Carbon Black Cloud Python API Documentation

Release 1.0a

Carbon Black Developer Network

Nov 06, 2020

User Guide

CHAPTER 1

In Alpha Development

Release v1.0a.

The Carbon Black Cloud Python SDK provides an easy interface to connect with Carbon Black Cloud products, including Endpoint Standard, Audit and Remediation, and Enterprise EDR. Use this SDK to more easily query and manage your endpoints, manipulate data as Python objects, and harness the full power of Carbon Black Cloud APIs.

CHAPTER 2

Major Features

- **Support For All Carbon Black Cloud Products** Endpoint Standard, Audit and Remediation, and Enterprise EDR
- **Reduced Complexity** The SDK manages the differences among Carbon Black Cloud APIs behind a single, consistent Python interface. Spend less time learning specific API calls, and more time controlling your environment.
- **More Efficient Performance** A built-in caching layer makes repeated access to the same resource more efficient. Instead of making identical API requests repeatedly, the SDK caches the results of the request the first time, and references the cache when you make future requests for the resource. This reduces the time required to access the resource later.

CHAPTER 3

API Credentials

To use the SDK and access data in Carbon Black Cloud, you must set up API keys with the correct permissions. Different APIs have different permission requirements for use, which is explained in the Developer Network Authentication Guide.

The SDK manages your API credentials for you. There are multiple ways to supply the SDK with your API credentials, which is explained in *Authentication*.

CHAPTER 4

User Guide

Get started with Carbon Black Cloud Python SDK here. For detailed information on the objects and methods exposed by Carbon Black Cloud Python SDK, see the full API Documentation below.

4.1 Installation

If you already have Python installed, skip to *Use Pip*.

4.1.1 Install Python

Carbon Black Cloud Python SDK is compatible with Python 3.6+. UNIX systems usually have Python installed by default; it will have to be installed on Windows systems separately.

If you believe you have Python installed already, run the following two commands at a command prompt:

```
$ python --version
Python 3.7.5

$ pip --version
pip 20.2.3 from /usr/local/lib/python3.7/site-packages (python 3.7)
```

If "python –version" reports back a version of 3.6.x or higher, you're all set. If "pip" is not found, follow the instructions on this guide.

If you're on Windows, and Python is not installed yet, download the latest Python installer from python.org.

Ensure that the "Add Python to PATH" option is checked.

4.1.2 Use Pip

Once Python and Pip are installed, open a command prompt and type:

```
$ pip install carbon-black-cloud-sdk
```

This will download and install the latest version of the SDK from the Python PyPI packaging server.

4.1.3 Virtual Environments (optional)

If you are installing the SDK with the intent to contribute to it's development, it is recommended that you use virtual environments to manage multiple installations.

A virtual environment is a Python environment such that the Python interpreter, libraries and scripts installed into it are isolated from those installed in other virtual environments, and (by default) any libraries installed in a "system" Python, i.e., one which is installed as part of your operating system[1].

See the python.org virtual environment guide for more information.

4.1.4 Get Source Code

Carbon Black Cloud Python SDK is actively developed on GitHub and the code is available from the Carbon Black GitHub repository. The version of the SDK on GitHub reflects the latest development version.

To clone the latest version of the SDK repository from GitHub:

```
$ git clone git@github.com:carbonblack/carbon-black-cloud-sdk-python.git
```

Once you have a copy of the source, you can install it in "development" mode into your Python site-packages:

```
$ cd carbon-black-cloud-sdk-python
$ python setup.py develop
```

[1] https://docs.python.org/3/library/venv.html

This will link the version of carbon-black-cloud-sdk-python you cloned into your Python site-packages directory. Any changes you make to the cloned version of the SDK will be reflected in your local Python installation. This is a good choice if you are thinking of changing or further developing carbon-black-cloud-sdk-python.

4.2 Authentication

Carbon Black Cloud APIs require authentication to secure your data.

There are a few methods for authentication listed below. Every method requires an API Key. See the Developer Network Authentication Guide to learn how to generate an API Key.

The SDK only uses one API Key at a time. It is recommeded to create API Keys for specific actions, and use them as needed.

For example, if using the Platform Devices API to search for mission critical devices, and the Endpoint Standard Live Response API to execute commands on those devices, generate two API Keys. The Platform API Key should have the Custom Access Level, and the Live Response Key should have the Live Response Access Level. Store these Keys with profile names, and reference the profile names when creating CBCloudAPI objects.

```
# import relevant modules
>>> from cbc_sdk.platform import Device
>>> from cbc_sdk import CBCloudAPI

# create Platform API object
>>> platform_api = CBCloudAPI(profile='platform')

# create Live Response API object
>>> live_response_api = CBCloudAPI(profile='live_response')

# search for specific devices with Platform Devices API
>>> important_devs = platform_api.select(Device).set_target_priorities("MISSION_
↪CRITICAL")

# execute commands with Live Response API
>>> for device in important_devs:
...       lr_session = live_response_api.live_response.request_session(device.id)
...       lr_session.create_process(r'cmd.exe /c "ping.exe 192.168.1.1"'))
...       lr_session.close()
```

4.2.1 Authentication Methods

With a File:

Credentials may be stored in a `credentials.cbc` file. With support for multiple profiles, this method makes it easy to manage multiple API Keys for different products and permission levels.

```
>>> cbc_api = CBCloudAPI('~/.carbonblack/myfile.cbc', profile='default')
```

With Windows Registry:

Windows Registry is a secure option for storing API credentials on Windows systems.

```
>>> provider = RegistryCredentialProvider()
>>> cbc_api = CBCloudAPI(credential_provider=provider, profile='default')
```

With an External Credential Provider:

Credential Providers allow for custom methods of loading API credentials. This method requires you to write your own Credential Provider.

```
>>> provider = MyCredentialProvider()
>>> cbc_api = CBCloudAPI(credential_provider=provider, profile='default')
```

Not Recommended:

At Runtime:

Credentials may be passed into `CBCloudAPI()` via keyword parameters. This method should be used with caution, taking care to not share your API credentials when managing code with source control.

```
>>> cbc_api = CBCloudAPI(url='defense.conferdeploy.net', token=ABCD/1234,
...                      org_key='ABCDEFGH')
```

Not Recommended:

With Environmental Variables:

Environmental variables can be used for authentication, but pose a security risk. This method is not recommended unless absolutely necessary.

With a File

Credentials may be supplied in a file that resembles a Windows `.INI` file in structure, which allows for multiple "profiles" or sets of credentials to be supplied in a single file. The file format is backwards compatible with CBAPI, so older files can continue to be used. This is an example of a credentials file:

```
[default]
url=http://example.com
token=ABCDEFGHIJKLMNOPQRSTUVWX/12345678
org_key=A1B2C3D4
ssl_verify=false
ssl_verify_hostname=no
ssl_cert_file=foo.certs
ssl_force_tls_1_2=1
proxy=proxy.example
ignore_system_proxy=on
integration_name=MyScript/0.9.0

[production]
url=http://example.com
token=QRSTUVWXYZABCDEFGHIJKLMN/76543210
org_key=A1B2C3D4
ssl_verify=false
ssl_verify_hostname=no
ssl_cert_file=foo.certs
ssl_force_tls_1_2=1
proxy=proxy.example
ignore_system_proxy=on
integration_name=MyApplication/1.3.1
```

Individual profiles or sections are delimited in the file by placing their name within square brackets: `[profile_name]`. Within each section, individual credential values are supplied in a `keyword=value` format.

Unrecognized keywords are ignored.

By default, the CBC SDK looks for credentials files in the following locations:

- The `.carbonblack` subdirectory of the current directory of the running process.
- The `.carbonblack` subdirectory of the user's home directory.
- The `/etc/carbonblack` subdirectory on Unix, or the `C:\Windows\carbonblack` subdirectory on Windows.

Within each of these directories, the SDK first looks for the `credentials.cbc` file, then the `credentials.psc` file (the older name for the credentials file under CBAPI).

You can override the file search logic and specify the full pathname of the credentials file in the keyword parameter `credential_file` when creating the `CBCloudAPI` object.

In all cases, you will have to specify the name of the profile to be retrieved from the credentials file in the keyword parameter `profile` when creating the `CBCloudAPI` object.

Example:

```
>>> cbc_api = CBCloudAPI(credential_file='~/.carbonblack/myfile.cbc', profile='default
↪')
```

Note on File Security: It is recommended that the credentials file be secured properly on Unix. It should be owned by the user running the process, as should the directory containing it, and neither one should specify any file permissions for "group" or "other." In numeric terms, that means the file should have `400` or `600` permissions, and its containing directory should have `500` or `700` permissions. This is similar to securing configuration or key files for `ssh`. If these permissions are incorrect, a warning message will be logged; a future version of the CBC SDK will disallow access to files altogether if they do not have the correct permissions.

Credential files *cannot* be properly secured in this manner under Windows; if they are used in that environment, a warning message will be logged.

With Windows Registry

CBC SDK also provides the ability to use the Windows Registry to supply credentials, a method which is more secure on Windows than other methods.

N.B.: Presently, to use the Windows Registry, you must supply its credential provider as an "external" credential provider. A future version of the CBC SDK will move to using this as a default provider when running on Windows.

By default, registry entries are stored under the key `HKEY_CURRENT_USER\Software\VMware Carbon Black\Cloud Credentials`. Under this key, there may be multiple subkeys, each of which specifies a "profile" (as with credential files). Within these subkeys, the following named values may be specified:

*** Required**

Keyword	Value Type	Default
`url` *	`REG_SZ`	
`token` *	`REG_SZ`	
`org_key` *	`REG_SZ`	
`ssl_verify`	`REG_DWORD`	1
`ssl_verify_hostname`	`REG_DWORD`	1
`ignore_system_proxy`	`REG_DWORD`	0
`ssl_force_tls_1_2`	`REG_DWORD`	0
`ssl_cert_file`	`REG_SZ`	
`proxy`	`REG_SZ`	
`integration_name`	`REG_SZ`	

Unrecognized named values are ignored.

To use the Registry credential provider, create an instance of it, then pass the reference to that instance in the `credential_provider` keyword parameter when creating `CBCloudAPI`. As with credential files, the name of the profile to be retrieved from the Registry should be specified in the keyword parameter `profile`.

Example:

```
>>> provider = RegistryCredentialProvider()
>>> cbc_api = CBCloudAPI(credential_provider=provider, profile='default')
```

Advanced Usage: The parameters `keypath` and `userkey` to `RegistryCredentialProvider` may be used to control the exact location of the "base" registry key where the sections of credentials are located. The `keypath` parameter allows specification of the path from `HKEY_CURRENT_USER` where the base registry key is located. If `userkey`, which is `True` by default, is `False`, the path will be interpreted as being rooted at `HKEY_LOCAL_MACHINE` rather than `HKEY_CURRENT_USER`.

Example:

```
>>> provider = RegistryCredentialProvider('Software\\Contoso\\My CBC Application')
>>> cbc_api = CBCloudAPI(credential_provider=provider, profile='default')
```

Note the use of doubled backslashes to properly escape them under Python.

With an External Credential Provider

Credentials may also be supplied by writing a class that conforms to the `CredentialProvider` interface protocol. When creating `CBCloudAPI`, pass a reference to a `CredentialProvider` object in the `credential_provider` keyword parameter. Then pass the name of the profile you want to retrieve from the provider object using the keyword parameter `profile`.

Example:

```
>>> provider = MyCredentialProvider()
>>> cbc_api = CBCloudAPI(credential_provider=provider, profile='default')
```

Details of writing a credential provider may be found in the *Developing a Custom Credential Provider* document.

At Runtime

The credentials may be passed into the `CBCloudAPI` object when it is created via the keyword parameters `url`, `token`, `org_key`, and (optionally) `ssl_verify` and `integration_name`.

Example:

```
>>> api = CBCloudAPI(url='https://example.com', token='ABCDEFGHIJKLMNOPQRSTUVWX/
↪12345678',
...                   org_key='A1B2C3D4', ssl_verify=False, integration_name='MyScript/
↪1.0')
```

The `integration_name` may be specified even if using another credential provider. If specified as a parameter, this overrides any integration name specified by means of the credential provider.

With Environmental Variables

The credentials may be supplied to CBC SDK via the environment variables `CBC_URL`, `CBC_TOKEN`, `CBC_ORG_KEY`, and `CBC_SSL_VERIFY`. For backwards compatibility with CBAPI, the environment variables

`CBAPI_URL`, `CBAPI_TOKEN`, `CBAPI_ORG_KEY`, and `CBAPI_SSL_VERIFY` may also be used; if both are specified, the newer `CBC_xxx` environment variables override their corresponding `CBAPI_xxx` equivalents. To use the environment variables, they must be set before the application is run (at least `CBC_URL` or `CBAPI_URL`, and `CBC_TOKEN` or `CBAPI_TOKEN`), and the `credential_file` keyword parameter to `CBCloudAPI` must be either `None` or left unspecified. (The `profile` keyword parameter will be ignored.)

N.B.: Passing credentials via the environment can be insecure, and, if this method is used, a warning message to that effect will be generated in the log.

4.2.2 Explanation of API Credential Components

When supplying API credentials to the SDK *at runtime*, *with a file*, or *with Windows Registry*, the credentials include these components:

*** Required**

Keyword	Definition	Default
`url` *	The URL used to access the Carbon Black Cloud.	
`token` *	The access token to authenticate with. Same structure as `X-Auth-Token` defined in the Developer Network Authentication Guide. Derived from an API Key's Secret Key and API ID.	
`org_key` *	The organization key specifying which organization to work with.	
`ssl_verify`	A Boolean value (see below) indicating whether or not to validate the SSL connection.	`True`
`ssl_verify_hostname`	A Boolean value (see below) indicating whether or not to verify the host name of the server being connected to.	`True`
`ignore_system_proxy`	A Boolean value (see below). If this is `True`, any system proxy settings will be ignored in making the connection to the server.	`False`
`ssl_force_tls_1_2`	A Boolean value (see below). If this is `True`, the connection will be forced to use TLS 1.2 rather than any later version.	`False`
`ssl_cert_file`	The name of an optional certificate file used to validate the certificates of the SSL connection. If not specified, the standard system certificate verification will be used.	
`proxy`	If specified, this is the name of a proxy host to be used in making the connection.	
`integration_name`	The name of the integration to use these credentials. The string may optionally end with a slash character, followed by the integration's version number. Passed as part of the `User-Agent:` HTTP header on all requests made by the SDK.	

When supplying API credentials to the SDK *with environmental variables*, the credentials include these components:

Alternative keywords are available to maintain backwards compatibility with CBAPI.

Boolean Values

Boolean values are specified by using the strings `true`, `yes`, `on`, or `1` to represent a `True` value, or the strings `false`, `no`, `off`, or `0` to represent a `False` value. All of these are case-insensitive. Any other string value specified will result in an error.

For example, to disable SSL connection validation, any of the following would work:

```
ssl_verify=False
ssl_verify=false
ssl_verify=No
```

(continues on next page)

(continued from previous page)

```
ssl_verify=no
ssl_verify=Off
ssl_verify=off
ssl_verify=0
```

4.3 Getting Started with the Carbon Black Cloud Python SDK - "Hello CBC"

This document will help you get started with the Carbon Black Cloud Python SDK by installing it, configuring authentication for it, and executing a simple example program that makes one API call.

4.3.1 Installation

Make sure you are using Python 3. Use the command `pip install carbon_black_cloud_sdk` to install the SDK and all its dependencies. (In some environments, the correct command will be `pip3 install carbon_black_cloud_sdk` to use Python 3.)

You can also access the SDK in development mode by cloning the GitHub repository, and then executing `python setup.py develop` (in some environments, `python3 setup.py develop`) from the top-level directory. Setting your `PYTHONPATH` environment variable to the directory `[sdk]/src`, where `[sdk]` is the top-level directory of the SDK, will also work for these purposes. (On Windows, use `[sdk]\src`.)

See also the *Installation* section of this documentation for more information.

4.3.2 Authentication

In order to make use of the API, you will need an *API token,* which you will get from the Carbon Black Cloud UI. For the purposes of our example, we will need a custom key with the ability to list devices.

Log into the Carbon Black Cloud UI and go to `Settings > API Access`. Start by selecting `Access Levels` at the top of the screen and press `Add Access Level`. Fill in a name and description for your sample access level, keep `Copy permissions from` set to `None`, and, under the permission category `Device` and permission name `General information`, check the `Read` check box. Press `Save` to save and create the new access level.

Now select `API Keys` at the top of the screen and press `Add API Key`. Enter a name for the key, and, optionally, a description. For `Access Level type`, select `Custom`, and for `Custom Access Level`, select the access level you created above. Press `Save` to save and create the new API key. An `API Credentials` dialog will be displayed with the new API ID and secret key; this dialog may also be re-displayed at any time by finding the API key in the list, clicking the drop-down arrow under the `Actions` column, and selecting `API Credentials`.

We will use a credentials file to store the credential information by default. Create a directory named `.carbonblack` under your user home directory. (On Windows, this directory is generally `C:\Users\[username]`, where `[username]` is your user name.) Within this directory create a file `credentials.cbc` to store your credentials. Copy the following template to this new file:

```
[default]
url=
token=
org_key=
ssl_verify=True
```

Following the `url=` keyword, add the top-level URL you use to access the Carbon Black Cloud, including the `https://` prefix and the domain name, but without any of the path information following it.

Following the `token=` keyword, add the `API Secret Key` from the `API Credentials` dialog, followed by a forward slash (/) character, followed by the `API ID` from the `API Credentials` dialog. (The secret key is always 24 characters in length, and the API ID is always 10 characters in length.)

Following the `org_key=` keyword, add the organization key from your organization, which may be seen under the `Org Key:` heading at the top of the `API Keys` display under `Settings > API Access`. It is always 8 characters in length.

Save the completed `credentials.cbc` file, which should look like this *(example text only)*:

```
[default]
url=https://example.net
token=ABCDEFGHGIJKLMNOPQRSTUVWX/ABCDEFGHIJ
org_key=A1B2C3D4
ssl_verify=True
```

On UNIX systems, you must make sure that the `credentials.cbc` file is properly secured. The simplest commands for doing so are:

```
$ chmod 600 ~/.carbonblack/credentials.cbc
$ chmod 700 ~/.carbonblack
```

For further information, please see the *Authentication* section of the documentation, as well as the Authentication Guide on the Carbon Black Cloud Developer Network.

4.3.3 Running the Example

The example we will be running is `list_devices.py`, located in the `examples/platform` subdirectory of the GitHub repository. If you cloned the repository, change directory to `[sdk]/examples/platform`, where `[sdk]` is the top-level directory of the SDK. (On Windows, use `[sdk]\examples\platform`.) Alternately, you may view the current version of that script in "raw" mode in GitHub, and use your browser's `Save As` function to save the script locally. In that case, change directory to whichever directory you saved the script to.

Execute the script by using the command `python list_devices.py -q '1'` (in some environments, `python3 list_devices.py -q '1'`). If all is well, you will see a list of devices (endpoints) registered in your organization, showing their numeric ID, host name, IP address, and last checkin time.

You can change what devices are shown by modifying the query value supplied to the `-q` parameter, and also by using additional parameters to modify the search criteria. Execute the command `python list_devices.py --help` (in some environments, `python3 list_devices.py --help`) for a list of all possible command line parameters.

4.3.4 Inside the Example Script

Once the command-line arguments are parsed, we create a Carbon Black Cloud API object with a call to the helper function `get_cb_cloud_object()`. The standard `select()` method is used to create a query object that queries for devices; the query string is passed to that object via the `where()` method, and other criteria are added using specific setters.

The query is an iterable object, and calling upon its iterator methods invokes the query, which, in this case, is the Search Devices API. The example script turns those results into an in-memory list, then iterates on that list, printing only certain properties of each retrieved Device object.

4.3.5 Calling the SDK Directly

Now we'll repeat this example, but using the Python command line directly without a script.

Access your Python interpreter with the `python` command (or `python3` if required) and type:

```
>>> from cbc_sdk.rest_api import CBCloudAPI
>>> from cbc_sdk.platform import Device
>>> cb = CBCloudAPI(profile='default')
```

This imports the necessary classes and creates an instance of the base `CBCloudAPI` object. By default, the file credentials provider is used. We set it to use the `default` profile in your `credentials.cbc` file, which you set up earlier.

N.B.: On Windows, a security warning message will be generated about file access to CBC SDK credentials being inherently insecure.

```
>>> query = cb.select(Device).where('1')
```

This creates a query object that searches for all devices (the '1' causes all devices to be matched, as in SQL).

```
>>> devices = list(query)
```

For convenience, we load the entirety of the query results into an in-memory list.

```
>>> for device in devices:
...     print(device.id, device.name, device.last_internal_ip_address, device.last_
↪contact_time)
...
```

Using a simple `for` loop, we print out the ID, host name, internal IP address, and last contact time from each returned device. Note that the contents of the list are `Device` objects, not dictionaries, so we access individual properties with the `object.property_name` syntax, rather than `object['property_name']`.

4.4 Concepts

4.4.1 Platform Devices vs Endpoint Standard Devices

For most use cases, Platform Devices are sufficient to access information about devices and change that information. If you want to connect to a device using Live Response, then you must use Endpoint Standard Devices and a Live Response API Key.

```
# Device information is accessible with Platform Devices
>>> api = CBCloudAPI(profile='platform')
>>> platform_devices = api.select(platform.Device).set_os(["WINDOWS", "LINUX"])
>>> for device in platform_devices:
...   print(
      f'''
      Device ID: {device.id}
      Device Name: {device.name}

      '''
Device ID: 1234
Device Name: Win10x64
```

(continues on next page)

(continued from previous page)

```
Device ID: 5678
Device Name: UbuntuDev


# Live Response is accessible with Endpoint Standard Devices
>>> api = CBCloudAPI(profile='live_response')
>>> endpoint_standard_device = api.select(endpoint_standard.Device, 1234)
>>> endpoint_standard_device.lr_session()
url: /integrationServices/v3/cblr/session/428:1234 -> status: PENDING
[...]
```

4.4.2 Queries

Generally, to retrieve information from your Carbon Black Cloud instance you will:

1. *Create a Query*
2. *Refine the Query*
3. *Execute the Query*

Create Queries with `CBCloudAPI.select()`

Data is retrieved from the Carbon Black Cloud with `CBCloudAPI.select()` statements. A `select()` statement creates a `query`, which can be further *refined with parameters or criteria*, and then *executed*.

```
# Create a query for devices
>>> device_query = api.select(platform.Device).where('avStatus:AV_ACTIVE')

# The query has not yet been executed
>>> type(device_query)
<class cbc_sdk.platform.devices.DeviceSearchQuery>
```

This query will search for Platform Devices with antivirus active.

Refine Queries with `where()`, `and_()`, and `or_()`

Queries can be refined during or after declaration with `where()`, `and_()`, and `or_()`.

```
# Create a query for events
>>> event_query = api.select(endpoint_standard.Event).where(hostName='Win10').and_
↪(ipAddress='10.0.0.1')

# Refine the query
>>> event_query.and_(applicationName='googleupdate.exe')
>>> event_query.and_(eventType='REGISTRY_ACCESS')
>>> event_query.and_(ownerNameExact='DevRel')
```

This query will search for Endpoint Standard Events created by the application `googleupdate.exe` accessing the registry on a device with a hostname containing `Win10`, an IP Address of `10.0.0.1`, and owned by `DevRel`.

Be Consistent When Refining Queries

All queries are of type `QueryBuilder()`, with support for either raw string-based queries , or keyword arguments.

```
# Equivalent queries
>>> string_query = api.select(platform.Device).where("avStatus:AV_ACTIVE")
>>> keyword_query = api.select(platform.Device).where(avStatus="AV_ACTIVE").
```

Queries must be consistent in their use of strings or keywords; do not mix strings and keywords.

```
# Not allowed
>>> mixed_query = api.select(platform.Device).where(avStatus='Win7x').and_(
↪"virtualMachine:true")
cbc_sdk.errors.ApiError: Cannot modify a structured query with a raw parameter
```

Execute a Query

A query is not executed on the server until it's accessed, either as an iterator (where it will generate results on demand as they're requested) or as a list (where it will retrieve the entire result set and save to a list).

```
# Create and Refine a query
>>> device_query = api.select(platform.Device).where('avStatus:AV_ACTIVE').set_os([
↪"WINDOWS"])

# Execute the query by accessing as a list
>>> matching_devices = [device for device in device_query]

>>> print(f"First matching device ID: {matching_devices[0].id}")
First matching device ID: 1234

# Or as an iterator
>>> for matching_device in device_query:
...   print(f"Matching device ID: {matching_device.id})
Matching device ID: 1234
Matching device ID: 5678
```

You can also call the Python built-in `len()` on this object to retrieve the total number of items matching the query.

```
# Retrieve total number of matching devices
>>> len(device_query)
2
```

In this example, the matching device ID's are accessed with `device.id`. If using Endpoint Standard Devices, the device ID's are accessed with `device.deviceId`.

Query Parameters vs Criteria

For queries, some Carbon Black Cloud APIs use `GET` requests with parameters, and some use `POST` requests with criteria.

Parameters

Parameters modify a query. When modifying a query with `where()`, `and_()`, and `or_()`, those modifications become query parameters when sent to Carbon Black Cloud.

```
>>> device_query = api.select(endpoint_standard.Device).where(hostName='Win7').and_
↪(ipAddress='10.0.0.1')
```

Executing this query results in an API call similar to `GET /integrationServices/v3/device?hostName='Win7'&ipAddress='10.0.0.1'`

Criteria

Criteria also modify a query, and can be used with or without parameters. When using CBC SDK, there are API-specific methods you can use to add criteria to queries.

```
# Create a query for alerts
>>> alert_query = api.select(cbc_sdk.Platform.Alert)

# Refine the query with parameters
>>> alert_query.where(alert_severity=9).or_(alert_severity=10)

# Refine the query with criteria
>>> alert_query.set_device_os(["MAC"]).set_device_os_versions(["10.14.6"])
```

Executing this query results in an API call to `POST /appservices/v6/orgs/{org_key}/alerts/_search` with this JSON Request Body:

```
{
  "query": "alert_severity:9 OR alert_severity:10",
  "criteria": {
    "device_os": ["MAC"],
    "device_os_version": ["10.14.6"]
  }
}
```

The query parameters are sent in `"query"`, and the criteria are sent in `"criteria"`.

Modules with Support for Criteria

Run

- `cbc_sdk.audit_remediation.base.RunQuery.device_ids()`
- `cbc_sdk.audit_remediation.base.RunQuery.device_types()`
- `cbc_sdk.audit_remediation.base.RunQuery.policy_id()`

`Result` and `Device Summary`

- `cbc_sdk.audit_remediation.base.ResultQuery.set_device_ids()`
- `cbc_sdk.audit_remediation.base.ResultQuery.set_device_names()`
- `cbc_sdk.audit_remediation.base.ResultQuery.set_device_os()`
- `cbc_sdk.audit_remediation.base.ResultQuery.set_policy_ids()`
- `cbc_sdk.audit_remediation.base.ResultQuery.set_policy_names()`
- `cbc_sdk.audit_remediation.base.ResultQuery.set_status()`

`ResultFacet` and `DeviceSummaryFacet`

- cbc_sdk.audit_remediation.base.FacetQuery.set_device_ids()
- cbc_sdk.audit_remediation.base.FacetQuery.set_device_names()
- cbc_sdk.audit_remediation.base.FacetQuery.set_device_os()
- cbc_sdk.audit_remediation.base.FacetQuery.set_policy_ids()
- cbc_sdk.audit_remediation.base.FacetQuery.set_policy_names()
- cbc_sdk.audit_remediation.base.FacetQuery.set_status()

Alert

- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_categories()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_create_time()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_device_ids()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_device_names()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_device_os()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_device_os_versions()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_device_username()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_group_results()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_alert_ids()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_legacy_alert_ids()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_minimum_severity()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_policy_ids()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_policy_names()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_process_names()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_process_sha256()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_reputations()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_tags()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_target_priorities()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_threat_ids()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_types()
- cbc_sdk.platform.alerts.BaseAlertSearchQuery.set_workflows()

WatchlistAlert

- cbc_sdk.platform.alerts.WatchlistAlertSearchQuery.set_watchlist_ids()
- cbc_sdk.platform.alerts.WatchlistAlertSearchQuery.set_watchlist_names()

CBAnalyticsAlert

- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_blocked_threat_categories()
- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_device_locations()
- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_kill_chain_statuses()
- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_not_blocked_threat_categories(

- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_policy_applied()
- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_reason_code()
- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_run_states()
- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_sensor_actions()
- cbc_sdk.platform.alerts.CBAnalyticsAlertSearchQuery.set_threat_cause_vectors()

VMwareAlert

- cbc_sdk.platform.alerts.VMwareAlertSearchQuery.set_group_ids()

Modules not yet Supported for Criteria

RunHistory Event Process

4.5 Porting Applications from CBAPI to Carbon Black Cloud SDK

Applications using the Carbon Black Cloud via the CBAPI can be ported to use the Carbon Black Cloud SDK. CBAPI applications using CB Protection or CB Response cannot be ported, as support for on-premise products is not present in the CBC SDK.

4.5.1 Import Changes

A number of packages have new name equivalents in the CBC SDK.

- Package cbapi.example_helpers -> cbc_sdk.helpers
- Package cbapi.psc -> cbc_sdk
- Package cbapi.psc.alerts_query -> cbc_sdk.platform
- Package cbapi.psc.devices_query -> cbc_sdk.platform
- Package cbapi.psc.livequery -> cbc_sdk.audit_remediation
- Package cbapi.psc.livequery.models -> cbc_sdk.audit_remediation
- Package cbapi.psc.defense -> cbc_sdk.endpoint_standard
- Package cbapi.psc.defense.models -> cbc_sdk.endpoint_standard
- Package cbapi.psc.threathunter -> cbc_sdk.enterprise_edr
- Package cbapi.psc.threathunter.models -> cbc_sdk.enterprise_edr

4.5.2 Code Changes

Helper Functions: Replace all calls to get_cb_defense_object(), get_cb_livequery_object(), get_cb_psc_object(), and get_cb_threathunter_object() with get_cb_cloud_object().

Audit/Remediation Queries:

- Replace cb.query(sql_query) with cb.select(Run).where(sql=sql_query).
- Replace cb.query_history(query_string) with cb.select(RunHistory).where(query_string).

- On a query object, use the `policy_id()` method instead of `policy_ids()`. Only one policy ID can be specified.

Base API Object

The different API objects, `CbDefenseAPI`, `CbLiveQueryAPI`, `CbPSCBaseAPI`, and `CbThreatHunterAPI` are replaced with `CBCloudAPI`. Import this object from the `cbc_sdk` package, i.e. `from cbc_sdk import CBCloudAPI`.

4.6 Logging & Diagnostics

4.7 Changelog

CHAPTER 5

SDK Documentation

5.1 Audit and Remediation

5.1.1 Submodules

5.1.2 cbc_sdk.audit_remediation.base module

5.1.3 Module contents

5.2 Credential Providers

5.2.1 Submodules

5.2.2 cbc_sdk.credential_providers.default module

5.2.3 cbc_sdk.credential_providers.environ_credential_provider module

5.2.4 cbc_sdk.credential_providers.file_credential_provider module

5.2.5 cbc_sdk.credential_providers.registry_credential_provider module

5.2.6 Module contents

5.3 Developing New Credential Providers

The credentials management framework for the CBC SDK is designed to allow different handlers to be implemented, which may supply credentials to the `CBCloudAPI` in ways not implemented by existing credential handlers.

5.3.1 Writing the Credential Provider

Find all classes required to implement a new credential provider in the `cbc_sdk.credentials` package. See below for descriptions of the classes. It is recommended, but not required, that your new credential provider inherit from the `CredentialProvider` abstract class, and that you implement the methods from that abstract class as detailed.

The arguments to the standard `__init__()` method are not defined by the interface specification; those may be used to initialize your credential provider in any desired fashion.

5.3.2 Using the Credential Provider

Create an instance of your credential provider object and pass it as the keyword parameter `credential_provider` when creating your `CBCloudAPI` object. Example:

```
>>> provider = MyCredentialProvider()
>>> cbc_api = CBCloudAPI(credential_provider=provider, profile='default')
```

Your credential provider's `get_credentials()` method will be called, passing in any profile specified in the `profile` keyword parameter used when creating `CBCloudAPI`.

5.3.3 Credential Provider Reference

These are the classes from the `cbc_sdk.credentials` package that are used in making a credential provider.

CredentialValue class

This class is of an enumerated type, and represents the various credential items loaded by the credential provider and fed to the rest of the SDK code. The possible values are:

- `URL` - The URL used to access the Carbon Black Cloud. This value *must* be specified.
- `TOKEN` - The access token to be used to authenticate to the server. It is the same structure as the `X-Auth-Token:` defined for direct API access in the developer documentation. This value *must* be specified.
- `ORG_KEY` - The organization key specifying which organization to work with. This value *must* be specified.
- `SSL_VERIFY` - A Boolean value indicating whether or not to validate the SSL connection. The default is `True`.
- `SSL_VERIFY_HOSTNAME` - A Boolean value indicating whether or not to verify the host name of the server being connected to. The default is `True`.
- `SSL_CERT_FILE` - The name of an optional certificate file used to validate the certificates of the SSL connection. If not specified, the standard system certificate verification will be used.
- `SSL_FORCE_TLS_1_2` - A Boolean value. If this is `True`, the connection will be forced to use TLS 1.2 rather than any later version. The default is `False`.
- `PROXY` - If specified, this is the name of a proxy host to be used in making the connection.
- `IGNORE_SYSTEM_PROXY` - A Boolean value. If this is `True`, any system proxy settings will be ignored in making the connection to the server. The default is `False`.

- `INTEGRATION` - The name of the integration to use these credentials. The string may optionally end with a slash character, followed by the integration's version number. Passed as part of the `User-Agent:` HTTP header on all requests made by the SDK.

Values of this type have one method:

requires_boolean_value

```
def requires_boolean_value(self):
```

Returns whether or not this particular credential item takes a Boolean value.

Returns: `True` if the credential item takes a Boolean value, `False` if the credential item takes a string value.

Credentials class

The class that holds credentials retrieved from the credential provider, and is used by the rest of the SDK. It is effectively immutable after creation.

__init__

```
def __init__(self, values=None):
```

Initializes a new `Credentials` object.

Parameters:

- `values` (type `dict`): A dictionary containing the values to initialize the `Credentials` object with. The keys of this dictionary may be either `CredentialValue` objects or their lowercase string equivalents, e.g. `CredentialValue.URL` or `"url"`. The values in the dict are strings for those credential items with string values. For credential items with Boolean values, the values may be either `bool` values, numeric values (with 0 being treated as `False` and non-zero values treated as `True`), or string values. In the case of string values, the value must be "0", "false", "off", or "no" to be treated as a `False` falue, or "1", "true", "on", or "yes" to be treated as a `True` value (all values case-insensitive). If an unrecognized string is used for a Boolean value, `CredentialError` will be raised. Unrecognized keys in the dict are ignored. Any missing items will be replaced by the default for that item.

Raises:

- `CredentialError` - If there is an error parsing a Boolean value string.

get_value

```
def get_value(self, key):
```

Retrieves a specific credential value from this object.

Parameters:

- `key` (type `CredentialValue`): Indicates which item to retrieve.

Returns: The value of that credential item (`str` or `bool` type).

__getattr__

```
def __getattr__(self, name):
```

Retrieves a specific credential value from this object. This is a bit of "syntactic sugar" allowing other code to access credential values, for instance, as `cred_object.url` instead of `cred_object.get_value(CredentialValue.URL)`.

Parameters:

- `name` (type `str`): Indicates which item to retrieve.

Returns: The value of that credential item (`str` or `bool` type).

Raises:

- `AttributeError` - If the credential item `name` was unrecognized.

CredentialProvider class

All credential providers *should* extend this abstract class, but, in any event, *must* implement the protocol it defines.

get_credentials

```
def get_credentials(self, section=None):
```

Return a Credentials object containing the configured credentials.

Parameters:

- `section` (type `str`): Indicates the credential section to retrieve. May be interpreted by the credential provider in amy manner it likes; may also be ignored.

Returns: A `Credentials` object containing the retrieved credentials.

Raises:

- `CredentialError` - If there is an error retrieving the credentials.

5.4 Endpoint Standard

5.4.1 Submodules

5.4.2 cbc_sdk.endpoint_standard.base module

5.4.3 Module contents

5.5 Enterprise EDR

5.5.1 Submodules

5.5.2 cbc_sdk.enterprise_edr.base module

5.5.3 cbc_sdk.enterprise_edr.threat_intelligence module

5.5.4 cbc_sdk.enterprise_edr.ubs module

5.5.5 Module contents

5.6 Platform

5.6.1 Submodules

5.6.2 cbc_sdk.platform.alerts module

5.6.3 cbc_sdk.platform.base module

5.6.4 cbc_sdk.platform.devices module

5.6.5 Module contents

5.7 CBC SDK

5.7.1 Subpackages

cbc_sdk.cache package

Submodules

cbc_sdk.cache.lru module

Module contents

5.7.2 Submodules

5.7.3 cbc_sdk.base module

5.7.4 cbc_sdk.connection module

5.7.5 cbc_sdk.credentials module

5.7.6 cbc_sdk.errors module

5.7.7 cbc_sdk.example_helpers module

5.7.8 cbc_sdk.live_response_api module

5.7.9 cbc_sdk.rest_api module

5.7.10 cbc_sdk.utils module

5.7.11 cbc_sdk.winerror module

5.7.12 Module contents

5.8 Exceptions

If an error occurs, the API attempts to roll the error into an appropriate Exception class.

5.8.1 Exception Classes

CHAPTER 6

Indices and tables

- genindex
- modindex
- search