Mandiant

USE THE DIGITAL THREAT MONITORING API

Overview

In this document, we're providing common ways to use the Digital Threat Monitoring (DTM) API. The examples here use the CURL command, but you could also use Postman or your favorite API tool.

The maximum number of API requests per 24 hours is limited to 50K. If you require more than 50K requests in a 24-hour period, contact **Support (https://docs.mandiant.com/home/mandiant-support-cases)**.

Authentication

The DTM API supports two forms of authentication:

- Bearer-based authentication using a valid **JWT (https://en.wikipedia.org/wiki/JSON_Web_Token)**.
- Basic authentication using an API access Key ID and Secret.

Bearer Authentication

The first thing that you need to interact with the REST API is a valid Bearer token (JWT) that includes the DTM grant within the token. For usage within this document we've stored that Bearer token in an environment variable called `DTM_BEARER`.

Basic Authentication

Basic authentication requires obtaining and using an API Key ID and Secret.

Get API Key ID and Secret

To obtain a **Service API Key** (which is tied to an organization rather than an individual user) for use with third-party security technologies such as a SIEM, contact **Support (https://www.mandiant.com/support)**.

To obtain an API Key ID and Secret for an individual user account, perform the following:

1. Navigate to the Mandiant Threat Intelligence web console.
2. Click **Account Settings**.
3. Select **API Access and Keys** from the navigation menu.
4. Click **Get Key ID and Secret**.
5. Copy and store the displayed values in a secure location.

When performing basic authentication with these values, the ID should be used as the username and the Secret as the password.

Create Monitors

A Monitor contains one or more conditions that let you target the types of content (called topics) to alert on. It also includes metadata about the monitor (name, description, status, email alert notification status). When the conditions of a Monitor are met for a given document, an Alert is generated for that Monitor.

The following code block creates a Monitor that matches on all documents that have the `mandiant.com` domain, and include any of the following text: `hack`, `password`, `root`, `malware`, `sell`, or `ransomware`.

```
curl -k -s -H 'Content-Type: application/json' -H "Authorization: Bearer ${DTM_BEARER}" https://api.intelligence.mandiant.com/v4/dtm/monitors --request POST --data-binary @- <<BODY
{
   "name": "monitor mandiant.com",
   "description": "monitor mandiant.com domain for suspicious keywords",
   "doc_condition": {
      "topic": "match_conditions",
      "operator": "all",
      "match": [
         {
            "topic": "group_network",
            "operator": "must_equal",
            "match": [
               "mandiant.com"
            ]
         },
         {
            "topic": "keyword",
            "operator": "must_contain",
            "match": [
               "hack",
               "password",
               "root",
               "malware",
               "sell",
               "ransomware"
            ]
         }
      ]
   }
}
BODY
```

The response includes your newly created Monitor:

```
{
  "id": "cad5l885cdt77b0idivg",
  "name": "monitor mandiant.com",
  "description": "monitor mandiant.com domain for suspicious keywords",
  "enabled": true,
  "doc_condition": {
    "operator": "all",
    "topic": "match_conditions",
    "match": [
      {
        "operator": "must_equal",
        "topic": "group_network",
        "match": [
          "mandiant.com"
        ]
      },
      {
        "operator": "must_contain",
        "topic": "keyword",
        "match": [
          "hack",
          "password",
          "root",
          "malware",
          "sell",
          "ransomware"
        ]
      }
    ]
  },
  "created_at": "2022-06-03T19:01:53.442Z",
  "updated_at": "2022-06-03T19:01:53.442Z",
  "email_notify_enabled": false,
  "email_notify_immediate": false
}
```


This next code block creates a Monitor that watches for organizations `army` or `navy` and targets potentially dark activity in content found within the `United States`:

```
curl -k -s -H 'Content-Type: application/json' -H "Authorization: Bearer ${DTM_BEARER}" https://api.intelligence.mandiant.com/v4/dtm/monitors --request POST --data-binary @- <<BODY
{
   "name": "army navy us dark",
   "description": "Monitor army or navy for dark activity in the united states",
   "doc_condition": {
      "operator": "all",
      "topic": "match_conditions",
      "match": [
         {
            "operator": "must_equal",
            "topic": "group_brand",
            "match": [
               "army",
               "navy"
            ]
         },
         {
            "operator": "must_equal",
            "topic": "group_location",
            "match": [
               "us",
               "usa",
               "united states"
            ]
         },
         {
            "operator": "must_equal",
            "topic": "keyword",
            "match": [
               "hack",
               "password",
               "root",
               "malware",
               "sell",
               "ransomware"
            ]
         }
      ]
   }
}
BODY
```

The response includes your newly created Monitor:

```
{
  "id": "cad5n8o5cdt77b0idjlg",
  "name": "army navy us dark",
  "description": "Monitor army or navy for dark activity in the united states",
  "enabled": true,
  "doc_condition": {
     "operator": "all",
     "topic": "match_conditions",
     "match": [
         {
            "operator": "must_equal",
            "topic": "group_brand",
            "match": [
               "army",
               "navy"
            ]
         },
         {
            "operator": "must_equal",
            "topic": "group_location",
            "match": [
               "us",
               "usa",
               "united states"
            ]
         },
         {
            "operator": "must_equal",
            "topic": "keyword",
            "match": [
               "hack",
               "password",
               "root",
               "malware",
               "sell",
               "ransomware"
            ]
         }
      ]
  },
  "created_at": "2022-06-03T19:06:11.086Z",
  "updated_at": "2022-06-03T19:06:11.086Z",
  "email_notify_enabled": false,
  "email_notify_immediate": false
}
```


More on Queries and Searches

Our system collects and ingests potential threats from the “dark web” (both public and private sources). After ingestion, the collected documents are run through a Machine Learning (ML) pipeline which extracts “topics” and “labels” from these documents. These topics/labels provide more context about the document. For example, our Analysis Machine Learning pipeline extracts things such as product/brand names, URLs, IP addresses, person names, locations, and source language.

Once extraction is complete, all active Monitors are run against the given document. Any such Monitors that match/hit produce an Alert (per Monitor). This all happens in near real-time.

To build Conditions, a Condition object is used. This JSON object has the form:

```
{
  "operator": "the operation for the condition",
  "topic": "the topic to perform the operation on",
  "match": [
     "either values to match, or nested conditions for building complex conditions"
  ]
}
```

The first listed Condition on a Monitor must be used to specify whether the nested Condition must match `any` or `all` of the match conditions. Using `any` can be thought of as OR operator, while `all` can be thought of as AND.

To create a complex Condition, use the `match_conditions` topic, and then specify either `all` or `any` for the operator. In a complex Condition, the `match` values must themselves be Conditions to specify the matching criteria.

For example, you have the following complex `doc_condition`.

- The domain is `gt.co`
- OR the organization is `acme` or `myorg`
- OR
    - The domain is `t.co` or `mm.co`
    - AND the organization is `dyk` or `mke`

The conditions would look like this:

```
"doc_condition": {
  "operator": "any",
  "topic": "match_conditions",
  "match": [
    {
      "match": [
        "gt.co"
      ],
      "operator": "must_equal",
      "topic": "group_network"
    },
    {
      "match": [
        "acme",
        "myorg"
      ],
      "operator": "must_equal",
      "topic": "group_brand"
    },
    {
      "operator": "all",
      "topic": "match_conditions",
      "match": [
        {
          "match": [
            "t.co",
            "mm.co"
          ],
          "operator": "must_equal",
          "topic": "group_network"
        },
        {
          "match": [
            "dyk",
            "mke"
          ],
          "operator": "must_not_equal",
          "topic": "group_brand"
        }
      ]
    }
  ]
}
```

In this example, the values of the `match` list are either specific values to match on a particular `topic`, or they are Conditions themselves when using the `match_conditions` topic with the operator of `all` or `any`.

So far, the topics we've been using are called topic groups (indicated by the topic name starting with `group_`). Topic groups are actually searching multiple individual topics.

Monitor Condition Topics

Topic	Description	Analyzer	Topic Group	Examples

Topic	Description	Analyzer	Topic Group	Examples
access_token	Access token used by applications to authenticate against protected resources.	Keyword	group_keys	somesecretkey
atom_address	Wallet address for the Cosmos (ATOM) cryptocurrency.	Keyword	group_crypto	cosmos1mnyn7x24xj6vraxeeq56dfkxa009tvhgknhm04
bch_address	Wallet address for the Bitcoin Cash (BCH) cryptocurrency.	Keyword	group_crypto	qp3wjpa3tjlj042z2wv7hahsldgwhwy0rq9sywjpyy
bin	Complete or partial Bank Identifier.	Numeric	group_bin	6277540029070332960
brand	Brand name or trademark.	Standard	group_brand	Acme, Inc.
btc_address	Wallet address for the Bitcoin (BTC) cryptocurrency.	Keyword	group_crypto	1BvBMSEYstWetqTFn5Au4m4GFg7xJaNVN2
city	A city or locality name.	Standard	group_location	Atlanta
client_identifier	An OpenID client identifier.	Keyword	group_identity	my_client_id
country	A country or nationality name.	Standard	group_location	United States of America
crypto_key_private	Asymmetric cryptography private key.	Keyword	group_keys	-----BEGIN PRIVATE KEY-----
crypto_key_public	Asymmetric cryptograph public key.	Keyword	group_keys	-----BEGIN PUBLIC KEY-----
cve	A **Common Vulnerabilities and Exposures (CVE) (https://www.cve.org/)** Identifier.	Keyword	group_threats	CVE-2021-42013

Topic	Description	Analyzer	Topic Group	Examples
cwe	A **Common Weakness Enumeration (CWE)** (https://cwe.mitre.org/) Identifier.	Keyword	group_threats	CWE-269
dash_address	Wallet address for the Dash cryptocurrency.	Keyword	group_crypto	XdAUmwtig27HBG6WfYyHAzP8n6XC9jESEw
doge_address	Wallet address for the Doge cryptocurrency	Keyword	group_crypto	DLCDJhnh6aGotar6b182jpzbNEyXb3C361
domain	An **RFC1035** (https://datatracker.ietf.org/doc/html/rfc1035) domain name.	Keyword	group_network	http://mandiant.com
email_address	An **RFC5322** (https://datatracker.ietf.org/doc/html/rfc5322) email address	Keyword	group_identity	john.smith@mandiant.com
filename	A name or identifier for a file	Keyword	group_paths	my_spreadsheet.xlsx
hashtag	A hash tag used to identify a topic of conversation	Keyword	group_social	security
icq_uin	An ICQ user identifier	Keyword	group_social	123456
identity_name	A name of a person, place, company, or thing	Standard	group_brand, group_identity	John Smith
ipv4_address	An IPv4 Address	IP	group_network	192.168.1.1
ipv6_address	An IPv6 Address	IP	group_network	::ffff:c0a8:101
jid	A Jabber user identifier	Keyword	group_social	jsmith@jabber.com
location_name	The name of a physical place or location	Standard	group_location	Pleasant Village

Topic	Description	Analyzer	Topic Group	Examples
ltc_address	Wallet address for the Litecoin cryptocurrency	Keyword	group_crypto	MGxNPPB7eBoWPUaprtX9v9CXJZoD2465zN
mac_address	A Media Access Control (MAC) Address for a network interface card	Hash		00:01:02:03:04:05
md5_hash	An MD5 cryptographic hash	Hash	group_hash	01020304050708090a0b0c0d0e0f1011
name	A name of a person, place, company, or thing	Standard	group_brand, group_identity, group_social	Digital Threat Monitoring
organization	The name of an organization.	Standard	group_brand	Mandiant, Inc.
password_plaintext	A detected plaintext password.	Keyword	group_keys	mypassword123
path	A location of a file or folder on a filesystem	Keyword	group_paths	C:\folder\file.txt, /home/path/file.txt
phone_number	A partial or complete phone number	Numeric	group_identity	123 (456)-7890
predict_password_plaintext	A detected plaintext password (lower confidence)	Keyword	group_keys	mypassword123
product	The name of a product.	Standard	group_brand	Widgets
product_batch_name	A batch number for a product	Standard	group_brand	Feb15#Batch5
registry_key	A path in the Windows registry	Keyword	group_paths	HKCU:\Software\Microsoft\Windows\CurrentVersion\Explorer
service_name	The name of a service	Standard	group_threats	Telegram
sha1_hash	A SHA1 cryptographic hash	Hash	group_hash	01020304050708090a0b0c0d0e0f101112131415

Topic	Description	Analyzer	Topic Group	Examples
sha256_hash	A SHA256 cryptographic hash	Hash	group_hash	0102030405070809 0a0b0c0d0e0f101112131415161718191a1b1c1d1e1f2021
telegram_user_name	A username for the Telegram messaging platform	Keyword	group_social, group_identity	johnsmith
threat_group_name	The name of a threat group	Standard	group_threats	FIN7, APT10, REvil
threat_name	The name of a particular type of threat	Standard	group_threats	cmd.exe, nmap, ngrok
twitter_handle	A username for the Twitter platform	Keyword	group_social, group_identity	Mandiant
url	An **RFC1738 (https://www.ietf.org/rfc/rfc1738.txt)** uniform resource locator (URL)	Keyword	group_network	https://mandiant.com/
xlm_address	Wallet address for the Stellar (XLM) cryptocurrency	Keyword	group_crypto	GDQP2KPQGKIHYJGXNUIYOMHARUARCA7DJT5FO2FFOOKY3B2WSQHG4W37
xmr_address	Wallet address for the Monero (XMR) cryptocurrency	Keyword	group_crypto	888tNkZrPN6JsEgekjMnABU4TBzc2Dt29EPAvkRxbANsAnjyPbb3iQ1YBRk1UXcdRsiKc9dhwMVgN5S9cQUiyoogDavup3H

Group Topic List

Topic	Description
group_brand	A name of a brand, or product, or line of business
group_identity	The name of a person, place, or thing
group_network	A network identifier: IP address, mac address, or domain name, for example
group_bin	A bank identifier
group_location	A physical location, city, state, province, or locality, example
group_paths	A digital location of a file or resource
group_threats	A threat type, class, or identifier
group_keys	A password, key, or other credential

Topic	Description
group_hash	A cryptographic hash
group_social	A social media identifier
group_crypto	A cryptocurrency wallet address

Label Topic List

Topic	Description
label_type	Detected MIME type of the originating document
label_language	Two-character **ISO 639-1** **(https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes)** language code specifying the detected language type
label_industry	Industry code of the affected industries of the original document
label_threat	Threat specifier of the original document
label_content	Detected content type of the source document: prose, structured, json, code

Special Topic List

Topic	Description
doc_type	This defines the specific document type to match. Valid types include: • account_discovery • domain_discovery • forum_post • tweet • document_analysis • web_content_publish • email_analysis • paste • message • shop_listing
source	The name of the collector that collected the source document
match_conditions	A nested matcher condition
typosquatted_domain	This option accepts a plain fully qualified domain name (not URL's) and will attempt to detect and alert when similar domains are registered.
lucene	This topic accepts a Lucene query that should run against documents. Multiple queries can be specified, operator must be 'all' or 'any'.

Topic	Description
keyword	Takes a list of keywords and performs full text search against the documents using the specified operator.

Text Searching

The `lucene` topic lets you construct arbitrary queries using **ElasticSearch text query string (https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html)** style syntax. This topic is used in your Monitor Conditions just like any other topics. But this topic will permit advanced text queries on documents using regular expressions, phrase distance matching, or conjunctions.

While the general form of text searching will just search for keywords (for example, `theword`), you can also target specific JSON paths in our documents. Targeting specific paths in the model can be useful to target very specific values in the document. However, by default all text paths in documents can be searched without specifying model search paths. For example, if you use a text query string of `hack OR password` both "hack" and "password" will be searched for in all text paths of specified documents.

When specifying model paths in the query string, the format is a dotted notation where each value in the path is the property name in the document. For example, if you wanted only search for a specific hashtag value in a "tweet" document, the following path can be used: `tweet_hashtags.hashtag`. This specifies the document path to search is the `hashtag` property of the `tweet_hashtags` object in a tweet document.

To understand which paths exist, you must look at the Schemas in the API specification. There you can see the document types that correspond to our models. Today DTM supports the following top-level models:

- document_analysis
- domain_discovery
- email_analysis
- forum_post
- message
- paste
- shop_listing
- tweet
- web_content_publish

Finally, it's important that your monitor topic conditions are set up to only match on the respective document type when you want to use document model paths in your query string. For example, to search on the hashtags of tweets as mentioned previously, you would add a monitor topic condition "Search Collection Type must equal Social Media". This ensures that when your query string is used to search the document, it will always be of type tweet.

View Monitors, Searches and Alerts

To view an individual Monitor, Search, or Alert, you can `GET` it by its ID. For example, to view the Monitor we created previously, we can use its ID `cad5n8o5cdt77b0idjlg` to fetch it:

```
1 curl -k -s -H 'Accept: application/json' -H "Authorization: Bearer ${DTM_BEARER}" https://api.intelligence.mandiant.com/v4/dtm/monitors/cad5n8o5cdt77b0idjlg
```

Our response is the Monitor:

```
{
  "id": "cad5n8o5cdt77b0idjlg",
  "name": "army navy us dark",
  "description": "Monitor army or navy for dark activity in the united states",
  "enabled": true,
  "doc_condition": {
    "operator": "all",
    "topic": "match_conditions",
    "match": [
      {
        "operator": "must_equal",
        "topic": "group_brand",
        "match": [
          "army",
          "navy"
        ]
      },
      {
        "operator": "must_equal",
        "topic": "group_location",
        "match": [
          "us",
          "usa",
          "united states"
        ]
      },
      {
        "operator": "must_equal",
        "topic": "keyword",
        "match": [
          "hack",
          "password",
          "root",
          "malware",
          "sell",
          "ransomware"
        ]
      }
    ]
  },
  "created_at": "2022-06-03T19:06:11.086Z",
  "updated_at": "2022-06-03T19:06:11.086Z",
  "email_notify_enabled": false,
  "email_notify_immediate": false
}
```


The same approach shown here can be used to fetch individual Monitors and Alerts by their ID.

Update Monitors, Searches and Alerts

To update a Monitor or Alert, the API provides the `PUT` and `PATCH` methods. Use `PUT` to completely replace the resource, and use `PATCH` only update the fields specified in the request body.

For example, to disable our `cad5n8o5cdt77b0idjlg` Monitor, we can use the Monitor and Search ID to set its `enabled` boolean flag using `PATCH`:

```
curl -k -s -H 'Content-Type: application/json' -H "Authorization: Bearer ${DTM_BEARER}" https://api.intelligence.mandiant.com/v4/dtm/monitors/cad5n8o5cdt77b0idjlg --request PATCH --data-binary @- <<BODY
{
 "enabled": false
}
BODY
```

This request says to set the `enabled` field of the given Monitor ID to the value `false`. Since we are using `PATCH` no other fields are changed.

Pagination

The REST API supports pagination of results in the form of scrolling. With scrolling, you can page through your results page-by-page, but you cannot jump ahead multiple pages, or back in your page results. The APIs that support LISTing resources (for example, `GET /monitors` or `GET /alerts`) all support pagination.

The APIs that support pagination also support query parameters to define the pagination behavior. The following query parameters are most common:

- `life`: The amount of time in seconds or minutes to keep the pagination context open for. By default its `1m`. If you haven't fetched next page of results within this time period; the page context is closed and cannot be used. Simply LIST the resources again to restart. Currently the max life is `10m`.
- `sort`: The field to sort the results by. For example, to sort the results by their updated timestamp, you can use `sort=updated_at`. You can also sort by `name` or `id`, for example. The default is `created_at`.
- `order`: The sort order of the results. The possible values are `asc` and `desc` for ascending and descending respectively. The default is `desc`. This order applies the `sort` field.
- `size`: The number of resources to include in each page of results. The default is `10`.
- `since`: Used for filtering the list based on `created_at` date ranges. This is an RFC3339 timestamp that specifies the starting date range for the results. If not specified, all results are returned.
- `until`: Used for filtering the list based on `created_at` date ranges. This is an RFC3339 timestamp that specifies the ending date range for the results. If not specified, all results are returned.

The API uses the **HTTP Link Header (https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Link)** to relay the URI to the next page of results when pagination. The `rel` used in the Link is `next` and will only be included when there's another page of results. API consumers should use the Link URI exactly as provided in the Link header, as it includes information about your pagination context. No other query parameters can be used when fetching subsequent pages through the Link header URI.

To illustrate pagination, let's paginate through the two Monitors we created previously. For this example, we'll paginate the `/monitors` API and tell it to only return one Monitor per page (just for illustration purposes) using the `size` query parameter:

```
1 curl -k -s -H 'Accept: application/json' -H "Authorization: Bearer ${DTM_BEARER}" "https://api.intelligence.mandiant.com/v4/dtm/monitors?size=1"
```

This returns a list with our most recent Monitor:

```
{
  "monitors":[
    {
      "id":"cad5n8o5cdt77b0idjlg",
      "name":"army navy us dark",
      "description":"Monitor army or navy for dark activity in the united states",
      "enabled":true,
      "doc_condition":{
        "operator":"all",
        "topic":"match_conditions",
        "match":[
          {
            "operator":"must_equal",
            "topic":"group_brand",
            "match":[
              "army",
              "navy"
            ]
          },
          {
            "operator":"must_equal",
            "topic":"group_location",
            "match":[
              "us",
              "usa",
              "united states"
            ]
          },
          {
            "operator":"must_equal",
            "topic":"keyword",
            "match":[
              "hack",
              "password",
              "root",
              "malware",
              "sell",
              "ransomware"
            ]
          }
        ]
      },
      "created_at":"2022-06-03T19:06:11.086Z",
      "updated_at":"2022-06-03T19:06:11.086Z",
      "email_notify_enabled":false,
      "email_notify_immediate":false
    }
  ]
}
```


The response Headers also include the following `Link` header:

```
1 <https://api.intelligence.mandiant.com/v4/dtm/monitors?page=cad5n8o5cdt77b0idjlg>; rel="next"
```

If we use the `Link` header URI, we can then fetch our next page of results:

```
1 curl -k -s -H 'Accept: application/json' -H "Authorization: Bearer ${DTM_BEARER}" "https://api.intelligence.mandiant.com/v4/dtm/monitors?page=cad5n8o5cdt77b0idjlg"
```

Which returns a list with our other Monitor:


```
{
  "monitors":[
    {
      "id":"cad5l885cdt77b0idivg",
      "name":"monitor mandiant.com",
      "description":"monitor mandiant.com domain for suspicious keywords",
      "enabled":true,
      "doc_condition":{
        "operator":"all",
        "topic":"match_conditions",
        "match":[
          {
            "operator":"must_equal",
            "topic":"group_network",
            "match":[
              "mandiant.com"
            ]
          },
          {
            "operator":"must_contain",
            "topic":"keyword",
            "match":[
              "hack",
              "password",
              "root",
              "malware",
              "sell",
              "ransomware"
            ]
          }
        ]
      },
      "created_at":"2022-06-03T19:01:53.442Z",
      "updated_at":"2022-06-03T19:01:53.442Z",
      "email_notify_enabled":false,
      "email_notify_immediate":false
    }
  ]
}
```


This time the response does not include the `Link` header, because there is not another page of Monitors to scroll through.

Get Alerts Generated by DTM

It is often necessary to check for new alerts created by DTM and inspect those alerts at regular intervals. This process outlines the most effective way to do so using the **DTM Alerts** **(https://docs.mandiant.com/home/digital-threat-monitoring-api#tag/Alerts/operation/get-alerts)** API endpoint, filtering for relevant alerts, paging through those results, and capturing

the relevant details.

Use pagination with alerts

1. The **/alerts endpoint** (https://docs.mandiant.com/home/digital-threat-monitoring-api#tag/Alerts/operation/get-alerts) requires **authentication**, therefore, make sure you have valid authentication credentials when making a GET call to the URL `/alerts` endpoint.
2. Determine a timeboxing window for your alerts. Make sure the date and times provided are in **RFC 3339 format** (https://datatracker.ietf.org/doc/html/rfc3339).
3. Build the proper query parameters and headers to use with making an alerts API call.
   For example, you can use these parameters to affect the response:
   - `refs=true` returns the actual document that generated the alert.
   - `size=25` returns up to 25 results per page in the response.

     When using `refs=true`, `size` must be less than `25`.

   - `sanitize=true` removes dangerous links from documents.

   For a list of all query parameters and their use, refer to **pagination** (https://docs.mandiant.com/home/digital-threat-monitoring-api#tag/Alerts/operation/get-alerts).
4. Make the API call for the alerts, setting the proper timeboxing and pagination query parameters. Then, stepping through the results, the system looks for the existence of a next page's page ID in the Link header and if it exists, queries the next page of results and continue to do so until no more pages exist.

Using this search template, you can create a request for alerts for a given time window and paginate through those results. You can then wait until the next time window has elapsed and repeat the process. This should help prevent running into **rate limiting** (https://docs.mandiant.com/home/dtm-api-limits-and-quotas) issues and reduce the amount of compute processing required when compared to repeatedly requesting alerts without using pagination.

Bulk update alerts that have been read

It may be useful to update the status of alerts once they have been retrieved from the `GET /alerts` endpoint. You can achieve this by using the **bulk update alerts endpoint** (https://docs.mandiant.com/home/digital-threat-monitoring-api#tag/Alerts/operation/post-alerts-bulk).

To do this, make a POST call to `/alerts/bulk` and provide it with a list of objects containing the alert ID and the status to set for that alert ID.

If you set the status to `read`, the status of all the alert IDs provided is set to `read`. In turn, all of those alerts are excluded from the results when calling `/alerts` in the future.