Library provided by FortiNDR Cloud Services was decoupled from the logging mechanism. Therefore, the client library allows the option to provide a different logger's instance when the client is created.

The provided logger must extend the `FncClientLogger` class that can be imported from `fnc.logger` and implement all its methods.

```
class FncClientLogger:
     def set_level(self, level):
       raise NotImplementedError()

     def critical(self, log: str):
       raise NotImplementedError()

     def error(self, log: str):
       raise NotImplementedError()

     def warning(self, log: str):
       raise NotImplementedError()

     def info(self, log: str):
       raise NotImplementedError()

     def debug(self, log: str):
       raise NotImplementedError()
```

There are two additional methods that could be implemented to configure console or file logging if required.

```
def set_console_logging(self, enable: bool = False):
       pass

     def set_log_to_file(self, enable: bool = False):
       pass
```

Errors

Any failure in the Client Library is reported as a `FncClientError`. This class extends the *Exception* class and can be imported from `fnc.errors`. The exact reason of the failure could be distinguished by looking into its internal fields.

Field	Type	Description
`error_type`	ErrorType	This field will contain the specific type of error that cause the failure. The complete list can be found in the appendix FncClient Library Error Types on page 37. In case of any unexpected exception a `FncClientError` with error type `GENERIC_ERROR` will be raised.
`error_message`	String	A meaningful error message specific to the error type is included in this field. This message will contain relevant information (i.e., misconfigured values) if any.
`error_data`	Dict	This field is a dictionary of the information, relevant to the specific error, required to create the error message.
`exception`	Exception	If an exception was received and wrapped into a `FncClientError`, this field will contain such exception. It could be retrieved using the `get_original_exception` method.

Sample Code:

```
class FncClientError(Exception):
    error_type: ErrorType
    error_message: str
    error_data: dict
    exception: Exception

    def get_original_exception(self):
```

How to use the FncApiClient

As mentioned in Getting the client on page 10, the `FncApiClient` instance can be created by calling the `get_api_client` class method of the `FncClient` class. The FortiNDR Cloud Service exposes REST APIs that can be leveraged to manage the services. The provided Python Client Library and specifically the `FncApiClient` can be used to access those APIs.

There are two methods that define the main use of the FncApiClient:

- **call_endpoint**:
  This method allows you to call specific endpoints provided by the REST APIs exposed by the FortiNDR Cloud Services. The client, simplifies the use of the REST APIs by wrapping all of the required logic and validations into a simple method call. However, it is important to note that not all the endpoints provided by the REST APIs are currently supported by the `FncApiClient`. The list of supported endpoints can be found in FncClient Library Supported Endpoints on page 38.
- **continuous_polling**:
  This allows to continuously poll detections from the FortiNDR Cloud Service to be imported into a third-party security solution. The retrieved information could be enriched during the process according to the received arguments.

Calling Specific Endpoint

In this section, we describe how to use the `FncApiClient` to call specific endpoints. The steps involved in calling specific endpoints are:

1. Imports

The main classes required while calling specific endpoint are shown below.

```
from fnc.fnc_client import FncClient
from fnc.api import EndpointKey, FncApiClient
from fnc.errors import FncClientError
```

2. Get the Client

The `FncApiClient` is created using the `FncClient` class’s method and providing the required arguments. For a detailed description, see Getting the client on page 10.

3. Call Endpoint

The FncApiClient’s `call_endpoint` method is used to call specific endpoints. The method requires two arguments: the endpoint to be called and a dictionary with its required arguments.

```
client.call_endpoint( endpoint: EndpointKey, args: dict )
```

The full list of supported endpoints and its description is provided in FncClient Library Supported Endpoints on page 38.

4. Handle Errors

Any exception occurring while calling the endpoint will be raised as a `FncClientError` exception. The specific problem can be identified by using the `FncClientError` fields.

The code below shows a full example of using the client while calling a specific endpoint to resolve a detection.

```
from fnc.fnc_client import FncClient
from fnc.api import EndpointKey, FncApiClient
from fnc.errors import FncClientError

client_name = ''
api_token = ''
domain = ''
log_level = None
client: FncApiClient = None
try:
    client = FncClient.get_api_client(
      name=client_name,
      domain=domain,
      api_token=api_token
    )
    client.get_logger().set_level(level=log_level)

    detection_id = ''
    resolution = ''
    comment = ''

    response = client.call_endpoint(
      EndpointKey.RESOLVE_DETECTION,
      {
        'detection_id': detection_id,
        'resolution': resolution,
        'resolution_comment': comment
      }
    )
    client.get_logger().info(response)

except FncClientError as e:
    client.get_logger().error(e)
```

Polling Detections

Detections are the primary mechanism by which the network event data is searched on a continuous basis for anomalies or suspicious behaviors. For this reason, in order to integrate with the FortiNDR Cloud Service, we need to be able to poll and to import all the detections regularly reporting any detections within a short period of time. This makes the polling strategy we follow extremely important since it not only needs to avoid duplication but it must ensure that no detection is missing.

In this section, we provide a quick overview of the detection object and the polling strategy used in the Client Library. We then show how to use the continuous polling method to pull and enrich the detections information.

Detections

A detection is an alert mechanism that notifies you of events impacting your device and network. Detections allow you to quickly identify and respond to suspicious or known malicious activity in your network.

Detections are identified based on both the IP address and the Sensor ID to avoid issues with overlapping IP space. A duplicate detection is not generated if an active detection already exists for the IP address and sensor ID pair. Instead, the *Last Seen timestamp* is updated and the event is added to the rule's *Latest Events*. Detections can be resolved either manually or automatically to show that the underlying cause of the events should no longer be triggering.

Detection Object

There are many fields in a Detection object. In this document we will focus on only a few of them. The Detections API documentation can be reviewed for additional details.

- **Account uuid:**
  The `account_uuid` field, is a unique identifier of the account owning this detection. Users can only see detections owned by or shared with the account they have access to.
- **Device ip:**
  The `device_ip` field contains the IP address or CIDR of the devices associated to this particular detection.
- **Muted:**
  Detections, as well as rules and devices can be muted to avoid receiving notifications whenever they are created or updated. The `muted`, `muted_rule` and `muted_device_uuid` are boolean fields that show if the detection is muted.
  - *Muted_rule*: All instances of that detection are muted regardless of device
  - *Muted*: This specific detection with this device is muted
  - *Muted_device_uuid* This IP is muted across all detections
- **Timestamps:**
  There are four timestamps on a detection object (`created`, `first_seen`, `last_seen` and `updated`). These timestamps are stored in UTC with microseconds precision (`yyyy-mm-ddThh:mm:ss.000000Z`). Of all of these timestamps, the `created` timestamp is the only one that is immutable.

Polling Strategy

The polling strategy we follow while pulling detections is very important since it not only needs to avoid duplication but it must ensure that no detection is missing. Below we show the key aspects for the polling strategy used in the `FncApiClient`'s `continuous_polling` method:

- **Search for detections regularly:**
  Since the network's events are constantly being processed, we need to search for new detections using the *GET Detections* endpoint. This search needs to be performed regularly so we get the detections as fast as possible. The recommended search interval is 5 minutes.
- **Use of a fixed window:**
  It is recommended to fix the end date as well as the start date. In this way we can ensure there is no overlap in the search window the next time the search is performed.
- **Allow delay (recommended 10 minutes):**
  While processing the detections, there is a time difference between when the detection is created and when the detection can be retrieved by the API. This delay allows some time for the detection to be processed, avoiding missing any detection. It is recommended to make this delay configurable so it can be updated if needed.
- **Use created_or_shared_start_date and created_or_shared_end_date:**
  The `created` timestamp is the only immutable one in the detection object. For this reason, it is recommended you use it for the fixed search window. However, there could be a time difference between when the detection is created and when it is shared with the specific `account_uuid`. The `created_or_shared_start_date` and `created_or_shared_end_date` filters in the Get detections endpoint take both into account. Using these timestamps helps avoid missing detections.

Taking into account the above key aspects, the recommended polling strategy would be:

- Get the last checkpoint or the configured start date if it is the first time the search is called.
- Set the fix window as:
  - `start_date = last_checkpoint` or the configured start date
  - `end_date = utc_now() - 10min`
    Ensure `start_date` is before the `end_date`.
- Use the Get Detections endpoint to search for detections incrementally with the following arguments:
  - `created_or_shared_start_date = start_date`
  - `created_or_shared_end_date = end_date`
  - `sort_by = device_ip` (This is required to ensure order is kept for every page)
  - `sort_order = asc`
  - `limit = 10000` (This is the maximum page size)
  - `include = rules` (Use the `include` argument to provide extra information for the rules and/or indicators. See, GET_DETECTIONS on page 51
  - `offset` = (Started in 0 and incremented 10000 each time until no detection is received)
- Import each detection using the detection['`create`'] as the timestamp.
- Update the `last_checkpoint` to be the `end_date` of the fixed window
- Repeat every 5 minutes

Important:
It is important to know that all the timestamps handled by FortiNDR Cloud Service APIs do not have any time zone information and are assumed to be in UTC. Therefore, you must ensure that any timestamp you provide is in UTC. Also, if you need to use any retrieved timestamp, the time zone information needs to be added prior to using it.

Polling Detections Continuously

In this section, we show how to use the `FncApiClient` to continuously poll detections.

Polling Arguments

When polling detections continuously, it is possible to enrich the information being retrieved or filter it if necessary. This can be done with the arguments passed to the polling method in a dictionary. The allowed arguments are described below.

```
polling_args = {
         'polling_delay': 10,
         'status':  'active',
         'pull_muted_detections': 'false',
         'pull_muted_rules':  'false',
         'pull_muted_devices':  'false',
         'include_description': True,
         'include_signature': True,
         'include_pdns': True,
         'include_annotations': True,
         'include_events': True,
         'filter_training_detections': True,
         'start_date': ''
}
```

Attribute	Type	Default	Description
`polling_delay`	int	10	This is expressed in minutes and represent a delay in retrieving detections to allow time for them to be processed by the backend services. It is recommended to be 10 minutes which is the default value.
`status`	string	‘active’	This argument is used to filter the retrieved detections according to the detections’ status (*active* or *resolved*). Only active detections are retrieved by default.
`pull_muted_detections`, `pull_muted_rules`, `pull_muted_devices`	string	‘false’	These arguments are used to filter the retrieved detections depending on whether they are muted or not. The supported values are (‘*false*’, ‘*true*’, ‘*all*’).

Attribute	Type	Default	Description
`include_descriptions, include_signature, include_pdns, include_annotations, include_events`	bool	False	These arguments are used to enrich the information being retrieved. If set to *true*, they tell the method to include the rule's description or signature, entity's pdns or annotations information and detection's associated events.
`limit`	int		This argument is used to limit the number of detections retrieved with each piece of historical data. It should not be used for continuous polling or some detections might be missed.
`start_date`	datetime	now	This argument is used to state since when detections need to be retrieved. This timestamp is used only if no checkpoint is provided in the context.

API Context

The context used during the continuous polling provides information about what was already retrieved and what needs to be retrieved. It holds three pieces of information:

Attribute	Type	Description
`checkpoint`	string	The value contained in the checkpoint, was the `end_date` in the last performed search. If it is included in the context while calling to the polling method, its value will be used as the `start_date`. The checkpoint is updated every time the polling method is called.
`Polling_args`	dictionary	After the polling method is called, the context will update the `polling_args` value with a dictionary containing all the arguments that were send to the FortiNDR Cloud REST API when the detections were requested. This allows you to perform the same request in case of a failure.
`history`	dictionary	The FncClient Library allows to pull historical data when pulling detections. However, it is possible that the amount of historical information is too big that it causes delays on the retrieval of current data. To avoid this delay, the client library allows you to split the context in historical and current data. This way current data and historical data can be retrieved separately. If the history value is provided in the context, it means we are retrieving historical data.

The context also includes metrics information and a cache for requested entity enrichments, which helps reduce the number of calls to the entity API services. Cached records are not persisted; they are stored in memory only and expire after 24 hours.

Splitting Historical and Current Data

The client library provides a method (`get_splitted_context`) that provide two contexts: one with the history value set to pull data from the provided `start_date` up to now and a second one without a history value and the `checkpoint` set to `now`. In this way we can pull the historical data using the first context while pulling the current data using the second one.

Example:

```
# Split the poling interval in history and current
h_context, context = client.get_splitted_context(args=polling_args)
```

Poll History Method

Once the context has been split, we are ready to start pulling the historical data using the `FncApiClient.poll_history` method. This method retrieves a piece of the historical data. The size of the piece is determined by the interval argument which default to one day. If the limit is not reached, the method tries to pull the next piece until the limit is reached or overpassed. Every time the `poll_history` method is called it updates the `start_date` information in the context's history value until it reaches the `end_date`. At this point, the whole historical data would have been polled.

The `get_splitted_context` method is only called the first time. After that, the history value in the context is used. This is done to avoid pulling the same information over and over again.

The steps below show and describe the process for pulling the historical data.

1. Imports

The main classes required while calling specific endpoint are shown below.

```
from fnc.fnc_client import FncClient
from fnc.api import ApiContext, FncApiClient
from fnc.errors import FncClientError
from fnc.utils import datetime_to_utc_str
```

Get the Client

The `FncApiClient` is created using the `FncClient` class's method and providing the required arguments. For a detailed description, Getting the client on page 10.

3. Prepare Polling Arguments and Context

When calling the `poll_history` method, we need to pass a dictionary with all the arguments. The arguments are used for filtering and enriching the detection's information. For a detailed description, see . *Polling Arguments* in Polling Detections Continuously on page 19.

```
polling_args = {
    'polling_delay': 10,
    'status':  'active',
    'pull_muted_detections': 'false',
    'pull_muted_rules':  'false',
    'pull_muted_devices':  'false',
    'include_description': True,
    'include_signature': True,
    'include_pdns': True,
    'include_annotations': True,
    'include_events': True,
    'filter_training_detections': True,
    'limit': 500,
    'start_date': ''
}
```

4. Split the Context

We also need to pass the context with the history value to the `poll_history` method.

```
# Split the poling interval in history and current
h_context, context = client.get_splitted_context(args=polling_args)
```

The history field in the context contains the start and end date to be pulled. It can be created manually but using the method above ensures duplications will be avoided since the end date of the history context will be the checkpoint in the current one. However, it can only be used once. After that, if the context needs to be recreated it will need to be updated manually with the last history value.

5. Poll the next piece of historical data

The FncApiClient's `poll_history` method is used to poll historical data. This method is a generator function and need to be used within a loop.

```
# The args for poll_history should be the same as for the continuous polling
for response in client.poll_history(
    context=h_context,
    args=polling_args,
    interval= timedelta(days=1)
):
    # Do Something...
    client.get_logger().info(response)
```

6. Clear Context's Arguments

The `polling_args` value in the context contains the la arguments used in the last request. If they are present in the context while the polling method is called, they will be used. Therefore, they need to be cleared unless it is required to perform the same call as before.

```
# Ensure each iteration start without polling_args in the context
h_context.clear_args()
```

7. Handle Errors

Any exception occurring while calling the `poll_history` method will be raised as a `FncClientError` exception. The specific problem can be identified by using the `FncClientError` fields.

The code below shows a full example of polling detection's historical data.

```
from fnc.api import ApiContext, FncApiClient
from fnc.errors import FncClientError

client_name = ''
api_token = ''
domain = ''
log_level = None

client: FncApiClient = FncClient.get_api_client(
    name=client_name,
    domain=domain,
    api_token=api_token
)
client.get_logger().set_level(level=log_level)

try:
    polling_args = {
      'polling_delay': 10,
      'status':  'active',
      'pull_muted_detections': 'false',
      'pull_muted_rules':  'false',
      'pull_muted_devices':  'false',
      'include_description': True,
      'include_signature': True,
      'include_pdns': True,
      'include_annotations': True,
      'include_events': True,
      'filter_training_detections': True,
      'limit': 500,
      'start_date': '2024-01-01T00:00:00.000000Z'
    }

    # Split the poling interval in history and current
    h_context, context = client.get_splitted_context(args=polling_args)

    # The polling args for poll_history should be the same as
    # for the continuous polling
    for response in client.poll_history(context=h_context, args=polling_args):
      # Do Something...
      client.get_logger().info(response)

    # Ensure each iteration start without polling_args in the context
    h_context.clear_args()

    # This is the end of the iteration. It can be called in a loop until
    # completed or wait for some time between iterations. It only requires
    # the context with the history value
```

```
except FncClientError as e:
    # Any exception will be reported as FncClientError. Specific Error
    # message will be added to the exception depending on its Error Type
    client.get_logger().error(e)
```

Continuous Polling Method

Once the context is split, we are ready to start pulling detections using the `FncApiClient.continuous_polling` method. This method retrieves detections since the timestamp stored in the context's checkpoint or the provided start date. If none of those values are provided the `start_date` will be set to the current time. After the call is completed, the checkpoint value in the context is updated to the used `end_date` to avoid overlaps in the search windows.

The steps below show and describe the process for pulling detections continuously.

1. Imports

The main classes required while polling detections are shown below.

```
from fnc.fnc_client import FncClient
from fnc.api import ApiContext, FncApiClient
from fnc.errors import FncClientError
```

2. Get the Client

The `FncApiClient` is created using the `FncClient` class's method and providing the required arguments. For a detailed description, see Getting the client on page 10.

3. Prepare Polling Arguments

When calling the `continuous_polling` method, we need to pass a dictionary with all the arguments. The arguments are used for filtering and enriching the detection's information. For a detailed description, see . *Polling Arguments* in Polling Detections Continuously on page 19.

```
polling_args = {
   'polling_delay': 10,
   'status':  'active',
   'pull_muted_detections': 'false',
   'pull_muted_rules':  'false',
   'pull_muted_devices':  'false',
   'include_description': True,
   'include_signature': True,
   'include_pdns': True,
   'include_annotations': True,
   'include_events': True,
   'filter_training_detections': True,
   'start_date': '' # Get configured Start Date or now as utc
}
```

4. Prepare the Context

When calling the `continuous_polling` method, we need to pass a context. Its checkpoint value will be used as the `start_date` for the search. It needs to be updated to the last call's checkpoint if it has not been updated already.

```
context = ApiContext()
```

```
context.update_checkpoint(checkpoint= checkpoint)
```

5. Poll detections

Using the polling strategy described above, the `continuous_polling` method will perform the next search and retrieve the detections. The FncApiClient's `continuous_polling` method is a generator function and need to be used within a loop.

```
for response in client.continuous_polling(context=context, args=polling_args):
       # Do Something...
       client.get_logger().info(response)
```

6. Persist checkpoint and clear Context's Arguments

The `polling_args` value in the context contains the la arguments used in the last request. If they are present in the context while the polling method is called, they will be used. Therefore, they need to be cleared unless it is required to perform the same call as before.

```
# Persist checkpoint if needed
checkpoint = context.get_checkpoint()

# Ensure each iteration start without polling_args in the context
context.clear_args()
```

7. Handle Errors

Any exception occurring while calling the `continuous_polling` method will be raised as a `FncClientError` exception. The specific problem can be identified by using the `FncClientError` fields.

The code below shows a full example of continuously polling detections.

```
from fnc.api import ApiContext, FncApiClient
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient

client_name = ''
api_token = ''
domain = ''
log_level = None

try:
    client: FncApiClient = FncClient.get_api_client(
      name=client_name,
      domain=domain,
        api_token=api_token
    )
    client.get_logger().set_level(level=log_level)

    polling_args = {
      'polling_delay': 10,
      'status':  'active',
      'pull_muted_detections': 'false',
      'pull_muted_rules':  'false',
      'pull_muted_devices':  'false',
      'include_description': True,
      'include_signature': True,
      'include_pdns': True,
      'include_annotations': True,
      'include_events': True,
      'filter_training_detections': True
    }

    checkpoint = # Get persisted checkpoint or ''
    start_date_str = # Get configured Start Date or now as utc

    polling_args['start_date'] = start_date_str

    context = ApiContext()
    context.update_checkpoint(checkpoint= checkpoint)

    for response in client.continuous_polling(context=context, args=polling_args):
      client.get_logger().info(response)

    # Persist checkpoint if needed
    checkpoint = context.get_checkpoint()
    # Ensure each iteration start without polling_args in the context
    context.clear_args()

except FncClientError as e:
    client.get_logger().error(e)
```

How to use the FncMetastreamClient

Overview

FortiNDR Cloud Service collects multiple Information and there are many different types of events stored in AWS S3 Buckets. The Metastream Client Library allows for the pulling of events from those buckets. However, the latest version (v1.0.0) of the library only supports the following event types:

- Suricata: A match for a single Suricata with details.
- Observation: Observations are a result of multi-stage data pipelines that go across multiple events, do historical data lookups, draw correlations, compute several statistics and in some cases use machine learning algorithms to classify and predict outcomes.

Using the Metastream Client Library, we can import any of these types of events. The structure of these event types can be found in the Appendix on page 37.

FncMetstreamClient

As mentioned the Getting the client on page 10, the `FncMetastreamClient` instance can be created by calling the `get_metastream_client` method of the `FncClient` class. The `FncMetastreamClient`can be used to access AWS S3 bucket to retrieve raw events stored by the FortiNDR Cloud Services.

Below are the main methods exposed by this client:

Attribute	Description
`fetch_event_types(…)`	Returns a list of all the event types that can be retrieved using this client. Current version only supports *Suricata* and *Observation*.
`poll_history(…)`	Allows you to pull historical data in chunks smaller than a day. This is useful when the amount of information being pulled is too big and would take too much time to be pulled completely.
`fetch_events_by_day(…)`	Fetch all raw events observed during a specified day. The specified day must be within the last seven days.
`fetch_events(…)`	Fetch all raw events that were observed since the specified timestamp up to the current timestamp. This interval must be less than 24 hours.

In the next sections we describe each of these methods in more details.

Fetching Events

There are several methods to fetch events exposed by the Metastream client. All of them, among other arguments, require a context and the event type needs to be retrieved. The whole list of supported event types can be retrieved using the `fetch_event_types` method shown in section Fetch Event Types. The context shown in *Metastream Context* is used to store specific session wide data such as metrics, checkpoint and the time window of historical data to be retrieved.

Metastream Context

The context used by the Metastream client provides information about what was already retrieved and what needs to be retrieved as well as some statistic information. It holds three pieces of information:

Attribute	Type	Description
`checkpoint`	string	The value contained in the checkpoint, was the `end_date` in the last performed search. It must be used as the start date for the next call of the fetching method. The checkpoint is updated every time the fetching method is called.
`history`	dictionary	The FncClient Library allows you to pull historical data when fetching events. However, it is possible that the amount of historical information is too big that it causes delays on the retrieval of current data. To avoid this delay, the client library allows you to split the context in historical and current data. This way current data and historical data can be retrieved separately.
`api_calls`, `file_downloads`	int	While fetching events, the statistical information of the number of API calls and downloaded files is updated in the context.

Splitting Historical and Current Data

The client library provides a method (`get_splitted_context`) that provide two context one with the `history` value set to pull data from the provided `start_date` up to now and a second one without history value and the checkpoint set to now. In this way we can pull the historical data using the first context while pulling the current data using the second one.

Example:

```
# Split the poling interval in history and current
h_context, context = client.get_splitted_context(start_date_str = <start_date of the history>)
```

Fetch Event Types

The `fetch_event_types()` method can be used to retrieve a list of supported event types. This function does not receive any argument and return a list of strings in which every element is an event type.

Example:

The code below will print the list of supported event types.

```
from fnc.metastream import FncMetastreamClient
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient

client_name = ''
access_key = ''
secret_key = ''
account_code = ''
bucket_name = ''
log_level = None

try:
    client: FncMetastreamClient = FncClient.get_metastream_client(
      name= client_name,
      access_key= access_key,
      secret_key= secret_key,
      account_code = account_code,
      bucket_name= bucket_name
    )

    client.get_logger().set_level(level=log_level)

     print(f'The supported event types are: {client.fetch_event_types()}')

except FncClientError as e:
    client.get_logger().error(e)
```

Fetch Events by Day

Given the large amount of information that might be stored in AWS S3 Buckets, it is recommended to retrieve events for one event type at a time and search for a period no longer than a day. This recommendation is enforced by the `fetch_events_by_day()` function. It allows you to fetch all raw events for a specified event type that were observed during a specified day.

Keep the following considerations in mind:

- The day is specified as a timestamp. The time zone information must be provided or UTC and it will be assumed by default.

- This is a generator function that produces a series of events usable in a `for-loop` or that can be retrieved one at a time with the `=next()`function.

In this section, we show how to use the `FncMetastreamClient` to fetch events for an specific day. The steps involved are:

1. Imports

The main classes required while calling specific endpoint are shown below.

```
from fnc.fnc_client import FncClient
from fnc.metastream import FncMetastreamClient
from fnc.errors import FncClientError
```

2. Get the Client

The `FncMetastreamClient` is created using the `FncClient` class's method and providing the required arguments. For a detailed description, see Getting the client on page 10.

3. Fetch events

The `FncMetastreamClient.fetch_events_by_day` method is used to fetch events for a specific day. The method requires two arguments the endpoint to be called and a dictionary with its required arguments.

```
client.call_endpoint(
    day: datetime,
    event_type: str,
    limit: int = 0,
    context: MetastreamContext = None
)
```

Property	Type	Required	Description
`day`	datetime	true	The day to download events from. Time is ignored if given. UTC is assumed if time zone information is not provided. Must be within the last 7 days.
`event_type`	string	true	The event type to download. Possible values are observation, suricata.
`limit`	int	false	The maximum number of events to fetch. Must be between 1 and 10000.
`context`	MetastreamContext	false	An object that stores specific session wide data such as metrics and checkpoint.

4. Handle Errors

Any exception occurring while fetching the events will be raised as a `FncClientError` exception. The specific problem can be identified by using the `FncClientError` fields.

Example

The code below will retrieve Observations from the previous two days until all the Observations have been retrieved and print the size of each piece.

```
from fnc.metastream import FncMetastreamClient
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient

client_name = ''
access_key = ''
secret_key = ''
account_code = ''
bucket_name = ''
log_level = None
client = None

try:
    client: FncMetastreamClient = FncClient.get_metastream_client(
      name= client_name,
      access_key= access_key,
      secret_key= secret_key,
      account_code = account_code,
      bucket_name= bucket_name
    )
    client.get_logger().set_level(level=log_level)
    day = datetime.now(timezone.utc) - timedelta(days=2)

    for events in client.fetch_events_by_day(event_type='observation', day=day):
      #process events
      print(f'num events: {len(events)}')

except FncClientError as e:
    client.get_logger().error(e)
```

Fetch Events History

The historical event data can be retrieved by calling the `fetch_events_by_days` method for each of the last 7 days. However, given the large amount of information being retrieved it could take a long time. For that reason, the `FncMetastreamClient` exposes a `poll_history` method that allows you to retrieve historical data in smaller chunks. This method requires a context with the history value set to pull data from the history`[‘start_date’]` up to history `[‘end_date’]`. To learn how to split the context into history and current, see Fetch Events History on page 31

Once the context is split, we are ready to start pulling the historical data using the `FncMetastreamClient.poll_history` method. The steps involved are:

1. Imports

The main classes required while calling specific endpoint are shown below.

```
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient
from fnc.metastream.metastream_client import FncMetastreamClient
from fnc.metastream.s3_client import MetastreamContext
```

2. Get the Client

The `FncMetastreamClient` is created using the `FncClient` class's method and providing the required arguments. For a detailed description, see Getting the client on page 10.

3. Split the Context

We need to pass the context with the history value to the poll_history method.

```
# Split the poling interval in history and current
h_context, context = client.get_splitted_context(start_date_str=start_date)
```

The history field in the context contains the start and end date to be pulled it can be manually created but using the above method, ensure duplications will be avoided since the end date of the history context will be the checkpoint in the current one. However, this method can only be used once

4. Fetch a piece of the historical data

The `FncMetastreamClient.poll_history` method is used to fetch apiece of the events for previous days. The method requires two arguments the endpoint to be called and a dictionary with its required arguments.

```
client.call_endpoint(day: datetime, event_type: str, limit: int = 0, context: MetastreamContext =
      None)
```

Property	Type	Required	Description
`context`	MetastreamContext	false	An object that stores specific session wide data such as metrics and checkpoint. It needs to contain the history value set.
`event_type`	string	true	The event type to download. Possible values are *observation*, *suricata*.
`interval`	timedelta	false	The maximum interval of time to search each time. It defaults to one day.

5. Handle Errors

Any exception occurring while fetching the events will be raised as a `FncClientError` exception. The specific problem can be identified by using the `FncClientError` fields.

Each call to this method retrieves a piece of the historical data. To fetch the whole history, it needs to be called while the `history['start_date']` is before the `history['end_date']`.

Example

```
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient
from fnc.metastream.metastream_client import FncMetastreamClient
from fnc.metastream.s3_client import MetastreamContext

api_token = ''
access_key = ''
secret_key = ''
account_code = ''
bucket = ''
event_type = ''
client = None

try:
    client: FncMetastreamClient = FncClient.get_metastream_client(
      name= client_name,
      access_key= access_key,
      secret_key= secret_key,
      account_code = account_code,
      bucket_name= bucket_name
    )
    client.get_logger().set_level(level=log_level)

    interval = timedelta(hours=1) # By default the size of the interval will be 1 day
    # 7 days is the maximum of events historical data that can be retrieve
    h_context, context = client.get_splitted_context(
      start_date_str=star_date_str
    )

    history = h_context.get_history(event_type=event_type)
    while h_context.history.get('start_date') < h_context.history.get('end_date'):
    for events in client.poll_history(
      context=h_context,
      event_type=event_type,
      interval=interval
    ):
      # Process events
      print(f'num events: {len(events)}')
except FncClientError as e:
    client.get_logger().error(e)
```

Fetch Events

Given the large amount of information that might be stored in AWS S3 Buckets, it is recommended to retrieve events for one event type at a time and search for a period no longer than a day. This recommendation is enforced by the `fetch_events()` function. This function allows you to fetch all raw events for the specified event type that were observed since the specified start date.

Keep the following considerations in mind:

- The start date must be less than a day before and it must have the timezone information or UTC will be assumed by default.
- This is a generator function that produces a series of events usable in a `for-loop` or that can be retrieved one at a time with the `next()` function.

In this section, we show how to use the `FncMetastreamClient` to fetch current events. The steps involved are:

1. Imports

The main classes required while calling specific endpoint are shown below.

```
from fnc.fnc_client import FncClient
from fnc.metastream import FncMetastreamClient
from fnc.errors import FncClientError
```

2. Get the Client

The `FncMetastreamClient` is created using the `FncClient` class's method and providing the required arguments. For a detailed description, see Getting the client on page 10.

3. Fetch events

The `FncMetastreamClient.fetch_events` method is used to fetch events for specific interval. The method requires five arguments the endpoint to be called and a dictionary with its required arguments.

Property	Type	Required	Description
`event_type`	string	true	The event type to download. Possible values are *observation*, *suricata*.
`start_date`	datetime	true	The start time to restrict results based on their timestamp. Value must have timezone information or UTC will be assumed.
`end_date`	datetime	false	The end time to restrict results based on their timestamp. Value must have timezone information or UTC will be assumed. It is set to now by default
`limit`	int	false	The maximum number of events to fetch. Must be between 1 and 10000.
`context`	MetastreamContext	false	An object that stores specific session wide data such as metrics and checkpoint.

```
for events in client.fetch_events(event_type=…, limit=…, start_date=…, context=…):
    # Do something …
```

4. Handle Errors

Any exception occurring while fetching the events will be raised as a `FncClientError` exception. The specific problem can be identified by using the `FncClientError` fields.

Example

The code below will retrieve Observations from the previous two days until all the Observations have been retrieved.

```
from fnc.metastream import FncMetastreamClient
from fnc.errors import FncClientError
from fnc.fnc_client import FncClient
from fnc.metastream.s3_client import MetastreamContext

client_name = ''
access_key = ''
secret_key = ''
account_code = ''
bucket_name = ''
log_level = None
client = None

try:
    client: FncMetastreamClient = FncClient.get_metastream_client(
      name= client_name,
      access_key= access_key,
      secret_key= secret_key,
      account_code = account_code,
      bucket_name= bucket_name
    )
    client.get_logger().set_level(level=log_level)

    evet_type = 'observation'
    start_date = datetime.now(timezone.utc) - timedelta(hours=2)
    end_date = datetime.now(timezone.utc)
    context = MetastreamContext()

    for events in client.fetch_events(
      event_type=event_type,
      start_date= start_date,
      end_date= end_date,
      context= context
    ):
      #process events
      print(f'num events: {len(events)}')

except FncClientError as e:
    client.get_logger().error(e)
```

Return Value

The three methods that fetch events from the AWS S3 buckets, return an array of events each call until all events have been returned. Each event will be a JSON object containing the specific event’s information. Below is a sample response.

```
response = [{
    'timestamp': '2022-10-16T21:59:53.998000Z',
    'uuid': '24fd131ec-85c9-4af0-b810-c541d2eff5a1',
    'event_type': 'observation',
    'customer_id': 'cid',
    'sensor_id': 'sid',
    'source': 'Fortinet',
    'evidence_start_timestamp': '2022-10-16T21:59:53.998000Z',
    'evidence_end_timestamp': '2022-10-16T22:59:54.814000Z',
    'observation_uuid': 'bf1e1203-ed35-4f22-865d-89e75a1c174a',
    'title': 'TCP Device Enumeration',
    'category': 'relationship',
    'confidence': 'high',
    'src_ip': '1.2.3.4',
    'src_ip_enrichments': {
      'internal': True,
      'geo': {'location': {
        'lat': 37.3541069,
        'lon': -121.955238
      },
      'country': None,
      'subdivision': None,
      'city': None},
      'asn': None,
      'annotations': None
    },
    'dst_ip': None,
    'dst_ip_enrichments': None,
    'geo_distance': None,
    'sensor_ids': ['chf1'],
    'evidence_iql': 'flow:ip = 1.2.3.4 AND proto = "tcp" AND customer_id = '
      '"cid" AND timestamp >= t"2022-10-16T21:59:53.998Z" AND '
      'timestamp <= t"2022-10-16T22:59:54.814Z"',
    'context': '{"Lowest '
      'ports":["0","1","2","3","4","5","7","9","11","13","15","17","18","19",
      "20","21","23","24","25","27","29","31","33","35","37","38"],"Count '
      'of distinct hosts":16646,"Duration (seconds) of '
      'activity":"3600.816","Average duration (seconds) between '
      'connections":"0.005"}',
    'intel': None,
    'class': 'specific'
}]
```

The next section provides more details, instructions and recommendations regarding how to build and use the client library to retrieve each event type in order to import them.

Appendix

This section contains the following topics:

FncClient Library Error Types

Any failure in the Client Library is reported as a `FncClientError`. This class extends the Exception class and can be imported from `fnc.errors`. The exact reason of the failure could be distinguished by looking into its internal fields and specifically its ErrorType. The entire list of identified error types is shown below. Any known error will be wrapped into a `FncClientError` with error type `GENERIC_ERROR`.

```
# Client related errors
    GENERIC_ERROR
    CLIENT_VALIDATION_ERROR
    CLIENT_API_TOKEN_VALIDATION_ERROR

# Continuous Polling related errors
    POLLING_VALIDATION_ERROR
    POLLING_TIME_WINDOW_ERROR
    POLLING_INVERTED_TIME_WINDOW_ERROR
    POLLING_EMPTY_TIME_WINDOW_ERROR
    POLLING_LIMIT_OVERPASSED
    MISSING_CONTEXT

# Endpoint related errors
    ENDPOINT_ERROR
    ENDPOINT_VALIDATION_ERROR
    ENDPOINT_RESPONSE_VALIDATION_ERROR

# Request related errors
    REQUEST_VALIDATION_ERROR
    REQUEST_CONNECTION_ERROR
    REQUEST_TIMEOUT_ERROR
    REQUEST_HTTP_ERROR
    REQUEST_CLOSING_SESSION_ERROR
    REQUEST_ERROR

# Metastream related errors
    EVENTS_FETCH_VALIDATION_ERROR
    EVENTS_UNKNOWN_DATE_PREFIX_FORMAT
```

FncClient Library Supported Endpoints

The FncClient Library and specifically the `FncApiClient`, leverage the REST APIs exposed by the FortiNDR Cloud services to manage their services. Only some of the endpoints available on those APIs are supported by the FncClient Library. A complete list of those supported endpoints and a description of each of them is presented in this section.

```
# Sensors API's Endpoints
   GET_SENSORS
   GET_DEVICES
   GET_TASK
   GET_TASKS
   CREATE_TASK
   GET_TELEMETRY_EVENTS
   GET_TELEMETRY_PACKETSTATS
   GET_TELEMETRY_NETWORK
# Entity API's Endpoints
   GET_ENTITY_SUMMARY
   GET_ENTITY_ANNOTATIONS
   GET_ENTITY_PDNS
   GET_ENTITY_DHCP
   GET_ENTITY_FILE
# Detections API's Endpoints
   GET_DETECTIONS
   RESOLVE_DETECTION
   GET_DETECTION_EVENTS
   GET_RULES
   GET_RULE
   CREATE_RULE
   GET_RULE_EVENTS
```

GET_SENSORS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a list of sensors.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_uuid`	string	false	false	UUID of account to filter by.
`account_code`	string	false	false	Account code to filter by.
`sensor_id`	string	false	false	ID of the sensor to filter by.

Argument	Type	Required	Multi-Value	Description
`include`	string	false	true	Include additional metadata such as `status`, `interfaces`, `admin.sensor`, `admin.zeek`, `admin.suricata`, and `network_ usage`. The values are provided as a comma-separated lists.
`enabled`	bool	false	false	Filter enabled or disabled sensors. If it is not provided, all the sensors are returned regardless their status.

Response

The return is a JSON containing the two fields. The sensors field contains the list of sensors retrieved and the `result_count` shows how many sensors were retrieved.

Example

```
{
    "result_count": 2222,
    "sensors": […]
}
```

GET_DEVICES

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a list of devices.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_uuid`	string	false	false	UUID of account to filter by.
`start_date`	string	false	false	Filter devices based on when they were seen.
`end_date`	string	false	false	Filter devices based on when they were seen.
`cidr`	string	false	false	Filter devices that are under a string specific CIDR.
`sensor_id`	string	false	false	Filter devices that were observed by a specific sensor.
`dedup_sensor_id`	string	false	false	Allowed values: `YES`, `NO`

Argument	Type	Required	Multi-Value	Description
`traffic_direction`	string	false	false	Filter devices that have been noted to only have a certain directionality of traffic (external or "internal). Allowed values: `internal`, `external`
`sort_by`	string	false	false	Allowed values: `ip_address`, `internal`, `external`
`sort_direction`	string	false	false	Allowed values: `desc`, `asc`

Response

The return is a JSON containing a devices field containing:

- `device_list`: List of the retrieved devices.
- `result_count`: Total amount of devices.
- `return_count`: The number of retrieved devices.

Example

```
{
    "devices": {
        "device_list": […],
        "result_count": 310992,
        "return_count": 10000
    }
}
```

GET_TASK

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a PCAP task.

Arguments

Argument	Type	Required	Multi-Value	Description
`task_id`	string	true	false	ID of the task to be retrieved

Response

The return is a JSON with a field `pcap_task` containing the retrieved task.

Example

```
{
    "pcap_task": {
        "task_uuid": "…",
        "description": "…",
        "name": "…",
        "sensor_ids": [],
        "account_code": "…",
        "bpf": "…",
        "created_uuid": "…",
        "updated_uuid": null,
        "created_email": "…",
        "updated_email": null,
        "created": "…",
        "updated": "…",
        "requested_start_time": "…",
        "requested_end_time": "…",
        "actual_start_time": "…",
        "actual_end_time": "…",
        "status": "…",
        "files": [],
        "has_files": false
    }
}
```

GET_TASKS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a list of all the PCAP tasks.

Arguments

Argument	Type	Required	Multi-Value	Description
`sensor_id`	string	false	false	ID of the sensor to filter by.
`created_start`	string	false	false	Filter tasks based on when they were created.
`created_end`	string	false	false	Filter tasks based on when they were created.
`search_text`	string	false	false	Filter tasks including the containing text.
`has_files_only`	bool	false	false	Filter tasks based on whether they have files or not.
`page_size`	int	false	false	
`page_num`	int	false	false	

Response

The return is a JSON with two fields:

- `pcaptasks`: containing a list with the retrieved tasks.
- `result_count`: containing the number of retrieved tasks

Example

```
{
    "pcaptasks": […]
    "result_count": 2
}
```

CREATE_TASK

This endpoint is used to communicate with the FortiNDR Cloud Service to create a new PCAP task.

Arguments

Argument	Type	Required	Multi-Value	Description
`description`	string	true	false	A description of the task.
`name`	string	true	false	The name of the task.
`sensor_ids`	string	true	false	Sensor IDs on which this task will run (separate multiple accounts by comma).
`account_uuid`	string	true	false	Filter tasks including the containing text.
`bpf`	string	true	false	The Berkeley Packet Filter for capture filtering.
`requested_start_date`	string	true	false	The date the task will become active. (2024-01-30T00:00:00.000Z).
`requested_end_date`	string	true	false	The date the task will become inactive. (2024-12-31T23:59:59.000Z).

Response

The return is a JSON with a field `pcaptask` containing the created task.

Example

```
{
    "pcap_task": {
        "task_uuid": "…",
        "description": "…",
        "name": "…",
        "sensor_ids": [],
        "account_code": "…",
        "bpf": "…",
        "created_uuid": "…",
        "updated_uuid": null,
        "created_email": "…",
        “updated_email”: null,
        "created": "…",
        "updated": "…",
        "requested_start_time": "…",
        "requested_end_time": "…",
        "actual_start_time": "…",
        "actual_end_time": "…",
        "status": "…",
        "files": [],
        "has_files": false
    }
}
```

GET_TELEMETRY_EVENTS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve event telemetry data grouped by time.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_uuid`	string	false	false	UUID of account to filter by.
`account_code`	string	false	false	Account code to filter by.
`sensor_id`	string	false	false	Sensor id to filter by.
`start_date`	string	false	false	Start date/time to query for. The string default is 1 day ago for `interval=hour` or 30 days ago for `interval=day`.
`end_date`	string	false	false	End date/time to query for. The default is the current time.

Argument	Type	Required	Multi-Value	Description
`event_type`	string	false	false	The type of event. Limited to flow, dns, http, ssl, and x509.
`interval`	string	false	false	Interval to group by: `hour` (default) or `day`.
`group_by`	string	false	false	Optionally group results by: `event_type`, `sensor_id`, `account_code`
`sort_direction`	string	false	false	Allowed values: `desc`, `asc`

Response

The return is a JSON with three fields:

- `columns`: Containing the column's headers for the retrieved information.
- `data`: A list containing the retrieved data. Each element is also a list with the values for each of the column's headers
- `result_count`: Containing the size of the retrieved list of data

Example

```
{
    "columns": […]
    "data": […]
    "result_count": 25
}
```

GET_TELEMETRY_PACKETSTATS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve packetstats telemetry data grouped by time.

Arguments

Argument	Type	Required	Multi-Value	Description
`sensor_id`	string	false	false	Sensor id to filter by.
`account_code`	string	false	false	Account code to filter by.
`start_date`	string	false	false	Start date/time to query for. The string default is 1 day ago for `interval=hour` or 30 days ago for `interval=day`.

Argument	Type	Required	Multi-Value	Description
`end_date`	string	false	false	End date/time to query for. The default is the current time.
`interval`	string	false	false	Interval to group by: `hour` (default) or `day`.
`group_by`	string	false	false	Optionally group results string by: interface_name, sensor_id or account_code.

Response

The return is a JSON with three fields:

- `columns`: Contains the column's headers for the retrieved information.
- `data`: A list containing the retrieved data. Each element is also a list with the values for each of the column's headers
- `result_count`: Contains the size of the retrieved list of data

Example

```
{
    "columns": […]
    "data": […]
    "result_count": 25
}
```

GET_TELEMETRY_NETWORK

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve network telemetry data grouped by time.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_code`	string	false	false	Account code to filter by.
`start_date`	string	false	false	Start date to filter by.
`end_date`	string	false	false	End date to filter by.
`interval`	string	false	false	The interval to filter by (`day`, `month_to_day`).

Argument	Type	Required	Multi-Value	Description
`latest_each_month`	string	false	false	Filters out all but the latest day and `month_to_date` for each month.
`sort_order`	string	false	false	Sorts by account code first, then timestamp. Allowed values: `asc` or `desc`. The default is `desc`.
`limit`	string	false	false	The number of records to return. Default=100 and Maximum=1000
`offset`	string	false	false	The number of records to skip. Default = `0`.

Response

The return is a JSON with six fields:

- `network_usage`: Contains a list with the retrieved information.
- `total_count`: he total amount of elements
- `result_count`: Contains the number of elements that were retrieved.
- `sort_order`:Either `asc` or `desc`.
- `limit`: The maximum number of elements to be retrieved.
- `offset`: The number of elements to be skipped.

Example

```
{
    "network_usage": […],
    "total_count": 10379,
    "result_count": 1000,
    "sort_order": "desc",
    "limit": 1000,
    "offset": 0
}
```

GET_ENTITY_SUMMARY

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve summary information about an IP or domain.

Arguments

Argument	Type	Required	Multi-Value	Description
`entity`	string	true	false	Entity for which you want to retrieve the information
`account_uuid`	string	false	true	Account UUIDs to filter by. If absent, all the caller's allowed accounts will be queried.
`entity_type`	string	false	false	Type of the entity we are searching. Allowed values are: `ip`, `domain`.

Response

The return is a JSON with two fields:

- `query_type`: The type of the query (i.e., `ip_address`, `domain`, etc.).
- `summary`: The retrieved summary information.

Example

```
{
    "query_type": "ip_address",
    "summary": {
        "entity": …,
        "first_seen": …,
        "last_seen": …,
        "prevalence_count_internal": …,
        "tags": […]
    }
}
```

GET_ENTITY_ANNOTATIONS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve annotations information about an IP address.

Arguments

Argument	Type	Required	Multi-Value	Description
account_uuid	string	false	true	Account uuid owning the annotations. If absent or null, all allowed accounts for the caller will be used.
entities	string	true	false	JSON string containing the list of entities to lookup. Each item containing the entity and its type. Example: `[ { "entity": "10.0.0.1", "entity_type": "ip" }, { "entity": "10.0.0.2", "entity_type": "ip" } ]`

Response

The return is a JSON containing the list of annotations for each entity. Each item containing three fields:

- `account_code`: account owning the annotations.
- `entity`: Contains the provided entity and its type.
- `annotations`: the list of annotations for the specific `entity.dhcp`

Example:

```
{
  "entity_annotations": [
    {
      "account_code": "…",
      "entity": {
        "entity": "10.0.0.1",
        "entity_type": "ip"
      },
      "annotations": [ … ]
    },
    {
      "account_code": "…",
      "entity": {
        "entity": "10.0.0.2",
        "entity_type": "ip"
      },
      "annotations": [ … ]
    }
  ]
}
```

GET_ENTITY_PDNS

This endpoint is used to communicate with FortiNDR Cloud Service to retrieve passive DNS information about an IP or domain.

Arguments

Argument	Type	Required	Multi-Value	Description
`entity`	string	true	false	Entity for which we want to retrieve the PDNS information.
`account_uuid`	string	false	true	Limit results to the specified account UUID(s). Defaults to all accounts for which the user has permission.
`record_type`	string	false	true	Limit results to the specified DNS query type(s). Supported types are: A, AAAA, CNAME, MX, NS. Case insensitive.
`source`	string	false	true	Limit the results to the specified data source(s). Note that not all Sources populate all fields. Supported sources are: ICEBRG_DNS. Case insensitive.
`start_date`	string	false	false	The earliest date before which to exclude results. Day granularity, inclusive.
`end_date`	string	false	false	The latest date before which to exclude results. Day granularity, inclusive.
`resolve_external`	string	false	false	Sorts by account code first, then timestamp. Allowed values: `asc`or `desc`. The default is `desc`.
`limit`	string	false	false	Maximum number of records to be returned. Default 1000.

Response

The return is a JSON with three fields:

- query_type: The type of the query (i.e., ip_address, domain, etc.).
- result_count: Contains the number of elements that were retrieved.
- passivedns: A list containing the retrieved PDNS information.

Example

```
{
    "query_type": "ip_address",
    "result_count": 1000,
    "passivedns": […]
}
```

GET_ENTITY_DHCP

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve DHCP information about an IP address.

Arguments

Argument	Type	Required	Multi-Value	Description
entity	string	true	false	Entity for which we want to retrieve the DHCP information
account_uuid	string	false	true	Limit results to the specified account UUID(s). Defaults to all accounts for which the user has permission.
sensor_id	string	false	false	Sensor id to filter by.
start_date	string	false	false	Limit the results to those DHCP leases that start after this start date.
end_date	string	false	false	Limit the results to those DHCP leases that start before this end date.

Response

The return is a JSON with three fields:

- query_type:The type of the query (i.e., ip_address, domain, etc.).
- result_count: Contains the number of elements that were retrieved.
- dhcp: A list containing the retrieved DHCP information.

Example

```
{
    "query_type": "ip_address",
    "result_count": 1000,
    "annotations": […]
}
```

GET_ENTITY_FILE

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve information about a file.

Arguments

Argument	Type	Required	Multi-Value	Description
`entity`	string	true	false	Entity for which we want to retrieve the information
`account_uuid`	string	false	true	Account UUID(s) to filter by. If absent, all the caller's allowed accounts will be queried.

Response

The return is a JSON with three fields:

- `query_type`: The type of the query (i.e., ip_address, domain, etc.).
- `hash_type`: The hash type that was applied.
- `file`: The retrieved file's information.

Example

```
{
    "query_type": "ip_address",
    "hash_type": 1000,
    "file": {…}
}
```

GET_DETECTIONS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a list of detections.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_uuid`	string	false	false	Account to filter results by.
`rule_uuid`	string	false	true	Rule UUID's to filter results by.
`status`	string	false	true	Status to filter results by (`active`, `resolved`).
`device_ip`	string	false	false	Device IP to filter by.
`sensor_id`	string	false	false	Sensor ID to filter by.
`include`	string	false	true	Include linked objects (`rules`, `indicators`).
`sort_order`	string	false	false	Direction of sort (`desc`, `asc`).
`sort_by`	string	false	false	Filed to sort by (`first_seen`, `last_seen`, `status`, `device_ip`, `indicator_count`).
`limit`	int	false	false	The number of records to include (default:100, max:10,000)
`offset`	int	false	false	The number of records to skip.
`is_muted`	bool	false	false	Filter results by whether a detection is muted by any way.
`muted`	bool	false	false	Filter results by whether or not the detection is muted.
`muted_rule`	bool	false	false	Filter results by whether the associated rule is muted for the account.
`muted_device`	bool	false	false	Filter results by whether the device is muted.
`rule_severity`	string	false	true	Filter results by rule severity (low, moderate, high).
`rule_category`	string	false	true	Filter results by rule category.
`rule_confidence`	string	false	true	Filter results by rule confidence (low, moderate, high).
`resolution`	string	false	true	Filter results by resolution in resolution table.
`resolution_user_uuid`	string	false	true	Filter results by resolution user UUID.
`uuid`	string	false	false	Filter results if there is resolution user UUID on the detection.
`resolve_start_date`	string	false	false	Resolved start date to filter by (inclusive).

Argument	Type	Required	Multi-Value	Description
`resolve_end_date`	string	false	false	Resolved end date to filter by (exclusive).
`active_start_date`	string	false	false	Active start date to filter by (inclusive).
`active_end_date`	string	false	false	Active end date to filter by (exclusive).
`created_start_date`	string	false	false	Created start date to filter by (inclusive).
`created_end_date`	string	false	false	Created end date to filter by (exclusive)
`created_or_shared_start_date`	string	false	false	Created or shared start date to filter by (inclusive).
`created_or_shared_end_date`	string	false	false	Created or shared end date to filter by (exclusive).
`rule_enabled`	string	false	false	Filter results if the rule is enabled.
`rule_attack_id`	string	false	true	Filter results by rule attack IDs.
`rule_account_uuid`	string	false	false	Filter results by rule account UUID.
`indicator_value`	string	false	false	Indicator value to filter by.

Response

The return is a JSON with eight fields:

- `detections`: A list with the retrieved detections.
- `rules`: A list with the rules associated to the retrieved detections.
- `total_count`: The total amount of elements
- `result_count`: Contains the number of elements that were retrieved.
- `sort_order`: Either `asc` or `desc`.
- `sort_by`: Feld used to sort the result (`first_seen`, `last_seen`, `status`, `device_ip`, `indicator_count`).
- `limit`: The maximum number of elements to be retrieved
- `offset`: The number of elements to be skipped.

Example

```
{
    "result_count": 6074,
    "total_count": 6074,
    "detections": […],
    "sort_by": "first_seen",
    "sort_order": "asc",
    "offset": 0,
    "limit": 10000,
    "rules": […]
}
```

RESOLVE_DETECTION

This endpoint is used to communicate with the FortiNDR Cloud Service to resolve a detection.

Arguments

Argument	Type	Required	Multi-Value	Description
`detection_id`	string	true	false	UUID for the detection to be resolved
`resolution`	string	true	false	Resolution category (`true_positive_mitigated`, `true_positive_no_action`, `false_positive`, `unknown`).
`resolution_comment`	string	false	false	Optional comment for the resolution.

Response

This endpoint returns an empty HTTP 204 response.

GET_DETECTION_EVENTS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a list of events associated to a detection.

Arguments

Argument	Type	Required	Multi-Value	Description
`detection_uuid`	string	true	false	UUID for the detection to filter by.
`offset`	string	false	false	The number of records to skip past. The default is 0.
`limit`	string	false	false	The maximum number of records to return. The default is 0 and the max is 1000.

Response

The return is a JSON with three fields:

- `events`: A list containing the retrieved events.
- `total_count`: The total amount of elements
- `result_count`: Contains the number of elements that were retrieved.

Example

```
{
    "result_count": 6074,
    "total_count": 6074,
    "events": […]
}
```

GET_RULES

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a list of rules.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_uuid`	string	false	false	Optional account to filter results by.
`rule_account_uuid`	string	false	false	Optional account to filter rule account sub-objects by.
`search`	string	false	false	Search string on name, category or description.
`has_detections`	bool	false	false	Include rules that have unmuted, unresolved detections.
`detection_device_ip`	string	false	false	Device IP of associated detections to filter by. Note the associated detections are determined by the `has_detections param`.
`indicator_value`	string	false	false	Case insensitive prefix search of related indicators.
`severity`	string	false	true	List of severities to filter by (low, moderate, high).
`confidence`	string	false	true	List of confidences to filter by (low, moderate, high).
`category`	string	false	true	Category to filter by (see above).
`rule_account_muted`	bool	false	false	Include muted rules accounts for the given `account_uuid`.
`enabled`	bool	false	false	Enabled rules only.
`attack_id`	string	false	true	List of MITRE ATT&CK technique ID to filter by. Leave blank or null to turn off this filter.

Argument	Type	Required	Multi-Value	Description
`sort_by`	string	false	false	The field to sort by: `created`, `updated`, `detections`, `severity`, `confidence`, `category`, `last_seen`, `detections_muted`. Defaults to updated.
`sort_order`	string	false	false	The sort order: `asc` or `desc`. The default is `desc`.
`limit`	int	false	false	The maximum number of records to return. Default: 100, Max: 1000.
`offset`	int	false	false	The number of records to skip past. Default: 0.

Response

The return is a JSON with ten fields:

- `rules`: The list of the rules retrieved.
- `accounts`: The list of the associated accounts.
- `active_rule_count`: The number of active rules
- `enabled_rule_count`: The number of enabled rules.
- `total_count`: The total amount of elements
- `result_count`: Contains the number of elements that were retrieved.
- `sort_order`: Either `asc` or `desc`.
- `sort_by`: Filed used to sort the result (`first_seen`, `last_seen`, `status`, `device_ip`, `indicator_count`).
- `limit`: The maximum number of elements to be retrieved
- `offset`: The number of elements to be skipped.

Example

```
{
    "rules": […],
    "accounts": […],
    "active_rule_count": 404,
    "enabled_rule_count": 1723,
    "result_count": 100,
    "total_count": 1843,
    "sort_by": "updated",
    "sort_order": "desc",
    "offset": 100,
    "limit": 100
}
```

GET_RULE

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a specific rule.

Arguments

Argument	Type	Required	Multi-Value	Description
`rule_id`	string	true	false	UUID for the rule to filter by.

Response

The return is a JSON with two fields:

- `rule`: Contains the retrieved rule.
- `accounts`: A list containing the associated accounts.

Example

```
{
    "rule": {…},
    "accounts": […],
}
```

CREATE_RULE

This endpoint is used to communicate with the FortiNDR Cloud Service to create a rule.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_uuid`	string	true	false	Rule's account
`name`	string	true	false	Rule's name
`category`	string	true	false	Rule's category
`query_signature`	string	true	false	Rule's query signature
`description`	string	false	false	Rule's description
`severity`	string	true	false	Rule's severity
`confidence`	string	true	false	Rule's confidence
`primary_attack_id`	string	false	false	
`secondary_attack_id`	string	false	false	

Argument	Type	Required	Multi-Value	Description
`specificity`	string	false	false	
`device_ip_fields`	string	true	false	
`indicator_fields`	string	false	false	
`run_account_uuids`	string	true	false	
`auto_resolution_minutes`	string	false	false	Time to be automatically resolved

Response

The return is a JSON with a field rule containing the created rule.

Example

```
{
    "rule": {
        "uuid": "…",
        "account_uuid": "…",
        "shared_account_uuids": null,
        "run_account_uuids": ["…"],
        "name": "…",
        "category": "…",
        "query_signature": "…",
        "description": null,
        "severity": "…",
        "confidence": "…",
        "auto_resolution_minutes": null,
        "enabled": true,
        "created_user_uuid": "…",
        "created": "…",
        "updated_user_uuid": "…",
        "updated": "…",
        "critical_updated": "…",
        "primary_attack_id": null,
        "secondary_attack_id": null,
        "specificity": null,
        "rule_accounts": null,
        "device_ip_fields": ["…"],
        "indicator_fields": null,
        "source_excludes": ["…"]
    }
}
```

GET_RULE_EVENTS

This endpoint is used to communicate with the FortiNDR Cloud Service to retrieve a list of events associated to a rule.

Arguments

Argument	Type	Required	Multi-Value	Description
`account_uuid`	string	false	false	UUID for the account to filter by.
`offset`	string	false	false	The number of records to skip past. The default is 0.
`limit`	string	false	false	The maximum number of records to return.The default is 0 and the max is 1000.

Response

The return is a JSON with three fields:

- `events`: A list containing the retrieved events.
- `total_count`: The total amount of elements
- `result_count`: Contains the number of elements that were retrieved.

Example

```
{
    "result_count": 6074,
    "total_count": 6074,
    "events": […]
}
```

Event Type Structure

Suricata

Field	Type	Description	Example
`alert`	suricata.alert		

Field	Type	Description	Example
customer_id	string	The code of the account that owns the event.	cust
dest_ip	string	The IP of the responder to the connection.	1.2.3.4
dest_port	integer	The port of the responder to the connection.	53
dst_ip_enrichments	ip_enrichments	Enrichments for an IP.	
event_type	string	The type of event recorded.	flow
geo_distance	number	The difference between src and dst geo values.	1410.373826280689
http	suricata.http		
intel	intel[ ]	Intel that matched entities in the event.	
payload	string	The raw payload from the traffic that matched the signature.	
proto	string	The transport layer protocol used.	tcp
sensor_id	string	The sensor that created the event.	sen1
source	string	The source of the event.	Zeek
src_ip	string	The IP of the initiator of the connection.	4.3.2.1
src_ip_enrichments	ip_enrichments	Enrichments for an IP.	
src_port	integer	The port of the initiator of the connection.	52843
timestamp	string	The time at which traffic for the event began.	2019-01-01T00:00:00.000000Z
Uuid	string	A unique identifier for the event.	1ea156cb-9462-16e9-f5cf-02372fae0a1a

Suricata.alert

Field	Type	Description	Example
`category`	string	The signature's category.	A Network Trojan was Detected
`rev`	integer	The signature's revision number.	2
`severity`	integer	The signature's severity rating (1 = high, 3 = low).	1
`signature`	string	The signature's name.	ET TROJAN Jaff Ransomware Checkin M1
`signature_id`	integer	The signature's ID.	2024290

Suricata.http

Field	Type	Description	Example
`hostname`	string	The content Host header.	www.google.com
`hostname_enrichments`	ip_enrichments or domain_enrichments	Enrichments for an IP or domain.	
`http_content_type`	string	The fingerprinted MIME-type of the response content, use instead of response_mime.	text/html
`http_method`	string	The HTTP method selected.	GET
`http_refer`	string	The content of the Referrer header.	http://au.search.yahoo.com/search?p=planetside.co.uk&fr=sfp&fr2=sb-top-search

Field	Type	Description	Example
`http_user_agent`	string	The content of the `UserAgentheader`.	Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko
`length`	integer	The length in bytes of the response.	24
`protocol`	string	The protocol version.	HTTP/1.1
`redirect`	string	The target location of the redirect.	http://dmd.metaservices.microsoft.com/metadata.svc
`status`	integer	The numeric code of the server's response.	200
`url`	string	The full URI of the request.	/index.php
`xff`	string	The content of the X-FORWARDED-FOR header	http://dmd.metaservices.microsoft.com/metadata.svc

Observations

Field	Type	Description	Example
`category`	string	The subject of an observation.	relationship
`class`	string		
`confidence`	string	The confidence in the model output to what was attempted to be observed.	high
`context`	string	A set of key value pairs providing additional details for this specific observation occurrence.	

Field	Type	Description	Example
`customer_id`	string	The code of the account that owns the event.	chg
`dst_ip`	string	The IP of the responder to the connection.	1.2.3.4
`dst_ip_enrichments`	ip_enrichments	Enrichments for an IP.	
`event_type`	string	The type of event recorded.	flow
`evidence_end_timestamp`	string	The timestamp for which the flagged activity ended.	2019-01-01T00:00:00.000000Z
`evidence_iql`	string	An IQL statement that attempts to identify the events used to generate the observation.	src.ip = 4.3.2.1' AND customer_id = 'abc' AND dce_rpc:dce_rpc_operation = 'NetrSessionEnum' AND timestamp >= t'2019-01-01T22:00:00.000000Z' AND timestamp <= t'2019-01-01T22:10:00.000000Z'
`evidence_start_timestamp`	string	The timestamp for which the flagged activity began.	2019-01-01T00:00:00.000000Z
`geo_distance`	number	The difference between `src` and `dst`geovalues.	1410.373826280689
`intel`	intel[]	Intel that matched entities in the event.	
`observation_uuid`	string	A unique identifier for the model used to generate the observation. Multiple models may exist for the same title.	ac33589c-ee31-4f5e-b6a1-dcb23da37205
`sensor_id`	string	The sensor that created the event.	sen1
`sensor_ids`	string[]	A list of sensors from which activity was used as part of the observation.	[abc1,abc2,abc3]
`source`	string	The source of the event.	Zeek

Field	Type	Description	Example
`src_ip`	string	The IP of the initiator of the connection.	4.3.2.1
`src_ip_enrichments`	ip_enrichments	Enrichments for an IP.	
`timestamp`	string	The time at which traffic for the event began.	2019-01-01T00:00:00.000000Z
`title`	string	The title of what was attempted to be detected (similar to a suricata `sig` name).	High Count of NetSession Destinations
`uuid`	string	A unique identifier for the event.	ac33589c-ee31-4f5e-b6a1-dcb23da37205
`category`	string	The subject of an observation.	relationship

www.fortinet.com

Copyright© 2025 Fortinet, Inc. All rights reserved. Fortinet®, FortiGate®, FortiCare® and FortiGuard®, and certain other marks are registered trademarks of Fortinet, Inc., and other Fortinet names herein may also be registered and/or common law trademarks of Fortinet. All other product or company names may be trademarks of their respective owners. Performance and other metrics contained herein were attained in internal lab tests under ideal conditions, and actual performance and other results may vary. Network variables, different network environments and other conditions may affect performance results. Nothing herein represents any binding commitment by Fortinet, and Fortinet disclaims all warranties, whether express or implied, except to the extent Fortinet enters a binding written contract, signed by Fortinet's Chief Legal Officer, with a purchaser that expressly warrants that the identified product will perform according to certain expressly-identified performance metrics and, in such event, only the specific performance metrics expressly identified in such binding written contract shall be binding on Fortinet. For absolute clarity, any such warranty will be limited to performance in the same ideal conditions as in Fortinet's internal lab tests. Fortinet disclaims in full any covenants, representations, and guarantees pursuant hereto, whether express or implied. Fortinet reserves the right to change, modify, transfer, or otherwise revise this publication without notice, and the most current version of the publication shall be applicable.