
cbapi Documentation

Release 1.7.10

Carbon Black Developer Network

Apr 18, 2024

CONTENTS

1	Major Features	3
2	API Credentials	5
3	Backwards & Forwards Compatibility	7
4	User Guide	9
4.1	Installation	9
4.2	Getting Started	11
4.3	Concepts	12
4.4	Logging & Diagnostics	17
4.5	EDR (CB Response) API Examples	18
4.6	CbAPI and Live Response	27
4.7	CbAPI Changelog	28
5	API Documentation	43
5.1	EDR (CB Response) API	43
5.2	Carbon Black App Control (CB Protection) API	63
5.3	Exceptions	83
6	Indices and tables	85
	Python Module Index	87
	Index	89

Release v1.7.10.

CBAPI provides a straightforward interface to the VMware Carbon Black products Carbon Black EDR and Carbon Black App Control. This library provides a Pythonic layer to access the raw power of the REST APIs of these Carbon Black products, making it easier to query data from 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'
>>> #
```

(continues on next page)

(continued from previous page)

```
>>> # change the policy ID
>>> #
>>> fi.computer.policyId = 3
>>> fi.computer.save()
```

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 and Carbon Black App Control 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 two 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 or Carbon Black App Control servers.

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 \(CB Response\)](#), or [Carbon Black App Control \(CB Protection\)](#).

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.response` for Carbon Black EDR (CB Response)
- `credentials.protection` for Carbon Black App Control (CB Protection)

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

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

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.
- **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 envvar for holding the EDR (CbR) or App Control (CbP) api token.

The *CBAPI_URL* envvar holds the FQDN of the target, an EDR (CbR) server specified just as they are in the configuration file format specified above.

The optional *CBAPI_SSL_VERIFY* envvar 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.

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).

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` or `cbapi-protection` 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.

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 from the “cb” to “cbisawesome”:

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

(continues on next page)

(continued from previous page)

```
>>> 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
-----
```

(continues on next page)

(continued from previous page)

```
(*)      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: {'result': '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: {'result': 'success'}
```

(continues on next page)

(continued from previous page)

```
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
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.
```

(continues on next page)

(continued from previous page)

```

→ 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)
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
...         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)
```

(continues on next page)

(continued from previous page)

```

...     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-IA9NQ1GN80I (part of sensor group Default Group):
- At 2016-11-11 20:59:31.631000, process net use y: \\test\blah was executed by WIN-
↳ IA9NQ1GN80I\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/000000003-0000-036c-01d2-2efd3af51186/1/event took 0.045s
↳ (response 200)
On host WIN-IA9NQ1GN80I (part of sensor group Default Group):
- At 2016-10-25 20:20:29.790000, process "C:\Windows\system32\cmd.exe" was executed by
↳ WIN-IA9NQ1GN80I\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-IA9NQ1GN80I").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-IA9NQ1GN80I+username%3Abit9rad&rows=0&start=0 took 0.
↳ 024s (response 200)
    chrome.exe:  23.4% =====
thumbnailextrac: 15.4% =====

```

(continues on next page)

(continued from previous page)

```

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: {'result': '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}
↳ 1}
HTTP POST /api/group took 0.282s (response 200)
Received response: {'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-IA9NQ1GN80I").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-IA9NQ1GN80I").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 Documentation - Release Feb 14, 2024

Updates

- Removed references to and documentation about Carbon Black Cloud. CBAPI (this SDK) is not maintained for Carbon Black Cloud.

Users of Carbon Black Cloud must transition to the Carbon Black Cloud Python SDK. Please see [Carbon Black Cloud Python SDK on the Developer Network](#) for details.

4.7.2 CbAPI 1.7.10 - Release Feb 1, 2023

Bug Fixes

- Update CbAPI to use packaging instead of distutils for python3.7+

4.7.3 CbAPI 1.7.9 - Release Sept 29, 2022

Bug Fixes

- Adjust Live Response Worker creation for EDR sensors to optimize for sensor specific jobs

4.7.4 CbAPI 1.7.8 - Release Jun 17, 2022

Bug Fixes

- Vendor the *attrdict* module because of *ImportError* for Python3.10

4.7.5 CbAPI 1.7.7 - Release Jan 28, 2022

Bug Fixes

- Changed the sort order for EDR sensor searches from 'last_checkin_time' (default when none provided explicitly) to 'hostname' to make the sort stable as sensors checkin during paging

4.7.6 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

4.7.7 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.

4.7.8 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)

4.7.9 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

4.7.10 CbAPI 1.7.2 - Released July 22, 2020

Updates

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

4.7.11 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

4.7.12 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

4.7.13 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](#).

4.7.14 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

4.7.15 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

4.7.16 CbAPI 1.5.6 - Released November 19, 2019

Updates

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

4.7.17 CbAPI 1.5.5 - Released November 12, 2019

Updates

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

4.7.18 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

4.7.19 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

4.7.20 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

4.7.21 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

4.7.22 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

4.7.23 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

4.7.24 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

4.7.25 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](#)

4.7.26 CbAPI 1.4.2 - Released March 27, 2019

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

- Threat Intelligence APIs

4.7.27 CbAPI 1.4.1 - Released January 10, 2019

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

4.7.28 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

4.7.29 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

4.7.30 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.

4.7.31 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.

4.7.32 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`.

4.7.33 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.

4.7.34 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.

4.7.35 CbAPI 1.3.0 - Released July 27, 2017

This release introduces the Live Response API for CB Defense. A sample `cbldr_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.

4.7.36 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

4.7.37 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
```

(continues on next page)

(continued from previous page)

```
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
```

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")
```

Initialize the base API information.

Args:

*args: TBD **kwargs: Additional arguments.

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

property 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
```

(continues on next page)

(continued from previous page)

```

>>> 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*)

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object. initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-initialized from the server.

property all_events

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

Returns

list of CbEvent objects

property all_events_segment

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

Returns

list of CbEvent objects

property binaryJoins 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
```

property childprocs

Generator that returns CbChildProcEvent objects associated with this process

property children

Generator that returns CbChildProcEvent objects associated with this process

property cmdline**Returns**

Returns the command line of the process

Return type

string

property comms_ip

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

property crossprocs

Generator that returns CbCrossProcEvent objects associated with this process

property 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.

property 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.

property 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

property 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.

property last_server_update

Returns a pretty version of when this process last updated

property last_update

Returns a pretty version of when this process last updated

property max_last_server_update

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

property max_last_update

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

property min_last_server_update

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

property min_last_update

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

property modloads

Generator that returns `:py:class:CbModLoadEvent` associated with this process

property 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.

property parent

Returns the parent Process object if one exists

property parent_md5

Workaround since `parent_md5` silently disappeared in EDR 6.x

property processblocks

Generator that returns `CbProcessBlockEvent` objects associated with this process

refresh()

Refresh the object from the Carbon Black server.

property regmods

Generator that returns `CbRegModEvent` objects associated with this process

property 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
```

property start

Returns the start time of the process

property unsigned_modloads

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

property 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]
```

(continues on next page)

(continued from previous page)

```
>>> 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

property 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*)

Initialize the BaseModel object.

Args:

`cb` (BaseAPI): Reference to the CBAPI object. `model_unique_id` (object): Unique ID for the model object. `initial_data` (dict): Initial data for the model object. `force_init` (bool): True to force the object to be re-initialized from the server.

class `FrequencyData`(*computer_count, process_count, all_process_count, module_frequency*)

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`(*result, publisher, issuer, subject, sign_time, program_name*)

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`(*file_desc, file_version, product_name, product_version, company_name, legal_copyright, original_filename*)

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`)

property banned

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

property digsig_issuer

Returns the Digital Signature Issuer

property digsig_prog_name

Returns the Digital Signature Program Name

property digsig_publisher

Returns the Digital Signature Publisher

property digsig_sign_time

Returns the Digital Signature signing time

property digsig_subject

Returns the Digital Signature subject

property endpoints

Return a list of endpoints this binary resides

property 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
```

property 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=0.001128923007450892)
```

property icon

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

property is_64bit

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

property 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.

property observed_filenames

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

property signed

Returns True if the binary is signed.

property signing_data

Returns [SigningData](#) object which contains: Digital Signature Result, Digital Signature publisher, Issuer, Subject, Signing Time, Program Name

property size

Returns the size of the Binary

property version_info

Returns a [VersionInfo](#) object containing detailed information: File Description, File Version, Product Name, Product Version, Company Name, Legal Copyright, and Original FileName

property 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.

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

class NetworkAdapter(macaddr, ipaddr)

Create new instance of NetworkAdapter(macaddr, ipaddr)

ipaddr

Alias for field number 1

macaddr

Alias for field number 0

property activity_stats

Returns a list of activity statistics from the associated EDR Sensor

property 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

property group**Getter**

Returns the sensor's group id.

Setter

Allows access to set the sensor's group id

property 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

property network_interfaces

Returns a list of networks adapters on the sensor

property os

Returns the operating system display string of the sensor

property 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'}]
```

property 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.

property 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

property 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:

property 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:

property 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)

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

property facets

Returns facets from the search associated with the watchlist query

Returns

dictionary of facets as keys

Return type

dict

property 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.

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

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")
```

Initialize the base API information.

Args:

args*: TBD *kwargs*: Additional arguments.

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

property 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*)

Initialize the MutableModel.

Args:*cb* (BaseAPI): Reference to the CBAPI object. *model_unique_id* (object): Unique ID for the model object.*initial_data* (dict): Initial data for the model object.**ResolutionApproved** = 2**ResolutionInstaller** = 4**ResolutionNotResolved** = 0**ResolutionOther** = 7**ResolutionPublisher** = 6**ResolutionRejected** = 1**ResolutionRuleChange** = 3**ResolutionUpdater** = 5**StatusClosed** = 3**StatusOpen** = 2**StatusSubmitted** = 1**property** `computer`**property** `fileCatalog`**property** `installerFileCatalog`**property** `processFileCatalog`**urlobject** = `'/api/bit9platform/v1/approvalRequest '`

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

Initialize the MutableModel.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.

initial_data (dict): Initial data for the model object.

StateApproved = 2

StateBanned = 3

StateMixed = 4

StateUnapproved = 1

property firstSeenComputer

property parent

property 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.

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

property fileInstances

property policy

resetCLIPassword()

property 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 usually 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

property 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.

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

property fileCatalog

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

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

Initialize the MutableModel.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object.

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.

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

property certificate

property computer

property fileHash

property 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.

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

property computer

property fileCatalog

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

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

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-
initialized from the server.

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

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

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-
initialized from the server.

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

property createdByUser

createdById = None

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

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

description = None

property 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)
```

Initialize the MutableModel.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object.

property 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)
```

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-
initialized from the server.

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

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

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-
initialized from the server.

```
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)
```

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-initialized from the server.

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

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

Initialize the MutableModel.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object.

```
ResultClean = 1
```

```
ResultMalicious = 3
```

```
ResultNotAvailable = 0
```

```
ResultPotentialThreat = 2
```

```
StatusAnalyzed = 3
```

```
StatusCancelled = 5
```

```
StatusError = 4
```

```
StatusProcessed = 2
```

```
StatusScheduled = 0
```

```
StatusSubmitted = 1
```

```
create_notification(**kwargs)
```

```
property file
```

```
property fileCatalog
```

```
property 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)
```

Initialize the MutableModel.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object.

```
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)
```

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object. initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-initialized from the server.

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

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

Initialize the BaseModel object.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object. initial_data (dict): Initial data for the model object. force_init (bool): True to force the object to be re-initialized from the server.

```
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
- **userId** – 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*)

Initialize the MutableModel.

Args:

cb (BaseAPI): Reference to the CBAPI object. model_unique_id (object): Unique ID for the model object.
initial_data (dict): Initial data for the model object.

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 Exceptions

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

5.3.1 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.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

C

`cbapi.protection.models`, [67](#)

A

actions (*cbapi.response.models.Feed* property), 58
 activity_stats (*cbapi.response.models.Sensor* property), 56
 adminComments (*cbapi.protection.models.User* attribute), 81
 Alert (class in *cbapi.response.models*), 59
 AlertQuery (class in *cbapi.response.models*), 50
 all_events (*cbapi.response.models.Process* property), 50
 all_events_segment (*cbapi.response.models.Process* property), 50
 allowAgentUpgrades (*cbapi.protection.models.Policy* attribute), 77
 analysisEnabled (*cbapi.protection.models.Connector* attribute), 69
 analysisName (*cbapi.protection.models.Connector* attribute), 69
 analysisResult (*cbapi.protection.models.Notification* attribute), 74
 analysisTargets (*cbapi.protection.models.Connector* attribute), 69
 and_() (*cbapi.protection.rest_api.Query* method), 66
 and_() (*cbapi.response.query.Query* method), 47
 anomaly (*cbapi.protection.models.Notification* attribute), 74
 api_json_request() (*cbapi.protection.rest_api.CbProtectionAPI* method), 64
 api_json_request() (*cbapi.response.rest_api.CbResponseAPI* method), 43
 ApiError, 83
 apiToken (*cbapi.protection.models.User* attribute), 81
 appliance (*cbapi.protection.models.Notification* attribute), 74
 ApprovalRequest (class in *cbapi.protection.models*), 67
 atEnforcementComputers (*cbapi.protection.models.Policy* attribute), 77
 automatic (*cbapi.protection.models.Policy* attribute), 77
 automatic (*cbapi.protection.models.User* attribute), 81
 automaticApprovalsOnTransition (*cbapi.protection.models.Policy* attribute),

77

automaticCount (*cbapi.protection.models.UserGroup* attribute), 83

B

backupCellPhone (*cbapi.protection.models.User* attribute), 81
 backupPager (*cbapi.protection.models.User* attribute), 81
 banned (*cbapi.response.models.Binary* property), 54
 BannedHash (class in *cbapi.response.models*), 58
 binary (*cbapi.response.models.BannedHash* property), 59
 binary (*cbapi.response.models.Process* property), 50
 Binary (class in *cbapi.response.models*), 53
 Binary.FrequencyData (class in *cbapi.response.models*), 54
 Binary.SigningData (class in *cbapi.response.models*), 54
 Binary.VersionInfo (class in *cbapi.response.models*), 54

C

canAnalyze (*cbapi.protection.models.Connector* attribute), 69
 cbapi.protection.models
 module, 67
 CbLRSessionBase (class in *cbapi.live_response_api*), 60
 CbProtectionAPI (class in *cbapi.protection.rest_api*), 63
 CbResponseAPI (class in *cbapi.response.rest_api*), 43
 cellPhone (*cbapi.protection.models.User* attribute), 81
 certificate (*cbapi.protection.models.FileCatalog* property), 70
 Certificate (class in *cbapi.protection.models*), 67
 childprocs (*cbapi.response.models.Process* property), 50
 children (*cbapi.response.models.Process* property), 50
 clVersion (*cbapi.protection.models.FileRule* attribute), 72
 clVersion (*cbapi.protection.models.TrustedUser* attribute), 80

`clVersionMax` (*cbapi.protection.models.Policy* attribute), 77

`cmdline` (*cbapi.response.models.Process* property), 50

`comments` (*cbapi.protection.models.User* attribute), 81

`comms_ip` (*cbapi.response.models.Process* property), 50

`computer` (*cbapi.protection.models.ApprovalRequest* property), 67

`computer` (*cbapi.protection.models.FileCatalog* property), 70

`computer` (*cbapi.protection.models.FileInstance* property), 70

`Computer` (class in *cbapi.protection.models*), 68

`connectedComputers` (*cbapi.protection.models.Policy* attribute), 77

`Connector` (class in *cbapi.protection.models*), 68

`connectorId` (*cbapi.protection.models.Notification* attribute), 75

`connectorVersion` (*cbapi.protection.models.Connector* attribute), 69

`create()` (*cbapi.protection.rest_api.CbProtectionAPI* method), 64

`create()` (*cbapi.response.rest_api.CbResponseAPI* method), 44

`create_directory()` (*cbapi.live_response_api.CbLRSessionBase* method), 61

`create_new_partition()`
(*cbapi.response.rest_api.CbResponseAPI* method), 44

`create_notification()`
(*cbapi.protection.models.PendingAnalysis* method), 76

`create_process()` (*cbapi.live_response_api.CbLRSessionBase* method), 63

`create_registry_key()`
(*cbapi.live_response_api.CbLRSessionBase* method), 61

`createdBy` (*cbapi.protection.models.FileRule* attribute), 72

`createdBy` (*cbapi.protection.models.TrustedUser* attribute), 80

`createdBy` (*cbapi.protection.models.UserGroup* attribute), 83

`createdByUser` (*cbapi.protection.models.FileRule* property), 72

`createdByUserId` (*cbapi.protection.models.FileRule* attribute), 72

`createdByUserId` (*cbapi.protection.models.Policy* attribute), 77

`createdByUserId` (*cbapi.protection.models.TrustedUser* attribute), 80

`createdByUserId` (*cbapi.protection.models.UserGroup* attribute), 83

`CredentialError`, 83

`crossprocs` (*cbapi.response.models.Process* property), 51

`customLogo` (*cbapi.protection.models.Policy* attribute), 77

D

`dashboard_statistics()`
(*cbapi.response.rest_api.CbResponseAPI* method), 44

`dateCreated` (*cbapi.protection.models.FileRule* attribute), 72

`dateCreated` (*cbapi.protection.models.Policy* attribute), 78

`dateCreated` (*cbapi.protection.models.TrustedUser* attribute), 80

`dateCreated` (*cbapi.protection.models.UserGroup* attribute), 83

`dateModified` (*cbapi.protection.models.FileRule* attribute), 72

`dateModified` (*cbapi.protection.models.Policy* attribute), 78

`dateModified` (*cbapi.protection.models.TrustedUser* attribute), 80

`dateModified` (*cbapi.protection.models.UserGroup* attribute), 83

`delete_file()` (*cbapi.live_response_api.CbLRSessionBase* method), 60

`delete_object()` (*cbapi.protection.rest_api.CbProtectionAPI* method), 64

`delete_object()` (*cbapi.response.rest_api.CbResponseAPI* method), 44

`delete_registry_key()`
(*cbapi.live_response_api.CbLRSessionBase* method), 62

`delete_registry_value()`
(*cbapi.live_response_api.CbLRSessionBase* method), 61

`department` (*cbapi.protection.models.User* attribute), 81

`depth` (*cbapi.response.models.Process* property), 51

`description` (*cbapi.protection.models.FileRule* attribute), 72

`description` (*cbapi.protection.models.Policy* attribute), 78

`description` (*cbapi.protection.models.TrustedUser* attribute), 80

`description` (*cbapi.protection.models.UserGroup* attribute), 83

`destIp` (*cbapi.protection.models.Notification* attribute), 75

`destUsername` (*cbapi.protection.models.Notification* attribute), 75

`digsig_issuer` (*cbapi.response.models.Binary* property), 55

`digsig_prog_name` (*cbapi.response.models.Binary* property), 55

- digsig_publisher (*cbapi.response.models.Binary property*), 55
 digsig_sign_time (*cbapi.response.models.Binary property*), 55
 digsig_subject (*cbapi.response.models.Binary property*), 55
 directories (*cbapi.protection.models.Notification attribute*), 75
 disconnectedEnforcementLevel (*cbapi.protection.models.Policy attribute*), 78
 dns_name (*cbapi.response.models.Sensor property*), 56
 DriftReport (*class in cbapi.protection.models*), 69
 DriftReportContents (*class in cbapi.protection.models*), 69
- ## E
- editable (*cbapi.protection.models.UserGroup attribute*), 83
 emailAddress (*cbapi.protection.models.User attribute*), 81
 enabled (*cbapi.protection.models.Connector attribute*), 69
 enabled (*cbapi.protection.models.User attribute*), 81
 enabled (*cbapi.protection.models.UserGroup attribute*), 83
 end (*cbapi.response.models.Process property*), 51
 endpoints (*cbapi.response.models.Binary property*), 55
 enforcementLevel (*cbapi.protection.models.Policy attribute*), 78
 EnforcementLevel (*class in cbapi.protection.models*), 69
 Event (*class in cbapi.protection.models*), 69
 external (*cbapi.protection.models.User attribute*), 81
 externalId (*cbapi.protection.models.Notification attribute*), 75
 externalUrl (*cbapi.protection.models.Notification attribute*), 75
- ## F
- facets (*cbapi.response.models.Watchlist property*), 59
 facets() (*cbapi.response.query.Query method*), 47
 Feed (*class in cbapi.response.models*), 58
 file (*cbapi.protection.models.FileUpload property*), 73
 file (*cbapi.protection.models.PendingAnalysis property*), 76
 file (*cbapi.response.models.Binary property*), 55
 FileAnalysis (*class in cbapi.protection.models*), 70
 fileAnalysisId (*cbapi.protection.models.Notification attribute*), 75
 fileCatalog (*cbapi.protection.models.ApprovalRequest property*), 67
 fileCatalog (*cbapi.protection.models.Event property*), 69
 fileCatalog (*cbapi.protection.models.FileInstance property*), 70
 fileCatalog (*cbapi.protection.models.FileRule property*), 72
 fileCatalog (*cbapi.protection.models.PendingAnalysis property*), 76
 FileCatalog (*class in cbapi.protection.models*), 70
 fileCatalogId (*cbapi.protection.models.FileRule attribute*), 72
 fileHash (*cbapi.protection.models.FileCatalog property*), 70
 fileHash (*cbapi.protection.models.PendingAnalysis property*), 76
 FileInstance (*class in cbapi.protection.models*), 70
 FileInstanceDeleted (*class in cbapi.protection.models*), 70
 FileInstanceGroup (*class in cbapi.protection.models*), 70
 fileInstances (*cbapi.protection.models.Computer property*), 68
 filemods (*cbapi.response.models.Process property*), 51
 fileName (*cbapi.protection.models.FileRule attribute*), 73
 fileName (*cbapi.protection.models.Notification attribute*), 75
 FileRule (*class in cbapi.protection.models*), 70
 fileRuleType (*cbapi.protection.models.FileRule attribute*), 73
 files (*cbapi.protection.models.Notification attribute*), 75
 fileState (*cbapi.protection.models.FileRule attribute*), 73
 fileTrackingEnabled (*cbapi.protection.models.Policy attribute*), 78
 FileUpload (*class in cbapi.protection.models*), 73
 find_file_writes() (*cbapi.response.models.Process method*), 51
 firstName (*cbapi.protection.models.User attribute*), 81
 firstSeenComputer (*cbapi.protection.models.Certificate property*), 68
 flags (*cbapi.protection.models.Notification attribute*), 75
 flush_events() (*cbapi.response.models.Sensor method*), 56
 forceInstaller (*cbapi.protection.models.FileRule attribute*), 73
 forceNotInstaller (*cbapi.protection.models.FileRule attribute*), 73
 frequency (*cbapi.response.models.Binary property*), 55
 from_ui() (*cbapi.response.rest_api.CbResponseAPI method*), 44
- ## G
- get_file() (*cbapi.live_response_api.CbLRSessionBase*

method), 60
get_object() (cbapi.protection.rest_api.CbProtectionAPI method), 64
get_object() (cbapi.response.rest_api.CbResponseAPI method), 45
get_raw_data() (cbapi.protection.rest_api.CbProtectionAPI method), 64
get_raw_data() (cbapi.response.rest_api.CbResponseAPI method), 45
get_registry_value()
(cbapi.live_response_api.CbLRSessionBase method), 61
GrantedUserPolicyPermission (class in cbapi.protection.models), 73
group (cbapi.response.models.Sensor property), 56
group_by() (cbapi.response.models.ProcessQuery method), 48

H

hash (cbapi.protection.models.FileRule attribute), 73
helpDeskUrl (cbapi.protection.models.Policy attribute), 78
hidden (cbapi.protection.models.Policy attribute), 78
homePhone (cbapi.protection.models.User attribute), 81
hostname (cbapi.response.models.Sensor property), 56
httpHeader (cbapi.protection.models.Notification attribute), 75

I

icon (cbapi.response.models.Binary property), 55
id (cbapi.protection.models.Connector attribute), 69
id (cbapi.protection.models.FileRule attribute), 73
id (cbapi.protection.models.Policy attribute), 78
id (cbapi.protection.models.TrustedUser attribute), 80
id (cbapi.protection.models.User attribute), 81
id (cbapi.protection.models.UserGroup attribute), 83
idUnique (cbapi.protection.models.FileRule attribute), 73
imageUrl (cbapi.protection.models.Policy attribute), 78
info() (cbapi.response.rest_api.CbResponseAPI method), 45
installerFileCatalog
(cbapi.protection.models.ApprovalRequest property), 67
interface_ip (cbapi.response.models.Process property), 51
InternalEvent (class in cbapi.protection.models), 74
InvalidObjectError, 84
ipaddr (cbapi.response.models.Sensor.NetworkAdapter attribute), 56
is_64bit (cbapi.response.models.Binary property), 55
is_executable_image (cbapi.response.models.Binary property), 55
isInternal (cbapi.protection.models.Connector attribute), 69
isolate() (cbapi.response.models.Sensor method), 56

K

kill_process() (cbapi.live_response_api.CbLRSessionBase method), 63

L

last_server_update (cbapi.response.models.Process property), 51
last_update (cbapi.response.models.Process property), 51
lastName (cbapi.protection.models.User attribute), 81
lazyApproval (cbapi.protection.models.FileRule attribute), 73
LevelHigh (cbapi.protection.models.EnforcementLevel attribute), 69
LevelLow (cbapi.protection.models.EnforcementLevel attribute), 69
LevelMedium (cbapi.protection.models.EnforcementLevel attribute), 69
LevelNone (cbapi.protection.models.EnforcementLevel attribute), 69
license_request() (cbapi.response.rest_api.CbResponseAPI method), 45
list_directory() (cbapi.live_response_api.CbLRSessionBase method), 60
list_processes() (cbapi.live_response_api.CbLRSessionBase method), 63
list_registry_keys()
(cbapi.live_response_api.CbLRSessionBase method), 62
list_registry_keys_and_values()
(cbapi.live_response_api.CbLRSessionBase method), 62
loadAgentInSafeMode (cbapi.protection.models.Policy attribute), 78
lr_session() (cbapi.response.models.Sensor method), 57

M

macaddr (cbapi.response.models.Sensor.NetworkAdapter attribute), 56
malwareName (cbapi.protection.models.Notification attribute), 75
malwareType (cbapi.protection.models.Notification attribute), 75
manualCount (cbapi.protection.models.UserGroup attribute), 83
max_children() (cbapi.response.models.ProcessQuery method), 48

max_last_server_update
 (*cbapi.response.models.Process* property),
 51
 max_last_server_update()
 (*cbapi.response.models.ProcessQuery*
 method), 48
 max_last_update (*cbapi.response.models.Process*
 property), 51
 max_last_update() (*cbapi.response.models.ProcessQuery*
 method), 48
 md5 (*cbapi.protection.models.Notification* attribute), 75
 MeteredExecution (class in *cbapi.protection.models*),
 74
 min_last_server_update
 (*cbapi.response.models.Process* property),
 51
 min_last_server_update()
 (*cbapi.response.models.ProcessQuery*
 method), 49
 min_last_update (*cbapi.response.models.Process*
 property), 51
 min_last_update() (*cbapi.response.models.ProcessQuery*
 method), 49
 modifiedBy (*cbapi.protection.models.FileRule* at-
 tribute), 73
 modifiedBy (*cbapi.protection.models.TrustedUser* at-
 tribute), 80
 modifiedBy (*cbapi.protection.models.UserGroup*
 attribute), 83
 modifiedByUserId (*cbapi.protection.models.FileRule*
 attribute), 73
 modifiedByUserId (*cbapi.protection.models.Policy* at-
 tribute), 78
 modifiedByUserId (*cbapi.protection.models.TrustedUser*
 attribute), 80
 modifiedByUserId (*cbapi.protection.models.UserGroup*
 attribute), 83
 modloads (*cbapi.response.models.Process* property), 51
 module
 cbapi.protection.models, 67
 MoreThanOneResultError, 84
 msgFormat (*cbapi.protection.models.Notification* at-
 tribute), 75

N

name (*cbapi.protection.models.Connector* attribute), 69
 name (*cbapi.protection.models.FileRule* attribute), 73
 name (*cbapi.protection.models.Policy* attribute), 78
 name (*cbapi.protection.models.TrustedUser* attribute), 80
 name (*cbapi.protection.models.User* attribute), 81
 name (*cbapi.protection.models.UserGroup* attribute), 83
 netconns (*cbapi.response.models.Process* property), 52
 network_interfaces (*cbapi.response.models.Sensor*
 property), 57

new_object() (*cbapi.response.models.Binary* class
 method), 55
 new_object() (*cbapi.response.models.Process* class
 method), 52
 Notification (class in *cbapi.protection.models*), 74
 Notifier (class in *cbapi.protection.models*), 75

O

ObjectNotFoundError, 84
 observed_filenames (*cbapi.response.models.Binary*
 property), 56
 origIdUnique (*cbapi.protection.models.FileRule*
 attribute), 73
 os (*cbapi.response.models.Sensor* property), 57

P

packageName (*cbapi.protection.models.Policy* attribute),
 78
 pager (*cbapi.protection.models.User* attribute), 82
 parent (*cbapi.protection.models.Certificate* property),
 68
 parent (*cbapi.response.models.Process* property), 52
 parent_md5 (*cbapi.response.models.Process* property),
 52
 passwordHash (*cbapi.protection.models.User* attribute),
 82
 passwordSalt (*cbapi.protection.models.User* attribute),
 82
 pendingAnalyses (*cbapi.protection.models.Connector*
 property), 69
 PendingAnalysis (class in *cbapi.protection.models*), 76
 permissions (*cbapi.protection.models.UserGroup* at-
 tribute), 83
 platformFlags (*cbapi.protection.models.FileRule* at-
 tribute), 73
 platformId (*cbapi.protection.models.TrustedUser* at-
 tribute), 80
 PlatformLinux (*cbapi.protection.models.FileRule* at-
 tribute), 72
 PlatformMac (*cbapi.protection.models.FileRule* at-
 tribute), 72
 PlatformWindows (*cbapi.protection.models.FileRule* at-
 tribute), 72
 policy (*cbapi.protection.models.Computer* property), 68
 Policy (class in *cbapi.protection.models*), 76
 policyIds (*cbapi.protection.models.FileRule* attribute),
 73
 policyIds (*cbapi.protection.models.UserGroup* at-
 tribute), 83
 post_object() (*cbapi.protection.rest_api.CbProtectionAPI*
 method), 65
 post_object() (*cbapi.response.rest_api.CbResponseAPI*
 method), 45
 Process (class in *cbapi.response.models*), 50

processblocks (*cbapi.response.models.Process* property), 52

processFileCatalog (*cbapi.protection.models.ApprovalRequest* property), 67

ProcessQuery (class in *cbapi.response.models*), 48

product (*cbapi.protection.models.Notification* attribute), 75

publisher (*cbapi.protection.models.Certificate* property), 68

publisher (*cbapi.protection.models.FileCatalog* property), 70

Publisher (class in *cbapi.protection.models*), 78

PublisherCertificate (class in *cbapi.protection.models*), 78

put_file() (*cbapi.live_response_api.CbLRSessionBase* method), 60

put_object() (*cbapi.protection.rest_api.CbProtectionAPI* method), 65

put_object() (*cbapi.response.rest_api.CbResponseAPI* method), 45

Q

query (*cbapi.response.models.Watchlist* property), 59

Query (class in *cbapi.protection.rest_api*), 66

Query (class in *cbapi.response.query*), 46

queued_stats (*cbapi.response.models.Sensor* property), 57

R

raise_unless_json() (*cbapi.protection.rest_api.CbProtectionAPI* method), 65

raise_unless_json() (*cbapi.response.rest_api.CbResponseAPI* method), 46

readOnly (*cbapi.protection.models.Policy* attribute), 78

readOnly (*cbapi.protection.models.User* attribute), 82

refresh() (*cbapi.response.models.Process* method), 52

registrationDate (*cbapi.protection.models.User* attribute), 82

regKeys (*cbapi.protection.models.Notification* attribute), 75

regmods (*cbapi.response.models.Process* property), 52

reportOnly (*cbapi.protection.models.FileRule* attribute), 73

reputationApprovalsEnabled (*cbapi.protection.models.FileRule* attribute), 73

reputationEnabled (*cbapi.protection.models.Policy* attribute), 78

resetCLIPassword() (*cbapi.protection.models.Computer* method), 68

ResolutionApproved (*cbapi.protection.models.ApprovalRequest* attribute), 67

ResolutionInstaller (*cbapi.protection.models.ApprovalRequest* attribute), 67

ResolutionNotResolved (*cbapi.protection.models.ApprovalRequest* attribute), 67

ResolutionOther (*cbapi.protection.models.ApprovalRequest* attribute), 67

ResolutionPublisher (*cbapi.protection.models.ApprovalRequest* attribute), 67

ResolutionRejected (*cbapi.protection.models.ApprovalRequest* attribute), 67

ResolutionRuleChange (*cbapi.protection.models.ApprovalRequest* attribute), 67

ResolutionUpdater (*cbapi.protection.models.ApprovalRequest* attribute), 67

resource_status (*cbapi.response.models.Sensor* property), 57

restart_sensor() (*cbapi.response.models.Sensor* method), 57

ResultClean (*cbapi.protection.models.Notification* attribute), 74

ResultClean (*cbapi.protection.models.PendingAnalysis* attribute), 76

ResultMalicious (*cbapi.protection.models.Notification* attribute), 74

ResultMalicious (*cbapi.protection.models.PendingAnalysis* attribute), 76

ResultNotAvailable (*cbapi.protection.models.Notification* attribute), 74

ResultNotAvailable (*cbapi.protection.models.PendingAnalysis* attribute), 76

ResultPotentialThreat (*cbapi.protection.models.Notification* attribute), 74

ResultPotentialThreat (*cbapi.protection.models.PendingAnalysis* attribute), 76

S

salutation (*cbapi.protection.models.User* attribute), 82

ScriptRule (class in *cbapi.protection.models*), 78

search() (*cbapi.response.models.Watchlist* method), 59

search_binaries() (*cbapi.response.models.Feed* method), 58

search_processes() (*cbapi.response.models.Feed* method), 58

select() (*cbapi.protection.rest_api.CbProtectionAPI* method), 65

select() (*cbapi.response.rest_api.CbResponseAPI* method), 46

sensor (*cbapi.response.models.Process* property), 52

Sensor (class in *cbapi.response.models*), 56
 Sensor.NetworkAdapter (class in *cbapi.response.models*), 56
 ServerConfig (class in *cbapi.protection.models*), 79
 ServerError, 83
 ServerPerformance (class in *cbapi.protection.models*), 79
 set_ignored() (*cbapi.response.models.Alert* method), 59
 set_ignored() (*cbapi.response.models.AlertQuery* method), 50
 set_registry_value() (*cbapi.live_response_api.CbLRSessionBase* method), 61
 severity (*cbapi.protection.models.Notification* attribute), 75
 sha1 (*cbapi.protection.models.Notification* attribute), 75
 sha256 (*cbapi.protection.models.Notification* attribute), 75
 sid (*cbapi.response.models.Sensor* property), 57
 signed (*cbapi.response.models.Binary* property), 56
 signing_data (*cbapi.response.models.Binary* property), 56
 size (*cbapi.response.models.Binary* property), 56
 sort() (*cbapi.protection.rest_api.Query* method), 66
 sort() (*cbapi.response.query.Query* method), 47
 sourceId (*cbapi.protection.models.FileRule* attribute), 73
 sourceType (*cbapi.protection.models.FileRule* attribute), 73
 SourceTypeApplicationTemplate (*cbapi.protection.models.FileRule* attribute), 72
 SourceTypeEventRule (*cbapi.protection.models.FileRule* attribute), 72
 SourceTypeExternal (*cbapi.protection.models.FileRule* attribute), 72
 SourceTypeImported (*cbapi.protection.models.FileRule* attribute), 72
 SourceTypeManual (*cbapi.protection.models.FileRule* attribute), 72
 SourceTypeReputation (*cbapi.protection.models.FileRule* attribute), 72
 SourceTypeTrustedDirectory (*cbapi.protection.models.FileRule* attribute), 72
 SourceTypeUnifiedManagement (*cbapi.protection.models.FileRule* attribute), 72
 srcHost (*cbapi.protection.models.Notification* attribute), 75
 srcIp (*cbapi.protection.models.Notification* attribute), 75
 srcUsername (*cbapi.protection.models.Notification* attribute), 75
 start (*cbapi.response.models.Process* property), 52
 StateApproved (*cbapi.protection.models.Certificate* attribute), 68
 StateApproved (*cbapi.protection.models.FileRule* attribute), 72
 StateBanned (*cbapi.protection.models.Certificate* attribute), 68
 StateBanned (*cbapi.protection.models.FileRule* attribute), 72
 StateMixed (*cbapi.protection.models.Certificate* attribute), 68
 StateUnapproved (*cbapi.protection.models.Certificate* attribute), 68
 StateUnapproved (*cbapi.protection.models.FileRule* attribute), 72
 status (*cbapi.protection.models.Notification* attribute), 75
 StatusAnalyzed (*cbapi.protection.models.PendingAnalysis* attribute), 76
 StatusCancelled (*cbapi.protection.models.PendingAnalysis* attribute), 76
 StatusClosed (*cbapi.protection.models.ApprovalRequest* attribute), 67
 StatusError (*cbapi.protection.models.PendingAnalysis* attribute), 76
 StatusOpen (*cbapi.protection.models.ApprovalRequest* attribute), 67
 StatusProcessed (*cbapi.protection.models.PendingAnalysis* attribute), 76
 StatusScheduled (*cbapi.protection.models.PendingAnalysis* attribute), 76
 StatusSubmitted (*cbapi.protection.models.ApprovalRequest* attribute), 67
 StatusSubmitted (*cbapi.protection.models.PendingAnalysis* attribute), 76

T

targetApp (*cbapi.protection.models.Notification* attribute), 75
 targetOS (*cbapi.protection.models.Notification* attribute), 75
 templateComputer (*cbapi.protection.models.Computer* property), 68
 ThreatReportQuery (class in *cbapi.response.models*), 50
 time (*cbapi.protection.models.Notification* attribute), 75
 TimeoutError, 84
 title (*cbapi.protection.models.User* attribute), 82
 totalComputers (*cbapi.protection.models.Policy* attribute), 78

TrustedDirectory (class in *cbapi.protection.models*), 79

TrustedUser (class in *cbapi.protection.models*), 79
type (*cbapi.protection.models.Notification* attribute), 75

U

unified (*cbapi.protection.models.User* attribute), 82

unifiedFlag (*cbapi.protection.models.FileRule* attribute), 73

unifiedSource (*cbapi.protection.models.FileRule* attribute), 73

unisolate() (*cbapi.response.models.Sensor* method), 57

unsigned_modloads (*cbapi.response.models.Process* property), 52

update_license() (*cbapi.response.rest_api.CbResponseAPI* method), 46

Updater (class in *cbapi.protection.models*), 80

url (*cbapi.protection.rest_api.CbProtectionAPI* property), 65

url (*cbapi.response.rest_api.CbResponseAPI* property), 46

urlobject (*cbapi.protection.models.ApprovalRequest* attribute), 67

urlobject (*cbapi.protection.models.Certificate* attribute), 68

urlobject (*cbapi.protection.models.Computer* attribute), 68

urlobject (*cbapi.protection.models.Connector* attribute), 69

urlobject (*cbapi.protection.models.DriftReport* attribute), 69

urlobject (*cbapi.protection.models.DriftReportContents* attribute), 69

urlobject (*cbapi.protection.models.Event* attribute), 70

urlobject (*cbapi.protection.models.FileAnalysis* attribute), 70

urlobject (*cbapi.protection.models.FileCatalog* attribute), 70

urlobject (*cbapi.protection.models.FileInstance* attribute), 70

urlobject (*cbapi.protection.models.FileInstanceDeleted* attribute), 70

urlobject (*cbapi.protection.models.FileInstanceGroup* attribute), 70

urlobject (*cbapi.protection.models.FileRule* attribute), 73

urlobject (*cbapi.protection.models.FileUpload* attribute), 73

urlobject (*cbapi.protection.models.GrantedUserPolicyPermission* attribute), 74

urlobject (*cbapi.protection.models.InternalEvent* attribute), 74

urlobject (*cbapi.protection.models.MeteredExecution* attribute), 74

urlobject (*cbapi.protection.models.Notification* attribute), 75

urlobject (*cbapi.protection.models.Notifier* attribute), 76

urlobject (*cbapi.protection.models.PendingAnalysis* attribute), 76

urlobject (*cbapi.protection.models.Policy* attribute), 78

urlobject (*cbapi.protection.models.Publisher* attribute), 78

urlobject (*cbapi.protection.models.PublisherCertificate* attribute), 78

urlobject (*cbapi.protection.models.ScriptRule* attribute), 78

urlobject (*cbapi.protection.models.ServerConfig* attribute), 79

urlobject (*cbapi.protection.models.ServerPerformance* attribute), 79

urlobject (*cbapi.protection.models.TrustedDirectory* attribute), 79

urlobject (*cbapi.protection.models.TrustedUser* attribute), 80

urlobject (*cbapi.protection.models.Updater* attribute), 80

urlobject (*cbapi.protection.models.User* attribute), 82

urlobject (*cbapi.protection.models.UserGroup* attribute), 83

use_comprehensive_search()
(*cbapi.response.models.ProcessQuery* method), 49

User (class in *cbapi.protection.models*), 80

UserGroup (class in *cbapi.protection.models*), 82

userGroupIds (*cbapi.protection.models.User* attribute), 82

username (*cbapi.response.models.Process* property), 52

userSid (*cbapi.protection.models.TrustedUser* attribute), 80

V

version (*cbapi.protection.models.FileRule* attribute), 73

version (*cbapi.protection.models.Notification* attribute), 75

version_info (*cbapi.response.models.Binary* property), 56

visible (*cbapi.protection.models.FileRule* attribute), 73

W

walk() (*cbapi.live_response_api.CbLRSessionBase* method), 61

walk_children() (*cbapi.response.models.Process* method), 52

walk_parents() (*cbapi.response.models.Process* method), 53

`Watchlist` (*class in `cbapi.response.models`*), [59](#)
`webui_link` (*`cbapi.response.models.Binary` property*),
[56](#)
`webui_link` (*`cbapi.response.models.Process` property*),
[53](#)
`webui_link` (*`cbapi.response.models.Sensor` property*),
[58](#)
`where()` (*`cbapi.protection.rest_api.Query` method*), [67](#)
`where()` (*`cbapi.response.query.Query` method*), [47](#)