cbapi Documentation

Release 1.7.6

Carbon Black Developer Network

Dec 20, 2021

Contents

Release v1.7.6.

CBAPI provides a straightforward interface to the VMware Carbon Black products: Carbon Black EDR, Carbon Black App Control, and Carbon Black Cloud Endpoint Standard(formerly CB Response, CB Protection, and CB Defense). This library provides a Pythonic layer to access the raw power of the REST APIs of all Carbon Black products, making it easier to query data from any platform or on-premise APIs, combine data from multiple API calls, manage all API credentials in one place, and manipulate data as Python objects. Take a look:

```
>>> from cbapi.response import CbResponseAPI, Process, Binary, Sensor
>>> #
>>> # Create our EDR API object
>>> #
>>> c = CbResponseAPI()
>>> #
>>> # take the first process that ran notepad.exe, download the binary and read the␣
↪first two bytes
>>> #
>>> c.select(Process).where('process_name:notepad.exe').first().binary.file.read(2)
'MZ'
>>> #
>>> # if you want a specific ID, you can put it straight into the .select() call:
>>> #
>>> binary = c.select(Binary, "24DA05ADE2A978E199875DA0D859E7EB")
>>> #
>>> # select all sensors that have ran notepad
>>> #
>>> sensors = set()
>>> for proc in c.select(Process).where('process_name:evil.exe'):
...     sensors.add(proc.sensor)
>>> #
>>> # iterate over all sensors and isolate
>>> #
>>> for s in sensors:
...     s.network_isolation_enabled = True
...     s.save()
```

If you're a Carbon Black App Control customer (formerly CB Protection), you may use:

```
>>> from cbapi.protection.models import FileInstance
>>> from cbapi.protection import CbProtectionAPI
>>> #
>>> # Create our App Control API object
>>> #
>>> p = CbProtectionAPI()
>>> #
>>> # Select the first file instance
>>> #
>>> fi = p.select(FileInstance).first()
>>> #
>>> # print that computer's hostname. This automatically "joins" with the Computer␣
↪API object.
>>> #
>>> fi.computer.name
u'DOMAIN\\MYHOSTNAME'
>>> #
>>> # change the policy ID
>>> #
>>> fi.computer.policyId = 3
```

(continues on next page)

(continued from previous page)

```
>>> fi.computer.save()
```

As of version 1.2, CBAPI also supports Carbon Black Cloud Endpoint Standard (formerly CB Defense):

```
>>> from cbapi.psc.defense import *
>>> #
>>> # Create our Cloud Endpoint Standard API object
>>> #
>>> p = CbDefenseAPI()
>>> #
>>> # Select any devices that have the hostname WIN-IA9NQ1GN8OI and an internal IP␣
↪address of 192.168.215.150
>>> #
>>> devices = c.select(Device).where('hostNameExact:WIN-IA9NQ1GN8OI').and_(
↪"ipAddress:192.168.215.150").first()
>>> #
>>> # Change those devices' policy into the Windows_Restrictive_Workstation policy.
>>> #
>>> for dev in devices:
>>>     dev.policyName = "Restrictive_Windows_Workstation"
>>>     dev.save()
```

CHAPTER 1

Major Features

- **Enhanced Live Response API** The new CBAPI now provides a robust interface to the Carbon Black EDR Live Response capability. Easily create Live Response sessions, initiate commands on remote hosts, and pull down data as necessary to make your Incident Response process much more efficient and automated.
- **Consistent API across VMware Carbon Black platforms** CBAPI supports Carbon Black EDR, Carbon Black App Control, and Carbon Black Cloud Endpoint Standard customers from a single API layer. Even better, the object model is the same for all three, and if you know one API, you can easily transition to another. CBAPI manages the differences among the three REST APIs behind a single, consistent Python-like interface.
- **Enhanced Performance** CBAPI now provides a built in caching layer to reduce the query load on the Carbon Black server. This is especially useful when taking advantage of CBAPI's new "joining" features. You can transparently access, for example, the binary associated with a given process in Carbon Black EDR. Since many processes may be associated with the same binary, it does not make sense to repeatedly request the same binary information from the server over and over again. Therefore CBAPI now caches this information to avoid unnecessary requests.
- **Reduce Complexity** CBAPI provides a friendly interface for accessing Carbon Black data. This greatly improves developer productivity and lowers the bar to entry.
- **Python 3 and Python 2 compatible** Use all the new features and modules available in Python 3 with CBAPI. This module is compatible with Python versions 2.6.6 and above, 2.7.x, 3.4.x, and 3.5.x.
- **Better support for multiple CB servers** CBAPI introduces the concept of Credential Profiles; named collections of URL, API keys, and optional proxy configuration for connecting to any number of Carbon Black EDR, Carbon Black App Control, or Carbon Black Cloud Endpoint Standard servers.

CHAPTER 2

API Credentials

CBAPI version 0.9.0 enforces the use of credential files.

In order to perform any queries via the API, you will need to get the API token for your CB user. See the documentation on the Developer Network website on how to acquire the API token for Carbon Black EDR (CB Response), Carbon Black App Control (CB Protection), or Carbon Black Cloud Endpoint Standard (CB Defense).

Once you acquire your API token, place it in one of the default credentials file locations:

- `/etc/carbonblack/`
- `~/.carbonblack/`
- `/current_working_directory/.carbonblack/`

For distinction between credentials of different Carbon Black products, use the following naming convention for your credentials files:

- `credentials.psc` for Carbon Black Cloud Endpoint Standard, Audit & Remediation, and Enterprise EDR (CB Defense, CB LiveOps, and CB ThreatHunter)
- `credentials.response` for Carbon Black EDR (CB Response)
- `credentials.protection` for Carbon Black App Control (CB Protection)

For example, if you use a Carbon Black Cloud product, you should have created a credentials file in one of these locations:

- `/etc/carbonblack/credentials.psc`
- `~/.carbonblack/credentials.psc`
- `/current_working_directory/.carbonblack/credentials.psc`

Credentials found in a later path will overwrite earlier ones.

The credentials are stored in INI format. The name of each credential profile is enclosed in square brackets, followed by key-value pairs providing the necessary credential information:

```
[default]
url=https://localhost
token=abcdef0123456789abcdef
ssl_verify=False

[prod]
url=https://cbserver.prod.corp.com
token=aaaaaa
ssl_verify=True

[otheruser]
url=https://localhost
token=bbbbbb
ssl_verify=False
```

The possible options for each credential profile are:

- **url**: The base URL of the Carbon Black server. This should include the protocol (https) and the hostname, and nothing else.
- **token**: The API token for the user ID. More than one credential profile can be specified for a given server, with different tokens for each.
- **ssl_verify**: True or False; controls whether the SSL/TLS certificate presented by the server is validated against the local trusted CA store.
- **org_key**: The organization key. This is required to access the Carbon Black Cloud, and can be found in the console. The format is `123ABC45`.
- **proxy**: A proxy specification that will be used when connecting to the Carbon Black server. The format is: `http://myusername:mypassword@proxy.company.com:8001/` where the hostname of the proxy is `proxy.company.com`, port 8001, and using username/password `myusername` and `mypassword` respectively.
- **ignore_system_proxy**: If you have a system-wide proxy specified, setting this to True will force CBAPI to bypass the proxy and directly connect to the Carbon Black server.

Future versions of CBAPI will also provide the ability to "pin" the TLS certificate so as to provide certificate verification on self-signed or internal CA signed certificates.

Environment Variable Support

The latest CBAPI for Python supports specifying API credentials in the following three environment variables:

CBAPI_TOKEN the envar for holding the EDR (CbR) or App Control (CbP) api token or the ConnectorId/APIKEY combination for Endpoint Standard (CB Defense)/Carbon Black Cloud.

The *CBAPI_URL* envar holds the FQDN of the target, an EDR (CbR), CBD, or CbD/Carbon Black Cloud server specified just as they are in the configuration file format specified above.

The optional *CBAPI_SSL_VERIFY* envar can be used to control SSL validation(True/False or 0/1), which will default to ON when not explicitly set by the user.

For environments where complex outbound network filters and proxy configurations are used (eg. anything other than an unauthenticated or basic password authenticated proxy) a prepared *requests.Session* object may be supplied as a *proxy_session* parameter. This session will then be used for all communication with the API. Construction of such a *Session* is beyond the scope of this document, consult your local network/security administrators for assistance.

CHAPTER 3

Backwards & Forwards Compatibility

The previous versions (0.8.x and earlier) of CBAPI and bit9Api are now deprecated and will no longer receive updates. However, existing scripts will work without change as CBAPI includes both in its legacy package. The legacy package is imported by default and placed in the top level CBAPI namespace when the CBAPI module is imported on a Python 2.x interpreter. Therefore, scripts that expect to import cbapi.CbApi will continue to work exactly as they had previously.

Since the old API was not compatible with Python 3, the legacy package is not importable in Python 3.x and therefore legacy scripts cannot run under Python 3.

Once CBAPI 1.0.0 is released, the old `cbapi.legacy.CbApi` will be deprecated and removed entirely no earlier than January 2017. New scripts should use the *`cbapi.response.rest_api.CbResponseAPI`* (for Carbon Black EDR (CB Response)), *`cbapi.protection.rest_api.CbProtectionAPI`* (for Carbon Black App Control (CB Protection)), or `cbapi.defense.rest_api.CbDefenseAPI` API entry points.

The API is frozen as of version 1.0; any changes in the 1.x version branch will be additions/bug fixes only. Breaking changes to the API will increment the major version number (2.x).

CHAPTER 4

User Guide

Get started with CBAPI here. For detailed information on the objects and methods exposed by CBAPI, see the full API Documentation below.

4.1 Installation

Before installing cbapi, make sure that you have access to a working EDR (CB Response) or App Control (CB Protection) server. The server can be either on-premise or in the cloud. EDR (CB Response) clusters are also supported. Once you have access to a working can use the standard Python packaging tools to install cbapi on your local machine.

Documentation is also available on the *Developer Network <https://developer.carbonblack.com/reference/enterprise-response/guide/>*.

If you already have Python installed, you can skip right down to "Using Pip".

4.1.1 Installing Python

Obviously the first thing you'll need to do is install Python on your workstation or server. We recommend using the latest version of Python 3 (as of this writing, 3.6.4) for maximum performance and compatibility. Linux and Mac OS X systems will most likely have Python installed; it will have to be installed on Windows separately.

Note that cbapi is compatible with both Python 2.7 and Python 3.x. If you already have Python 3 installed on your system, you're good to go!

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

```
$ python --version
Python 3.6.4

$ pip --version
pip 9.0.1 from /usr/local/lib/python3.6/site-packages (python 3.6)
```

If "python" reports back a version of 2.6.x, 2.7.x, or 3.x.x, you're in luck. If "pip" is not found, don't worry, we'll install that shortly.

If you're on Windows, and Python is not installed yet, download the latest Python installer from the python.org website. We recommend using the latest version of Python 3. As of this writing, the latest version available is 3.6.4. The direct link for the Python 3.6.4 installer for Windows 64-bit platforms is https://www.python.org/ftp/python/3.6.4/python-3.6.4-amd64.exe.

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

If for some reason you do not have pip installed, follow the instructions at this handy guide.

4.1.2 Using Pip

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

```
$ pip install cbapi
```

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

4.1.3 Getting the Source Code

cbapi is actively developed on GitHub and the code is available from the carbonblack GitHub repository. The version of cbapi on GitHub will reflect the latest development version of cbapi and may contain bugs not present in the currently released version. On the other hand, it may contain exactly the goodies you're looking for (or you'd like to contribute back; we are happy to accept pull requests!)

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

```
$ git clone https://github.com/carbonblack/cbapi-python.git
```

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

```
$ cd cbapi-python
$ python setup.py develop
```

This will link the version of cbapi-python you checked out into your Python site-packages directory. Any changes you make to the checked out version of cbapi will be reflected in your local Python installation. This is a good choice if you are thinking of changing or developing on cbapi itself.

4.2 Getting Started

First, let's make sure that your API authentication tokens have been imported into cbapi. Once that's done, then read on for the key concepts that will explain how to interact with Carbon Black APIs via cbapi.

Feel free to follow along with this document or watch the Development Environment Setup video on the Developer Network website.

4.2.1 API Authentication

EDR (CB Response) and App Control (CB Protection) use a per-user API secret token to authenticate requests via the API. The API token confers the same permissions and authorization as the user it is associated with, so protect the API token with the same care as a password.

To learn how to obtain the API token for a user, see the Developer Network website: there you will find instructions for obtaining an API token for EDR (CB Response) and App Control (CB Protection).

Once you have the API token, cbapi helps keep your credentials secret by enforcing the use of a credential file. To encourage sharing of scripts across the community while at the same time protecting the security of our customers, cbapi strongly discourages embedding credentials in individual scripts. Instead, you can place credentials for several EDR (CB Response) or App Control (CB Protection) servers inside the API credential file and select which "profile" you would like to use at runtime.

To create the initial credential file, a simple-to-use script is provided. Just run the `cbapi-response`, `cbapi-protection`, or `cbapi-psc` script with the `configure` argument. On Mac OS X and Linux:

```
$ cbapi-response configure
```

Alternatively, if you're using Windows (change `c:\python27` if Python is installed in a different directory):

```
C:\> python c:\python27\scripts\cbapi-response configure
```

This configuration script will walk you through entering your API credentials and will save them to your current user's credential file location, which is located in the `.carbonblack` directory in your user's home directory.

If using cbapi-psc, you will also be asked to provide an org key. An org key is required to access the Carbon Black Cloud, and can be found in the console under Settings -> API Keys.

4.2.2 Your First Query

Now that you have cbapi installed and configured, let's run a simple query to make sure everything is functional:

```
$ python
Python 2.7.10 (default, Jun 22 2015, 12:25:23)
[GCC 4.2.1 Compatible Apple LLVM 6.1.0 (clang-602.0.53)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from cbapi.response import *
>>> c = CbResponseAPI()
>>> print(c.select(Process).first().cmdline)
C:\Windows\system32\services.exe
```

That's it! Now on to the next step, learning the concepts behind cbapi.

4.3 Concepts

There are a few critical concepts that will make understanding and using the cbapi easier. These concepts are explained below, and also covered in a slide deck presented at the Carbon Black regional User Exchanges in 2016. You can see the slide deck here.

At a high level, the cbapi tries to represent data in EDR (CB Response) or App Control (CB Protection) as Python objects. If you've worked with SQL Object-relational Mapping (ORM) frameworks before, then this structure may seem familiar – cbapi was designed to operate much like an ORM such as SQLAlchemy or Ruby's ActiveRecord. If you haven't worked with one of these libraries, don't worry! The concepts will become clear after a little practice.

4.3.1 Model Objects

Everything in cbapi is represented in terms of "Model Objects". A Model Object in cbapi represents a single instance of a specific type of data in EEDR (CB Response) or App Control (CB Protection). For example, a process document from EDR (as seen on an Analyze Process page in the Web UI) is represented as a *cbapi.response.models.Process* Model Object. Similarly, a file instance in App Control (CB Protection) is represented as a *cbapi.protection.models.FileInstance* Model Object.

Once you have an instance of a Model Object, you can access all of the data contained within as Python properties. For example, if you have a Process Model Object named `proc` and you want to print its command line (which is stored in the `cmdline` property), you would write the code:

```
>>> print(proc.cmdline)
```

This would automatically retrieve the `cmdline` attribute of the process and print it out to your screen.

The data in EDR (CB Response) or App Control (CB Protection) may change rapidly, and so a comprehensive list of valid properties is difficult to keep up-to-date. Therefore, if you are curious what properties are available on a specific Model Object, you can print that Model Object to the screen. It will dump all of the available properties and their current values. For example:

```
>>> print(binary)
cbapi.response.models.Binary:
-> available via web UI at https://cbserver/#binary/08D1631FAF39538A133D94585644D5A8
host_count            : 1
digsig_result         : Signed
observed_filename     : [u'c:\\windows\\syswow64\\appwiz.cpl']
product_version       : 6.2.9200.16384
legal_copyright       : © Microsoft Corporation. All rights reserved.
digsig_sign_time      : 2012-07-26T08:56:00Z
orig_mod_len          : 669696
is_executable_image   : False
is_64bit              : False
digsig_publisher      : Microsoft Corporation
...
```

In this example, `host_count`, `orig_mod_len`, etc. are all properties available on this Binary Model Object. Sometimes, properties are not available on every instance of a Model Object. In this case, you can use the `.get()` method to retrieve the property, and return a default value if the property does not exist on the Model Object:

```
>>> print(binary.get("product_version", "<unknown>"))
6.2.9200.16384
```

In summary, Model Objects contain all the data associated with a specific type of API call. In this example, the *cbapi.response.models.Binary* Model Object reflects all the data available via the `/api/v1/binary`

API route on an EDR (CB Response) server.

4.3.2 Joining Model Objects

Many times, there are relationships between different Model Objects. To make navigating these relationships easy, cbapi provides special properties to "join" Model Objects together. For example, a *cbapi.response.models.Process* Model Object can reference the *cbapi.response.models.Sensor* or *cbapi.response.models.Binary* associated with this Process.

In this case, special "join" properties are provided for you. When you use one of these properties, cbapi will automatically retrieve the associated Model Object, if necessary.

This capability may sound like a performance killer, causing many unnecessary API calls in order to gather this data. However, cbapi has extensive Model Object caching built-in, so multiple requests for the same data will be eliminated and an API request is only made if the cache does not already contain the requested data.

For example, to print the name of the Sensor Group assigned to the Sensor that ran a specific Process:

```
>>> print(proc.sensor.group.name)
Default Group
```

Behind the scenes, this makes at most two API calls: one to obtain the Sensor associated with the Process, then another to obtain the Sensor Group that Sensor is part of. If either the Sensor or Sensor Group are already present in cbapi's internal cache, the respective API call is not made and the data is returned directly from the internal cache.

In summary, some Model Objects have special "join" properties that provide easy access to related Model Objects. A list of "join" properties is included as part of the documentation for each Model Object.

4.3.3 Queries

Now that we've covered how to get data out of a specific Model Object, we now need to learn how to obtain Model Objects in the first place! To do this, we have to create and execute a Query. cbapi Queries use the same query syntax accepted by EDR (CB Response) or App Control (CB Protection) APIs, and add a few little helpful features along the way.

To create a query in cbapi, use the `.select()` method on the CbResponseAPI or CbProtectionAPI object. Pass the Model Object type as a parameter to the `.select()` call and optionally add filtering criteria with `.where()` clauses.

Let's start with a simple query for EDR (CB Response):

```
>>> from cbapi.response import *
>>> cb = CbResponseAPI()
>>> cb.select(Process).where("process_name:cmd.exe")
<cbapi.response.rest_api.Query object at 0x1068815d0>
```

This returns a prepared Query object with the query string `process_name:cmd.exe`. Note that at this point no API calls have been made. The cbapi Query objects are "lazy" in that they are only evaluated when you use them. If you create a Query object but never attempt to retrieve any results, no API call is ever made (I suppose that answers the age-old question; if a Query object is created, but nobody uses it, it does not make a sound, after all).

What can we do with a Query? The first thing we can do is compose new Queries. Most Query types in cbapi can be "composed"; that is, you can create a new query from more than one query string. This can be useful if you have a "base" query and want to add additional filtering criteria. For example, if we take the query above and add the additional filtering criteria `(filemod:*.exe or filemod:*.dll)`, we can write:

```
>>> base_query = cb.select(Process).where("process_name:cmd.exe")
>>> composed_query = base_query.where("(filemod:*.exe or filemod:*.dll")
```

Now the `composed_query` is equivalent to a query of `process_name:cmd.exe (filemod:*.exe or filemod:*.dll)`. You can also add sorting criteria to a query:

```
>>> sorted_query = composed_query.sort("last_update asc")
```

Now when we execute the `sorted_query`, the results will be sorted by the last server update time in ascending order.

Ok, now we're ready to actually execute a query and retrieve the results. You can think of a Query as a kind of "infinite" Python list. Generally speaking, you can use all the familiar ways to access a Python list to access the results of a cbapi query. For example:

```
>>> len(base_query)     # How many results were returned for the query?
3

>>> base_query[:2]      # I want the first two results
[<cbapi.response.models.Process: id 00000003-0000-036c-01d2-2efd3af51186-00000001> @␣
↪https://cbserver,
<cbapi.response.models.Process: id 00000003-0000-07d4-01d2-2efcd4949dfc-00000001> @␣
↪https://cbserver]

>>> base_query[-1:]     # I want the last result
[<cbapi.response.models.Process: id 00000002-0000-0f2c-01d2-2a57625ca0dd-00000001> @␣
↪https://cbserver]

>>> for proc in base_query:  # Loop over all the results
>>>     print(proc.cmdline)
"C:\Windows\system32\cmd.exe"
"C:\Windows\system32\cmd.exe"
"C:\Windows\system32\cmd.exe"

>>> procs = list(base_query) # Just make a list of all the results
```

In addition to using a Query object as an array, two helper methods are provided as common shortcuts. The first method is `.one()`. The `.one()` method is useful when you know only one result should match your query; it will throw a `MoreThanOneResultError` exception if there are zero or more than one results for the query. The second method is `.first()`, which will return the first result from the result set, or None if there are no results.

Every time you access a Query object, it will perform a REST API query to the Carbon Black server. For large result sets, the results are retrieved in batches- by default, 100 results per API request on EDR (CB Response) and 1,000 results per API request on App Control (CB Protection). The search queries themselves are not cached, but the resulting Model Objects are.

4.3.4 Retrieving Objects by ID

Every Model Object (and in fact any object addressable via the REST API) has a unique ID associated with it. If you already have a unique ID for a given Model Object, for example, a Process GUID for CB Response, or a Computer ID for App Control (CB Protection), you can ask cbapi to give you the associated Model Object for that ID by passing that ID to the `.select()` call. For example:

```
>>> binary = cb.select(Binary, "CA4FAFFA957C71C006B59E29DFE3EB8B")
>>> print(binary.file_desc)
PNRP Name Space Provider
```

Note that retrieving an object via `.select()` with the ID does not automatically request the object from the server via the API. If the Model Object is already in the local cache, the locally cached version is returned. If it is not, a "blank" Model Object is created and is initialized only when an attempt is made to read a property. Therefore, assuming an empty cache, in the example above, the REST API query would not happen until the second line (the `print` statement). If you want to ensure that an object exists at the time you call `.select()`, add the `force_init=True` keyword parameter to the `.select()` call. This will cause cbapi to force a refresh of the object and if it does not exist, cbapi will throw a `ObjectNotFoundError` exception.

4.3.5 Creating New Objects

The EDR (CB Response) and App Control (CB Protection) REST APIs provide the ability to insert new data under certain circumstances. For example, the EDR REST API allows you to insert a new banned hash into its database. Model Objects that represent these data types can be "created" in cbapi by using the `create()` method:

```
>>> bh = cb.create(BannedHash)
```

If you attempt to create a Model Object that cannot be created, you will receive a `ApiError` exception.

Once a Model Object is created, it's blank (it has no data). You will need to set the required properties and then call the `.save()` method:

```
>>> bh = cb.create(BannedHash)
>>> bh.text = "Banned from API"
>>> bh.md5sum = "CA4FAFFA957C71C006B59E29DFE3EB8B"
>>> bh.save()
```

If you don't fill out all the properties required by the API, then you will receive an `InvalidObjectError` exception with a list of the properties that are required and not currently set.

Once the `.save()` method is called, the appropriate REST API call is made to create the object. The Model Object is then updated to the current state returned by the API, which may include additional data properties initialized by EDR (CB Response) or App Control (CB Protection).

4.3.6 Modifying Existing Objects

The same `.save()` method can be used to modify existing Model Objects if the REST API provides that capability. If you attempt to modify a Model Object that cannot be changed, you will receive a `ApiError` exception.

For example, if you want to change the "jgarman" user's password to "cbisawesome":

```
>>> user = cb.select(User, "jgarman")
>>> user.password = "cbisawesome"
>>> user.save()
```

4.3.7 Deleting Objects

Simply call the `.delete()` method on a Model Object to delete it (again, if you attempt to delete a Model Object that cannot be deleted, you will receive a `ApiError` exception).

Example:

```
>>> user = cb.select(User, "jgarman")
>>> user.delete()
```

4.3.8 Tracking Changes to Objects

Internally, Model Objects track all changes between when they were last refreshed from the server up until `.save()` is called. If you're interested in what properties have been changed or added, simply `print` the Model Object.

You will see a display like the following:

```
>>> user = cb.create(User)
>>> user.username = "jgarman"
>>> user.password = "cbisawesome"
>>> user.first_name = "Jason"
>>> user.last_name = "Garman"
>>> user.teams = []
>>> user.global_admin = False
>>> print(user)
User object, bound to https://cbserver.
 Partially initialized. Use .refresh() to load all attributes
--------------------------------------------------------------------------------

(+)                email: jgarman@carbonblack.com
(+)           first_name: Jason
(+)         global_admin: False
                     id: None
(+)            last_name: Garman
(+)             password: cbisawesome
(+)                teams: []
(+)             username: jgarman
```

Here, the `(+)` symbol before a property name means that the property will be added the next time that `.save()` is called. Let's call `.save()` and modify one of the Model Object's properties:

```
>>> user.save()
>>> user.first_name = "J"
>>> print(user)
print(user)
User object, bound to https://cbserver.
 Last refreshed at Mon Nov  7 16:54:00 2016
--------------------------------------------------------------------------------

                   email: jgarman@carbonblack.com
(*)           first_name: J
            global_admin: False
                     id: jgarman
               last_name: Garman
                   teams: []
                username: jgarman
```

The `(*)` symbol means that a property value will be changed the next time that `.save()` is called. This time, let's forget about our changes by calling `.reset()` instead:

```
>>> user.reset()
>>> print(user.first_name)
Jason
```

Now the user Model Object has been restored to the original state as it was retrieved from the server.

4.4 Logging & Diagnostics

The cbapi provides extensive logging facilities to track down issues communicating with the REST API and understand potential performance bottlenecks.

4.4.1 Enabling Logging

The cbapi uses Python's standard `logging` module for logging. To enable debug logging for the cbapi, you can do the following:

```
>>> import logging
>>> root = logging.getLogger()
>>> root.addHandler(logging.StreamHandler())
>>> logging.getLogger("cbapi").setLevel(logging.DEBUG)
```

All REST API calls, including the API endpoint, any data sent via POST or PUT, and the time it took for the call to complete:

```
>>> user.save()
Creating a new User object
Sending HTTP POST /api/user with {"email": "jgarman@carbonblack.com", "first_name":
↪"Jason", "global_admin": false, "id": null, "last_name": "Garman", "password":
↪"cbisawesome", "teams": [], "username": "jgarman"}
HTTP POST /api/user took 0.079s (response 200)
Received response: {u'result': u'success'}
HTTP GET /api/user/jgarman took 0.011s (response 200)
```

4.5 EDR (CB Response) API Examples

```
>>> import logging
>>> root = logging.getLogger()
>>> root.addHandler(logging.StreamHandler())
>>> logging.getLogger("cbapi").setLevel(logging.DEBUG)
```

```
>>> from cbapi.response import *
>>> cb = CbResponseAPI()
```

With that boilerplate out of the way, let's take a look at a few examples.

4.5.1 Download a Binary from EDR (CB Response)

Let's grab a binary that EDR (CB Response) has collected from one of the endpoints. This can be useful if you want to send this binary for further automated analysis or pull it down for manual reverse engineering. You can see a full example with command line options in the examples directory: `binary_download.py`.

Let's step through the example:

```
>>> import shutil
>>> md5 = "7FB55F5A62E78AF9B58D08AAEEAEF848"
>>> binary = cb.select(Binary, md5)
>>> shutil.copyfileobj(binary.file, open(binary.original_filename, "wb"))
```

First, we select the binary by its primary key: the MD5 hash of the binary contents. The third line requests the binary file data by accessing the `file` property on the Binary Model Object. The `file` property acts as a read-only, Python file-like object. In this case, we use the Python `shutil` library to copy one file object to another. The advantage of using `shutil` is that the file is copied in chunks, and the full file does not have to be read into memory before saving it to disk.

Another way to use the `file` property is to call `.read()` on it just like any other Python file object. The following code will read the first two bytes from the Binary:

```
>> binary.file.read(2)
"MZ"
```

4.5.2 Ban a Binary

Now let's take this binary and add a Banning rule for it. To do this, we create a new BannedHash Model Object:

```
>>> bh = cb.create(BannedHash)
>>> bh.md5hash = binary.md5
>>> bh.text = "Banned from API"
>>> bh.enabled = True
>>> bh.save()
Creating a new BannedHash object
Sending HTTP POST /api/v1/banning/blacklist with {"md5hash":
↪"7FB55F5A62E78AF9B58D08AAEEAEF848", "text": "banned from API"}
HTTP POST /api/v1/banning/blacklist took 0.035s (response 200)
Received response: {u'result': u'success'}
HTTP GET /api/v1/banning/blacklist/7FB55F5A62E78AF9B58D08AAEEAEF848 took 0.039s␣
↪(response 200)
```

Note that if the hash is already banned in EDR (CB Response), then you will receive a *ServerError* exception with the message that the banned hash already exists.

4.5.3 Isolate a Sensor

Switching gears, let's take a Sensor and quarantine it from the network. The EDR (CB Response) network isolation functionality allows administrators to isolate endpoints that may be actively involved in an incident, while preserving access to perform Live Response on that endpoint and collect further endpoint telemetry.

To isolate a sensor, we first need to acquire its Sensor Model Object:

```
>>> sensor = cb.select(Sensor).where("hostname:HOSTNAME").first()
```

This will select the first sensor that matches the hostname `HOSTNAME`. Now we can isolate that machine:

```
>>> sensor.isolate()
Updating Sensor with unique ID 4
Sending HTTP PUT /api/v1/sensor/4 with {"boot_id": "0", "build_id": 5, "build_version_
↪string": "005.002.000.61003", ...}
HTTP PUT /api/v1/sensor/4 took 0.129s (response 204)
HTTP GET /api/v1/sensor/4 took 0.050s (response 200)
...
True
```

The `.isolate()` method will keep polling the EDR (CB Response) server until the sensor has confirmed that it is now isolated from the network. If the sensor is offline or otherwise unreachable, this call could never return. Therefore,

there is also a `timeout=` keyword parameter that can be used to set an optional timeout that, if reached, will throw a `TimeoutError` exception. The `.isolate()` function returns True when the sensor is successfully isolated.

When you're ready to restore full network connectivity to the sensor, simply call the `.unisolate()` method:

```
>>> sensor.unisolate()
Updating Sensor with unique ID 4
Sending HTTP PUT /api/v1/sensor/4 with {"boot_id": "0", "build_id": 5, "build_version_
↪string": "005.002.000.61003", ...}
HTTP PUT /api/v1/sensor/4 took 0.077s (response 204)
HTTP GET /api/v1/sensor/4 took 0.020s (response 200)
...
True
```

Again, once the sensor is back on the network, the `.unisolate()` method will return True. Just like `.isolate()`, you can optionally specify a timeout using the `timeout=` keyword parameter.

4.5.4 Querying Processes and Events

Now, let's do some queries into the EDR (CB Response) database. The true power of EDR (CB Response) is its continuous recording and powerful query language that allows you to go back in time and track the root cause of any security incident on your endpoints. Let's start with a simple query to find instances of a specific behavioral IOC, where our attacker used the built-in Windows tool `net.exe` to mount an internal network share. We will iterate over all uses of `net.exe` to mount our target share, printing out the parent processes that led to the execution of the offending command:

```
>>> query = cb.select(Process).where("process_name:net.exe").and_(r
↪"cmdline:\\test\blah").group_by("id")
>>> def print_details(proc, depth):
...     print("%s%s: %s ran %s" % (" "*depth, proc.start, proc.username, proc.
↪cmdline))
...
>>> for proc in query:
...     print_details(proc, 0)
...     proc.walk_parents(print_details)
...
HTTP GET /api/v1/process?cb.urlver=1&facet=false&q=process_name%3Anet.exe+cmdline%3A
↪%5C%5Ctest%5Cblah&rows=100&sort=last_update+desc&start=0 took 0.462s (response 200)
2016-11-11 20:59:31.631000: WIN-IA9NQ1GN8OI\bit9rad ran net  use y: \\test\blah
HTTP GET /api/v3/process/00000003-0000-036c-01d2-2efd3af51186/1/event took 0.036s␣
↪(response 200)
2016-10-25 20:20:29.790000: WIN-IA9NQ1GN8OI\bit9rad ran "C:\Windows\system32\cmd.exe"
HTTP GET /api/v3/process/00000003-0000-0c34-01d2-2ec94f09cae6/1/event took 0.213s␣
↪(response 200)
 2016-10-25 14:08:49.651000: WIN-IA9NQ1GN8OI\bit9rad ran C:\Windows\Explorer.EXE
HTTP GET /api/v3/process/00000003-0000-0618-01d2-2ec94edef208/1/event took 0.013s␣
↪(response 200)
  2016-10-25 14:08:49.370000: WIN-IA9NQ1GN8OI\bit9rad ran␣
↪C:\Windows\system32\userinit.exe
HTTP GET /api/v3/process/00000003-0000-02ec-01d2-2ec9412b4b70/1/event took 0.017s␣
↪(response 200)
   2016-10-25 14:08:26.382000: SYSTEM ran winlogon.exe
HTTP GET /api/v3/process/00000003-0000-02b0-01d2-2ec94115df7a/1/event took 0.012s␣
↪(response 200)
    2016-10-25 14:08:26.242000: SYSTEM ran \SystemRoot\System32\smss.exe 00000001␣
↪00000030
```

(continues on next page)

(continued from previous page)

```
HTTP GET /api/v3/process/00000003-0000-0218-01d2-2ec93f813429/1/event took 0.021s␣
↪(response 200)
     2016-10-25 14:08:23.590000: SYSTEM ran \SystemRoot\System32\smss.exe
HTTP GET /api/v3/process/00000003-0000-0004-01d2-2ec93f7c7181/1/event took 0.081s␣
↪(response 200)
      2016-10-25 14:08:23.559000: SYSTEM ran c:\windows\system32\ntoskrnl.exe
HTTP GET /api/v3/process/00000003-0000-0000-01d2-2ec93f6051ee/1/event took 0.011s␣
↪(response 200)
       2016-10-25 14:08:23.374000:  ran c:\windows\system32\ntoskrnl.exe
HTTP GET /api/v3/process/00000003-0000-0004-01d2-2ec93f6051ee/1/event took 0.011s␣
↪(response 200)
2016-11-11 20:59:25.667000: WIN-IA9NQ1GN8OI\bit9rad ran net  use z: \\test\blah
2016-10-25 20:20:29.790000: WIN-IA9NQ1GN8OI\bit9rad ran "C:\Windows\system32\cmd.exe"
 2016-10-25 14:08:49.651000: WIN-IA9NQ1GN8OI\bit9rad ran C:\Windows\Explorer.EXE
  2016-10-25 14:08:49.370000: WIN-IA9NQ1GN8OI\bit9rad ran␣
↪C:\Windows\system32\userinit.exe
   2016-10-25 14:08:26.382000: SYSTEM ran winlogon.exe
    2016-10-25 14:08:26.242000: SYSTEM ran \SystemRoot\System32\smss.exe 00000001␣
↪00000030
     2016-10-25 14:08:23.590000: SYSTEM ran \SystemRoot\System32\smss.exe
      2016-10-25 14:08:23.559000: SYSTEM ran c:\windows\system32\ntoskrnl.exe
       2016-10-25 14:08:23.374000:  ran c:\windows\system32\ntoskrnl.exe
```

That was a lot in one code sample, so let's break it down part-by-part.

First, we set up the `query` variable by creating a new `Query` object using the `.where()` and `.and_()` methods. Next, we define a function that will get called on each parent process all the way up the chain to the system kernel loading during the boot process. This function, `print_details`, will print a few data points about each process: namely, the local endpoint time when that process started, the user who spawned the process, and the command line for the process.

Finally, we execute our query by looping over the result set with a Python for loop. For each process that matches the query, first we print details of the process itself (the process that called `net.exe` with a command line argument of our target share `\\test\blah`), then calls the `.walk_parents()` helper method to walk up the chain of all parent processes. Each level of parent process (the "depth") is represented by an extra space; therefore, reading backwards, you can see that `ntoskrnl.exe` spawned `smss.exe`, which in turn spawned `winlogon.exe`, and so on. You can see the full backwards chain of events that ultimately led to the execution of each of these `net.exe` calls.

Remember that we have logging turned on for these examples, so you see each of the HTTP GET requests to retrieve process event details as they happen. Astute observers will note that walking the parents of the second `net.exe` command, where the `\\test\blah` share was mounted on the `z:` drive, did not trigger additional HTTP GET requests. This is thanks to cbapi's caching layer. Since both `net.exe` commands ran as part of the same command shell session, the parent processes are shared between the two executions. Since the parent processes were already requested as part of the previous walk up the chain of parent processes, cbapi did not re-request the data from the server, instead using its internal cache to satisfy the process information requests from this script.

New Filters: Group By, Time Restrictions

In the query above, there is an extra `.group_by()` method. This method is new in cbapi 1.1.0 and is part of five new query filters available when communicating with a EDR (CB Response) 6.1 server. These filters are accessible via methods on the `Process` Query object. These new methods are:

- `.group_by()` - Group the result set by a field in the response. Typically you will want to group by `id`, which will ensure that the result set only has one result per *process* rather than one result per *event segment*. For more information on processes, process segments, and how segments are stored in EDR (CB Response) 6.0, see the Process API Changes for EDR (CB Response) 6.0 page on the Developer Network website.

- `.min_last_update()` - Only return processes that have events after a given date/time stamp (relative to the individual sensor's clock)
- `.max_last_update()` - Only return processes that have events before a given date/time stamp (relative to the individual sensor's clock)
- `.min_last_server_update()` - Only return processes that have events after a given date/time stamp (relative to the EDR (CB Response) server's clock)
- `.max_last_server_update()` - Only return processes that have events before a given date/time stamp (relative to the EDR (CB Response) server's clock)

EDR (CB Response) 6.1 uses a new way of recording process events that greatly increases the speed and scale of collection, allowing you to store and search data for more endpoints on the same hardware. Details on the new database format can be found on the Developer Network website at the Process API Changes for EDR (CB Response) 6.0 page.

The `Process` Model Object traditionally referred to a single "segment" of events in the CB Response database. In EDR (CB Response) versions prior to 6.0, a single segment will include up to 10,000 individual endpoint events, enough to handle over 95% of the typical event activity for a given process. Therefore, even though a `Process` Model Object technically refers to a single *segment* in a process, since most processes had less than 10,000 events and therefore were only comprised of a single segment, this distinction wasn't necessary.

However, now that processes are split across many segments, a better way of handling this is necessary. Therefore, EDR (CB Response) 6.0 introduces the new `.group_by()` method.

More on Filters

Querying for a process will return *all* segments that match. For example, if you search for `process_name:cmd.exe`, the result set will include *all* segments of *all* `cmd.exe` processes. Therefore, EDR (CB Response) 6.1 introduced the ability to "group" result sets by a field in the result. Typically you will want to group by the internal process id (the `id` field), and this is what we did in the query above. Grouping by the `id` field will ensure that only one result is returned per *process* rather than per *segment*.

Let's take a look at an example:

```
>>> from datetime import datetime, timedelta
>>> yesterday = datetime.utcnow() - timedelta(days=1)       # Get "yesterday" in GMT
>>> for proc in c.select(Process).where("process_name:cmd.exe").min_last_
↪update(yesterday):
...     print proc.id, proc.segment
DEBUG:cbapi.connection:HTTP GET /api/v1/process?cb.min_last_update=2017-05-21T18%3A41
↪%3A58Z&cb.urlver=1&facet=false&q=process_name%3Acmd.exe&rows=100&sort=last_
↪update+desc&start=0 took 2.164s (response 200)
00000001-0000-0e48-01d2-c2a397f4cfe0 1495465643405
00000001-0000-0e48-01d2-c2a397f4cfe0 1495465407157
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463680155
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463807694
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463543944
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463176570
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463243492
```

Notice that the "same" process ID is returned seven times, but with seven different segment IDs. EDR (CB Response) will return *every* process event segment that matches a given query, in this case, any event segment that contains the process command name `cmd.exe`.

That is, however, most likely not what you wanted. Instead, you'd like a list of the *unique* processes associated with the command name `cmd.exe`. Just add the `.group_by("id")` filter to your query:

```
>>> for proc in c.select(Process).where("process_name:cmd.exe").min_last_
↪update(yesterday).group_by("id"):
...     print proc.id, proc.segment
DEBUG:cbapi.connection:HTTP GET /api/v1/process?cb.group=id&cb.min_last_update=2017-
↪05-21T18%3A41%3A58Z&cb.urlver=1&facet=false&q=process_name%3Acmd.exe&rows=100&
↪sort=last_update+desc&start=0 took 2.163s (response 200)
00000001-0000-0e48-01d2-c2a397f4cfe0 1495465643405
```

4.5.5 Feed and Watchlist Maintenance

The cbapi provides several helper functions to assist in creating watchlists and feeds.

Watchlists are simply saved Queries that are automatically run on the EDR (CB Response) server on a periodic basis. Results of the watchlist are tagged in the database and optionally trigger alerts. Therefore, a cbapi Query can easily be converted into a watchlist through the Query `.create_watchlist()` function:

```
>>> new_watchlist = query.create_watchlist("[WARN] Attempts to mount internal share")
Creating a new Watchlist object
Sending HTTP POST /api/v1/watchlist with {"id": null, "index_type": "events", "name":
↪"[WARN] Attempts to mount internal share", "search_query": "facet=false&q=process_
↪name%3Anet.exe+cmdline%3A%5C%5Ctest%5Cblah&cb.urlver=1&sort=last_update+desc"}
HTTP POST /api/v1/watchlist took 0.510s (response 200)
Received response: {u'id': 222}
Only received an ID back from the server, forcing a refresh
HTTP GET /api/v1/watchlist/222 took 0.034s (response 200)
```

This helper function will automatically create a watchlist from the Query object with the given name.

If you have a watchlist that already exists, the Watchlist Model Object can help you extract the human-readable query from the watchlist. Just select the watchlist and access the `.query` property on the Watchlist Model Object:

```
>>> my_watchlist = cb.select(Watchlist).where("name:[WARN] Attempts to mount internal␣
↪share").one()
>>> print(my_watchlist.query)
process_name:net.exe cmdline:\\test\blah
```

You can also execute the query straight from the Watchlist Model Object:

```
>>> len(my_watchlist.search())
HTTP GET /api/v1/process?cb.urlver=1&facet=false&q=process_name%3Anet.exe+cmdline%3A
↪%5C%5Ctest%5Cblah&rows=0&start=0 took 0.477s (response 200)
2
```

And finally, you can of course enable and disable Watchlists:

```
>>> my_watchlist.enabled = False
>>> my_watchlist.save()
Updating Watchlist with unique ID 222
Sending HTTP PUT /api/v1/watchlist/222 with {"alliance_id": null, "date_added": "2016-
↪11-15 23:48:27.615993-05:00", "enabled": false, "from_alliance": false, "group_id":␣
↪-1, "id": "222", "index_type": "events", "last_hit": "2016-11-15 23:50:08.448685-
↪05:00", "last_hit_count": 2, "name": "[WARN] Attempts to mount internal share",
↪"readonly": false, "search_query": "facet=false&q=process_name%3Anet.exe%20cmdline
↪%3A%5C%5Ctest%5Cblah&cb.urlver=1", "search_timestamp": "2016-11-16T04:50:01.750240Z
↪", "total_hits": "2", "total_tags": "2"}
HTTP PUT /api/v1/watchlist/222 took 0.036s (response 200)
```

(continues on next page)

(continued from previous page)

```
Received response: {u'result': u'success'}
HTTP GET /api/v1/watchlist/222 took 0.029s (response 200)
```

You can see more examples of Feed and Watchlist maintenance in the `feed_operations.py` and `watchlist_operations.py` example scripts.

4.5.6 Managing Threat Reports & Alerts

The cbapi provides helper functions to manage alerts and threat reports in bulk. The Query objects associated with the ThreatReport and Alert Model Objects provide a few bulk operations to help manage large numbers of Threat Reports and Alerts, respectively.

To mark a large number of Threat Reports as false positives, create a query that matches the Reports you're interested in. For example, if every Report from the Feed named "SOC" that contains the word "FUZZYWOMBAT" in the report title should be considered a false positive (and no longer trigger Alerts), you can write the following code to do so:

```
>>> feed = c.select(Feed).where("name:SOC").one()
>>> report_query = feed.reports.where("title:FUZZYWOMBAT")
>>> report_query.set_ignored()
```

Similar actions can be taken on Alerts. The AlertQuery object exposes three helper methods to perform bulk operations on sets of Alerts: `.set_ignored()`, `.assign_to()`, and `.change_status()`.

4.5.7 Joining Everything Together

Now that we've examined how to request information on binaries, sensors, and processes through cbapi, let's chain this all together using the "join" functionality of cbapi's Model Objects. Let's just tweak the `print_details` function from above to add a few more contextual details. Our new function will now include the following data points for each process:

- The hostname the process was executed on
- The sensor group that host belongs to
- **If the binary was signed, also print out:**
  - The number of days between when the binary was signed and it was executed on the endpoint
  - The verified publisher name from the digital signature

We can transparently "join" between the Process Model Object and the Sensor, Sensor Group, and Binary Model Objects using the appropriately named helper properties. Here's the new function:

```
>>> import pytz

>>> def print_details(proc, depth):
...     print("On host {0} (part of sensor group {1}):".format(proc.hostname, proc.
↪sensor.group.name))
...     print("- At {0}, process {1} was executed by {2}".format(proc.start, proc.
↪cmdline, proc.username))
...     if proc.binary.signed:
...         # force local timestamp into UTC, we're just looking for an estimate here.
...         utc_timestamp = proc.start.replace(tzinfo=pytz.timezone("UTC"))
...         days_since_signed = (utc_timestamp - proc.binary.signing_data.sign_time).
↪days
```

(continues on next page)

(continued from previous page)

```
...         print("- That binary ({0}) was signed by {1} {2} days before it was␣
↪executed.".format(proc.process_md5,
...               proc.binary.signing_data.publisher, days_since_signed))
```

Now if we run our for loop from above again:

```
>>> for proc in query:
...     print_details(proc, 0)
...     proc.walk_parents(print_details)
...
HTTP GET /api/v1/process?cb.urlver=1&facet=false&q=process_name%3Anet.exe+cmdline%3A
↪%5C%5Ctest%5Cblah&rows=100&sort=last_update+desc&start=0 took 0.487s (response 200)
HTTP GET /api/v1/sensor/3 took 0.037s (response 200)
HTTP GET /api/group/1 took 0.022s (response 200)
On host WIN-IA9NQ1GN8OI (part of sensor group Default Group):
- At 2016-11-11 20:59:31.631000, process net  use y: \\test\blah was executed by WIN-
↪IA9NQ1GN8OI\bit9rad
HTTP GET /api/v1/binary/79B6D4C5283FC806387C55B8D7C8B762/summary took 0.016s␣
↪(response 200)
- That binary (79b6d4c5283fc806387c55b8d7c8b762) was signed by Microsoft Corporation␣
↪1569 days before it was executed.
HTTP GET /api/v3/process/00000003-0000-036c-01d2-2efd3af51186/1/event took 0.045s␣
↪(response 200)
On host WIN-IA9NQ1GN8OI (part of sensor group Default Group):
- At 2016-10-25 20:20:29.790000, process "C:\Windows\system32\cmd.exe"  was executed␣
↪by WIN-IA9NQ1GN8OI\bit9rad
HTTP GET /api/v1/binary/BF93A2F9901E9B3DFCA8A7982F4A9868/summary took 0.015s␣
↪(response 200)
- That binary (bf93a2f9901e9b3dfca8a7982f4a9868) was signed by Microsoft Corporation␣
↪1552 days before it was executed.
```

Those few lines of Python above are jam-packed with functionality. Now for each process execution, we have added contextual information on the source host, the group that host is part of, and details about the signing status of the binary that was executed. The magic is performed behind the scenes when we use the `.binary` and `.sensor` properties on the Process Model Object. Just like our previous example, cbapi's caching layer ensures that we do not overload the EDR (CB Response) server with duplicate requests for the same data. In this example, multiple redundant requests for sensor, sensor group, and binary data are all eliminated by cbapi's cache.

4.5.8 Facets

The cbapi also provides functionality to pull facet information from the database. You can use the `.facet()` method on a Query object to retrieve facet (ie. "group") information for a given query result set. Here's an example that pulls the most common process names for our sample host:

```
>>> def print_facet_histogram(facets):
...     for entry in facets:
...         print("%15s: %5s%% %s" % (entry["name"][:15], entry["ratio"], u"\u25A0
↪"*int((entry["percent"])/2)))
...

>>> facet_query = cb.select(Process).where("hostname:WIN-IA9NQ1GN8OI").and_(
↪"username:bit9rad")
>>> print_facet_histogram(facet_query.facets("process_name")["process_name"])

HTTP GET /api/v1/process?cb.urlver=1&facet=true&facet.field=process_name&facet.
↪field=username&q=hostname%3AWIN-IA9NQ1GN8OI+username%3Abit9rad&rows=0&start=0 took␣
↪0.024s (response 200)
```

(continues on next page)

(continued from previous page)

```
     chrome.exe:  23.4% ================================================
thumbnailextrac:  15.4% ================================
   adobearm.exe:   8.6% ==================
   taskhost.exe:   6.0% ============
    conhost.exe:   4.7% =========
       ping.exe:   4.0% ========
     wermgr.exe:   3.5% =======
```

In the above example, we just pulled one facet: the `process_name`; you can ask the server for faceting on multiple fields in one query by simply listing the fields in the call to `.facet()`: for example, `.facet("username", "process_name")` will produce a dictionary with two top-level keys: `username` and `process_name`.

4.5.9 Administrative Tasks

In addition to querying data, you can also perform various administrative tasks using cbapi.

Let's create a user on our EDR (CB Response) server:

```
>>> user = cb.create(User)
>>> user.username = "jgarman"
>>> user.password = "cbisawesome"
>>> user.first_name = "Jason"
>>> user.last_name = "Garman"
>>> user.email = "jgarman@carbonblack.com"
>>> user.teams = []
>>> user.global_admin = False
Creating a new User object
Sending HTTP POST /api/user with {"email": "jgarman@carbonblack.com", "first_name":
↪"Jason", "global_admin": false, "id": null, "last_name": "Garman", "password":
↪"cbisawesome", "teams": [], "username": null}
HTTP POST /api/user took 0.608s (response 200)
Received response: {u'result': u'success'}
```

How about moving a sensor to a new Sensor Group:

```
>>> sg = cb.create(SensorGroup)
>>> sg.name = "Critical Endpoints"
>>> sg.site = 1
>>> sg.save()
Creating a new SensorGroup object
Sending HTTP POST /api/group with {"id": null, "name": "Critical Endpoints", "site_id
↪": 1}
HTTP POST /api/group took 0.282s (response 200)
Received response: {u'id': 2}
Only received an ID back from the server, forcing a refresh
HTTP GET /api/group/2 took 0.011s (response 200)
>>> sensor = cb.select(Sensor).where("hostname:WIN-IA9NQ1GN8OI").first()
>>> sensor.group = sg
>>> sensor.save()
Updating Sensor with unique ID 3
Sending HTTP PUT /api/v1/sensor/3 with {"boot_id": "2", "build_id": 2, "build_version_
↪string": "005.002.000.60922", ...
HTTP PUT /api/v1/sensor/3 took 0.087s (response 204)
HTTP GET /api/v1/sensor/3 took 0.030s (response 200)
```

4.6 CbAPI and Live Response

Working with the Live Response REST API directly can be difficult. Thankfully, just like the rest of Carbon Black's REST APIs, cbapi provides Pythonic APIs to make working with the Live Response API much easier.

In addition to easy-to-use APIs to call into Live Response, cbapi also provides a "job-based" interface that allows cbapi to intelligently schedule large numbers of concurrent Live Response sessions across multiple sensors. Your code can then be notified when the jobs are complete, returning the results of the job if it succeeded or the Exception if it failed.

4.6.1 Getting Started with Live Response

The cbapi Live Response API is built around establishing a `cbapi.response.live_response.LiveResponseSession` object from a *cbapi.response.models.Sensor* Model Object. Then you can call methods on the `LiveResponseSession` object to perform Live Response actions on the target host. These calls are synchronous, meaning that they will wait until the action is complete and a result is available, before returning back to your script. Here's an example:

```
>>> from cbapi.response import *
>>> cb = CbResponseAPI()
>>> sensor = cb.select(Sensor).where("hostname:WIN-IA9NQ1GN8OI").first()
>>> with sensor.lr_session() as session:
...     print(session.get_file(r"c:\test.txt"))

this is a test
```

Since the Live Response API is synchronous, the script will not continue until either the Live Response session is established and the file contents are retrieved, or an exception occurs (in this case, either a timeout error or an error reading the file).

As seen in the example above, the `.lr_session()` method is context-aware. EDR (CB Response) has a limited number of concurrent Live Response session slots (by default, only ten). By wrapping the `.lr_session()` call within a `with` context, the session is automatically closed at the end of the block and frees that slot for another concurrent Live Response session in another script or user context.

A full listing of methods in the cbapi Live Response API is available in the documentation for the *cbapi.live_response_api.CbLRSessionBase* class.

4.6.2 Live Response Errors

There are four classes of errors that you will commonly encounter when working with the Live Response API:

- A *cbapi.errors.TimeoutError* is raised if a timeout is encountered when waiting for a response for a Live Response API request.
- A `cbapi.response.live_response_api.LiveResponseError` is raised if an error is returned during the execution of a Live Response command on an endpoint. The `LiveResponseError` includes detailed information about the error that occurred, including the exact error code that was returned from the endpoint and a textual description of the error.
- A *cbapi.errors.ApiError* is raised if you attempt to execute a command that is not supported by the sensor; for example, attempting to acquire a memory dump from a sensor running a pre-5.1 version of the agent will fail with an `ApiError` exception.
- A *cbapi.errors.ServerError* is raised if any other error occurs; for example, a 500 Internal Server Error is returned from the Live Response API.

4.6.3 Job-Based API

The basic Synchronous API described above in the Getting Started section works well for small tasks, targeting one sensor at a time. However, if you want to execute the same set of Live Response commands across a larger number of sensors, the cbapi provides a Job-Based Live Response API. The Job-Based Live Response API provides a straightforward API to submit Live Response jobs to a scheduler, schedule those Live Response jobs on individual endpoints concurrently, and return results and any errors back to you when the jobs complete. The Job-Based Live Response API is a natural fit with the Event-Based API to create IFTTT-style pipelines; if an event is received via the Event API, then perform Live Response actions on the affected endpoint via the Live Response Job-Based API.

The Job-Based API works by first defining a reusable "job" to perform on the endpoint. The Job is simply a class or function that takes a Live Response session object as input and performs a series of commands. Jobs can be as simple as retrieving a registry key, or as complex as collecting the Chrome browser history for any currently logged-in users.

Let's look at an example Job to retrieve a registry key. This example job is pulled from the `get_reg_autoruns.py` example script:

```
class GetRegistryValue(object):
    def __init__(self, registry_key):
        self.registry_key = registry_key

    def run(self, session):
        reg_info = session.get_registry_value(self.registry_key)
        return time.time(), session.sensor_id, self.registry_key, reg_info["value_data
↪"]
```

To submit this job, you instantiate an instance of a `GetRegistryValue` class with the registry key you want to pull back from the endpoint, and submit the `.run()` method to the Live Response Job API:

```
>>> job = GetRegistryValue(regmod_path)
>>> registry_job = cb.live_response.submit_job(job.run, sensor_id)
```

Your script resumes execution immediately after the call to `.submit_job()`. The job(s) that you've submitted will be executed in a set of background threads managed by cbapi.

4.7 CbAPI Changelog

4.7.1 CbAPI 1.7.6 - Release Dec 20, 2021

Bug Fixes

- Removed the requirement for an admin token to connect
- Added sensor paginated query

General

- Updated version of lxml library

4.7.2 CbAPI 1.7.5 - Released June 16, 2021

Updates

- **General**

  - Allow the CbAPI to accept a pre-configured Session object to be used for access, to get around unusual configuration requirements.

4.7.3 CbAPI 1.7.4 - Released April 7, 2021

Updates

- **General**
  - Fix example code in the documentation for Facets
- **EDR (CB Response)**
  - Add missing fields for SensorGroup class and fix example script to properly create SensorGroup
  - Fix example script sensor_group_operations.py to list groups without ipaddresses
  - Fix alert.save()
  - Allow blocked processes to be accessed through the Process (processblocks)

4.7.4 CbAPI 1.7.3 - Released January 15, 2021

Updates

- **General**
  - Fix resource warnings regarding unclosed file object
  - Notice added to readme for Carbon Black Cloud features moving to Carbon Black Cloud SDK repo
- **Carbon Black Cloud**
  - Increase default rows of alerts to 100
  - Add get_auditlogs function to API object
- **CB Threathunter**
  - Fix typo in process query
  - Bump lxml from 4.4.1 to 4.6.2 for Threat Intelligence example
- **EDR (CB Response)**
  - Add Sensor Builds
  - **Alert.set_ignored() and AlertQuery.set_ignored():**
    - Added a docstring to specify what happens with this method
    - Modified the payload keys based on manual testing
  - **Alert.change_status() and AlertQuery.change_status():**
    - Added a status check to ensure it's a valid status

4.7.5 CbAPI 1.7.2 - Released July 22, 2020

Updates

- **General**
  - Allow passing in proxy configuration as direct parameters during class instantiation of base API.

4.7.6 CbAPI 1.7.1 - Released July 22, 2020

Updates

- **General**
  - Documentation updates to indicate changed product names
- **Carbon Black Cloud**
  - Process Search v2 rows defaults to 10k to match UI behavior
- **EDR (CB Response)**
  - Add support for fetching alert by ID

4.7.7 CbAPI 1.7.0 - Released July 14, 2020

Updates

- **General**
  - Updates to pool defaults in base API.
  - Changes to exception handling to better discriminate ConnectionErrors and queries with invalid syntax.
  - Various minor bug fixes throughout.
- **Carbon Black Cloud**
  - Bug fixes to query implementation.
  - Live Response: Account for sensor queue depth when submitting jobs.
- **CB Defense**
  - Added examples for Dell BIOS verification.
- **CB ThreatHunter**
  - Bug fixes to query implementation.
  - Update process and event searches to v2.
  - examples/create_feed: Make report optional during feed creation
  - examples/process_exporter: Add headers to CSV file writer
  - examples/threat_intelligence: Simplify report validation, add severity conversion to percent

4.7.8 CbAPI 1.6.2 - Released April 08, 2020

Updates

- **CB Response**
  - Changes to align with limits placed on the sensor update function in CB Response 7.1.0. Release notes are available on User Exchange, the ID is CB 28683.

4.7.9 CbAPI 1.6.1 - Released January 13, 2020

Updates

- **CB Response**
  - Fix Alert.save() to use alert v1 API
- **Carbon Black Cloud**
  - Fix Live Response flow to use integrationServices/v3/device to prevent need for multiple API keys
- **CB ThreatHunter**
  - Update example for ThreatHunter Query

4.7.10 CbAPI 1.6.0 - Released December 3, 2019

Updates

- **New Carbon Black Cloud API Support**
  - **Support for Devices v6:**
    - List and search for devices
    - Export device information to CSV
    - Device control actions: quarantine, bypass, background scan, deregister/delete, update
  - **Support for Alerts v6:**
    - Search for and retrieve alerts
    - Update alert status (dismiss alerts)

Examples

- **Devices v6:**
  - psc/device_control.py
  - psc/download_device_list.py
  - psc/list_devices.py
- **Alerts v6:**
  - psc/alert_search_suggestions.py
  - psc/bulk_update_alerts.py
  - psc/bulk_update_cbanalytics_alerts.py
  - psc/bulk_update_threat_alerts.py
  - psc/bulk_update_vmware_alerts.py
  - psc/bulk_update_watchlist_alerts.py
  - psc/list_alert_facets.py
  - psc/list_alerts.py
  - psc/list_cbanalytics_alert_facets.py
  - psc/list_cbanalytics_alerts.py

  - psc/list_vmware_alert_facets.py
  - psc/list_vmware_alerts.py
  - psc/list_watchlist_alert_facets.py
  - psc/list_watchlist_alerts.py

4.7.11 CbAPI 1.5.6 - Released November 19, 2019

Updates

- **General**
  - Name change to Carbon Black Cloud from PSC.

4.7.12 CbAPI 1.5.5 - Released November 12, 2019

Updates

- **CB ThreatHunter**
  - Fix List object that was not callable.

4.7.13 CbAPI 1.5.4 - Released October 24, 2019

Updates

- **General**
  - Prevent pytest from blocking python2 install
- **CB Response**
  - Fix python2 function overwrite for max_children

4.7.14 CbAPI 1.5.3 - Released October 15, 2019

Updates

- **General**
  - Fix MoreThanOneResultError
  - Add environmental org key
- **CB ThreatHunter**
  - Fix iterating process search results
  - Fix watchlist reports fetch
  - Fix process.summary

4.7.15 CbAPI 1.5.2 - Released September 9, 2019

Updates

- **CB Response**
  - Add support for max_children on Process search
- **CB LiveOps**
  - Add LQ device summaries
  - Add faceting for LQ results and LQ device summaries
  - Add LQ run history
- **CB ThreatHunter**
  - Fix an invalid search job creation

4.7.16 CbAPI 1.5.1 - Released July 23, 2019

Updates

- **CB Response**
  - Require CBAPI users to obtain their API token from the CB Response console.
- **CB LiveOps**
  - Fixing a build issue

4.7.17 CbAPI 1.5.0 - Released July 23, 2019

Updates

- **CB LiveOps**
  - Start new LiveQuery (LQ) runs
  - Fetch LQ results
  - View LQ run status
  - Filter on LQ results
- **PSC Org Key Management**
  - Added support for org key management within CBAPI
  - Credentials utility for org keys
  - PR #166, #169, #170

Examples

- LiveQuery - manage_run.py
- LiveQuery - run_search.py

4.7.18 CbAPI 1.4.5 - Released July 11, 2019

Updates

- **CB ThreatHunter**
  - Route updates for process search, feed management, watchlist management
  - Enforce org_key presence
  - Org-based process search
  - Org-based event search
  - Org-based tree queries
- Minor updates for Python3 Compatibility

Examples

- Updated CB TH Process Search Example
- Added process_guid to process_tree example for ThreatHunter

4.7.19 CbAPI 1.4.4 - Released July 3, 2019

Updates

- Carbon Black UBS Support PR #142
- CB Response - Fixing bulk update for Alerts to use v1 route
- Updates to use yaml safe_load #157

Examples

- Refactored Carbon Black ThreatHunter examples
- Added process_guid to process_tree example for ThreatHunter

4.7.20 CbAPI 1.4.3 - Released May 7, 2019

Updates

- CB ThreatHunter - Feed fixes #156
- CB Response - Change Alert model object to use v2 route #155
- CB Response - Only view active LR sessions #154
- Removing refs to VT alliance feeds #144

Examples

- CB Defense - Create list_events_with_cmdline_csv.py #152
- CB Defense - Updated import link to proper module #148

4.7.21 CbAPI 1.4.2 - Released March 27, 2019

This release introduces additional support for CB PSC's ThreatHunter APIs

- Threat Intelligence APIs

4.7.22 CbAPI 1.4.1 - Released January 10, 2019

- Bug fixes
- Adding to authorized error to make it clear that users should check API creds

4.7.23 CbAPI 1.4.0 - Released January 10, 2019

This release introduces support for CB PSC's ThreatHunter APIs

- Process, Tree, and Search are supported with more to come

4.7.24 CbAPI 1.3.6 - Released February 14, 2018

This release has one critical fix:

- Fix a fatal exception when connecting to CB Response 6.1.x servers

4.7.25 CbAPI 1.3.5 - Released February 2, 2018

This release includes bugfixes and contributions from the Carbon Black community.

All products:

- More Python 3 compatibility fixes.
- Fix the `wait_for_completion` and `wait_for_output` options in the Live Response `.create_process()` method. If `wait_for_completion` is True, the call to `.create_process()` will block until the remote process has exited. If `wait_for_output` is True, then `.create_process()` will additionally wait until the output of the remote process is ready and return that output to the caller. Setting `wait_for_output` to True automatically sets `wait_for_completion` to True as well.
- The `BaseAPI` constructor now takes three new optional keyword arguments to control the underlying connection pool: `pool_connections`, `pool_maxsize`, and `pool_block`. These arguments are sent to the underlying `HTTPAdapter` used when connecting to the Carbon Black server. For more information on these parameters, see the Python requests module API documentation for HTTPAdapter.

CB Defense:

- Date/time stamps in the Device model object are now represented as proper Python datetime objects, rather than integers.
- The `policy_operations.py` example script's "Replace Rule" command is fixed.
- Add the CB Live Response job-based API.
- Add a new example script `list_devices.py`

CB Response:

- The `Process` and `Binary` model objects now return None by default when a non-existent attribute is referenced, rather than throwing an exception.
- Fixes to `walk_children.py` example script.
- Fix exceptions in enumerating child processes, retrieving path and MD5sums from processes.
- Multiple `.where()` clauses can now be used in the `Sensor` model object.
- Workaround implemented for retrieving/managing more than 500 banned hashes.

- Alert bulk operations now work on batches of 500 alerts.
- `.flush_events()` method on `Sensor` model object no longer throws an exception on CB Response 6.x servers.
- `.restart_sensor()` method now available for `Sensor` model object.
- Fix `user_operations.py` example script to eliminate exception when adding a new user to an existing team.
- Add `.remove_team()` method on `User` model object.
- Automatically set `cb.legacy_5x_mode` query parameter for all Process queries whenever a legacy Solr core (from CB Response 5.x) is loaded.
- Added `.use_comprehensive_search()` method to enable the "comprehensive search" option on a Process query. See the CB Developer Network documentation on Comprehensive Search for more information on "comprehensive search".
- Add `.all_childprocs()`, `.all_modloads()`, `.all_filemods()`, `.all_regmods()`, `.all_crossprocs()`, and `.all_netconns()` methods to retrieve process events from all segments, rather than the current process segment. You can also use the special segment "0" to retrieve process events across all segments.
- Fix `cmdline_filters` in the `IngressFilter` model object.

App Control (CB Protection):

- Tamper Protection can now be set and cleared in the `Computer` model object.

4.7.26 CbAPI 1.3.4 - Released September 14, 2017

This release includes a critical security fix and small bugfixes.

Security fix:

- The underlying CbAPI connection class erroneously disabled hostname validation by default. This does *not* affect code that uses CbAPI through the public interfaces documented here; it only affects code that accesses the new `CbAPISessionAdapter` class directly. This class was introduced in version 1.3.3. Regardless, it is strongly recommended that all users currently using 1.3.3 upgrade to 1.3.4.

Bug fixes:

- Add rule filename parameter to CB Defense `policy_operations.py` script's `add-rule` command.
- Add support for `tamperProtectionActive` attribute to App Control's (CB Protection) `Computer` object.
- Work around CB Response issue- the `/api/v1/sensor` route incorrectly returns an HTTP 500 if no sensors match the provided query. CbAPI now catches this exception and will instead return an empty set back to the caller.

4.7.27 CbAPI 1.3.3 - Released September 1, 2017

This release includes security improvements and bugfixes.

Security changes:

- CbAPI enforces the use of HTTPS when connecting to on-premise CB Response servers.
- CbAPI can optionally require TLSv1.2 when connecting to Carbon Black servers.

  - Note that some versions of Python and OpenSSL, notably the version of OpenSSL packaged with Mac OS X, do not support TLSv1.2. This will cause CbAPI to fail to connect to CB Response 6.1+ servers which require TLSv1.2 cipher suites.
  - A new command, `cbapi check-tls`, will report the TLS version supported by your platform.
  - To enforce the use of TLSv1.2 when connecting to a server, add `ssl_force_tls_1_2=True` to that server's credential profile.
- Add the ability to "pin" a specific server certificate to a credential profile.
  - You can now force TLS certificate verification on self-signed, on-premise installations of EDR (CB Response) or App Control (Protection) through the `ssl_cert_file` option in the credential profile.
  - To "pin" a server certificate, save the PEM-formatted server certificate to a file, and put the full path to that PEM file in the `ssl_cert_file` option of that server's credential profile.
  - When using this option with on-premise CB Response servers, you may also have to set `ssl_verify_hostname=False` as the hostname in the certificate generated at install time is `localhost` and will not match the server's hostname or IP address. This option will still validate that the server's certificate is valid and matches the copy in the `ssl_cert_file` option.

Changes for CB Protection:

- The API now sets the appropriate "GET" query fields when changing fields such as the `debugFlags` on the Computer object.
- The `.template` attribute on the Computer model object has been renamed `.templateComputer`.
- Remove AppCatalog and AppTemplate model objects.

Changes for CB Response:

- Added `.webui_link` property to CB Response Query objects.
- Added `ban_hash.py` example.

Bug Fixes:

- Error handling is improved on Python 3. Live Response auto-reconnect functionality is now fixed on Python 3 as a result.
- Workaround implemented for CB Response 6.1 where segment_ids are truncated on Alerts. The `.process` attribute on an Alert now ignores the `segment_id` and links to the first Process segment.
- Fixed issue with `Binary.signed` and `CbModLoadEvent.is_signed`.

4.7.28 CbAPI 1.3.2 - Released August 10, 2017

This release introduces the Policy API for CB Defense. A sample `policy_operations.py` script is now included in the `examples` directory for CB Defense.

Other changes:

- CB Response
  - Bugfixes to the `User` Model Object.
  - New `user_operations.py` example script to manage users & teams.
  - Additional `Team` Model Object to add/remove/modify user teams.
  - New `check_datasharing.py` example script to check if third party data sharing is enabled for binaries on any sensor groups.

  - Documentation fix for the `User` Model Object.
  - Fix to the `watchlist_operations.py` example script.

4.7.29 CbAPI 1.3.1 - Released August 3, 2017

This is a bugfix release with minor changes:

- CB Response
  - Add `partition_operations.py` script to demonstrate the use of the StoragePartition model object.
  - Fix errors when accessing the `.start` attribute of child processes.
  - Fix errors generated by the `walk_children.py` example script. The output has been changed as well to indicate the process lifetime, console UI link, and command lines.
  - Add an `.end` attribute to the Process model object. This attribute reports back either `None` if the process is still executing, or the last event time associated with the process if it has exited. See the `walk_children.py` script for an example of how to calculate process lifetime.
  - Fix errors when using the `.parents` attribute of a Process.
  - Add `wait_for_completion` flag to `create_process` Live Response method, and default to `True`. The `create_process` method will now wait for the target process to complete before returning.
- CB Defense
  - Add `wait_for_completion` flag to `create_process` Live Response method, and default to `True`. The `create_process` method will now wait for the target process to complete before returning.

4.7.30 CbAPI 1.3.0 - Released July 27, 2017

This release introduces the Live Response API for CB Defense. A sample `cblr_cli.py` script is now included in the `examples` directory for both CB Response and CB Defense.

Other changes:

- CB Protection
  - You can now create new `FileRule` and `Policy` model objects in cbapi.
- CB Response
  - Added `watchlist_exporter.py` and `watchlist_importer.py` scripts to the CB Response examples directory. These scripts allow you to export Watchlist data in a human- and machine-readable JSON format and then re-import them into another CB Response server.
  - The `Sensor` Model Object now uses the non-paginated (v1) API by default. This fixes any issues encountered when iterating over all the sensors and receiving duplicate and/or missing sensors.
  - Fix off-by-one error in `CbCrossProcess` object.
  - Fix issue iterating through `Process` Model Objects when accessing processes generated from a 5.2 server after upgrading to 6.1.
  - Reduce number of API requests required when accessing sibling information (parents, children, and siblings) from the `Process` Model Object.
  - Retrieve all events for a process when using `segment` ID of zero on a CB Response 6.1 server.
  - Behavior of `Process.children` attribute has changed:

    - Only one entry is present per child (before there were up to two; one for the spawn event, one for the terminate event)
    - The timestamp is derived from the start time of the process, not the timestamp from the spawn event. the two timestamps will be off by a few microseconds.
    - The old behavior is still available by using the `Process.childprocs` attribute instead. This incurs a performance penalty as another API call will have to be made to collect the childproc information.
  - `Binary` Model Object now returns False for `.is_signed` attribute if it is set to `(Unknown)`.
- Moved the `six` Python module into cbapi and removed the external dependency.

4.7.31 CbAPI 1.2.0 - Released June 22, 2017

This release introduces compatibility with our new product, CB Defense, as well as adding new Model Objects introduced in the CB Protection 8.0 APIs.

Other changes:

- CB Response
  - New method `synchronize()` added to the `Feed` Model Object
- Bug fixes and documentation improvements

4.7.32 CbAPI 1.1.1 - Released June 2, 2017

This release includes compatibility fixes for CB Response 6.1. Changes from 1.0.1 include:

- Substantial changes to the `Process` Model Object for CB Response 6.1. See details below.
- New `StoragePartition` Model Object to control Solr core loading/unloading in CB Response 6.1.
- New `IngressFilter` Model Object to control ingress filter settings in CB Response 6.1.
- Fix issues with `event_export.py` example script.
- Add `.all_events` property to the `Process` Model Object to expose a list of all events across all segments.
- Add example script to perform auto-banning based on watchlist hits from CB Event Forwarder S3 output files.
- Add bulk operations to the `ThreatReport` and `Alert` Query objects:
  - You can now call `.set_ignored()`, `.assign()`, and `.change_status()` on an `Alert` Query object to change the respective fields for every Alert that matches the query.
  - You can now call `.set_ignored()` on a `ThreatReport` Query object to set or clear the ignored flag for every ThreatReport that matches the query.

Changes to `Process` Model Object for CB Response 6.1

CB Response 6.1 uses a new way of recording process events that greatly increases the speed and scale of collection, allowing you to store and search data for more endpoints on the same hardware. Details on the new database format can be found on the Developer Network website at the Process API Changes for CB Response 6.0 page.

The `Process` Model Object traditionally referred to a single "segment" of events in the CB Response database. In CB Response versions prior to 6.0, a single segment will include up to 10,000 individual endpoint events, enough to handle over 95% of the typical event activity for a given process. Therefore, even though a `Process` Model Object

technically refers to a single *segment* in a process, since most processes had less than 10,000 events and therefore were only comprised of a single segment, this distinction wasn't necessary.

However, now that processes are split across many segments, a better way of handling this is necessary. Therefore, CB Response 6.0 introduces the new `.group_by()` method. This method is new in cbapi 1.1.0 and is part of five new query filters available when communicating with a CB Response 6.1 server. These filters are accessible via methods on the `Process` Query object. These new methods are:

- `.group_by()` - Group the result set by a field in the response. Typically you will want to group by `id`, which will ensure that the result set only has one result per *process* rather than one result per *event segment*. For more information on processes, process segments, and how segments are stored in CB Response 6.0, see the Process API Changes for CB Response 6.0 page on the Developer Network website.
- `.min_last_update()` - Only return processes that have events after a given date/time stamp (relative to the individual sensor's clock)
- `.max_last_update()` - Only return processes that have events before a given date/time stamp (relative to the individual sensor's clock)
- `.min_last_server_update()` - Only return processes that have events after a given date/time stamp (relative to the CB Response server's clock)
- `.max_last_server_update()` - Only return processes that have events before a given date/time stamp (relative to the CB Response server's clock)

Examples for new Filters

Let's take a look at an example:

```
>>> from datetime import datetime, timedelta
>>> yesterday = datetime.utcnow() - timedelta(days=1)       # Get "yesterday" in GMT
>>> for proc in c.select(Process).where("process_name:cmd.exe").min_last_
↪update(yesterday):
...     print proc.id, proc.segment
DEBUG:cbapi.connection:HTTP GET /api/v1/process?cb.min_last_update=2017-05-21T18%3A41
↪%3A58Z&cb.urlver=1&facet=false&q=process_name%3Acmd.exe&rows=100&sort=last_
↪update+desc&start=0 took 2.164s (response 200)
00000001-0000-0e48-01d2-c2a397f4cfe0 1495465643405
00000001-0000-0e48-01d2-c2a397f4cfe0 1495465407157
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463680155
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463807694
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463543944
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463176570
00000001-0000-0e48-01d2-c2a397f4cfe0 1495463243492
```

Notice that the "same" process ID is returned seven times, but with seven different segment IDs. CB Response will return *every* process event segment that matches a given query, in this case, any event segment that contains the process command name `cmd.exe`.

That is, however, most likely not what you wanted. Instead, you'd like a list of the *unique* processes associated with the command name `cmd.exe`. Just add the `.group_by("id")` filter to your query:

```
>>> for proc in c.select(Process).where("process_name:cmd.exe").min_last_
↪update(yesterday).group_by("id"):
...     print proc.id, proc.segment
DEBUG:cbapi.connection:HTTP GET /api/v1/process?cb.group=id&cb.min_last_update=2017-
↪05-21T18%3A41%3A58Z&cb.urlver=1&facet=false&q=process_name%3Acmd.exe&rows=100&
↪sort=last_update+desc&start=0 took 2.163s (response 200)
00000001-0000-0e48-01d2-c2a397f4cfe0 1495465643405
```

CHAPTER 5

API Documentation

Once you have read the User Guide, you can view examples on GitHub or try writing code of your own. You can use the full API documentation below to see all the methods available in CBAPI and unlock the full functionality of the SDK.

5.1 EDR (CB Response) API

5.1.1 Main Interface

To use cbapi with Carbon Black EDR (Response), you will be using the CbResponseAPI. The CbResponseAPI object then exposes two main methods to access data on the Carbon Black server: `select` and `create`.

class cbapi.response.rest_api.**CbResponseAPI**(**args*, ***kwargs*)

The main entry point into the Carbon Black EDR API. Note that calling this will automatically connect to the Carbon Black server in order to verify connectivity and get the server version.

Parameters

- **profile** (*str*) – (optional) Use the credentials in the named profile when connecting to the Carbon Black server. Uses the profile named 'default' when not specified.
- **url** (*str*) – (optional, discouraged) Instead of using a credential profile, pass URL and API token to the constructor.
- **token** (*str*) – (optional, discouraged) API token
- **ssl_verify** (*bool*) – (optional, discouraged) Enable or disable SSL certificate verification

Usage:

```
>>> from cbapi import CbResponseAPI
>>> cb = CbResponseAPI(profile="production")
```

api_json_request(*method*, *uri*, ***kwargs*)
Submit a request to the server.

Args: method (str): HTTP method to use. uri (str): URI to submit the request to. **kwargs (dict): Additional arguments.

Returns: object: Result of the operation.

Raises: ServerError: If there's an error output from the server.

create(*cls*, *data=None*)
Create a new object.

Args: cls (class): The Model class (only some models can be created, for example, Feed, Notification, ...) data (object): The data used to initialize the new object

Returns: Model: An empty instance of the model class.

Raises: ApiError: If the Model cannot be created.

create_new_partition()
Create a new Solr time partition for event storage. Available in Carbon Black EDR 6.1 and above. This will force roll-over current hot partition into warm partition (by renaming it to a time-stamped name) and create a new hot partition ("writer").

Returns Nothing if successful.

Raises

- ***ApiError*** – if there was an error creating the new partition.
- ***ServerError*** – if there was an error creating the new partition.

dashboard_statistics()
Retrieve dashboard statistics from the Carbon Black EDR server.

Returns Dictionary with information retrieved from the `/api/v1/dashboard/statistics` API route

Return type dict

delete_object(*uri*)
Send a DELETE request to the specified URI.

Args: uri (str): The URI to send the DELETE request to.

Returns: object: The return data from the DELETE request.

from_ui(*uri*)
Retrieve a Carbon Black EDR object based on URL from the Carbon Black EDR web user interface.

For example, calling this function with `https://server/#/analyze/00000001-0000-0554-01d1-3bc4553b8c9f/1` as the `uri` argument will return a new :py:class: cbapi.response.models.Process class initialized with the process GUID from the URL.

Parameters **uri** (*str*) – Web browser URL from the CB web interface

Returns the appropriate model object for the URL provided

Raises ***ApiError*** – if the URL does not correspond to a recognized model object

get_object(*uri*, *query_parameters=None*, *default=None*)
Submit a GET request to the server and parse the result as JSON before returning.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response.

Returns: object: Result of the GET request.

get_raw_data(*uri*, *query_parameters=None*, *default=None*, ***kwargs*)
Submit a GET request to the server and return the result without parsing it.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response. **kwargs:

Returns: object: Result of the GET request.

info()
Retrieve basic version information from the Carbon Black DER server.

Returns Dictionary with information retrieved from the `/api/info` API route

Return type dict

license_request()
Retrieve license request block from the Carbon Black EDR server.

Returns License request block

Return type str

post_object(*uri*, *body*, ***kwargs*)
Send a POST request to the specified URI.

Args: uri (str): The URI to send the POST request to. body (object): The data to be sent in the body of the POST request. **kwargs:

Returns: object: The return data from the POST request.

put_object(*uri*, *body*, ***kwargs*)
Send a PUT request to the specified URI.

Args: uri (str): The URI to send the PUT request to. body (object): The data to be sent in the body of the PUT request. **kwargs:

Returns: object: The return data from the PUT request.

raise_unless_json(*ret*, *expected*)
Raise a ServerError unless we got back an HTTP 200 response with JSON containing all the expected values.

Args: ret (object): Return value to be checked. expected (dict): Expected keys and values that need to be found in the JSON response.

Raises: ServerError: If the HTTP response is anything but 200, or if the expected values are not found.

select(*cls*, *unique_id=None*, **args*, ***kwargs*)
Prepare a query against the Carbon Black data store.

Args: cls (class): The Model class (for example, Computer, Process, Binary, FileInstance) to query unique_id (optional): The unique id of the object to retrieve, to retrieve a single object by ID *args: **kwargs:

Returns: object: An instance of the Model class if a unique_id is provided, otherwise a Query object

update_license(*license_block*)
Upload new license to the Carbon Black EDR server.

Parameters **license_block** (*str*) – Licence block provided by Carbon Black support

Raises ***ServerError*** – if the license is not accepted by the Carbon Black server

url
Return the connection URL.

Returns: str: The connection URL.

5.1.2 Queries

class cbapi.response.query.**Query**(*doc_class*, *cb*, *query=None*, *raw_query=None*)
Represents a prepared query to the Carbon Black EDR server.

This object is returned as part of a `CbResponseAPI.select()` operation on Process and Binary objects from the Carbon Black EDR server. You should not have to create this class yourself.

The query is not executed on the server until it's accessed, either as an iterator (where it will generate values on demand as they're requested) or as a list (where it will retrieve the entire result set and save to a list). You can also call the Python built-in `len()` on this object to retrieve the total number of items matching the query.

The syntax for query :py:meth:where and :py:meth:sort methods can be found in the Query Reference posted on the Carbon Black Developer Network website.

Examples:

```
>>> cb = CbResponseAPI()
>>> query = cb.select(Process)                          # returns a Query object␣
↪matching all Processes
>>> query = query.where("process_name:notepad.exe") # add a filter to this Query
>>> query = query.sort("last_update desc")              # sort by last update time,␣
↪most recent first
>>> for proc in query:                                  # uses the iterator to␣
↪retrieve all results
>>>     print("{0} {1}".format(proc.username, proc.hostname))
>>> processes = query[:10]                              # retrieve the first ten␣
↪results
>>> len(query)                                          # retrieve the total count
```

Notes:

- The slicing operator only supports start and end parameters, but not step. `[1:-1]` is legal, but `[1:2:-1]` is not.
- You can chain where clauses together to create AND queries; only objects that match all `where` clauses will be returned.

and_(*new_query*)
Add a filter to this query. Equivalent to calling *where()* on this object.

Parameters **new_query** (*str*) – Query string - see the Query Reference.

Returns Query object

Return type *Query*

facets(**args*)
Retrieve a dictionary with the facets for this query.

Parameters **args** – Any number of fields to use as facets

Returns Facet data

Return type dict

sort(*new_sort*)
Set the sort order for this query.

Parameters **new_sort** (*str*) – New sort order - see the Query Reference.

Returns Query object

Return type *Query*

where(*new_query*)
Add a filter to this query.

Parameters **new_query** (*str*) – Query string - see the Query Reference.

Returns Query object

Return type *Query*

class cbapi.response.models.**ProcessQuery**(*doc_class*, *cb*, *query=None*, *raw_query=None*)

group_by(*field_name*)
Set the group-by field name for this query. Typically, you will want to set this to 'id' if you only want one result per process.

This method is only available for EDR servers 6.0 and above. Calling this on a Query object connected to a EDR 5.x server will simply result in a no-op.

Parameters **field_name** (*str*) – Field name to group the result set by.

Returns Query object

Return type *ProcessQuery*

max_children(*num_children*)
Sets the number of children to fetch with the process

This method is only available for EDR servers 6.0 and above. Calling this on a Query object connected to a EDR 5.x server will simply result in a no-op.

Default 15

Parameters **num_children** (*int*) – Number of children to fetch with process

Returns Query object

Return type *ProcessQuery*

max_last_server_update(*v*)
Set the maximum last update time (relative to server) for this query. The timestamp can be expressed either as a `datetime` like object or as an ISO 8601 string formatted timestamp such as 2017-04-29T04:21:18Z. If a `datetime` like object is provided, it is assumed to be in GMT time zone.

This option will limit the number of Solr cores that need to be searched for events that match the query.

This method is only available for EDR servers 6.0 and above. Calling this on a Query object connected to a EDR 5.x server will simply result in a no-op.

Parameters **v** (*str*) – Timestamp (either string or datetime object).

Returns Query object

Return type *ProcessQuery*

max_last_update(*v*)
Set the maximum last update time (relative to sensor) for this query. The timestamp can be expressed either

as a `datetime` like object or as an ISO 8601 string formatted timestamp such as 2017-04-29T04:21:18Z. If a `datetime` like object is provided, it is assumed to be in GMT time zone.

This option will limit the number of Solr cores that need to be searched for events that match the query.

This method is only available for EDR servers 6.0 and above. Calling this on a Query object connected to a EDR 5.x server will simply result in a no-op.

Parameters **v** (*str*) – Timestamp (either string or datetime object).

Returns Query object

Return type *ProcessQuery*

min_last_server_update(*v*)

Set the minimum last update time (relative to server) for this query. The timestamp can be expressed either as a `datetime` like object or as an ISO 8601 string formatted timestamp such as 2017-04-29T04:21:18Z. If a `datetime` like object is provided, it is assumed to be in GMT time zone.

This option will limit the number of Solr cores that need to be searched for events that match the query.

This method is only available for EDR servers 6.0 and above. Calling this on a Query object connected to a EDR 5.x server will simply result in a no-op.

Parameters **v** (*str*) – Timestamp (either string or datetime object).

Returns Query object

Return type *ProcessQuery*

min_last_update(*v*)

Set the minimum last update time (relative to sensor) for this query. The timestamp can be expressed either as a `datetime` like object or as an ISO 8601 string formatted timestamp such as 2017-04-29T04:21:18Z. If a `datetime` like object is provided, it is assumed to be in GMT time zone.

This option will limit the number of Solr cores that need to be searched for events that match the query.

This method is only available for EDR servers 6.0 and above. Calling this on a Query object connected to a EDR 5.x server will simply result in a no-op.

Parameters **v** (*str*) – Timestamp (either string or datetime object).

Returns Query object

Return type *ProcessQuery*

use_comprehensive_search()

Set the *comprehensive_search* flag on the Process query.

Returns new Query object

Return type *ProcessQuery*

class cbapi.response.models.**ThreatReportQuery**(*doc_class*, *cb*, *query=None*, *raw_query=None*)

class cbapi.response.models.**AlertQuery**(*doc_class*, *cb*, *query=None*, *raw_query=None*)

set_ignored(*ignored_flag=True*, *status='False Positive'*)

Ignore all future Alerts from the Report that triggered this Alert.

5.1.3 Models

class cbapi.response.models.**Process**(*cb*, *procguid*, *segment=None*, *max_children=15*, *initial_data=None*, *force_init=False*, *suppressed_process=False*)

all_events
Returns a list of all events associated with this process across all segments, sorted by timestamp

Returns list of CbEvent objects

all_events_segment
Returns a list of all events associated with this process segment, sorted by timestamp

Returns list of CbEvent objects

binary
Joins this attribute with the *Binary* object associated with this Process object

Example

```
>>> process_obj = c.select(Process).where('process_name:svch0st.exe')[0]
>>> binary_obj = process_obj.binary
>>> print(binary_obj.signed)
False
```

childprocs
Generator that returns `CbChildProcEvent` objects associated with this process

children
Generator that returns `CbChildProcEvent` objects associated with this process

cmdline

Returns Returns the command line of the process

Return type string

comms_ip
Returns ascii representation of the ip address used to communicate with the EDR Server

crossprocs
Generator that returns `CbCrossProcEvent` objects associated with this process

depth
Returns the depth of this process from the "root" system process

Returns integer representing the depth of the process (0 is the root system process). To prevent infinite recursion, a maximum depth of 500 processes is enforced.

end
Returns the end time of the process (based on the last event received). If the process has not yet exited, "end" will return None.

Returns datetime object of the last event received for the process, if it has terminated. Otherwise, None.

filemods
Generator that returns `CbFileModEvent` objects associated with this process

find_file_writes(*filename*)
Returns a list of file writes with the specified filename

Parameters **filename** (*str*) – filename to match on file writes

Returns Returns a list of file writes with the specified filename

Return type list

interface_ip
Returns ascii representation of the ip address of the interface used to communicate with the EDR server. If using NAT, this will be the "internal" IP address of the sensor.

last_server_update
Returns a pretty version of when this process last updated

last_update
Returns a pretty version of when this process last updated

max_last_server_update
Returns a pretty version of the latest event in this process segment

max_last_update
Returns a pretty version of the latest event in this process segment

min_last_server_update
Returns a pretty version of the earliest event in this process segment

min_last_update
Returns a pretty version of the earliest event in this process segment

modloads
Generator that returns *:py:class:CbModLoadEvent* associated with this process

netconns
Generator that returns `CbNetConnEvent` objects associated with this process

classmethod new_object(*cb*, *item*, *max_children=15*)
Create a new instance of the object from item data.

Args: cb (BaseAPI): Reference to the CBAPI object. item (dict): Item data, as retrieved from the server.

Returns: object: New instance of the object.

parent
Returns the parent Process object if one exists

parent_md5
Workaround since parent_md5 silently disappeared in EDR 6.x

processblocks
Generator that returns `CbProcessBlockEvent` objects associated with this process

refresh()
Refresh the object from the Carbon Black server.

regmods
Generator that returns `CbRegModEvent` objects associated with this process

sensor
Joins this attribute with the *Sensor* object associated with this Process object

Example

```
>>> process_obj = c.select(Process).where('process_name:svch0st.exe')[0]
>>> sensor_obj = process.sensor
>>> print(sensor_obj.computer_dns_name)
hyperv-win7-x86
```

start
Returns the start time of the process

unsigned_modloads
Returns all unsigned module loads. This is useful to filter out all Microsoft signed DLLs

username
Returns the username of the owner of this process

walk_children(*callback*, *max_depth=0*, *depth=0*)
Walk down the execution chain while calling the specified callback function at each depth.

Example

```
>>> def proc_callback(parent_proc, depth):
...     print(parent_proc.cmdline, depth)
>>>
>>> process = c.select(Process).where('process_name:svch0st.exe')[0]
>>> process.walk_children(proc_callback, depth=2)
(u'cmd.exe \c ipconfig', 2)
(u'cmd.exe \\c ipconfig', 2)
(u'cmd.exe /c ipconfig', 2)
(u'ipconfig', 3)
(u'cmd.exe /c ipconfig.exe /all', 2)
(u'cmd.exe \c ipconfig', 2)
(u'cmd.exe \\c ipconfig', 2)
(u'cmd.exe /c ipconfig', 2)
(u'ipconfig', 3)
(u'cmd.exe /c ipconfig.exe /all', 2)
```

Parameters

- **callback** (*func*) – Callback function used for execution at each depth. This function is executed with the parent process object and depth as parameters.
- **max_depth** (*int*) – Max number of iterations down the execution chain.
- **depth** (*int*) – Number of iterations down the execution chain

Returns None

walk_parents(*callback*, *max_depth=0*, *depth=0*)
Walk up the execution chain while calling the specified callback function at each depth.

Example

```
>>> def proc_callback(parent_proc, depth):
...     print(parent_proc.cmdline, depth)
>>>
>>> process = c.select(Process).where('process_name:ipconfig.exe')[0]
>>> process.walk_parents(proc_callback)
(u'cmd.exe /c ipconfig.exe', 0)
(u'c:\windows\carbonblack\cb.exe', 1)
(u'C:\Windows\system32\services.exe', 2)
(u'wininit.exe', 3)
(u'\SystemRoot\System32\smss.exe 00000000 00000040 ', 4)
(u'\SystemRoot\System32\smss.exe', 5)
(u'', 6)
```

Parameters

- **callback** (*func*) – Callback function used for execution at each depth. This function is executed with the parent process object and depth as parameters.
- **max_depth** (*int*) – Max number of iterations up the execution chain
- **depth** (*int*) – Number of iterations up the execution chain.

Returns None

webui_link
Returns the Carbon Black EDR Web UI link associated with this process

class cbapi.response.models.**Binary**(*cb*, *md5sum*, *initial_data=None*, *force_init=False*)

class FrequencyData
Class containing frequency information about a binary

Parameters

- **computer_count** (*int*) – Number of endpoints this binary resides
- **process_count** (*int*) – Number of executions
- **all_process_count** (*int*) – Number of all process documents
- **module_frequency** (*int*) – process_count / all_process_count

Create new instance of FrequencyData(computer_count, process_count, all_process_count, module_frequency)

class SigningData
Class containing binary signing information

Parameters

- **result** (*str*) – Signed or Unsigned
- **publisher** (*str*) – Singnature publisher
- **issuer** (*str*) – Signature issuer
- **subject** (*str*) – Signing subject
- **sign_time** (*str*) – Binary signed time
- **program_name** (*str*) – Binary program name

Create new instance of SigningData(result, publisher, issuer, subject, sign_time, program_name)

class VersionInfo
Class containing versioning information about a binary

Parameters

- **file_desc** (*str*) – File description
- **file_version** (*str*) – File version
- **product_name** (*str*) – Product Name
- **product_version** (*str*) – Product version
- **company_name** (*str*) – Company Name
- **legal_copyright** (*str*) – Copyright
- **original_filename** (*str*) – Original File name of this binary

Create new instance of VersionInfo(file_desc, file_version, product_name, product_version, company_name, legal_copyright, original_filename)

banned
Returns *BannedHash* object if this Binary's hash has been banned, otherwise returns *False*

digsig_issuer
Returns the Digital Signature Issuer

digsig_prog_name
Returns the Digital Signature Program Name

digsig_publisher
Returns the Digital Signature Publisher

digsig_sign_time
Returns the Digital Signature signing time

digsig_subject
Returns the Digital Signature subject

endpoints
Return a list of endpoints this binary resides

file
Returns a file pointer to this binary

Example

```
>>> process_obj = c.select(Process).where("process_name:svch0st.exe").first()
>>> binary_obj = process_obj.binary
>>> print(binary_obj.file.read(2))
MZ
```

frequency
Returns *FrequencyData* information about the binary.

Example

```
>>> process_obj = c.select(Process).where('process_name:svch0st.exe').first()
>>> binary_obj = process_obj.binary
>>> print(binary_obj.frequency)
FrequencyData(computer_count=1, process_count=5, all_process_count=4429,␣
↪module_frequency=0.0011289230074508 92)
```

icon
Returns the raw icon of this Binary. This data is not encoded.

is_64bit
Returns True if the Binary is an AMD64 or x64 (64-bit) Executable

is_executable_image
Returns True if the Binary is executable

classmethod new_object(*cb*, *item*)
Create a new instance of the object from item data.

Args: cb (BaseAPI): Reference to the CBAPI object. item (dict): Item data, as retrieved from the server.

Returns: object: New instance of the object.

observed_filenames
Returns a list of all observed file names associated with this Binary

signed
Returns True if the binary is signed.

signing_data
Returns *SigningData* object which contains: Digital Signature Result, Digital Signature publisher, Issuer, Subject, Signing Time, Program Name

size
Returns the size of the Binary

version_info
Returns a *VersionInfo* object containing detailed information: File Descritpion, File Version, Product Name, Product Version, Company Name, Legal Copyright, and Original FileName

webui_link
Returns the Carbon Black EDR Web UI link associated with this Binary object

class cbapi.response.models.**Sensor**(**args*, ***kwargs*)
Represents a Sensor object in the Carbon Black server.

class NetworkAdapter(*macaddr*, *ipaddr*)
Create new instance of NetworkAdapter(macaddr, ipaddr)

ipaddr
Alias for field number 1

macaddr
Alias for field number 0

activity_stats
Returns a list of activity statistics from the associated EDR Sensor

dns_name
Returns the DNS name associated with this sensor object. This is the same as 'computer_dns_name'.

flush_events()
Performs a flush of events for this EDR Sensor

Warning This may cause a significant amount of network traffic from this sensor to the EDR Server

group

Getter

Returns the sensor's group id.

Setter

Allows access to set the sensor's group id

hostname
Returns the hostname associated with this sensor object. This is the same as 'computer_name'

isolate(*timeout=None*)
Turn on network isolation for this EDR Sensor.

This function will block and only return when the isolation is complete, or if a timeout is reached. By default, there is no timeout. You can specify a timeout period (in seconds) in the "timeout" parameter to this function. If a timeout is specified and reached before the sensor is confirmed isolated, then this function will throw a TimeoutError.

Returns True if sensor is isolated

Raises ***TimeoutError*** – if sensor does not isolate before timeout is reached

lr_session()
Retrieve a Live Response session object for this Sensor.

Returns Live Response session object

Return type cbapi.live_response_api.LiveResponseSession

Raises *ApiError* – if there is an error establishing a Live Response session for this Sensor

network_interfaces
Returns a list of networks adapters on the sensor

os
Returns the operating system display string of the sensor

queued_stats
Returns a list of status and size of the queued event logs from the associated EDR Sensor

Example

```
>>> sensor_obj = c.select(Sensor).where("ip:192.168").first()
>>> pprint.pprint(sensor_obj.queued_stats)
[{u'id': u'355509',
  u'num_eventlog_bytes': u'0',
  u'num_eventlogs': u'0',
  u'num_storefile_bytes': u'0',
  u'num_storefiles': 0,
  u'sensor_id': 1,
  u'timestamp': u'2016-10-17 19:08:09.645294-05:00'}]
```

resource_status
Returns a list of memory statistics used by the EDR Sensor

restart_sensor()
Restarts the Carbon Black sensor (*not* the underlying endpoint operating system).

This simply sets the flag to ask the sensor to restart the next time it checks into the EDR server, it does not wait for the sensor to restart.

sid
Security Identifier being used by the EDR Sensor

unisolate(*timeout=None*)
Turn off network isolation for this EDR Sensor.

This function will block and only return when the isolation is removed, or if a timeout is reached. By default, there is no timeout. You can specify a timeout period (in seconds) in the "timeout" parameter to this function. If a timeout is specified and reached before the sensor is confirmed unisolated, then this function will throw a TimeoutError.

Returns True if sensor is unisolated

Raises *TimeoutError* – if sensor does not unisolate before timeout is reached

webui_link
Returns the Carbon Black EDR Web UI link associated with this Sensor

class cbapi.response.models.**Feed**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)
Represents a Feed object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

actions

Returns Returns all `FeedAction` objects associated with this feed

Return type `response.rest_api.Query`

search_binaries(*min_score=None*, *max_score=None*)
Perform a *Binary* search within this feed that satisfies min_score and max_score :param min_score: minimum feed score :param max_score: maximum feed score :return: Returns a `response.rest_api.Query` object within the appropriate

search parameters for binaries

Return type `response.rest_api.Query`

search_processes(*min_score=None*, *max_score=None*)
Perform a *Process* search within this feed that satisfies min_score and max_score

Parameters

- **min_score** – minimum feed score
- **max_score** – maximum feed score

Returns Returns a `response.rest_api.Query` object with the appropriate search parameters for processes

Return type `response.rest_api.Query`

class cbapi.response.models.**BannedHash**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)
Represents a BannedHash object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

binary
Joins this attribute with the *Binary* object associated with this Banned Hash object

class cbapi.response.models.**Watchlist**(**args*, ***kwargs*)
Represents a Watchlist object in the Carbon Black server.

Variables

- ***description*** – A description of the watchlist.
- **search_query** – URL encoded search query associated with this watchlist.
- **index_type** – Index to search for this watchlist. Must be either 'events' (Processes) or 'modules' (Binaries)

facets
Returns facets from the search associated with the watchlist query

Returns dictionary of facets as keys

Return type dict

query

Getter

Returns the query associated with this watchlist.

Setter

Allows access to set the query associated with this watchlist

search()
Creates a search based on the watchlist's search parameter

Returns a *Process* response.rest_api.Query or Binary response.rest_api.Query

Return type response.rest_api.Query

class cbapi.response.models.**Alert**(*cb*, *alert_id*, *initial_data=None*)
Represents a Alert object in the Carbon Black server.

set_ignored(*ignored_flag=True*, *status='False Positive'*)
Ignore all future Alerts from the Report that triggered this Alert.

5.1.4 Live Response

class cbapi.live_response_api.**CbLRSessionBase**(*cblr_manager*, *session_id*, *sensor_id*, *session_data=None*)
A Live Response session that interacts with a remote machine.

Initialize the CbLRSessionBase.

Args: cblr_manager (CbLRManagerBase): The Live Response manager governing this session. session_id (str): The ID of this session. sensor_id (int): The ID of the sensor (remote machine) we're connected to. session_data (dict): Additional session data.

File Operations

CbLRSessionBase.**get_file**(*file_name*, *timeout=None*, *delay=None*)
Retrieve contents of the specified file on the remote machine.

Args: file_name (str): Name of the file to be retrieved. timeout (int): Timeout for the operation. delay (float): TBD

Returns: str: Contents of the specified file.

CbLRSessionBase.**delete_file**(*filename*)
Delete the specified file name on the remote machine.

Args: filename (str): Name of the file to be deleted.

CbLRSessionBase.**put_file**(*infp*, *remote_filename*)
Create a new file on the remote machine with the specified data.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: ... lr_session.put_file(open("test.txt", "rb"), r"c:test.txt")

Args: infp (object): Python file-like containing data to upload to the remote endpoint. remote_filename (str): File name to create on the remote endpoint.

CbLRSessionBase.**list_directory**(*dir_name*)
List the contents of a directory on the remote machine.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: ... pprint.pprint(lr_session.list_directory('C:\temp\')) [{u'attributes': [u'DIRECTORY'],

u'create_time': 1471897244, u'filename': u'.', u'last_access_time': 1476390670, u'last_write_time': 1476390670, u'size': 0},

{u'attributes': [u'DIRECTORY'], u'create_time': 1471897244, u'filename': u'..', u'last_access_time': 1476390670, u'last_write_time': 1476390670, u'size': 0},

{u'attributes': [u'ARCHIVE'], u'create_time': 1476390668, u'filename': u'test.txt', u'last_access_time': 1476390668, u'last_write_time': 1476390668, u'size': 0}]

Args: dir_name (str): Directory to list. This parameter should end with the path separator.

Returns: list: A list of dicts, each one describing a directory entry.

CbLRSessionBase.**create_directory**(*dir_name*)
Create a directory on the remote machine.

Args: dir_name (str): The new directory name.

CbLRSessionBase.**walk**(*top*, *topdown=True*, *onerror=None*, *followlinks=False*)
Perform a full directory walk with recursion into subdirectories on the remote machine.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: ... for entry in lr_session.walk(directory_name): ... print(entry) ('C:\temp', [u'dir1', u'dir2'], [u'file1.txt'])

Args: top (str): Directory to recurse on. topdown (bool): If True, start output from top level directory. onerror (func): Callback if an error occurs. This function is called with one argument (the exception

that occurred).

followlinks (bool): True to follow symbolic links.

Returns: list: List of tuples containing directory name, subdirectory names, file names.

Registry Operations

CbLRSessionBase.**get_registry_value**(*regkey*)
Return the associated value of the specified registry key on the remote machine.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: >>> pprint.pprint(lr_session.get_registry_value('HKLM\SYSTEM\CurrentControlSet\services\ACPI\Start')) {u'value_data': 0, u'value_name': u'Start', u'value_type': u'REG_DWORD'}

Args: regkey (str): The registry key to retrieve.

Returns: dict: A dictionary with keys of: value_data, value_name, value_type.

CbLRSessionBase.**set_registry_value**(*regkey*, *value*, *overwrite=True*, *value_type=None*)
Set a registry value on the specified registry key on the remote machine.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: ... lr_session.set_registry_value('HKLM\SYSTEM\CurrentControlSet\services\ACPI\testvalue', 1)

Args: regkey (str): The registry key to set. value (object): The value data. overwrite (bool): If True, any existing value will be overwritten. value_type (str): The type of value. Examples: REG_DWORD, REG_MULTI_SZ, REG_SZ

CbLRSessionBase.**delete_registry_value**(*regkey*)
Delete a registry value on the remote machine.

Args: regkey (str): The registry value to delete.

CbLRSessionBase.**create_registry_key**(*regkey*)
Create a new registry key on the remote machine.

Args: regkey (str): The registry key to create.

CbLRSessionBase.**delete_registry_key**(*regkey*)
Delete a registry key on the remote machine.

Args: regkey (str): The registry key to delete.

CbLRSessionBase.**list_registry_keys_and_values**(*regkey*)

Enumerate subkeys and values of the specified registry key on the remote machine.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: >>> pprint.pprint(lr_session.list_registry_keys_and_values('HKLM\SYSTEM\CurrentControlSet\services\ACPI')) {'sub_keys': [u'Parameters', u'Enum'],

'values': [{u'value_data': 0,

u'value_name': u'Start', u'value_type': u'REG_DWORD'},

{u'value_data': 1, u'value_name': u'Type', u'value_type': u'REG_DWORD'},

{u'value_data': 3, u'value_name': u'ErrorControl', u'value_type': u'REG_DWORD'},

{u'value_data': u'system32\drivers\ACPI.sys', u'value_name': u'ImagePath', u'value_type': u'REG_EXPAND_SZ'},

{u'value_data': u'Microsoft ACPI Driver', u'value_name': u'DisplayName', u'value_type': u'REG_SZ'},

{u'value_data': u'Boot Bus Extender', u'value_name': u'Group', u'value_type': u'REG_SZ'},

{u'value_data': u'acpi.inf_x86_neutral_ddd3c514822f1b21', u'value_name': u'DriverPackageId', u'value_type': u'REG_SZ'},

{u'value_data': 1, u'value_name': u'Tag', u'value_type': u'REG_DWORD'}]}

Args: regkey (str): The registry key to enumerate.

Returns:

dict: A dictionary with two keys, 'sub_keys' (a list of subkey names) and 'values' (a list of dicts containing value data, name, and type).

CbLRSessionBase.**list_registry_keys**(*regkey*)

Enumerate all registry values from the specified registry key on the remote machine.

Args: regkey (str): The registry key to enumerate.

Returns: list: List of values for the registry key.

Process Operations

CbLRSessionBase.**kill_process**(*pid*)

Terminate a process on the remote machine.

Args: pid (int): Process ID to be terminated.

Returns: bool: True if success, False if failure.

CbLRSessionBase.**create_process**(*command_string*, *wait_for_output=True*, *remote_output_file_name=None*, *working_directory=None*, *wait_timeout=30*, *wait_for_completion=True*)

Create a new process on the remote machine with the specified command string.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: ... print(lr_session.create_process(r'cmd.exe /c "ping.exe 192.168.1.1"')) Pinging 192.168.1.1 with 32 bytes of data: Reply from 192.168.1.1: bytes=32 time<1ms TTL=64

Args: command_string (str): Command string used for the create process operation. wait_for_output (bool): True to block on output from the new process (execute in foreground).

This will also set wait_for_completion (below).

remote_output_file_name (str): The remote output file name used for process output. working_directory (str): The working directory of the create process operation. wait_timeout (int): Timeout used for this command. wait_for_completion (bool): True to wait until the process is completed before returning.

Returns: str: The output of the process.

CbLRSessionBase.**list_processes**()

List currently running processes on the remote machine.

Example: >>> with c.select(Sensor, 1).lr_session() as lr_session: ... print(lr_session.list_processes()[0]) {u'command_line': u'',

u'create_time': 1476260500, u'parent': 0, u'parent_guid': u'00000001-0000-0000-0000-000000000000', u'path': u'', u'pid': 4, u'proc_guid': u'00000001-0000-0004-01d2-2461a85e4546', u'sid': u's-1-5-18', u'username': u'NT AUTHORITY\SYSTEM'}

Returns: list: A list of dicts describing the processes.

5.2 Carbon Black App Control (CB Protection) API

5.2.1 Main Interface

To use cbapi with Carbon Black App Control (CB Protection), you will be using the CbProtectionAPI. The CbProtectionAPI object then exposes two main methods to select data on the Carbon Black server:

class cbapi.protection.rest_api.**CbProtectionAPI**(**args*, ***kwargs*)

The main entry point into the Carbon Black App Control API.

Parameters **profile** (*str*) – (optional) Use the credentials in the named profile when connecting to the Carbon Black server. Uses the profile named 'default' when not specified.

Usage:

```
>>> from cbapi import CbProtectionAPI
>>> cb = CbProtectionAPI(profile="production")
```

api_json_request(*method*, *uri*, ***kwargs*)

Submit a request to the server.

Args: method (str): HTTP method to use. uri (str): URI to submit the request to. **kwargs (dict): Additional arguments.

Returns: object: Result of the operation.

Raises: ServerError: If there's an error output from the server.

create(*cls*, *data=None*)

Create a new object.

Args: cls (class): The Model class (only some models can be created, for example, Feed, Notification, ...) data (object): The data used to initialize the new object

Returns: Model: An empty instance of the model class.

Raises: ApiError: If the Model cannot be created.

delete_object(*uri*)
Send a DELETE request to the specified URI.

Args: uri (str): The URI to send the DELETE request to.

Returns: object: The return data from the DELETE request.

get_object(*uri*, *query_parameters=None*, *default=None*)
Submit a GET request to the server and parse the result as JSON before returning.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response.

Returns: object: Result of the GET request.

get_raw_data(*uri*, *query_parameters=None*, *default=None*, ***kwargs*)
Submit a GET request to the server and return the result without parsing it.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response. **kwargs:

Returns: object: Result of the GET request.

post_object(*uri*, *body*, ***kwargs*)
Send a POST request to the specified URI.

Args: uri (str): The URI to send the POST request to. body (object): The data to be sent in the body of the POST request. **kwargs:

Returns: object: The return data from the POST request.

put_object(*uri*, *body*, ***kwargs*)
Send a PUT request to the specified URI.

Args: uri (str): The URI to send the PUT request to. body (object): The data to be sent in the body of the PUT request. **kwargs:

Returns: object: The return data from the PUT request.

raise_unless_json(*ret*, *expected*)
Raise a ServerError unless we got back an HTTP 200 response with JSON containing all the expected values.

Args: ret (object): Return value to be checked. expected (dict): Expected keys and values that need to be found in the JSON response.

Raises: ServerError: If the HTTP response is anything but 200, or if the expected values are not found.

select(*cls*, *unique_id=None*, **args*, ***kwargs*)
Prepares a query against the Carbon Black data store.

Parameters

- **cls** (*class*) – The Model class (for example, Computer, Process, Binary, FileInstance) to query
- **unique_id** – (optional) The unique id of the object to retrieve, to retrieve a single object by ID

Returns An instance of the Model class if a unique_id is provided, otherwise a Query object

url
Return the connection URL.

Returns: str: The connection URL.

5.2.2 Queries

class cbapi.protection.rest_api.**Query**(*doc_class*, *cb*, *query=None*)

Represents a prepared query to the Carbon Black App Control server.

This object is returned as part of a *CbProtectionAPI.select()* operation on models requested from the Carbon Black App Control server. You should not have to create this class yourself.

The query is not executed on the server until it's accessed, either as an iterator (where it will generate values on demand as they're requested) or as a list (where it will retrieve the entire result set and save to a list). You can also call the Python built-in `len()` on this object to retrieve the total number of items matching the query.

The syntax for query :py:meth:where and :py:meth:sort methods can be found in the App Control REST API reference posted on the Carbon Black Developer Network website.

Examples:

```
>>> from cbapi.protection import CbProtectionAPI, Computer
>>> cb = CbProtectionAPI()
>>> query = cb.select(Computer)                        # returns a Query object␣
↪matching all Computers
>>> query = query.where("ipAddress:10.201.2.*")        # add a filter to this Query
>>> query = query.sort("processorSpeed DESC")          # sort by computer processor␣
↪speed, descending
>>> for comp in query:                                 # uses the iterator to␣
↪retrieve all results
>>>     print(comp.name)
>>> comps = query[:10]                                 # retrieve the first ten␣
↪results
>>> len(query)                                         # retrieve the total count
```

Notes:

- The slicing operator only supports start and end parameters, but not step. `[1:-1]` is legal, but `[1:2:-1]` is not.
- You can chain where clauses together to create AND queries; only objects that match all `where` clauses will be returned.

and_(*q*)

Add a filter to this query. Equivalent to calling *where()* on this object.

Parameters **q** (*str*) – Query string - see the App Control REST API reference.

Returns Query object

Return type *Query*

sort(*new_sort*)

Set the sort order for this query.

Parameters **new_sort** (*str*) – Sort order - see the App Control REST API reference.

Returns Query object

Return type *Query*

where(*q*)

Add a filter to this query.

Parameters **q** (*str*) – Query string - see the App Control REST API reference.

Returns Query object

Return type *Query*

5.2.3 Models

class cbapi.protection.models.**ApprovalRequest**(*cb*, *model_unique_id*, *initial_data=None*)

ResolutionApproved = 2

ResolutionInstaller = 4

ResolutionNotResolved = 0

ResolutionOther = 7

ResolutionPublisher = 6

ResolutionRejected = 1

ResolutionRuleChange = 3

ResolutionUpdater = 5

StatusClosed = 3

StatusOpen = 2

StatusSubmitted = 1

computer

fileCatalog

installerFileCatalog

processFileCatalog

urlobject = '/api/bit9platform/v1/approvalRequest'

class cbapi.protection.models.**Certificate**(*cb*, *model_unique_id*, *initial_data=None*)

StateApproved = 2

StateBanned = 3

StateMixed = 4

StateUnapproved = 1

firstSeenComputer

parent

publisher

urlobject = '/api/bit9platform/v1/certificate'

class cbapi.protection.models.**Computer**(*cb*, *model_unique_id*, *initial_data=None*)

Represents a Computer object in the Carbon Black server.

fileInstances

policy

resetCLIPassword()

templateComputer

urlobject = '/api/bit9platform/v1/computer'

class cbapi.protection.models.**Connector**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a Connector object in the Carbon Black server.

Variables

- ***id*** – Unique connector Id
- ***name*** – Name of the connector. Note that only non-internal connectors can be renamed
- ***analysisName*** – Name for analysis component of the connector (can be same as the name field)
- ***connectorVersion*** – Version of this connector
- ***canAnalyze*** – True if this connector can analyze files
- ***enabled*** – True if connector is enabled
- ***analysisEnabled*** – True if analysis component of this connector is enabled
- ***isInternal*** – True if this is internal connector
- ***analysisTargets*** – Array of possible analysis targets. Analysis targets are required when creating new fileAnalysis. They usualy represent different OS and configurations and are available only for some internal connectors.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

analysisEnabled = None

analysisName = None

analysisTargets = []

canAnalyze = None

connectorVersion = None

enabled = None

id = None

isInternal = None

name = None

pendingAnalyses

urlobject = '/api/bit9platform/v1/connector'

class cbapi.protection.models.**DriftReport**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a DriftReport object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

urlobject = '/api/bit9platform/v1/driftReport'

class cbapi.protection.models.**DriftReportContents**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a DriftReportContents object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

urlobject = '/api/bit9platform/v1/driftReportContents'

class cbapi.protection.models.**EnforcementLevel**

LevelHigh = 20

LevelLow = 40

LevelMedium = 30

LevelNone = 80

class cbapi.protection.models.**Event**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a Event object in the Carbon Black server.

fileCatalog

urlobject = '/api/bit9platform/v1/event'

class cbapi.protection.models.**FileAnalysis**(*cb*, *model_unique_id*, *initial_data=None*)

urlobject = '/api/bit9platform/v1/fileAnalysis'

class cbapi.protection.models.**FileCatalog**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a FileCatalog object in the Carbon Black server.

certificate

computer

fileHash

publisher

urlobject = '/api/bit9platform/v1/fileCatalog'

class cbapi.protection.models.**FileInstance**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a FileInstance object in the Carbon Black server.

computer

fileCatalog

urlobject = '/api/bit9platform/v1/fileInstance'

class cbapi.protection.models.**FileInstanceDeleted**(*cb*, *model_unique_id*, *initial_data=None*)

urlobject = '/api/bit9platform/v1/fileInstanceDeleted'

class cbapi.protection.models.**FileInstanceGroup**(*cb*, *model_unique_id*, *initial_data=None*)

urlobject = '/api/bit9platform/v1/fileInstanceGroup'

class cbapi.protection.models.**FileRule**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)
Represents a FileRule object in the Carbon Black server.

Variables

- ***id*** – Unique id of this fileRule

- ***fileCatalogId*** – Id of fileCatalog entry associated with this fileRule. Can be null if file hasn't been seen on any endpoints yet. This is foreign key and can be expanded to expose fields from the related fileCatalog object
- ***name*** – Name of this rule.
- ***description*** – Description of this rule.
- ***fileState*** – File state for this rule. Can be one of: 1=Unapproved 2=Approved 3=Banned
- ***sourceType*** – Mechanism that created this rule. Can be one of: 1 = Manual 2 = Trusted Directory 3 = Reputation 4 = Imported 5 = External (API) 6 = Event Rule 7 = Application Template 8 = Unified Management
- ***sourceId*** – Id of source of this rule. Can be event rule id or trusted directory id
- ***reportOnly*** – True if this has a report-only ban
- ***reputationApprovalsEnabled*** – True if reputation approvals are enabled for this file
- ***forceInstaller*** – True if this file is forced to act as installer, even if product detected it as 'not installer'
- ***forceNotInstaller*** – True if this file is forced to act as 'not installer', even if product detected it as installer
- ***policyIds*** – List of IDs of policies where this rule applies. Value will be empty if this is a global rule
- ***hash*** – Hash associated with this rule. Note that hash will be available only if rule was created through md5 or sha-1 hash. If rule was created through fileName, fileCatalogId or sha-256 hash that exists in the catalog, this field will be empty.
- ***fileName*** – File name associated with this rule. Note that file name will be available only if rule was created through file name. If rule was created through fileCatalogId or hash, this field will be empty.
- ***lazyApproval*** – This filed is valid only when creating approvals. When set to true, it will cause approval to be sent to agent only if file is marked as installer or if it blocked on any agent. This is useful when proactively creating lot of approvals that might or might not be required, since it is using less resources. Note that, as soone as lazy approval is sent to agents, this field will changed to 'false'.
- ***platformFlags*** – Set of platform flags where this file rule will be valid. combination of: 1 = Windows 2 = Mac 4 = Linux
- ***dateCreated*** – Date/time when this rule was created (UTC)
- ***createdBy*** – User that created this object
- ***createdByUserId*** – Id of user that created this object
- ***dateModified*** – Date/time when this object was last modified (UTC)
- ***modifiedBy*** – User that last modified this object
- ***modifiedByUserId*** – Id of user that last modified this object
- ***clVersion*** – CL version associated with this file rule
- ***idUnique*** – Unique GUID of this rule
- ***origIdUnique*** – Unique GUID of the original rule

- ***unifiedFlag*** – Local override flag for unified rule (0 - if rule is not unified, 1 - no override allowed, 3 - local override allowed)
- ***unifiedSource*** – Unified server name that created this rule
- ***fileRuleType*** – Text description of file rule type
- ***version*** – Version of this file rule
- ***visible*** – If rule should be visible in the UI or not

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

```
PlatformLinux = 4

PlatformMac = 2

PlatformWindows = 1

SourceTypeApplicationTemplate = 7

SourceTypeEventRule = 6

SourceTypeExternal = 5

SourceTypeImported = 4

SourceTypeManual = 1

SourceTypeReputation = 3

SourceTypeTrustedDirectory = 2

SourceTypeUnifiedManagement = 8

StateApproved = 2

StateBanned = 3

StateUnapproved = 1

clVersion = None

createdBy = None

createdByUser

createdByUserId = None

dateCreated = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())

dateModified = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())

description = None

fileCatalog

fileCatalogId = None

fileName = None

fileRuleType = None

fileState = None

forceInstaller = None

forceNotInstaller = None

hash = None
```

id = None

idUnique = None

lazyApproval = None

modifiedBy = None

modifiedByUserId = None

name = None

origIdUnique = None

platformFlags = None

policyIds = None

reportOnly = None

reputationApprovalsEnabled = None

sourceId = None

sourceType = None

unifiedFlag = None

unifiedSource = None

urlobject = '/api/bit9platform/v1/fileRule'

version = None

visible = None

class cbapi.protection.models.**FileUpload**(*cb*, *model_unique_id*, *initial_data=None*)

file

urlobject = '/api/bit9platform/v1/fileUpload'

class cbapi.protection.models.**GrantedUserPolicyPermission**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a GrantedUserPolicyPermission object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

urlobject = '/api/bit9platform/v1/grantedUserPolicyPermission'

class cbapi.protection.models.**InternalEvent**(*cb*, *model_unique_id*, *initial_data=None*)

urlobject = '/api/bit9platform/v1/internalEvent'

class cbapi.protection.models.**MeteredExecution**(*cb*, *model_unique_id*, *initial_data=None*)

urlobject = '/api/bit9platform/v1/meteredExecution'

class cbapi.protection.models.**Notification**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a Notification object in the Carbon Black server.

Variables

- ***connectorId*** – Id of connector object that sent the notification
- ***time*** – Date/time of the notification (UTC)
- ***analysisResult*** – Analysis result. Can be one of: 0 = Unknown, 1 = Not malicious, 2 = Potential risk, 3 = Malicious
- ***fileAnalysisId*** – Id of fileAnalysis object associated with the notification. This should be available if notification came as a result of the file analysis

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

ResultClean = 1

ResultMalicious = 3

ResultNotAvailable = 0

ResultPotentialThreat = 2

analysisResult = None

anomaly = None

appliance = None

connectorId = None

destIp = None

destUsername = None

directories = []

externalId = None

externalUrl = None

fileAnalysisId = None

fileName = None

files = []

flags = None

httpHeader = None

malwareName = None

malwareType = None

md5 = None

msgFormat = None

product = None

regKeys = []

severity = None

```
    sha1 = None
    sha256 = None
    srcHost = None
    srcIp = None
    srcUsername = None
    status = None
    targetApp = None
    targetOS = None
    time = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())
    type = None
    urlobject = '/api/bit9platform/v1/notification'
    version = None
```

class cbapi.protection.models.**Notifier**(*cb*, *model_unique_id*, *initial_data=None*)

```
    urlobject = '/api/bit9platform/v1/notifier'
```

class cbapi.protection.models.**PendingAnalysis**(*cb*, *model_unique_id*, *initial_data=None*)

```
    ResultClean = 1
    ResultMalicious = 3
    ResultNotAvailable = 0
    ResultPotentialThreat = 2
    StatusAnalyzed = 3
    StatusCancelled = 5
    StatusError = 4
    StatusProcessed = 2
    StatusScheduled = 0
    StatusSubmitted = 1
```

create_notification(***kwargs*)

```
    file
    fileCatalog
    fileHash
    urlobject = '/api/bit9platform/v1/pendingAnalysis'
```

class cbapi.protection.models.**Policy**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a Policy object in the Carbon Black server.

Variables

- ***id*** – Unique id of this policy
- ***name*** – Name of this policy.

- ***description*** – Description of this policy.
- ***packageName*** – Name of installer package for this policy
- ***enforcementLevel*** – Target enforcement level. Can be one of: 20=High (Block Unapproved) 30=Medium (Prompt Unapproved) 40=Low (Monitor Unapproved) 60=None (Visibility) 80=None (Disabled)
- ***disconnectedEnforcementLevel*** – Target enforcement level for disconnected computers. Can be one of: 20=High (Block Unapproved) 30=Medium (Prompt Unapproved) 40=Low (Monitor Unapproved) 60=None (Visibility) 80=None (Disabled)
- ***helpDeskUrl*** – Helpdesk URL for notifiers in this policy
- ***imageUrl*** – Image logo URL for notifiers in this policy
- ***dateCreated*** – Date/time when this rule was created (UTC)
- ***createdByUserId*** – Id of user that created this object
- ***dateModified*** – Date/time when this object was last modified (UTC)
- ***modifiedByUserId*** – Id of user that last modified this object
- ***readOnly*** – True if this policy is read-only
- ***hidden*** – True if this policy is hidden in the UI
- ***automatic*** – True if AD mapping is enabled for this policy
- ***loadAgentInSafeMode*** – True if agents in this policy will be loaded when machine is booted in 'safe mode'
- ***reputationEnabled*** – True if reputation approvals are enabled in this policy
- ***fileTrackingEnabled*** – True if file tracking enabled in this policy
- ***customLogo*** – True if notifiers in this policy use custom logo
- ***automaticApprovalsOnTransition*** – True if agents in this policy will automatically locally approve files when transitioning into High Enforcement
- ***allowAgentUpgrades*** – True if agents can be upgraded for this policy
- ***totalComputers*** – Total number of computers in this policy
- ***connectedComputers*** – Number of connected computers in this policy
- ***atEnforcementComputers*** – Number of computers that are at target enforcement level in this policy
- ***clVersionMax*** – Max target CL version for agents in this policy

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

allowAgentUpgrades = None

atEnforcementComputers = None

automatic = None

automaticApprovalsOnTransition = None

clVersionMax = None

connectedComputers = None

createdByUserId = None

```
customLogo = None
dateCreated = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())
dateModified = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())
description = None
disconnectedEnforcementLevel = None
enforcementLevel = None
fileTrackingEnabled = None
helpDeskUrl = None
hidden = None
id = None
imageUrl = None
loadAgentInSafeMode = None
modifiedByUserId = None
name = None
packageName = None
readOnly = None
reputationEnabled = None
totalComputers = None
urlobject = '/api/bit9platform/v1/policy'
```

class cbapi.protection.models.**Publisher**(*cb*, *model_unique_id*, *initial_data=None*)

```
urlobject = '/api/bit9platform/v1/publisher'
```

class cbapi.protection.models.**PublisherCertificate**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a PublisherCertificate object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

```
urlobject = '/api/bit9platform/v1/publisherCertificate'
```

class cbapi.protection.models.**ScriptRule**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a ScriptRule object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

```
urlobject = '/api/bit9platform/v1/scriptRule'
```

class cbapi.protection.models.**ServerConfig**(*cb*, *model_unique_id*, *initial_data=None*)

```
urlobject = '/api/bit9platform/v1/serverConfig'
```

class cbapi.protection.models.**ServerPerformance**(*cb*, *model_unique_id*, *initial_data=None*)

urlobject = '/api/bit9platform/v1/serverPerformance'

class cbapi.protection.models.**TrustedDirectory**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a TrustedDirectory object in the Carbon Black server.

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

urlobject = '/api/bit9platform/v1/trustedDirectory'

class cbapi.protection.models.**TrustedUser**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a TrustedUser object in the Carbon Black server.

Variables

- ***id*** – Unique id of this trustedUser
- ***name*** – Name of the user as it will appear on the console. This is not the name that will be enforced on the endpoint
- ***userSid*** – Id of the user that will be trusted on the endpoint. This field can be user name, user SID (Security identifier) on Windows platforms or user's ID on Linux and Mac platforms
- ***description*** – Description of this rule
- ***platformId*** – Platform where this trustedUser will be valid. it is one of: 1 = Windows, 2 = Mac, 4 = Linux
- ***dateCreated*** – Date/time when this object was created (UTC)
- ***createdByUserId*** – Id of user that created this object. This is foreign key and can be expanded to expose fields from the related user object
- ***createdBy*** – User that created this object
- ***dateModified*** – Date/time when this object was last modified (UTC)
- ***modifiedByUserId*** – Id of user that last modified this object. This is foreign key and can be expanded to expose fields from the related user object
- ***modifiedBy*** – User that last modified this object
- ***clVersion*** – CL version associated with this trustedUser

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

clVersion = None

createdBy = None

createdByUserId = None

dateCreated = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())

dateModified = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())

description = None

id = None

modifiedBy = None

```
    modifiedByUserId = None
    name = None
    platformId = None
    urlobject = '/api/bit9platform/v1/trustedUser'
    userSid = None
```

class cbapi.protection.models.**Updater**(*cb*, *model_unique_id*, *initial_data=None*)

```
    urlobject = '/api/bit9platform/v1/updater'
```

class cbapi.protection.models.**User**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a User object in the Carbon Black server.

Variables

- ***id*** – Unique id of this user
- ***name*** – Name of the user
- ***userGroupIds*** – Comma-separated list of IDs of corresponding userGroup objects
- ***eMailAddress*** – EMail address of this user
- ***firstName*** – First name of this user
- ***lastName*** – Last name of this user
- ***title*** – Title of this user
- ***salutation*** – Salutation of this user
- ***department*** – Department this user belongs to
- ***homePhone*** – User's home phone
- ***cellPhone*** – User's cell phone
- ***backupCellPhone*** – User's secondary cell phone
- ***pager*** – User's pager number
- ***backupPager*** – User's secondary pager number
- ***comments*** – Comments for this user
- ***adminComments*** – Administrator's comments for this user
- ***registrationDate*** – Date this user was first registered (UTC)
- ***readOnly*** – True if this user is one of internal users (System or Cb Collective Defense Cloud Service) or AD user. These users cannot be modified through the API
- ***external*** – True if this is externally generated user (e.g. from AD)
- ***automatic*** – True if this user's roles are assigned automatically through mappings (valid only for external users)
- ***unified*** – True if this user's token is already connected to a remote unified environment (token should not be changed)
- ***enabled*** – True if this user is enabled
- ***passwordHash*** – Hash of user password

- ***passwordSalt*** – Salt used to generate password hash
- ***apiToken*** – API token for this user

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

adminComments = None

apiToken = None

automatic = None

backupCellPhone = None

backupPager = None

cellPhone = None

comments = None

department = None

eMailAddress = None

enabled = None

external = None

firstName = None

homePhone = None

id = None

lastName = None

name = None

pager = None

passwordHash = None

passwordSalt = None

readOnly = None

registrationDate = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())

salutation = None

title = None

unified = None

urlobject = '/api/bit9platform/v1/user'

userGroupIds = None

class cbapi.protection.models.**UserGroup**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a UserGroup object in the Carbon Black server.

Variables

- ***id*** – Unique id of this user group
- ***name*** – Name of the user group
- ***description*** – Description of this user group

- ***permissions*** – Permissions associated with users of this user group as a hexadecimal string. See https://developer.carbonblack.com/reference/enterprise-protection/8.0/rest-api/#usergroup for more information.
- ***policyIds*** – List of IDs of policies where this user group applies. Value will be empty if this is a global user group
- ***enabled*** – True if this userGroup is enabled
- ***editable*** – True if this userGroup is editable
- ***dateCreated*** – Date/time when this object was created (UTC)
- ***createdByUserId*** – Id of user that created this object. This is foreign key and can be expanded to expose fields from the related user object
- ***createdBy*** – User that created this object
- ***dateModified*** – Date/time when this object was last modified (UTC)
- ***modifiedByUserId*** – Id of user that last modified this object. This is foreign key and can be expanded to expose fields from the related user object
- ***modifiedBy*** – User that last modified this object
- ***automaticCount*** – Number of users that belong to this group and have been assigned through AD rule (doesn't include internal users)
- ***manualCount*** – Number of users that belong to this group and have been assigned manually (doesn't include internal users)

Base model for :param cb: :param model_unique_id: :param initial_data: :param force_init: :param full_doc: :return:

```
automaticCount = None
createdBy = None
createdByUserId = None
dateCreated = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())
dateModified = datetime.datetime(1970, 1, 1, 0, 0, tzinfo=tzlocal())
description = None
editable = None
enabled = None
id = None
manualCount = None
modifiedBy = None
modifiedByUserId = None
name = None
permissions = None
policyIds = None
urlobject = '/api/bit9platform/v1/userGroup'
```

5.3 Cloud Endpoint Standard API

This page documents the public interfaces exposed by cbapi when communicating with a Cloud Endpoint Standard server.

5.3.1 Main Interface

To use cbapi with VMware Carbon Black Cloud Endpoint Standard, you will be using the CBDefenseAPI. The CBDefenseAPI object then exposes two main methods to select data on the Carbon Black server:

class cbapi.psc.defense.rest_api.**CbDefenseAPI**(**args*, ***kwargs*)

The main entry point into the Carbon Black Cloud Endpoint Standard Defense API.

Parameters **profile** (*str*) – (optional) Use the credentials in the named profile when connecting to the Carbon Black server. Uses the profile named 'default' when not specified.

Usage:

```
>>> from cbapi import CbDefenseAPI
>>> cb = CbDefenseAPI(profile="production")
```

alert_search_suggestions(*query*)

Returns suggestions for keys and field values that can be used in a search.

Parameters **str** (*query*) – A search query to use.

Returns A list of search suggestions expressed as dict objects.

api_json_request(*method*, *uri*, ***kwargs*)

Submit a request to the server.

Args: method (str): HTTP method to use. uri (str): URI to submit the request to. **kwargs (dict): Additional arguments.

Returns: object: Result of the operation.

Raises: ServerError: If there's an error output from the server.

bulk_threat_dismiss(*threat_ids*, *remediation=None*, *comment=None*)

Dismiss the alerts associated with multiple threat IDs. The alerts will be left in a DISMISSED state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

bulk_threat_update(*threat_ids*, *remediation=None*, *comment=None*)

Update the alert status of alerts associated with multiple threat IDs. The alerts will be left in an OPEN state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

create(*cls*, *data=None*)
Create a new object.

Args: cls (class): The Model class (only some models can be created, for example, Feed, Notification, ...) data (object): The data used to initialize the new object

Returns: Model: An empty instance of the model class.

Raises: ApiError: If the Model cannot be created.

delete_object(*uri*)
Send a DELETE request to the specified URI.

Args: uri (str): The URI to send the DELETE request to.

Returns: object: The return data from the DELETE request.

device_background_scan(*device_ids*, *scan*)
Set the background scan option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **scan** (*boolean*) – True to turn background scan on, False to turn it off.

device_bypass(*device_ids*, *enable*)
Set the bypass option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **enable** (*boolean*) – True to enable bypass, False to disable it.

device_delete_sensor(*device_ids*)
Delete the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be deleted.

device_quarantine(*device_ids*, *enable*)
Set the quarantine option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **enable** (*boolean*) – True to enable quarantine, False to disable it.

device_uninstall_sensor(*device_ids*)
Uninstall the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be uninstalled.

device_update_policy(*device_ids*, *policy_id*)
Set the current policy for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **policy_id** (*int*) – ID of the policy to set for the devices.

device_update_sensor_version(*device_ids*, *sensor_version*)
Update the sensor version for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **sensor_version** (*dict*) – New version properties for the sensor.

get_auditlogs()

Retrieve queued audit logs from the Carbon Black Cloud Endpoint Standard server. Note that this can only be used with a 'API' key generated in the CBC console.

Returns list of dictionary objects representing the audit logs, or an empty list if none available.

get_notifications()

Retrieve queued notifications (alerts) from the Cloud Endpoint Standard server. Note that this can only be used with a 'SIEM' key generated in the Carbon Black Cloud console.

Returns list of dictionary objects representing the notifications, or an empty list if none available.

get_object(*uri*, *query_parameters=None*, *default=None*)

Submit a GET request to the server and parse the result as JSON before returning.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response.

Returns: object: Result of the GET request.

get_raw_data(*uri*, *query_parameters=None*, *default=None*, ***kwargs*)

Submit a GET request to the server and return the result without parsing it.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response. **kwargs:

Returns: object: Result of the GET request.

notification_listener(*interval=60*)

Generator to continually poll the Cloud Endpoint Standard server for notifications (alerts). Note that this can only be used with a 'SIEM' key generated in the Carbon Black Cloud console.

post_object(*uri*, *body*, ***kwargs*)

Send a POST request to the specified URI.

Args: uri (str): The URI to send the POST request to. body (object): The data to be sent in the body of the POST request. **kwargs:

Returns: object: The return data from the POST request.

put_object(*uri*, *body*, ***kwargs*)

Send a PUT request to the specified URI.

Args: uri (str): The URI to send the PUT request to. body (object): The data to be sent in the body of the PUT request. **kwargs:

Returns: object: The return data from the PUT request.

raise_unless_json(*ret*, *expected*)

Raise a ServerError unless we got back an HTTP 200 response with JSON containing all the expected values.

Args: ret (object): Return value to be checked. expected (dict): Expected keys and values that need to be found in the JSON response.

Raises: ServerError: If the HTTP response is anything but 200, or if the expected values are not found.

select(*cls*, *unique_id=None*, **args*, ***kwargs*)
Prepare a query against the Carbon Black data store.

Args: cls (class): The Model class (for example, Computer, Process, Binary, FileInstance) to query unique_id (optional): The unique id of the object to retrieve, to retrieve a single object by ID *args: **kwargs:

Returns: object: An instance of the Model class if a unique_id is provided, otherwise a Query object

url
Return the connection URL.

Returns: str: The connection URL.

5.3.2 Queries

class cbapi.psc.defense.rest_api.**Query**(*doc_class*, *cb*, *query=None*)
Represents a prepared query to the Cloud Endpoint Standard server.

This object is returned as part of a *CbDefenseAPI.select()* operation on models requested from the Cloud Endpoint Standard server. You should not have to create this class yourself.

The query is not executed on the server until it's accessed, either as an iterator (where it will generate values on demand as they're requested) or as a list (where it will retrieve the entire result set and save to a list). You can also call the Python built-in `len()` on this object to retrieve the total number of items matching the query.

Examples:

```
>>> from cbapi.psc.defense import CbDefenseAPI
>>> cb = CbDefenseAPI()
```

Notes:

- The slicing operator only supports start and end parameters, but not step. `[1:-1]` is legal, but `[1:2:-1]` is not.
- You can chain where clauses together to create AND queries; only objects that match all `where` clauses will be returned.

and_(*q*)
Add a filter to this query. Equivalent to calling *where()* on this object.

Parameters **q** (*str*) – Query string

Returns Query object

Return type *Query*

where(*q*)
Add a filter to this query.

Parameters **q** (*str*) – Query string

Returns Query object

Return type *Query*

5.3.3 Models

class cbapi.psc.defense.models.**DefenseMutableModel**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a DefenseMutableModel object in the Carbon Black server.

class cbapi.psc.defense.models.**Device**(*cb*, *model_unique_id*, *initial_data=None*)

Represents a Device object in the Carbon Black server.

activationCode = None

activationCodeExpiryTime = datetime.datetime(1970, 1, 1, 0, 0)

assignedToId = None

assignedToName = None

avEngine = None

avLastScanTime = datetime.datetime(1970, 1, 1, 0, 0)

avMaster = None

avStatus = []

avUpdateServers = []

createTime = datetime.datetime(1970, 1, 1, 0, 0)

deregisteredTime = datetime.datetime(1970, 1, 1, 0, 0)

deviceGuid = None

deviceId = None

deviceOwnerId = None

deviceSessionId = None

deviceType = None

email = None

firstName = None

firstVirusActivityTime = datetime.datetime(1970, 1, 1, 0, 0)

info_key = 'deviceInfo'

lastContact = datetime.datetime(1970, 1, 1, 0, 0)

lastExternalIpAddress = None

lastInternalIpAddress = None

lastLocation = None

lastName = None

lastReportedTime = datetime.datetime(1970, 1, 1, 0, 0)

lastResetTime = datetime.datetime(1970, 1, 1, 0, 0)

lastShutdownTime = datetime.datetime(1970, 1, 1, 0, 0)

lastVirusActivityTime = datetime.datetime(1970, 1, 1, 0, 0)

linuxKernelVersion = None

lr_session()
Retrieve a Live Response session object for this Device.

Returns Live Response session object

Return type cbapi.defense.cblr.LiveResponseSession

Raises *ApiError* – if there is an error establishing a Live Response session for this Device

messages = []

middleName = None

name = None

organizationId = None

organizationName = None

osVersion = None

passiveMode = None

policyId = None

policyName = None

primary_key = 'deviceId'

quarantined = None

registeredTime = datetime.datetime(1970, 1, 1, 0, 0)

rootedByAnalytics = None

rootedByAnalyticsTime = datetime.datetime(1970, 1, 1, 0, 0)

rootedBySensor = None

rootedBySensorTime = datetime.datetime(1970, 1, 1, 0, 0)

scanLastActionTime = datetime.datetime(1970, 1, 1, 0, 0)

scanLastCompleteTime = datetime.datetime(1970, 1, 1, 0, 0)

scanStatus = None

sensorStates = []

sensorVersion = None

status = None

targetPriorityType = None

testId = None

uninstalledTime = datetime.datetime(1970, 1, 1, 0, 0)

urlobject = '/integrationServices/v3/device'

vdiBaseDevice = None

windowsPlatform = None

class cbapi.psc.defense.models.**Event**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a Event object in the Carbon Black server.

info_key = 'eventInfo'

primary_key = 'eventId'

urlobject = '/integrationServices/v3/event'

class cbapi.psc.defense.models.**Policy**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=False*)

Represents a Policy object in the Carbon Black server.

add_rule(*new_rule*)

delete_rule(*rule_id*)

description = None

id = None

info_key = 'policyInfo'

latestRevision = None

name = None

policy = {}

priorityLevel = None

replace_rule(*rule_id*, *new_rule*)

rules

systemPolicy = None

urlobject = '/integrationServices/v3/policy'

version = None

5.4 VMware Carbon Black Cloud Enterprise EDR API

This page documents the public interfaces exposed by cbapi when communicating with a VMware Carbon Black Cloud Enterprise EDR server.

5.4.1 Main Interface

To use cbapi with Enterprise EDR, you use CbThreatHunterAPI objects. These objects expose two main methods to access data on the Enterprise EDR server: `select` and `create`.

class cbapi.psc.threathunter.rest_api.**CbThreatHunterAPI**(**args*, ***kwargs*)

The main entry point into the Carbon Black Cloud Enterprise EDR API.

Parameters **profile** (*str*) – (optional) Use the credentials in the named profile when connecting to the Carbon Black server. Uses the profile named 'default' when not specified.

Usage:

```
>>> from cbapi.psc.threathunter import CbThreatHunterAPI
>>> cb = CbThreatHunterAPI(profile="production")
```

alert_search_suggestions(*query*)

Returns suggestions for keys and field values that can be used in a search.

Parameters **str** (*query*) – A search query to use.

Returns A list of search suggestions expressed as dict objects.

api_json_request(*method*, *uri*, ***kwargs*)
Submit a request to the server.

Args: method (str): HTTP method to use. uri (str): URI to submit the request to. **kwargs (dict): Additional arguments.

Returns: object: Result of the operation.

Raises: ServerError: If there's an error output from the server.

bulk_threat_dismiss(*threat_ids*, *remediation=None*, *comment=None*)
Dismiss the alerts associated with multiple threat IDs. The alerts will be left in a DISMISSED state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

bulk_threat_update(*threat_ids*, *remediation=None*, *comment=None*)
Update the alert status of alerts associated with multiple threat IDs. The alerts will be left in an OPEN state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

convert_query(*query*)
Converts a legacy Carbon Black EDR query to an Enterprise EDR query.

Parameters **query** (*str*) – the query to convert

Returns the converted query

Return type str

create(*cls*, *data=None*)
Creates a new model.

```
>>> feed = cb.create(Feed, feed_data)
```

Parameters

- **cls** – The model being created
- **data** (*dict(str, object)*) – The data to pre-populate the model with

Returns an instance of *cls*

custom_severities
Returns a list of active `ReportSeverity` instances

Return type list[`ReportSeverity`]

delete_object(*uri*)
Send a DELETE request to the specified URI.

Args: uri (str): The URI to send the DELETE request to.

Returns: object: The return data from the DELETE request.

device_background_scan(*device_ids*, *scan*)
Set the background scan option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **scan** (*boolean*) – True to turn background scan on, False to turn it off.

device_bypass(*device_ids*, *enable*)
Set the bypass option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **enable** (*boolean*) – True to enable bypass, False to disable it.

device_delete_sensor(*device_ids*)
Delete the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be deleted.

device_quarantine(*device_ids*, *enable*)
Set the quarantine option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **enable** (*boolean*) – True to enable quarantine, False to disable it.

device_uninstall_sensor(*device_ids*)
Uninstall the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be uninstalled.

device_update_policy(*device_ids*, *policy_id*)
Set the current policy for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **policy_id** (*int*) – ID of the policy to set for the devices.

device_update_sensor_version(*device_ids*, *sensor_version*)
Update the sensor version for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **sensor_version** (*dict*) – New version properties for the sensor.

get_object(*uri*, *query_parameters=None*, *default=None*)
Submit a GET request to the server and parse the result as JSON before returning.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response.

Returns: object: Result of the GET request.

get_raw_data(*uri*, *query_parameters=None*, *default=None*, ***kwargs*)
Submit a GET request to the server and return the result without parsing it.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response. **kwargs:

Returns: object: Result of the GET request.

limits()
Returns a dictionary containing API limiting information.

Example:

```
>>> cb.limits()
{u'status_code': 200, u'time_bounds': {u'upper': 1545335070095, u'lower':␣
↪1542779216139}}
```

Returns a dict of limiting information

Return type dict(str, str)

post_object(*uri*, *body*, ***kwargs*)
Send a POST request to the specified URI.

Args: uri (str): The URI to send the POST request to. body (object): The data to be sent in the body of the POST request. **kwargs:

Returns: object: The return data from the POST request.

put_object(*uri*, *body*, ***kwargs*)
Send a PUT request to the specified URI.

Args: uri (str): The URI to send the PUT request to. body (object): The data to be sent in the body of the PUT request. **kwargs:

Returns: object: The return data from the PUT request.

queries()
Retrieves a list of queries, active or complete, known by the Enterprise EDR server.

Returns a list of query ids

Return type list(str)

raise_unless_json(*ret*, *expected*)
Raise a ServerError unless we got back an HTTP 200 response with JSON containing all the expected values.

Args: ret (object): Return value to be checked. expected (dict): Expected keys and values that need to be found in the JSON response.

Raises: ServerError: If the HTTP response is anything but 200, or if the expected values are not found.

select(*cls*, *unique_id=None*, **args*, ***kwargs*)
Prepare a query against the Carbon Black data store.

Args: cls (class): The Model class (for example, Computer, Process, Binary, FileInstance) to query unique_id (optional): The unique id of the object to retrieve, to retrieve a single object by ID *args: **kwargs:

Returns: object: An instance of the Model class if a unique_id is provided, otherwise a Query object

url
Return the connection URL.

Returns: str: The connection URL.

validate_query(*query*)
Validates the given IOC query.

```
>>> cb.validate_query("process_name:chrome.exe") # True
```

Parameters **query** (*str*) – the query to validate

Returns whether or not the query is valid

Return type bool

5.4.2 Queries

The Enterprise EDR API uses QueryBuilder instances to construct structured or unstructured (i.e., raw string) queries. You can either construct these instances manually, or allow `CbThreatHunterAPI.select()` to do it for you:

class cbapi.psc.threathunter.query.**QueryBuilder**(***kwargs*)
Provides a flexible interface for building prepared queries for the Carbon Black Enterprise EDR backend.

This object can be instantiated directly, or can be managed implicitly through the `CbThreatHunterAPI.select()` API.

Examples:

```
>>> from cbapi.psc.threathunter import QueryBuilder
>>> # build a query with chaining
>>> query = QueryBuilder().where(process_name="malicious.exe").and_(device_name=
↪"suspect")
>>> # start with an initial query, and chain another condition to it
>>> query = QueryBuilder(device_os="WINDOWS").or_(process_username="root")
```

and_(*q*, ***kwargs*)
Adds a conjunctive filter to a query.

Parameters

- **q** – string or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

not_(*q*, ***kwargs*)
Adds a negative filter to a query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

or_(*q*, ***kwargs*)
Adds a disjunctive filter to a query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

where(*q*, ***kwargs*)
Adds a conjunctive filter to a query.

Parameters

- **q** – string or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

class cbapi.psc.threathunter.query.**Query**(*doc_class*, *cb*)
Represents a prepared query to the Carbon Black Enterprise EDR backend.

This object is returned as part of a `CbThreatHunterPI.select()` operation on models requested from the Enterprise EDR backend. You should not have to create this class yourself.

The query is not executed on the server until it's accessed, either as an iterator (where it will generate values on demand as they're requested) or as a list (where it will retrieve the entire result set and save to a list). You can also call the Python built-in `len()` on this object to retrieve the total number of items matching the query.

Examples:

```
>>> from cbapi.psc.threathunter import CbThreatHunterAPI,Process
>>> cb = CbThreatHunterAPI()
>>> query = cb.select(Process)
>>> query = query.where(process_name="notepad.exe")
>>> # alternatively:
>>> query = query.where("process_name:notepad.exe")
```

Notes:

- The slicing operator only supports start and end parameters, but not step. `[1:-1]` is legal, but `[1:2:-1]` is not.
- You can chain where clauses together to create AND queries; only objects that match all `where` clauses will be returned.

and_(*q=None*, ***kwargs*)
Add a conjunctive filter to this query.

Parameters

- **q** – Query string or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type *Query*

not_(*q=None*, ***kwargs*)
Adds a negated filter to this query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type *Query*

or_(*q=None*, ***kwargs*)
Add a disjunctive filter to this query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type *Query*

where(*q=None*, ***kwargs*)
Add a filter to this query.

Parameters

- **q** – Query string, *QueryBuilder*, or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type *Query*

class cbapi.psc.threathunter.models.**AsyncProcessQuery**(*doc_class*, *cb*)
Represents the query logic for an asychronous Process query.

This class specializes `Query` to handle the particulars of process querying.

and_(*q=None*, ***kwargs*)
Add a conjunctive filter to this query.

Parameters

- **q** – Query string or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type `Query`

not_(*q=None*, ***kwargs*)
Adds a negated filter to this query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type `Query`

or_(*q=None*, ***kwargs*)
Add a disjunctive filter to this query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type `Query`

sort_by(*key*, *direction='ASC'*)
Sets the sorting behavior on a query's results.

Example:

```
>>> cb.select(Process).where(process_name="cmd.exe").sort_by("device_timestamp
↪")
```

Parameters

- **key** – the key in the schema to sort by
- **direction** – the sort order, either "ASC" or "DESC"

Return type *AsyncProcessQuery*

timeout(*msecs*)
Sets the timeout on a process query.

Example:

```
>>> cb.select(Process).where(process_name="foo.exe").timeout(5000)
```

Param msecs: the timeout duration, in milliseconds

Returns AsyncProcessQuery object

Return type *AsyncProcessQuery*

where(*q=None*, ***kwargs*)
Add a filter to this query.

Parameters

- **q** – Query string, `QueryBuilder`, or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type `Query`

class cbapi.psc.threathunter.query.**FeedQuery**(*doc_class*, *cb*)
Represents the logic for a `Feed` query.

```
>>> cb.select(Feed)
>>> cb.select(Feed, id)
>>> cb.select(Feed).where(include_public=True)
```

class cbapi.psc.threathunter.query.**ReportQuery**(*doc_class*, *cb*)
Represents the logic for a `Report` query.

```
>>> cb.select(Report).where(feed_id=id)
```

Note: Only feed reports can be queried. Watchlist reports should be interacted with via `Watchlist.reports()`.

class cbapi.psc.threathunter.query.**WatchlistQuery**(*doc_class*, *cb*)
Represents the logic for a `Watchlist` query.

```
>>> cb.select(Watchlist)
```

5.4.3 Models

class cbapi.psc.threathunter.models.**Process**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=True*)
Represents a Process object in the Carbon Black server.

class Summary(*cb*, *model_unique_id*)
Represents a Summary object in the Carbon Black server.

children
Returns a list of child processes for this process.

Returns Returns a list of process objects

Return type list of *Process*

events(***kwargs*)
Returns a query for events associated with this process's process GUID.

Parameters **kwargs** – Arguments to filter the event query with.

Returns Returns a Query object with the appropriate search parameters for events

Return type *cbapi.psc.threathunter.query.Query*

Example:

```
>>> [print(event) for event in process.events()]
>>> [print(event) for event in process.events(event_type="modload")]
```

parents
Returns a query for parent processes associated with this process.

Returns Returns a Query object with the appropriate search parameters for parent processes, or None if the process has no recorded parent

Return type `cbapi.psc.threathunter.query.AsyncProcessQuery` or None

process_md5
Returns a string representation of the MD5 hash for this process.

Returns A string representation of the process's MD5.

Return type str

process_pids
Returns a list of PIDs associated with this process.

Returns A list of PIDs

Return type list of ints

process_sha256
Returns a string representation of the SHA256 hash for this process.

Returns A string representation of the process's SHA256.

Return type str

siblings
Returns a list of sibling processes for this process.

Returns Returns a list of process objects

Return type list of *Process*

summary
Returns organization-specific information about this process.

tree()
Returns a *Tree* of children (and possibly siblings) associated with this process.

Returns Returns a *Tree* object

Return type *Tree*

Example:

```
>>> tree = process.tree()
```

class cbapi.psc.threathunter.models.**Event**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=True*)
Represents a Event object in the Carbon Black server.

class cbapi.psc.threathunter.models.**Tree**(*cb*, *model_unique_id=None*, *initial_data=None*, *force_init=False*, *full_doc=True*)
Represents a Tree object in the Carbon Black server.

children
Returns all of the children of the process that this tree is centered around.

Returns A list of *Process* instances

Return type list of *Process*

class cbapi.psc.threathunter.models.**Feed**(*cb*, *model_unique_id=None*, *initial_data=None*)
Represents a Feed object in the Carbon Black server.

Variables

- ***name*** – A human-friendly name for this feed
- **owner** – The feed owner's connector ID
- **provider_url** – A URL supplied by the feed's provider
- ***summary*** – A human-friendly summary for the feed
- ***category*** – The feed's category
- **source_label** – The feed's source label
- **access** – The feed's access (public or private)
- ***id*** – The feed's unique ID

append_reports(*reports*)
Append the given reports to this feed's current reports.

Parameters **reports** (list(*Report*)) – the reports to append

Raises ***InvalidObjectError*** – if *id* is missing

delete()
Deletes this feed from the Enterprise EDR server.

Raises ***InvalidObjectError*** – if *id* is missing

replace_reports(*reports*)
Replace this feed's reports with the given reports.

Parameters **reports** (list(*Report*)) – the reports to replace with

Raises ***InvalidObjectError*** – if *id* is missing

reports
Returns a list of *Report* associated with this feed.

Returns a list of reports

Return type list(*Report*)

save(*public=False*)
Saves this feed on the Enterprise EDR server.

Parameters **public** – Whether to make the feed publicly available

Returns The saved feed

Return type *Feed*

update(***kwargs*)
Update this feed's metadata with the given arguments.

```
>>> feed.update(access="private")
```

Parameters **kwargs** (*dict(str, str)*) – The fields to update

Raises

- ***InvalidObjectError*** – if *id* is missing or *validate()* fails
- ***ApiError*** – if an invalid field is specified

validate()
Validates this feed's state.

Raises ***InvalidObjectError*** – if the feed's state is invalid

class cbapi.psc.threathunter.models.**Report**(*cb*, *model_unique_id=None*, *initial_data=None*, *feed_id=None*, *from_watchlist=False*)
Represents a Report object in the Carbon Black server.

Variables

- ***id*** – The report's unique ID
- **timestamp** – When this report was created
- ***title*** – A human-friendly title for this report

- ***description*** – A human-friendly description for this report
- ***severity*** – The severity of the IOCs within this report
- **link** – A URL for some reference for this report
- ***tags*** – A list of tags for this report
- **iocs_v2** – A list of IOC_V2 dicts associated with this report
- **visibility** – The visibility of this report

custom_severity
Returns the custom severity for this report.

Returns The custom severity for this report, if it exists

Return type *ReportSeverity*

Raises ***InvalidObjectError*** – if *id* is missing or this report is from a watchlist

delete()
Deletes this report from the Enterprise EDR server.

```
>>> report.delete()
```

Raises ***InvalidObjectError*** – if *id* is missing, or *feed_id* is missing and this report is a feed report

ignore()
Sets the ignore status on this report.

Only watchlist reports have an ignore status.

Raises ***InvalidObjectError*** – if *id* is missing or this report is not from a watchlist

ignored
Returns the ignore status for this report.

Only watchlist reports have an ignore status.

```
>>> if report.ignored:
...     report.unignore()
```

Returns whether or not this report is ignored

Return type bool

Raises ***InvalidObjectError*** – if *id* is missing or this report is not from a watchlist

iocs_
Returns a list of *IOC_V2* associated with this report.

```
>>> for ioc in report.iocs_:
...     print(ioc.values)
```

Returns a list of IOCs

Return type list(*IOC_V2*)

save_watchlist()
Saves this report *as a watchlist report*.

Note: This method **cannot** be used to save a feed report. To save feed reports, create them with *cb.create* and use `Feed.replace()`.

Raises ***InvalidObjectError*** – if *validate()* fails

unignore()
Removes the ignore status on this report.

Only watchlist reports have an ignore status.

Raises ***InvalidObjectError*** – if *id* is missing or this report is not from a watchlist

update(***kwargs*)
Update this report with the given arguments.

Note: The report's timestamp is always updated, regardless of whether passed explicitly.

```
>>> report.update(title="My new report title")
```

Parameters **kwargs** (*dict(str, str)*) – The fields to update

Returns The updated report

Return type *Report*

Raises ***InvalidObjectError*** – if *id* is missing, or *feed_id* is missing and this report is a feed report, or *validate()* fails

validate()
Validates this report's state.

Raises ***InvalidObjectError*** – if the report's state is invalid

class cbapi.psc.threathunter.models.**IOC**(*cb*, *model_unique_id=None*, *initial_data=None*, *report_id=None*)
Represents a IOC object in the Carbon Black server.

Variables

- ***md5*** – A list of MD5 checksums
- **ipv4** – A list of IPv4 addresses
- **ipv6** – A list of IPv6 addresses
- **dns** – A list of domain names
- ***query*** – A list of dicts, each containing an IOC query

Creates a new IOC instance.

Raises ***ApiError*** – if *initial_data* is *None*

validate()
Validates this IOC structure's state.

Raises ***InvalidObjectError*** – if the IOC structure's state is invalid

class cbapi.psc.threathunter.models.**IOC_V2**(*cb*, *model_unique_id=None*, *initial_data=None*, *report_id=None*)

Represents a IOC_V2 object in the Carbon Black server.

Variables

- ***id*** – The IOC_V2's unique ID
- **match_type** – How IOCs in this IOC_V2 are matched
- **values** – A list of IOCs
- **field** – The kind of IOCs contained in this IOC_V2
- **link** – A URL for some reference for this IOC_V2

Creates a new IOC_V2 instance.

Raises ***ApiError*** – if *initial_data* is *None*

ignore()

Sets the ignore status on this IOC.

Only watchlist IOCs have an ignore status.

Raises ***InvalidObjectError*** – if *id* is missing or this IOC is not from a watchlist

ignored

Returns whether or not this IOC is ignored

```
>>> if ioc.ignored:
...     ioc.unignore()
```

Returns the ignore status

Return type bool

Raises ***InvalidObjectError*** – if this IOC is missing an *id* or is not a watchlist IOC

unignore()

Removes the ignore status on this IOC.

Only watchlist IOCs have an ignore status.

Raises ***InvalidObjectError*** – if *id* is missing or this IOC is not from a watchlist

validate()

Validates this IOC_V2's state.

Raises ***InvalidObjectError*** – if the IOC_V2's state is invalid

class cbapi.psc.threathunter.models.**Watchlist**(*cb*, *model_unique_id=None*, *initial_data=None*)

Represents a Watchlist object in the Carbon Black server.

Variables

- ***name*** – A human-friendly name for the watchlist
- ***description*** – A short description of the watchlist
- ***id*** – The watchlist's unique id
- **tags_enabled** – Whether tags are currently enabled
- **alerts_enabled** – Whether alerts are currently enabled

- **create_timestamp** – When this watchlist was created
- **last_update_timestamp** – Report IDs associated with this watchlist
- **report_ids** – Report IDs associated with this watchlist
- **classifier** – A key, value pair specifying an associated feed

classifier_
Returns the classifier key and value, if any, for this watchlist.

Return type tuple(str, str) or None

delete()
Deletes this watchlist from the Enterprise EDR server.

Raises ***InvalidObjectError*** – if *id* is missing

disable_alerts()
Disable alerts for this watchlist.

Raises ***InvalidObjectError*** – if *id* is missing

disable_tags()
Disable tagging for this watchlist.

Raises ***InvalidObjectError*** – if *id* is missing

enable_alerts()
Enable alerts for this watchlist. Alerts are not retroactive.

Raises ***InvalidObjectError*** – if *id* is missing

enable_tags()
Enable tagging for this watchlist.

Raises ***InvalidObjectError*** – if *id* is missing

feed
Returns the feed linked to this watchlist, if there is one.

Returns the feed linked to this watchlist, if any

Return type *Feed* or None

reports
Returns a list of *Report* instances associated with this watchlist.

Note: If this watchlist is a classifier (i.e. feed-linked) watchlist, *reports* will be empty. To get the reports associated with the linked feed, use *feed* like:

```
>>> for report in watchlist.feed.reports:
...     print(report.title)
```

Returns A list of reports

Return type list(*Report*)

save()
Saves this watchlist on the Enterprise EDR server.

Returns The saved watchlist

Return type *Watchlist*

Raises ***InvalidObjectError*** – if *validate()* fails

update(***kwargs*)
Updates this watchlist with the given arguments.

```
>>> watchlist.update(name="New Name")
```

Parameters **kwargs** (*dict(str, str)*) – The fields to update

Raises

- ***InvalidObjectError*** – if *id* is missing or *validate()* fails
- ***ApiError*** – if *report_ids* is given *and* is empty

validate()
Validates this watchlist's state.

Raises ***InvalidObjectError*** – if the watchlist's state is invalid

class cbapi.psc.threathunter.models.**ReportSeverity**(*cb*, *initial_data=None*)
Represents a ReportSeverity object in the Carbon Black server.

Variables

- **report_id** – The unique ID for the corresponding report
- ***severity*** – The severity level

class cbapi.psc.threathunter.models.**Binary**(*cb*, *model_unique_id*)
Represents a Binary object in the Carbon Black server.

Variables

- ***sha256*** – The SHA-256 hash of the file
- ***md5*** – The MD5 hash of the file
- **file_available** – If true, the file is available for download
- **available_file_size** – The size of the file available for download
- **file_size** – The size of the actual file (represented by the hash)
- **os_type** – The OS that this file is designed for
- **architecture** – The set of architectures that this file was compiled for
- **lang_id** – The Language ID value for the Windows VERSIONINFO resource
- **charset_id** – The Character set ID value for the Windows VERSIONINFO resource
- **internal_name** – The internal name from FileVersionInformation
- **product_name** – The product name from FileVersionInformation
- **company_name** – The company name from FileVersionInformation
- **trademark** – The trademark from FileVersionInformation
- **file_description** – The file description from FileVersionInformation
- **file_version** – The file version from FileVersionInformation
- ***comments*** – Comments from FileVersionInformation

- **original_filename** – The original filename from FileVersionInformation
- **product_description** – The product description from FileVersionInformation
- **product_version** – The product version from FileVersionInformation
- **private_build** – The private build from FileVersionInformation
- **special_build** – The special build from FileVersionInformation

class Summary(*cb*, *model_unique_id*)
Represents a Summary object in the Carbon Black server.

download_url
Returns a URL that can be used to download the file for this binary. Returns None if no download can be found.

Parameters **expiration_seconds** – How long the download should be valid for

Raises *InvalidObjectError* – if URL retrieval should be retried

Returns A pre-signed AWS download URL

Return type str

summary
Returns organization-specific information about this binary.

class cbapi.psc.threathunter.models.**Downloads**(*cb*, *shas*, *expiration_seconds=3600*)
Represents a Downloads object in the Carbon Black server.

class FoundItem(*cb*, *item*)
Represents a FoundItem object in the Carbon Black server.

found
Returns a list of *Downloads.FoundItem*, one for each binary found in the binary store.

5.5 VMware Carbon Black Cloud API

This page documents the public interfaces exposed by cbapi when communicating with the VMware Carbon Black Cloud.

5.5.1 Main Interface

To use cbapi with the VMware Carbon Black Cloud, you use CbPSCBaseAPI objects.

class cbapi.psc.rest_api.**CbPSCBaseAPI**(**args*, ***kwargs*)
The main entry point into the Cb PSC API.

Parameters **profile** (*str*) – (optional) Use the credentials in the named profile when connecting to the Carbon Black server. Uses the profile named 'default' when not specified.

Usage:

```
>>> from cbapi import CbPSCBaseAPI
>>> cb = CbPSCBaseAPI(profile="production")
```

alert_search_suggestions(*query*)
Returns suggestions for keys and field values that can be used in a search.

Parameters **str** (*query*) – A search query to use.

Returns A list of search suggestions expressed as dict objects.

api_json_request(*method*, *uri*, ***kwargs*)
Submit a request to the server.

Args: method (str): HTTP method to use. uri (str): URI to submit the request to. **kwargs (dict): Additional arguments.

Returns: object: Result of the operation.

Raises: ServerError: If there's an error output from the server.

bulk_threat_dismiss(*threat_ids*, *remediation=None*, *comment=None*)
Dismiss the alerts associated with multiple threat IDs. The alerts will be left in a DISMISSED state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

bulk_threat_update(*threat_ids*, *remediation=None*, *comment=None*)
Update the alert status of alerts associated with multiple threat IDs. The alerts will be left in an OPEN state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

create(*cls*, *data=None*)
Create a new object.

Args: cls (class): The Model class (only some models can be created, for example, Feed, Notification, ...) data (object): The data used to initialize the new object

Returns: Model: An empty instance of the model class.

Raises: ApiError: If the Model cannot be created.

delete_object(*uri*)
Send a DELETE request to the specified URI.

Args: uri (str): The URI to send the DELETE request to.

Returns: object: The return data from the DELETE request.

device_background_scan(*device_ids*, *scan*)
Set the background scan option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **scan** (*boolean*) – True to turn background scan on, False to turn it off.

device_bypass(*device_ids*, *enable*)
Set the bypass option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **enable** (*boolean*) – True to enable bypass, False to disable it.

device_delete_sensor (*device_ids*)
Delete the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be deleted.

device_quarantine (*device_ids*, *enable*)
Set the quarantine option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **enable** (*boolean*) – True to enable quarantine, False to disable it.

device_uninstall_sensor (*device_ids*)
Uninstall the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be uninstalled.

device_update_policy (*device_ids*, *policy_id*)
Set the current policy for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **policy_id** (*int*) – ID of the policy to set for the devices.

device_update_sensor_version (*device_ids*, *sensor_version*)
Update the sensor version for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **sensor_version** (*dict*) – New version properties for the sensor.

get_object (*uri*, *query_parameters=None*, *default=None*)
Submit a GET request to the server and parse the result as JSON before returning.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response.

Returns: object: Result of the GET request.

get_raw_data (*uri*, *query_parameters=None*, *default=None*, ***kwargs*)
Submit a GET request to the server and return the result without parsing it.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response. **kwargs:

Returns: object: Result of the GET request.

post_object (*uri*, *body*, ***kwargs*)
Send a POST request to the specified URI.

Args: uri (str): The URI to send the POST request to. body (object): The data to be sent in the body of the POST request. **kwargs:

Returns: object: The return data from the POST request.

put_object (*uri*, *body*, ***kwargs*)
Send a PUT request to the specified URI.

Args: uri (str): The URI to send the PUT request to. body (object): The data to be sent in the body of the PUT request. **kwargs:

Returns: object: The return data from the PUT request.

raise_unless_json (*ret*, *expected*)
Raise a ServerError unless we got back an HTTP 200 response with JSON containing all the expected values.

Args: ret (object): Return value to be checked. expected (dict): Expected keys and values that need to be found in the JSON response.

Raises: ServerError: If the HTTP response is anything but 200, or if the expected values are not found.

select (*cls*, *unique_id=None*, **args*, ***kwargs*)
Prepare a query against the Carbon Black data store.

Args: cls (class): The Model class (for example, Computer, Process, Binary, FileInstance) to query unique_id (optional): The unique id of the object to retrieve, to retrieve a single object by ID *args: **kwargs:

Returns: object: An instance of the Model class if a unique_id is provided, otherwise a Query object

url
Return the connection URL.

Returns: str: The connection URL.

5.5.2 Device API

The Carbon Black Cloud can be used to enumerate devices within your organization, and change their status via a control request.

You can use the select() method on the CbPSCBaseAPI to create a query object for Device objects, which can be used to locate a list of Devices.

Example:

```
>>> cbapi = CbPSCBaseAPI(...)
>>> devices = cbapi.select(Device).set_os("LINUX").status("ALL")
```

Selects all devices running Linux from the current organization.

Query Object:

class cbapi.psc.devices_query.**DeviceSearchQuery** (*doc_class*, *cb*)
Represents a query that is used to locate Device objects.

background_scan (*scan*)
Set the background scan option for the specified devices.

Parameters **scan** (*boolean*) – True to turn background scan on, False to turn it off.

bypass (*enable*)
Set the bypass option for the specified devices.

Parameters **enable** (*boolean*) – True to enable bypass, False to disable it.

delete_sensor ()
Delete the specified sensor devices.

download()
Uses the query parameters that have been set to download all device listings in CSV format.

Example:

```
>>> cb.select(Device).set_status(["ALL"]).download()
```

Returns The CSV raw data as returned from the server.

quarantine(*enable*)
Set the quarantine option for the specified devices.

Parameters **enable** (*boolean*) – True to enable quarantine, False to disable it.

set_ad_group_ids(*ad_group_ids*)
Restricts the devices that this query is performed on to the specified AD group IDs.

Parameters **ad_group_ids** – list of ints

Returns This instance

set_device_ids(*device_ids*)
Restricts the devices that this query is performed on to the specified device IDs.

Parameters **ad_group_ids** – list of ints

Returns This instance

set_exclude_sensor_versions(*sensor_versions*)
Restricts the devices that this query is performed on to exclude specified sensor versions.

Parameters **sensor_versions** – List of sensor versions to exclude

Returns This instance

set_last_contact_time(**args*, ***kwargs*)
Restricts the devices that this query is performed on to the specified last contact time (either specified as a start and end point or as a range).

Returns This instance

set_os(*operating_systems*)
Restricts the devices that this query is performed on to the specified operating systems.

Parameters **operating_systems** – list of operating systems

Returns This instance

set_policy_ids(*policy_ids*)
Restricts the devices that this query is performed on to the specified policy IDs.

Parameters **policy_ids** – list of ints

Returns This instance

set_status(*statuses*)
Restricts the devices that this query is performed on to the specified status values.

Parameters **statuses** – list of strings

Returns This instance

set_target_priorities(*target_priorities*)
Restricts the devices that this query is performed on to the specified target priority values.

Parameters **target_priorities** – list of strings

Returns This instance

sort_by(*key*, *direction='ASC'*)
Sets the sorting behavior on a query's results.

Example:

```
>>> cb.select(Device).sort_by("name")
```

Parameters

- **key** – the key in the schema to sort by
- **direction** – the sort order, either "ASC" or "DESC"

Return type *DeviceSearchQuery*

uninstall_sensor()
Uninstall the specified sensor devices.

update_policy(*policy_id*)
Set the current policy for the specified devices.

Parameters **policy_id** (*int*) – ID of the policy to set for the devices.

update_sensor_version(*sensor_version*)
Update the sensor version for the specified devices.

Parameters **sensor_version** (*dict*) – New version properties for the sensor.

Model Object:

class cbapi.psc.models.**Device**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a Device object in the Carbon Black server.

Variables

- ***activation_code*** – Device activation code
- ***activation_code_expiry_time*** – When the expiration code expires and cannot be used to register a device
- ***ad_group_id*** – Device's AD group
- ***av_ave_version*** – AVE version (part of AV Version)
- ***av_engine*** – Current AV version
- ***av_last_scan_time*** – Last AV scan time
- ***av_master*** – Whether the device is an AV Master (?)
- ***av_pack_version*** – Pack version (part of AV Version)
- ***av_product_version*** – AV Product version (part of AV Version)
- ***av_status*** – AV Statuses
- ***av_update_servers*** – Device's AV servers
- ***av_vdf_version*** – VDF version (part of AV Version)
- ***current_sensor_policy_name*** – Current MSM policy name
- ***deregistered_time*** – When the device was deregistered with the PSC backend
- ***device_id*** – ID of the device

- ***device_meta_data_item_list*** – MSM Device metadata
- ***device_owner_id*** – ID of the user who owns the device
- ***email*** – Email of the user who owns the device
- ***encoded_activation_code*** – Encoded device activation code
- ***first_name*** – First name of the user who owns the device
- ***id*** – ID of the device
- ***last_contact_time*** – Time the device last checked into the PSC backend
- ***last_device_policy_changed_time*** – Last time the device's policy was changed
- ***last_device_policy_requested_time*** – Last time the device requested policy updates
- ***last_external_ip_address*** – Device's external IP
- ***last_internal_ip_address*** – Device's internal IP
- ***last_location*** – Location of the device (on-/off-premises)
- ***last_name*** – Last name of the user who owns the device
- ***last_policy_updated_time*** – Last time the device was MSM processed
- ***last_reported_time*** – Time when device last reported an event to PSC backend
- ***last_reset_time*** – When the sensor was last reset
- ***last_shutdown_time*** – When the device last shut down
- ***linux_kernel_version*** – Linux kernel version
- ***login_user_name*** – Last acive logged in username
- ***mac_address*** – Device's hardware MAC address
- ***middle_name*** – Middle name of the user who owns the device
- ***name*** – Device Hostname
- ***organization_id*** – Org ID to which the device belongs
- ***organization_name*** – Name of the org that owns this device
- ***os*** – Device type
- ***os_version*** – Version of the OS
- ***passive_mode*** – Whether the device is in passive mode (bypass?)
- ***policy_id*** – ID of the policy this device is using
- ***policy_name*** – Name of the policy this device is using
- ***policy_override*** – Manually assigned policy (overrides mass sensor management)
- ***quarantined*** – Whether the device is quarantined
- ***registered_time*** – When the device was registered with the PSC backend
- ***scan_last_action_time*** – When the background scan was last active
- ***scan_last_complete_time*** – When the background scan was last completed
- ***scan_status*** – Background scan status

- ***sensor_out_of_date*** – Whether the device is out of date
- ***sensor_states*** – Active sensor states
- ***sensor_version*** – Version of the PSC sensor
- ***status*** – Device status
- ***target_priority_type*** – Priority of the device
- ***uninstall_code*** – Code to enter to uninstall this device
- ***vdi_base_device*** – VDI Base device
- ***virtual_machine*** – Whether this device is a Virtual Machine (VMware AppDefense integration
- ***virtualization_provider*** – VM Virtualization Provider
- ***windows_platform*** – Type of windows platform (client/server, x86/x64)

activation_code = None

activation_code_expiry_time = None

ad_group_id = None

av_ave_version = None

av_engine = None

av_last_scan_time = None

av_master = None

av_pack_version = None

av_product_version = None

av_status = []

av_update_servers = []

av_vdf_version = None

background_scan(*flag*)
Set the background scan option for this device.

Parameters **flag** (*boolean*) – True to turn background scan on, False to turn it off.

bypass(*flag*)
Set the bypass option for this device.

Parameters **flag** (*boolean*) – True to enable bypass, False to disable it.

current_sensor_policy_name = None

delete_sensor()
Delete this sensor device.

deregistered_time = None

device_id = None

device_meta_data_item_list = []

device_owner_id = None

email = None

encoded_activation_code = None

first_name = None

id = None

last_contact_time = None

last_device_policy_changed_time = None

last_device_policy_requested_time = None

last_external_ip_address = None

last_internal_ip_address = None

last_location = None

last_name = None

last_policy_updated_time = None

last_reported_time = None

last_reset_time = None

last_shutdown_time = None

linux_kernel_version = None

login_user_name = None

lr_session()

Retrieve a Live Response session object for this Device.

Returns Live Response session object

Return type cbapi.defense.cblr.LiveResponseSession

Raises ***ApiError*** – if there is an error establishing a Live Response session for this Device

mac_address = None

middle_name = None

name = None

organization_id = None

organization_name = None

os = None

os_version = None

passive_mode = None

policy_id = None

policy_name = None

policy_override = None

primary_key = 'id'

quarantine(*flag*)

Set the quarantine option for this device.

Parameters **flag** (*boolean*) – True to enable quarantine, False to disable it.

quarantined = None

registered_time = None

scan_last_action_time = None

scan_last_complete_time = None

scan_status = None

sensor_out_of_date = None

sensor_states = []

sensor_version = None

status = None

target_priority_type = None

uninstall_code = None

uninstall_sensor()
Uninstall this sensor device.

update_policy(*policy_id*)
Set the current policy for this device.

Parameters **policy_id** (*int*) – ID of the policy to set for the devices.

update_sensor_version(*sensor_version*)
Update the sensor version for this device.

Parameters **sensor_version** (*dict*) – New version properties for the sensor.

urlobject = '/appservices/v6/orgs/{0}/devices'

urlobject_single = '/appservices/v6/orgs/{0}/devices/{1}'

vdi_base_device = None

virtual_machine = None

virtualization_provider = None

windows_platform = None

5.5.3 Alerts API

Using the API, you can search for alerts within your organization, and dismiss or undismiss them, either individually or in bulk.

You can use the select() method on the CbPSCBaseAPI to create a query object for BaseAlert objects, which can be used to locate a list of alerts. You can also search for more specialized alert types:

- CBAnalyticsAlert - Alerts from CB Analytics
- VMwareAlert - Alerts from VMware
- WatchlistAlert - Alerts from watch lists

Example:

```
>>> cbapi = CbPSCBaseAPI(...)
>>> alerts = cbapi.select(BaseAlert).set_device_os(["WINDOWS"]).set_process_name([
↪"IEXPLORE.EXE"])
```

Selects all alerts on a Windows device running the Internet Explorer process.

Individual alerts may have their status changed using the dismiss() or update() methods on the BaseAlert object. To dismiss multiple alerts at once, you can use the dismiss() or update() methods on the standard query, after adding criteria to it. This method returns a request ID, which can be used to create a WorkflowStatus object; querying this object's "finished" property will let you know when the operation is finished.

Example:

```
>>> cbapi = CbPSCBaseAPI(...)
>>> query = cbapi.select(BaseAlert).set_process_name(["IEXPLORE.EXE"])
>>> reqid = query.dismiss("Using Chrome")
>>> stat = cbapi.select(WorkflowStatus, reqid)
>>> while not stat.finished:
>>>     # wait for it to finish
```

This dismisses all alerts which reference the Internet Explorer process.

Query Objects:

class cbapi.psc.alerts_query.**BaseAlertSearchQuery**(*doc_class*, *cb*)

Represents a query that is used to locate BaseAlert objects.

dismiss(*remediation=None*, *comment=None*)

Dismiss all alerts matching the given query. The alerts will be left in a DISMISSED state after this request.

Parameters

- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

facets(*fieldlist*, *max_rows=0*)

Return information about the facets for this alert by search, using the defined criteria.

Parameters

- **list** (*fieldlist*) – List of facet field names. Valid names are "ALERT_TYPE", "CATEGORY", "REPUTATION", "WORKFLOW", "TAG", "POLICY_ID", "POLICY_NAME", "DEVICE_ID", "DEVICE_NAME", "APPLICATION_HASH", "APPLICATION_NAME", "STATUS", "RUN_STATE", "POLICY_APPLIED_STATE", "POLICY_APPLIED", and "SENSOR_ACTION".
- **int** (*max_rows*) – The maximum number of rows to return. 0 means return all rows.

Returns A list of facet information specified as dicts.

set_alert_ids(*alert_ids*)

Restricts the alerts that this query is performed on to the specified alert IDs.

Parameters **list** (*alert_ids*) – List of string alert IDs.

Returns This instance

set_categories(*categories*)

Restricts the alerts that this query is performed on to the specified categories.

Parameters **list** (*categories*) – List of categories to be restricted to. Valid categories are "THREAT", "MONITORED", "INFO", "MINOR", "SERIOUS", and "CRITICAL."

Returns This instance

set_create_time(**args*, ***kwargs*)
Restricts the alerts that this query is performed on to the specified creation time (either specified as a start and end point or as a range).

Returns This instance

set_device_ids(*device_ids*)
Restricts the alerts that this query is performed on to the specified device IDs.

Parameters **list** (*device_ids*) – list of integer device IDs

Returns This instance

set_device_names(*device_names*)
Restricts the alerts that this query is performed on to the specified device names.

Parameters **list** (*device_names*) – list of string device names

Returns This instance

set_device_os(*device_os*)
Restricts the alerts that this query is performed on to the specified device operating systems.

Parameters **list** (*device_os*) – List of string operating systems. Valid values are "WINDOWS", "ANDROID", "MAC", "IOS", "LINUX", and "OTHER."

Returns This instance

set_device_os_versions(*device_os_versions*)
Restricts the alerts that this query is performed on to the specified device operating system versions.

Parameters **list** (*device_os_versions*) – List of string operating system versions.

Returns This instance

set_device_username(*users*)
Restricts the alerts that this query is performed on to the specified user names.

Parameters **list** (*users*) – List of string user names.

Returns This instance

set_group_results(*do_group*)
Specifies whether or not to group the results of the query.

Parameters **boolean** (*do_group*) – True to group the results, False to not do so.

Returns This instance

set_legacy_alert_ids(*alert_ids*)
Restricts the alerts that this query is performed on to the specified legacy alert IDs.

Parameters **list** (*alert_ids*) – List of string legacy alert IDs.

Returns This instance

set_minimum_severity(*severity*)
Restricts the alerts that this query is performed on to the specified minimum severity level.

Parameters **int** (*severity*) – The minimum severity level for alerts.

Returns This instance

set_policy_ids(*policy_ids*)
Restricts the alerts that this query is performed on to the specified policy IDs.

Parameters **list** (*policy_ids*) – list of integer policy IDs

Returns This instance

set_policy_names(*policy_names*)
Restricts the alerts that this query is performed on to the specified policy names.

Parameters **list** (*policy_names*) – list of string policy names

Returns This instance

set_process_names(*process_names*)
Restricts the alerts that this query is performed on to the specified process names.

Parameters **list** (*process_names*) – list of string process names

Returns This instance

set_process_sha256(*shas*)
Restricts the alerts that this query is performed on to the specified process SHA-256 hash values.

Parameters **list** (*shas*) – list of string process SHA-256 hash values

Returns This instance

set_reputations(*reps*)
Restricts the alerts that this query is performed on to the specified reputation values.

Parameters **list** (*reps*) – List of string reputation values. Valid values are "KNOWN_MALWARE", "SUSPECT_MALWARE", "PUP", "NOT_LISTED", "ADAPTIVE_WHITE_LIST", "COMMON_WHITE_LIST", "TRUSTED_WHITE_LIST", and "COMPANY_BLACK_LIST".

Returns This instance

set_tags(*tags*)
Restricts the alerts that this query is performed on to the specified tag values.

Parameters **list** (*tags*) – list of string tag values

Returns This instance

set_target_priorities(*priorities*)
Restricts the alerts that this query is performed on to the specified target priority values.

Parameters **list** (*priorities*) – List of string target priority values. Valid values are "LOW", "MEDIUM", "HIGH", and "MISSION_CRITICAL".

Returns This instance

set_threat_ids(*threats*)
Restricts the alerts that this query is performed on to the specified threat ID values.

Parameters **list** (*threats*) – list of string threat ID values

Returns This instance

set_types(*alerttypes*)
Restricts the alerts that this query is performed on to the specified alert type values.

Parameters **list** (*alerttypes*) – List of string alert type values. Valid values are "CB_ANALYTICS", "VMWARE", and "WATCHLIST".

Returns This instance

set_workflows(*workflow_vals*)
Restricts the alerts that this query is performed on to the specified workflow status values.

Parameters **list** (*workflow_vals*) – List of string alert type values. Valid values are "OPEN" and "DISMISSED".

Returns This instance

sort_by(*key*, *direction='ASC'*)
Sets the sorting behavior on a query's results.

Example:

```
>>> cb.select(BaseAlert).sort_by("name")
```

Parameters

- **key** – the key in the schema to sort by
- **direction** – the sort order, either "ASC" or "DESC"

Return type *BaseAlertSearchQuery*

update(*remediation=None*, *comment=None*)
Update all alerts matching the given query. The alerts will be left in an OPEN state after this request.

Parameters

- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

class cbapi.psc.alerts_query.**CBAnalyticsAlertSearchQuery**(*doc_class*, *cb*)
Represents a query that is used to locate CBAnalyticsAlert objects.

set_blocked_threat_categories(*categories*)
Restricts the alerts that this query is performed on to the specified threat categories that were blocked.

Parameters **list** (*categories*) – List of threat categories to look for. Valid values are "UNKNOWN", "NON_MALWARE", "NEW_MALWARE", "KNOWN_MALWARE", and "RISKY_PROGRAM".

Returns This instance.

set_device_locations(*locations*)
Restricts the alerts that this query is performed on to the specified device locations.

Parameters **list** (*locations*) – List of device locations to look for. Valid values are "ON-SITE", "OFFSITE", and "UNKNOWN".

Returns This instance.

set_kill_chain_statuses(*statuses*)
Restricts the alerts that this query is performed on to the specified kill chain statuses.

Parameters **list** (*statuses*) – List of kill chain statuses to look for. Valid values are "RECONNAISSANCE", "WEAPONIZE", "DELIVER_EXPLOIT", "IN-STALL_RUN","COMMAND_AND_CONTROL", "EXECUTE_GOAL", and "BREACH".

Returns This instance.

set_not_blocked_threat_categories(*categories*)
Restricts the alerts that this query is performed on to the specified threat categories that were NOT blocked.

Parameters **list** (*categories*) – List of threat categories to look for. Valid values are "UNKNOWN", "NON_MALWARE", "NEW_MALWARE", "KNOWN_MALWARE", and "RISKY_PROGRAM".

Returns This instance.

set_policy_applied(*applied_statuses*)
Restricts the alerts that this query is performed on to the specified status values showing whether policies were applied.

Parameters **list** (*applied_statuses*) – List of status values to look for. Valid values are "APPLIED" and "NOT_APPLIED".

Returns This instance.

set_reason_code(*reason*)
Restricts the alerts that this query is performed on to the specified reason codes (enum values).

Parameters **list** (*reason*) – List of string reason codes to look for.

Returns This instance.

set_run_states(*states*)
Restricts the alerts that this query is performed on to the specified run states.

Parameters **list** (*states*) – List of run states to look for. Valid values are "DID_NOT_RUN", "RAN", and "UNKNOWN".

Returns This instance.

set_sensor_actions(*actions*)
Restricts the alerts that this query is performed on to the specified sensor actions.

Parameters **list** (*actions*) – List of sensor actions to look for. Valid values are "POLICY_NOT_APPLIED", "ALLOW", "ALLOW_AND_LOG", "TERMINATE", and "DENY".

Returns This instance.

set_threat_cause_vectors(*vectors*)
Restricts the alerts that this query is performed on to the specified threat cause vectors.

Parameters **list** (*vectors*) – List of threat cause vectors to look for. Valid values are "EMAIL", "WEB", "GENERIC_SERVER", "GENERIC_CLIENT", "REMOTE_DRIVE", "REMOVABLE_MEDIA", "UNKNOWN", "APP_STORE", and "THIRD_PARTY".

Returns This instance.

class cbapi.psc.alerts_query.**VMwareAlertSearchQuery**(*doc_class*, *cb*)
Represents a query that is used to locate VMwareAlert objects.

set_group_ids(*groupids*)
Restricts the alerts that this query is performed on to the specified AppDefense-assigned alarm group IDs.

Parameters **list** (*groupids*) – List of (integer) AppDefense-assigned alarm group IDs.

Returns This instance.

class cbapi.psc.alerts_query.**WatchlistAlertSearchQuery**(*doc_class*, *cb*)
Represents a query that is used to locate WatchlistAlert objects.

set_watchlist_ids(*ids*)
Restricts the alerts that this query is performed on to the specified watchlist ID values.

Parameters **list** (*ids*) – list of string watchlist ID values

Returns This instance

set_watchlist_names(*names*)
Restricts the alerts that this query is performed on to the specified watchlist name values.

Parameters **list** (*names*) – list of string watchlist name values

Returns This instance

Model Objects:

class cbapi.psc.models.**Workflow**(*cb*, *initial_data=None*)
Represents a Workflow object in the Carbon Black server.

Variables

- ***changed_by*** – Username of the user who changed the workflow
- ***comment*** – Comment when updating the workflow
- ***last_update_time*** – When the workflow was last updated
- ***remediation*** – Alert remediation code. Indicates the result of the investigation into the alert
- ***state*** – State of the workflow

changed_by = None

comment = None

last_update_time = None

remediation = None

state = None

class cbapi.psc.models.**BaseAlert**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a BaseAlert object in the Carbon Black server.

Variables

- ***category*** – Alert category - Monitored vs Threat
- ***create_time*** – Time the alert was created
- ***device_id*** – ID of the device
- ***device_name*** – Device name
- ***device_os*** – Device OS
- ***device_os_version*** – Device OS Version
- ***device_username*** – Logged on user during the alert. This is filled on a best-effort approach. If the user is not available it may be populated with the device owner
- ***first_event_time*** – Time of the first event in an alert
- ***group_details*** – Group details for when alert grouping is on
- ***id*** – Unique ID for this alert
- ***last_event_time*** – Time of the last event in an alert
- ***last_update_time*** – Time the alert was last updated
- ***legacy_alert_id*** – Unique short ID for this alert. This is deprecated and only available on alerts stored in the old schema.

- ***notes_present*** – Are notes present for this threatId
- ***org_key*** – Unique identifier for the organization to which the alert belongs
- ***policy_id*** – ID of the policy the device was in at the time of the alert
- ***policy_name*** – Name of the policy the device was in at the time of the alert
- ***severity*** – Threat ranking
- ***tags*** – Tags for the alert
- ***target_value*** – Device priority as assigned via the policy
- ***threat_id*** – ID of the threat to which this alert belongs. Threats are comprised of a combination of factors that can be repeated across devices.
- ***type*** – Type of the alert
- ***workflow*** – User-updatable status of the alert

category = None

create_time = None

device_id = None

device_name = None

device_os = None

device_os_version = None

device_username = None

dismiss (*remediation=None*, *comment=None*)
Dismiss this alert.

Parameters

- **str** (*comment*) – The remediation status to set for the alert.
- **str** – The comment to set for the alert.

dismiss_threat (*remediation=None*, *comment=None*)
Dismiss alerts for this threat.

Parameters

- **str** (*comment*) – The remediation status to set for the alert.
- **str** – The comment to set for the alert.

first_event_time = None

group_details = {}

id = None

last_event_time = None

last_update_time = None

legacy_alert_id = None

notes_present = None

org_key = None

policy_id = None

policy_name = None

primary_key = 'id'

severity = None

tags = []

target_value = None

threat_id = None

type = None

update(*remediation=None*, *comment=None*)
Update this alert.

Parameters

- **str** (*comment*) – The remediation status to set for the alert.
- **str** – The comment to set for the alert.

update_threat(*remediation=None*, *comment=None*)
Update alerts for this threat.

Parameters

- **str** (*comment*) – The remediation status to set for the alert.
- **str** – The comment to set for the alert.

urlobject = '/appservices/v6/orgs/{0}/alerts'

urlobject_single = '/appservices/v6/orgs/{0}/alerts/{1}'

workflow = {}

workflow_

class cbapi.psc.models.**CBAnalyticsAlert**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a CBAnalyticsAlert object in the Carbon Black server.

urlobject = '/appservices/v6/orgs/{0}/alerts/cbanalytics'

class cbapi.psc.models.**VMwareAlert**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a VMwareAlert object in the Carbon Black server.

urlobject = '/appservices/v6/orgs/{0}/alerts/vmware'

class cbapi.psc.models.**WatchlistAlert**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a WatchlistAlert object in the Carbon Black server.

urlobject = '/appservices/v6/orgs/{0}/alerts/watchlist'

class cbapi.psc.models.**WorkflowStatus**(*cb*, *model_unique_id*, *initial_data=None*)
Represents a WorkflowStatus object in the Carbon Black server.

Variables

- ***errors*** – Errors for dismiss alerts or threats, if no errors it won't be included in response
- ***failed_ids*** – Failed ids
- ***id*** – Time based id for async job, it's not unique across the orgs
- ***num_hits*** – Total number of alerts to be operated on
- ***num_success*** – Successfully operated number of alerts

- ***status*** – Status for the async progress
- ***workflow*** – Requested workflow change

errors = []

failed_ids = []

finished

id = None

id_

in_progress

num_hits = None

num_success = None

primary_key = 'id'

queued

status = None

urlobject_single = '/appservices/v6/orgs/{0}/workflow/status/{1}'

workflow = {}

workflow_

5.6 CB LiveQuery API

This page documents the public interfaces exposed by cbapi when communicating with Carbon Black LiveQuery devices.

5.6.1 Main Interface

To use cbapi with Carbon Black LiveQuery, you use CbLiveQueryAPI objects.

The LiveQuery API is used in two stages: run submission and result retrieval.

class cbapi.psc.livequery.rest_api.**CbLiveQueryAPI**(**args*, ***kwargs*)
The main entry point into the Carbon Black Cloud LiveQuery API.

Parameters **profile** (*str*) – (optional) Use the credentials in the named profile when connecting to the Carbon Black server. Uses the profile named 'default' when not specified.

Usage:

```
>>> from cbapi.psc.livequery import CbLiveQueryAPI
>>> cb = CbLiveQueryAPI(profile="production")
```

alert_search_suggestions(*query*)
Returns suggestions for keys and field values that can be used in a search.

Parameters **str** (*query*) – A search query to use.

Returns A list of search suggestions expressed as dict objects.

api_json_request(*method*, *uri*, ***kwargs*)
Submit a request to the server.

Args: method (str): HTTP method to use. uri (str): URI to submit the request to. **kwargs (dict): Additional arguments.

Returns: object: Result of the operation.

Raises: ServerError: If there's an error output from the server.

bulk_threat_dismiss(*threat_ids*, *remediation=None*, *comment=None*)
Dismiss the alerts associated with multiple threat IDs. The alerts will be left in a DISMISSED state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

bulk_threat_update(*threat_ids*, *remediation=None*, *comment=None*)
Update the alert status of alerts associated with multiple threat IDs. The alerts will be left in an OPEN state after this request.

Parameters

- **list** (*threat_ids*) – List of string threat IDs.
- **str** (*comment*) – The remediation state to set for all alerts.
- **str** – The comment to set for all alerts.

Returns The request ID, which may be used to select a WorkflowStatus object.

create(*cls*, *data=None*)
Create a new object.

Args: cls (class): The Model class (only some models can be created, for example, Feed, Notification, ...) data (object): The data used to initialize the new object

Returns: Model: An empty instance of the model class.

Raises: ApiError: If the Model cannot be created.

delete_object(*uri*)
Send a DELETE request to the specified URI.

Args: uri (str): The URI to send the DELETE request to.

Returns: object: The return data from the DELETE request.

device_background_scan(*device_ids*, *scan*)
Set the background scan option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **scan** (*boolean*) – True to turn background scan on, False to turn it off.

device_bypass(*device_ids*, *enable*)
Set the bypass option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.

- **enable** (*boolean*) – True to enable bypass, False to disable it.

device_delete_sensor(*device_ids*)
Delete the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be deleted.

device_quarantine(*device_ids*, *enable*)
Set the quarantine option for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be set.
- **enable** (*boolean*) – True to enable quarantine, False to disable it.

device_uninstall_sensor(*device_ids*)
Uninstall the specified sensor devices.

Parameters **device_ids** (*list*) – List of IDs of devices to be uninstalled.

device_update_policy(*device_ids*, *policy_id*)
Set the current policy for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **policy_id** (*int*) – ID of the policy to set for the devices.

device_update_sensor_version(*device_ids*, *sensor_version*)
Update the sensor version for the specified devices.

Parameters

- **device_ids** (*list*) – List of IDs of devices to be changed.
- **sensor_version** (*dict*) – New version properties for the sensor.

get_object(*uri*, *query_parameters=None*, *default=None*)
Submit a GET request to the server and parse the result as JSON before returning.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response.

Returns: object: Result of the GET request.

get_raw_data(*uri*, *query_parameters=None*, *default=None*, ***kwargs*)
Submit a GET request to the server and return the result without parsing it.

Args: uri (str): The URI to send the GET request to. query_parameters (object): Parameters for the query. default (object): What gets returned in the event of an empty response. **kwargs:

Returns: object: Result of the GET request.

post_object(*uri*, *body*, ***kwargs*)
Send a POST request to the specified URI.

Args: uri (str): The URI to send the POST request to. body (object): The data to be sent in the body of the POST request. **kwargs:

Returns: object: The return data from the POST request.

put_object(*uri*, *body*, ***kwargs*)
Send a PUT request to the specified URI.

Args: uri (str): The URI to send the PUT request to. body (object): The data to be sent in the body of the PUT request. **kwargs:

Returns: object: The return data from the PUT request.

raise_unless_json(*ret*, *expected*)
Raise a ServerError unless we got back an HTTP 200 response with JSON containing all the expected values.

Args: ret (object): Return value to be checked. expected (dict): Expected keys and values that need to be found in the JSON response.

Raises: ServerError: If the HTTP response is anything but 200, or if the expected values are not found.

select(*cls*, *unique_id=None*, **args*, ***kwargs*)
Prepare a query against the Carbon Black data store.

Args: cls (class): The Model class (for example, Computer, Process, Binary, FileInstance) to query unique_id (optional): The unique id of the object to retrieve, to retrieve a single object by ID *args: **kwargs:

Returns: object: An instance of the Model class if a unique_id is provided, otherwise a Query object

url
Return the connection URL.

Returns: str: The connection URL.

5.6.2 Queries

The LiveQuery API uses QueryBuilder instances to construct structured or unstructured (i.e., raw string) queries. You can either construct these instances manually, or allow `CbLiveQueryAPI.select()` to do it for you:

class cbapi.psc.livequery.query.**QueryBuilder**(***kwargs*)
Provides a flexible interface for building prepared queries for the CB PSC backend.

This object can be instantiated directly, or can be managed implicitly through the `select()` API.

and_(*q*, ***kwargs*)
Adds a conjunctive filter to a query.

Parameters

- **q** – string or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

not_(*q*, ***kwargs*)
Adds a negative filter to a query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

or_(*q*, ***kwargs*)
Adds a disjunctive filter to a query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

where(*q*, ***kwargs*)
Adds a conjunctive filter to a query.

Parameters

- **q** – string or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns QueryBuilder object

Return type *QueryBuilder*

class cbapi.psc.livequery.query.**RunQuery**(*doc_class*, *cb*)
Represents a query that either creates or retrieves the status of a LiveQuery run.

device_ids(*device_ids*)
Restricts the devices that this LiveQuery run is performed on to the given IDs.

Parameters **device_ids** – list of ints

Returns This instance

device_types(*device_types*)
Restricts the devices that this LiveQuery run is performed on to the given device types.

Parameters **device_types** – list of strs

Returns This instance

name(*name*)
Sets this LiveQuery run's name. If no name is explicitly set, the run is named after its SQL.

Parameters **name** – The run name

Returns This instance

notify_on_finish()
Sets the notify-on-finish flag on this LiveQuery run.

Returns This instance

policy_ids(*policy_ids*)
Restricts this LiveQuery run to the given policy IDs.

Parameters **policy_ids** – list of ints

Returns This instance

submit()
Submits this LiveQuery run.

Returns A new Run instance containing the run's status

where(*sql*)
Sets this LiveQuery run's underlying SQL.

Parameters **sql** – The SQL to execute

Returns This instance

class cbapi.psc.livequery.models.**ResultQuery**(*doc_class*, *cb*)
Represents a query that retrieves results from a LiveQuery run.

all()
Returns all the items of a query as a list.

Returns List of query items

and_(*q=None*, ***kwargs*)
Add a conjunctive filter to this query.

Parameters

- **q** – Query string or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type Query

criteria(***kwargs*)
Sets the filter criteria on a query's results.

Example:

```
>>> cb.select(Result).run_id(my_run).criteria(device_id=[123, 456])
```

first()
Returns the first item that would be returned as the result of a query.

Returns First query item

not_(*q=None*, ***kwargs*)
Adds a negated filter to this query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type Query

one()
Returns the only item that would be returned by a query.

Returns Sole query return item

Raises ***MoreThanOneResultError*** – If the query returns zero items, or more than one item

or_(*q=None*, ***kwargs*)
Add a disjunctive filter to this query.

Parameters

- **q** – *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type `Query`

run_id(*run_id*)
Sets the run ID to query results for.

Example:

```
>>> cb.select(Result).run_id(my_run)
```

sort_by(*key*, *direction='ASC'*)
Sets the sorting behavior on a query's results.

Example:

```
>>> cb.select(Result).run_id(my_run).where(username="foobar").sort_by("uid")
```

Parameters

- **key** – the key in the schema to sort by
- **direction** – the sort order, either "ASC" or "DESC"

Return type *ResultQuery*

where(*q=None*, ***kwargs*)
Add a filter to this query.

Parameters

- **q** – Query string, `QueryBuilder`, or *solrq.Q* object
- **kwargs** – Arguments to construct a *solrq.Q* with

Returns Query object

Return type `Query`

5.6.3 Models

class cbapi.psc.livequery.models.**Run**(*cb*, *model_unique_id=None*, *initial_data=None*)
Represents a Run object in the Carbon Black server.

Variables

- **template_id** – Placeholder
- ***org_key*** – The organization key for this run
- ***name*** – The name of the LiveQuery run
- ***id*** – The run's unique ID
- **sql** – The LiveQuery query
- **created_by** – Placeholder
- ***create_time*** – When this run was created
- **status_update_time** – When the status of this run was last updated
- **timeout_time** – Placeholder
- **cancellation_time** – Placeholder

- **cancelled_by** – Placeholder
- **archive_time** – Placeholder
- **archived_by** – Placeholder
- ***notify_on_finish*** – Whether or not to send an email on query completion
- **active_org_devices** – The number of devices active in the organization
- ***status*** – The run status
- **device_filter** – Any device filter rules associated with the run
- **schedule** – Placeholder
- **last_result_time** – When the most recent result for this run was reported
- **total_results** – Placeholder
- **match_count** – Placeholder
- **no_match_count** – Placeholder
- **error_count** – Placeholder
- **not_supported_count** – Placeholder
- **cancelled_count** – Placeholder

class cbapi.psc.livequery.models.**Result**(*cb*, *initial_data*)

Represents a Result object in the Carbon Black server.

Variables

- ***id*** – The result's unique ID
- **device** – The device associated with the result
- ***status*** – The result's status
- **time_received** – The time at which this result was received
- **device_message** – Placeholder
- **fields** – The fields returned by the backing osquery query
- **metrics** – Metrics associated with the result's host

class Device(*cb*, *initial_data*)

Represents a Device object in the Carbon Black server.

class Fields(*cb*, *initial_data*)

Represents a Fields object in the Carbon Black server.

class Metrics(*cb*, *initial_data*)

Represents a Metrics object in the Carbon Black server.

device_

Returns the reified `Result.Device` for this result.

fields_

Returns the reified `Result.Fields` for this result.

metrics_

Returns the reified `Result.Metrics` for this result.

5.7 Exceptions

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

5.7.1 Exception Classes

exception cbapi.errors.**ApiError**(*message=None*, *original_exception=None*)
Base class for all CBAPI errors; also raised for generic internal errors.

Initialize the ApiError.

Args: message (str): The actual error message. original_exception (Exception): The exception that caused this one to be raised.

exception cbapi.errors.**CredentialError**(*message=None*, *original_exception=None*)
The credentials had an unspecified error.

Initialize the ApiError.

Args: message (str): The actual error message. original_exception (Exception): The exception that caused this one to be raised.

exception cbapi.errors.**ServerError**(*error_code*, *message*, *result=None*, *original_exception=None*)
A ServerError is raised when an HTTP 5xx error code is returned from the Carbon Black server.

Initialize the ServerError.

Args: error_code (int): The error code that was received from the server. message (str): The actual error message. result (object): The result of the operation from the server. original_exception (Exception): The exception that caused this one to be raised.

exception cbapi.errors.**ObjectNotFoundError**(*uri*, *message=None*, *original_exception=None*)
The requested object could not be found in the Carbon Black datastore.

Initialize the ObjectNotFoundError.

Args: uri (str): The URI of the action that failed. message (str): The error message. original_exception (Exception): The exception that caused this one to be raised.

exception cbapi.errors.**MoreThanOneResultError**(*message=None*, *original_exception=None*)
Only one object was requested, but multiple matches were found in the Carbon Black datastore.

Initialize the ApiError.

Args: message (str): The actual error message. original_exception (Exception): The exception that caused this one to be raised.

exception cbapi.errors.**InvalidObjectError**(*message=None*, *original_exception=None*)
An invalid object was received by the server.

Initialize the ApiError.

Args: message (str): The actual error message. original_exception (Exception): The exception that caused this one to be raised.

exception cbapi.errors.**TimeoutError**(*uri=None*, *error_code=None*, *message=None*, *original_exception=None*)
A requested operation timed out.

Initialize the TimeoutError.

Args: uri (str): The URI of the action that timed out. error_code (int): The error code that was received from the server. message (str): The error message. original_exception (Exception): The exception that caused this one to be raised.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

A

B

C

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W