cloud security alliance℠

The CTP prototype back office API, rev. 0.2

October 2015

Contents

Contact Author: Dr. Alain Pannetrat <apannetrat@cloudsecurityalliance.org>

Chapter 1

Introduction

This document describes the *CTP prototype back office API* used in the ctpd prototype, hereafter simply referred as the *back office API*. As such it complements the *CTP API data model and API* [client_API], hereafter simply referred as the *client API*. Most concepts and reference material defined in the *client API* will not be repeated here.

Warning
The *CTP back office API* is not part of the official CTP API specifications published by Cloud Security Alliance (CSA). It is only intended to be used for the prototype ctpd implementation available at https://github.com/cloudsecurityalliance/-ctpd

There are two key differences between the *back office API* and the *client API* that are worth noting. First — with few exceptions — the *client API* mainly describes read-only API calls with the `GET` method, while the *back office API* contains many write calls with the `POST` or `PUT` methods, mirroring the read-only methods in the *client API*. With an understanding of the *client API* it should be straightforward to understand related calls in the *back office API*. Second, the *back office API* contains specific administrative API methods, for the purpose of configuring access control, through the creation of credentials (i.e. *access tokens*) and the definition of access rights (through the use of *tags*).

1.1 Notations

1.1.1 Identifiers

As described in the *client API*, CTP API method URLs are composed of two parts:

1. A CTP entry point denoted `{ctpBase}`.
2. A relative path to the entry point that represents a CTP API resource, accessible through specific HTTP methods.

The relative URL is itself further broken down into resource types (e.g. `assets` in `/assets/a1b2c3`) and resource identifiers (e.g. `a1b2c3` in `/assets/a1b2c3`).

Example 1.1 Breaking down a CTP URL

The URL `https://ctp.ikialab.net:4443/ctp/assets/a1b2c3` is broken down into:

- `{ctpBase}=https://ctp.ikialab.net:4443/ctp/`, and
- the relative URL path `/assets/a1b2c3`.

The relative URL path is further broken down into a resource type "`asset`" and a resource identifier "`a1b2c3`".

See the *client API* for details.

1.1.2 API query signatures

In this document, we will often describe API calls through their *signature*.

Given an HTTP query to a fully qualified URL, a CTP API query signature is created as the direct concatenation of the following elements:

- The HTTP method in uppercase (i.e. `GET`, `POST`, `PUT` or `DELETE`).
- A space (U+0020).
- The relative URL path to `{ctpBase}`, not including the query string, where all resource identifiers are replaced with the symbol `{id}`.
- If and only if the query string contains a variable named `x`:
  - The character `?` (U+003F).
  - The value of the query string variable `x`.

Example 1.2 A signature of a URL

Assuming `{ctpBase}` = \http://example.com/ctp/, a GET query to the URL *http//example.com/ctp/assets/ab01?x=foo&y=bar* will have the following signature:

```
GET /assets/{id}?foo
```

Explanation:

- The HTTP method is `GET`
- The relative URL path is `/assets/ab01` where the resource identifier `ab01` is replaced by `{id}`.
- The query string does contain a variable `x` set to `foo` (but the variable `y` is ignored).

1.2 High-level view of the back office API

A typical use of the *back office API* can be devided into three types of activities:

1. Creating an account for a user and populating the data accessible to that user through ctp.
2. Updating the data available to the user, in particular *measurement results*.
3. Removing a user and related data.

Creating an account for a user and populating the relevant data is done as follows:

Step 1
Create an account, with the `POST /accounts` API call.

Step 2
Create a *service view* and “tag” it with the identifier of the account created in Step 1, using the `POST /serviceViews` API call.

Step 3
Create one or more *assets* within the *service view* created in step 2, using the `POST /serviceViews/{id}/assets` API call, where `{id}` represents the identifier of the *service view*.

Step 4
Proceed similarly to step 3, to create *attributes* within *assets*, using the `POST /assets/{id}/attributes` API call, where `{id}` represents the identifier of an *asset*.

Step 5
Proceed similarly to step 3 or 4, to create *measurements* within *attributes*, using the `POST /attributes/{id}/measurements` API call, where `{id}` represents the identifier of an *attribute*.

In addition to these steps that are specific to each new user, it will be necessary to create the *metrics* that the ctpd server implements, which are usually shared by all users, using the `POST /metrics` API call.

In the ctpd server, access control is managed by tags. Each resource described in the *client API* has an extra hidden field called `accesTags` that is only available to administrators using the *back office API*. Creating a “tag” in step 2 is necessary as part of the access control system used in ctp to make sure that a user can only access its own resources, as detailed in Chapter 2. Tags are also created in the following steps of the process, but this is done implicitly: by default the tags of an *asset* are inherited from the *service view*, the tags of an *attribute* are inherited from the *asset*, and finally the tags of a *measurement* are inherited from the relevant *attribute*. Of course, if you don’t like the implicit behavior you can override it by specifying explicit `accessTags` in all steps described above.

Updating the data available to the user is mainly done by updating measurement results through the `PUT /measurements/{id}?x=result` API call, where `{id}` represents the identifier of the measurement.

Finally removing data is done through relevant calls using the `DELETE` method. Note that because there is a hierarchical dependency between many resources a delete call may affect more than one resource. For example, deleting a *service view* will result in the deletion of all *assets*, *attributes*, *measurements*, *triggers* and *log entries* within the scope of the *service view*.

1.3 Structure of this document

The remaining of this specification is organized as follows:

- Section 2 describes the principles of access control that apply to the *back office API*, including:
  - Token-based authentication,
  - Tag-based authorization.
- Section 3 describes the JSON resources that are used in the *back office API* and that are not described already in the *client API*.
- Section 4 details all the administrator API calls that compose the *back office API*.

Chapter 2

Access control

In this document we use the word *principal* to refer to any entity that can be authenticated by a computer system, and more specifically here by the ctpd server. Each principal is associated to an account in the ctpd server.

Access control in the *back office API* is controlled by two elements:

- **Tokens**: unpredictable random strings that serves to identify and authenticate principals. These are "Bearer" OAuth2 tokens (see [RFC_6750]).
- **Tags**: textual strings that can be associated to tokens, API query signatures or API resources, which represent access rights.

Authentication is controlled by the verification, creation and deletion of tokens.

Authorisation is controlled by tags. Tags are associated with user accounts, API resources and API query signatures. In order to determine if a principal is authorized to access an API resource, the server will perform tag matching verifications between the tags associated with the principal's account and the tag of the API resource as well as the tag of the API query signature as detailed hereafter.

2.1 Authentication

As described in the *client API* specification, making a CTP API call requires the principal to specify an OAuth2 authentication token, as shown in the example below using curl.

```
curl -H "Authtorization: Bearer qSWzVMzJ4Ee2nAKLWL5EWbnt" -X GET https://ctp.example.com/ ↩
    asset/01
```

With the *back office API*, administrators can create access tokens and associate tags to them, through the creation of an *account*.

This is done by issuing a `POST` request:

```
POST /accounts
```

This request must have a JSON body describing the tags associated with the newly created account, along with an optional name and annotation:

```
{
    "name": "jsmith@example.com",
    "annotation": "this token is created for John Smith",
    "accountTags": [
        "access:user",
              "access:anybody",
    ],
}
```

The server will acknowledge the creation of the new account with a 201 status code followed by a copy of the newly created resource, which will include a `self` property pointing to the account's newly created URL along with a randomly generated token:

```
{
    "self": "https://ctp.example.com/accounts/2931/",
    "name": "jsmith@example.com",
    "annotation": "this token is created for John Smith",
    "accountTags": [
        "access:user",
              "access:anybody",
        "id:2931"
    ],
    "token": "qSWzVMzJ4Ee2nAKLWL5EWbnt"
}
```

Tip
Alternatively it's possible to specify the value of "token" in the body of the POST request instead of letting the ctpd server choose it for you.

The URL returned in the `self` property of the response to the `POST` request can be used to later read the description of the account, e.g.:

```
GET /accounts/2931
```

The same URL can also be used to delete the account, e.g.:

```
DELETE /accounts/2931
```

Note
There is currently no way to modify an account with a PUT request. Modifications are done by deleting the account and reposting a new version of it.

Finally, the list of all existing accounts is accessible to administrators as a collection by issuing a `GET` request, e.g.:

```
GET /accounts
```

2.2 Authorisation

2.2.1 Principles

Authorisation is based on tag matching.

Tags can be any UTF-8 string. By convention they are typically composed of two parts separated by a column (e.g. `id:1234`).

Tags are associated with 3 kinds of objects:

1. **accounts** (i.e. principals) as described in Section 2.1.
2. **API query signatures**, as described in Section 1.1.2.
3. **API resources**, such as an "asset", a "service view", an "attribute", etc.

In CTP there are two types of resources accessible by the API: individual resources and collections. For individual resources, access is granted by matching tags from *accounts* with tags from both *API query signatures* and *API resources*. For collections, access is granted granted by matching tags from *accounts* with tags from both *API query signatures* and the tags of the *API resource* that represents the scope of the collection.

More precisely:

- Once authenticated, a **principal** will be authorized to perform a CTP API query to an *individual resource* if:
  - at least one account tag associated with the principal's *account* matches an access tag associated with the *API query signature*.
  - at least one account tag associated with the principal's *account* matches an access tag associated with the *API resource*.
- Once authenticated a **principal** will be authorized to perform a CTP API query to a *collection*:
  - at least one account tag associated with the prinicpal's *account* matches an access tag associated with the *API query signature*.
  - at least one account tag associated with the principal's *account* matches an access tag associated with the *API resource* representing the scope of the resource, following the CTP resource hierarchy defined in *client API*:
    - Accessing a collection of *assets* will require validation against the scoping *service view*.
    - Accessing a collection of *attributes* will require validation against the scoping *asset*.
    - Accessing a collection of *measurements* will require validation against the scoping *attribute*.
    - Accessing a collection of *triggers* or *logs entries* will require validation against the scoping *service view*.
    - This restriction does not apply to collection of *service views*, *metrics* or *accounts*, which have a global scope.

Note
To address a potential point of confusion, it is worth noting that accounts have both "access tags" and "account tags". The "account tags" are used to define what the principal can access following the rules defined above. The "access tags" defines who can access the *account* resource itself (e.g. usually the administrator only).

Two tags are considered as matching if either:

- they are strictly byte-to-byte equal, OR
- one of the tags is the wildcard value `*` (U+0042).

Warning
The wildcard value should be used with care, associating it only to tokens destined to administrators. The wildcard allows in effect to override all access control restrictions.

Example 2.1 Matching tags

Consider an authenticated principal, who has an *account* with the following account tags:

- `access:user`
- `id:1234`

This principal attempts to access the following CTP URL with a GET request: `https://ctp.example.com/assets/a1b2c3/`. Such a request has the following API signature: `GET /assets/{id}` with `{ctpBase}=https://ctp.example.com` here.
Further assume that:

- The CTP server associates the tag `access:user` to the API signature `GET /assets/{id}`,
- The CTP server associates the tag `id:1234` with the asset identified as `a1b2c3`.

The client will be allowed to access the resource because one tag of the token matches a tag associated to the API signature and another matches the resource itself. If the tag associated with the asset numbered `a1b2c3` was `id:6789` instead of `id:1234` for example, access would be refused.

2.2.2 Tags of API query signatures

The following table describes the tag that is associated with each API query signature, as currently hard-coded in the CTP server.

API query signature	**Tag**
`GET /`	`access:user`
`GET /serviceViews`	`access:user`
`GET /serviceViews/{id}`	`access:user`
`GET /serviceViews/{id}/dependencies`	`access:user`
`GET /serviceViews/{id}/assets`	`access:user`
`GET /assets/{id}`	`access:user`
`GET /assets/{id}/attributes`	`access:user`
`GET /attributes/{id}`	`access:user`
`GET /attributes/{id}/measurements`	`access:user`
`GET /measurements/{id}`	`access:user`
`GET /metrics`	`access:anybody`
`GET /metrics/{id}`	`access:anybody`
`GET /triggers/{id}`	`access:user`
`GET /dependencies/{id}`	*unimplemented API call*
`GET /logs/{id}`	`access:user`
`PUT /measurements/{id}?initiate`	`access:user`
`POST /ServiceViews/{id}/triggers`	`access:user`
`DELETE /triggers/{id}`	`access:admin`
`GET /views/{id}?tags`	`access:admin`
`PUT /views/{id}?tags`	`access:admin`
`GET /serviceViews/{id}?tags`	`access:admin`
`PUT /serviceViews/{id}?tags`	`access:admin`
`GET /assets/{id}?tags`	`access:admin`
`PUT /assets/{id}?tags`	`access:admin`
`GET /attributes/{id}?tags`	`access:admin`
`PUT /attributes/{id}?tags`	`access:admin`
`GET /measurements/{id}?tags`	`access:admin`
`PUT /measurements/{id}?tags`	`access:admin`
`GET /metrics/{id}?tags`	`access:admin`
`PUT /metrics/{id}?tags`	`access:admin`
`GET /triggers/{id}?tags`	`access:admin`
`PUT /triggers/{id}?tags`	`access:admin`
`GET /dependencies/{id}?tags`	*unimplemented API call*
`PUT /dependencies/{id}?tags`	*unimplemented API call*
`GET /logs/{id}?tags`	`access:admin`
`PUT /logs/{id}?tags`	`access:admin`
`GET /accounts/{id}?tags`	`access:admin`
`PUT /accounts/{id}?tags`	`access:admin`
`GET /logs`	`access:admin`
`PUT /measurements/{id}?result`	`access:agent`
`PUT /measurements/{id}?objective`	`access:admin`
`POST /serviceViews`	`access:admin`
`POST /serviceViews/{id}/assets`	`access:admin`
`POST /assets/{id}/attributes`	`access:admin`
`POST /attributes/{id}/measurements`	`access:agent`
`POST /metrics`	`access:admin`
`POST /dependencies`	*unimplemented API call*
`DELETE /serviceViews/{id}`	`access:admin`
`DELETE /assets/{id}`	`access:admin`
`DELETE /attributes/{id}`	`access:admin`
`DELETE /measurements/{id}`	`access:admin`
`DELETE /metrics/{id}`	`access:admin`

API query signature	**Tag**
`DELETE /dependencies/{id}`	*unimplemented API call*
`DELETE /logs/{id}`	*unimplemented API call*
`GET /accounts/{id}`	`access:admin`
`POST /accounts`	`access:admin`
`GET /accounts`	`access:admin`
`PUT /accounts/{id}`	*unimplemented API call*
`DELETE /accounts/{id}`	`access:admin`

2.2.3 Tags of resources

An arbitrary set of tags can be associated with a resource by using the tag related methods described in this specification in Chapter 4. These methods are easily distinguished from others because they have the query string parameter `x=tags` in their URL (i.e. their signature ends with `?tags`).

When an CTP resource is created using a `POST` request it is initiated with a the set of tags that are passed in the request body, in the `accessTags` JSON property. If the `accessTags` property is not specified, the tags are chosen implicitly according to the following rules:

- If the resource is a *service view*, it receives by default no tag.
- If the resource is a *metric*, it receives by default the predefined tag `access:anybody`.
- In all other cases, the resource will inherit all the tags of its parents:
  - An *asset* gets all of the tags of the *service-view* it is part of.
  - An *attribute* gets all of the tags of the *asset* it applies to.
  - A *measurement* gets all of the tags of the *attribute* it applies to.
  - A *trigger* gets all the tags of the *measurement* it applies to.
  - A *log entry* gets all the tags of the *trigger* it derives from.

Chapter 3

Resources

The JSON resources used in the *back office API* are the same as in the *client API* with the addition of two elements: *tags* and *accounts*. These two additional elements are detailed here.

3.1 Tag resources

Tags are extensions to the *client API* that are accessible only to administrators. All resources described in the *client API* contain an extra property `accessTags` that can be set during the creation of a resource with a `POST` request or later consulted and modified with specific tag-related API calls.

If `/foo/bar` is a link to a CTP resource, then the corresponding tags can be accessed simply by appending the query string `x=tags` to the URL: `/foo/bar?x=tags`. This does not apply to collections however, as detailed in Section 2.2.

When issuing a `PUT` or `GET` request to `/foo/bar?x=tags` the response will be of the following form.

```
{
  "self": URL,
  "accessTags": [
    string,
    ...
  ]
}
```

Property	Type	Description
`self`	*URL*	Self-link that can be queried to get an updated representation of the resource. It does not appear in request bodies, only in responses.
`accessTags`	*list*	List of strings that represent tags associated to the resource.
`accessTags[]`	*string*	A string representing a tag associated with the resource.

3.2 account resources

Each principal accessing the ctpd server must have an account. Account resources describe a binding between an authentication token and a set of access tags. The authentication token is represented as a random unpredictable string as described in [RFC_6750]. Administrators can read, create or delete accounts with the relevant methods described in this specification.

The format of a account is described hereafter.

```
{
  "self": URL,
  "accessTags": [
    string,
    ...
  ],
  "annotation": string,
  "token": string
}
```

Property	Type	Description
`self`	*URL*	Self-link that can be queried to get an updated representation of the resource. It does not appear in request bodies, only in responses.
`annotation`	*string*	An optional string representing textual information associated with the token.
`accessTags`	*list*	List of strings that represent tags associated to the resource.
`accessTags[]`	*string*	A string representing a tag associated with the resource.
`token`	*string*	A bearer OAuth2 token. When this token is omitted in a request body, it is chosen by the ctpd server randomly and returned in the response body.

Chapter 4

API calls

The following calls are already described in the *client API* documentation and are not detailed here:

- `GET /`
- `GET /serviceViews/`
- `GET /serviceViews/{id}`
- `GET /serviceViews/{id}/dependencies`
- `GET /serviceViews/{id}/assets`
- `GET /serviceViews/{id}/triggers`
- `GET /serviceViews/{id}/logs`
- `GET /assets/{id}`
- `GET /assets/{id}/attributes`
- `GET /attributes/{id}`
- `GET /attributes/{id}/measurements`
- `GET /measurements/{id}`
- `GET /metrics`
- `GET /metrics/{id}`
- `GET /triggers/{id}`
- `GET /dependencies/{id}`
- `GET /logs/{id}`
- `PUT /measurements/{id}?initiate`
- `POST /views/{id}/triggers`
- `DELETE /triggers/{id}`

4.1 GET /serviceViews/{id}?tags

Get the access tags associated with a service view.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: `x=tag`
- **Response body**: a tag resource
- **Success status code**: 200

Request example

```
GET /servceViews/0123?x=tags
```

There is no request body.

Response example

```
{
  "self": "https://ctp.ikialab.net:8080/ctp/views/VIYQUT1WG628fhbQ?x=tags",
  "accessTags": [
    "id:VIYQUT1WG628gbbP"
    "access:user"
  ]
}
```

4.2 PUT /views/{id}?tags

Modify the access tags associated with a service view.

METHOD CHARACTERISTICS

- **Request body**: a tag resource, excluding the `self` property
- **Query string**: `x=tag`
- **Response body**: a tag tag resource.
- **Success status code**: 200

Request example

```
PUT /views/0123?x=tags
```

The request body contains a tag resource.

```
{
  "accessTags": [
    "id:VIYQUT1WG628gbbP",
    "foo:bar"
  ]
}
```

Response example The response body is the same as the request body with the addition of a self-reference.

```
{
  "self": "https://ctp.ikialab.net:8080/ctp/views/VIYQUT1WG628fhbQ?x=tags",
  "accessTags": [
    "view:VIYQUT1WG628gbbP",
    "foo:bar"
  ]
}
```

4.3 GET /assets/{id}?tags

Read the access tags associated with an asset. It is similar to reading tags associated with a service view.

4.4 PUT /assets/{id}?tags

Change the access tags associated with an asset. It is similar to modifying tags associated with a service view.

4.5 GET /attributes/{id}?tags

Read the access tags associated with an attribute. It is similar to reading tags associated with a service view.

4.6 PUT /attributes/{id}?tags

Change the access tags associated with an attribute. It is similar to modifying tags associated with a service view.

4.7 GET /measurements/{id}?tags

Read the access tags associated with a measurement. It is similar to reading tags associated with a service view.

4.8 PUT /measurements/{id}?tags

Change the access tags associated with a measurement. It is similar to modifying tags associated with a service view.

4.9 GET /metrics/{id}?tags

Read the access tags associated with a metric. It is similar to reading tags associated with a service view.

4.10 PUT /metrics/{id}?tags

Change the access tags associated with a metric. It is similar to modifying tags associated with a service view.

4.11 GET /triggers/{id}?tags

Read the access tags associated with a trigger. It is similar to reading tags associated with a service view.

4.12 PUT /triggers/{id}?tags

Change the access tags associated with a trigger. It is similar to modifying tags associated with a service view.

4.13 GET /dependencies/{id}?tags

note: *unimplemented (June 2015).*

4.14 PUT /dependencies/{id}?tags

note: *unimplemented (June 2015).*

4.15 GET /logs/{id}?tags

Read the access tags associated with a log entry. It is similar to reading tags associated with a service view.

4.16 PUT /logs/{id}?tags

Change the access tags associated with a log entry. It is similar to modifying tags associated with a service view.

4.17 GET /accounts/{id}?tags

Read the access tags associated with an account. It is similar to reading tags associated with a service view.

Note the distinction between the "access tags" and the "account tags" of an account, highligted in [?note].

4.18 PUT /accounts/{id}?tags

Change the access tags associated with an account. It is similar to modifying tags associated with a service view.

Note the distinction between the "access tags" and the "account tags" of an account, highligted in [?note].

4.19 GET /logs

Read all the log entries in the system. This can be paged as several complementary requests as described in the section on *collections* in the *client API*.

4.20 PUT /measurements/{id}?result

Set or update the measurement result within a measurement.

METHOD CHARACTERISTICS

- **Request body**: the sub-part of a measurement defined as the content of the `result` property (see *client API*, section 4.2.6).

- **Query string**: `x=result`
- **Response body**: a measurement resource (see *client API*).
- **Success status code**: 200

The following properties may be omitted in the request body: `updateTime`, `authority` and `signature`. When omitted `updateTime` is set to the current UTC date, while `authority` and `signature` are set to "" (the empty string).

Request example

```
PUT /measurements/VYkcv8y-5zuVAAAH?x=result
```

The request body corresponds to the subset of a measurement resource formed by only keeping the `result` property, excluding all other elements (see *client API*, section 4.2.7).

```
{
  "result": {
    "value": [
        {
            "knots": 1
        }
    ],
    "updateTime": "2015-06-23T11:45:51Z",
    "authority": "net.ikialab",
    "signature": "eyJhbGciOiJSUz..." // (1)
  }
}
```

(1) shortened for clarity

Response example The response is a measurement resource as defined in the *client API* (section 4.2.6).

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/measurements/VYkcv8y-5zuVAAAH",
  "scope": "https://ctp.ikialab.net:4443/ctp/attributes/VYkcv8y-5zuVAAAF",
  "name": "",
  "annotation": "",
  "metric": "https://ctp.ikialab.net:4443/ctp/metrics/VYkcv8y-5zuVAAAG",
  "result": {
    "value": [
      {
        "knots": 1
      }
    ],
    "updateTime": "2015-06-23T11:45:51Z",
    "authority": "net.ikialab",
    "signature": "eyJhbGciOiJSUz..." // (1)
  },
  "objective": {
    "condition": "value[0].knots>5",
    "status": "false"
  },
  "createTrigger": null,
  "userActivated": false,
  "state": "activated"
}
```

(1) shortened for clarity

4.21 PUT /measurements/{id}?objective

Set or update the measurement objective defined for a measurement.

METHOD CHARACTERISTICS

- **Request body**: the sub-part of a measurement defined as the content of the `objective` property (see *client API*, section 4.2.6), without the status property.
- **Query string**: `x=objective`
- **Response body**: a measurement resource (see *client API*).
- **Success status code**: 200

Request example

```
PUT /measurements/VYkcv8y-5zuVAAAH?x=objective
```

The request body corresponds to the subset of a measurement resource formed by only keeping the `object` property, excluding all other elements (see *client API*, section 4.2.7) and excluding the `status` property within the objective.

```
{
    "objective": {
        "condition": "value[0].knots>0"
    }
}
```

Response example The response is a measurement resource as defined in the *client API* (section 4.2.6).

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/measurements/VYkcv8y-5zuVAAAH",
  "scope": "https://ctp.ikialab.net:4443/ctp/attributes/VYkcv8y-5zuVAAAF",
  "name": "",
  "annotation": "",
  "metric": "https://ctp.ikialab.net:4443/ctp/metrics/VYkcv8y-5zuVAAAG",
  "result": {
    "value": [
      {
        "knots": 1
      }
    ],
    "updateTime": "2015-06-23T11:45:51Z",
    "authority": "net.ikialab",
    "signature": "eyJhbGciOiJSUz..." // ❶
  },
  "objective": {
    "condition": "value[0].knots>0",
    "status": "true"
  },
  "createTrigger": null,
  "userActivated": false,
  "state": "activated"
}
```

❶ shortened for clarity

4.22 POST /serviceViews

Create a new service view.

METHOD CHARACTERISTICS

- **Request body**: a service view resource (see *client API*) with the addition of the optional `accessTags` field.
- **Query string**: none
- **Response body**: a view resource, with all properties defined.
- **Success status code**: 201

All properties in the request body are optional. The properties `self` and all other URL-type fields (i.e. `dependencies`, `assets`, `logs` and `triggers`) are ignored in the request body and are replaced by values constructed by the ctpd server.

When a service view is created, it is recommended to specify at least the following two tags in the request:

- `id:{id}`, where `{id}` is the unique identifier assigned to the account that will be allowed to access the service view.
- `access:user`, which allows authenticated principals to access the resources.

Note that the response will not contain the `accessTags` field, which can only be accessed with the adequate tag related API call.

Request example

```
POST /views/
```

The request body is a subset of the `serviceView` resource defined in the main API.

```
{
   "name": "storage service",
   "accessTags": [
      "id:1234",
      "access:user"
   ]
}
```

Response example

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/serviceViews/VYkcvsy-5zuVAAAB",
  "scope": "https://ctp.ikialab.net:4443/ctp/",
  "name": "my view",
  "annotation": "this is a view",
  "provider": "net.ikialab",
  "dependencies": "https://ctp.ikialab.net:4443/ctp/serviceViews/VYkcvsy-5zuVAAAD/ ↩
      dependencies",
  "logs": "https://ctp.ikialab.net:4443/ctp/views/VYkcvsy-5zuVAAAB/logs",
  "triggers": "https://ctp.ikialab.net:4443/ctp/views/VYkcvsy-5zuVAAAB/triggers",
  "assets": "https://ctp.ikialab.net:4443/ctp/serviceViews/VYkcvsy-5zuVAAAD/assets",
  "serviceClass": ""
}
```

4.23 POST /serviceViews/{id}/assets

Create an asset as part of the service-view identified by `{id}`.

The newly created asset automatically inherits all tags from the parent service-view, unless tags are specified in the request body.

METHOD CHARACTERISTICS

- **Request body**: An asset resource as defined in the *client API* (section 4.2.3), with only the following properties: `name`, `annotation`, `assetClass` and the additional optional `accessTags` field.
- **Query string**: none
- **Response body**: An asset resource.
- **Success status code**: 201

All properties in the request body are optional. The properties `self` and all other URL-type fields are ignored in the request body and are replaced by values constructed by the ctpd server.

Note that the response will not contain the `accessTags` field, which can only be accessed with the adequate tag related API call.

Request example

```
POST /serviceViews/VYkcvsy-5zuVAAAD/assets
```

The request body is a subset of an asset resource.

```
{
  "name": "name of asset",
  "annotation": "testing insertion of asset",
  "assetClass": ""
}
```

Response example The response body is a fully defined asset resource, which includes a `self` property with the URL of the newly created object.

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/assets/VYk6dMy-5zr9AAAE",
  "scope": "https://ctp.ikialab.net:4443/ctp/serviceViews/VYk6c8y-5zr9AAAD",
  "annotation": "testing insertion of asset",
  "name": "name of asset",
  "attributes": "https://ctp.ikialab.net:4443/ctp/assets/VYk6dMy-5zr9AAAE/attributes",
  "assetClass": ""
}
```

4.24 POST /assets/{id}/attributes

Create an attribute attached to the asset identified by `{id}`.

The newly created attribute automatically inherits all tags from the parent asset, unless tags are specified in the request body.

METHOD CHARACTERISTICS

- **Request body**: an attribute resource as defined in the *client API* (section 4.2.4), with only the following properties defined: `name`, `annotation` and the additional optional `accessTags` field.
- **Query string**: none

- **Response body**: An attribute resource.
- **Success status code**: 201

All properties in the request body are optional. The properties `self` and all other URL-type fields are ignored in the request body and are replaced by values constructed by the ctpd server.

Note that the response will not contain the `accessTags` field, which can only be accessed with the adequate tag related API call.

Request example The request body is a subset of an attribute resource.

```
{
  "name": "velocity",
  "annotation": "",
}
```

Response example The response body is a fully defined attribute resource, which includes a `self` property with the URL of the newly created object.

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/attributes/VYk6dMy-5zr9AAAF",
  "scope": "https://ctp.ikialab.net:4443/ctp/assets/VYk6dMy-5zr9AAAE",
  "name": "velocity",
  "annotation": "",
  "measurements": "https://ctp.ikialab.net:4443/ctp/attributes/VYk6dMy-5zr9AAAF/ ↩
      measurements"
}
```

4.25 POST /attributes/{id}/measurements

Create a measurement that applies to the attribute identified by `{id}`.

The newly created measurement automatically inherits all tags from the parent attribute, unless tags are specified in the request body.

METHOD CHARACTERISTICS

- **Request body**: a measurement resource as defined in the *client API* (section 4.2.6), with the addition of the optional `accessTags` field.
- **Query string**: none
- **Response body**: a measurement resource.
- **Success status code**: 201

All properties in the request body are optional. The properties `self` and all other URL-type fields are ignored in the request body and are replaced by values constructed by the ctpd server.

Note that the response will not contain the `accessTags` field, which can only be accessed with the adequate tag related API call.

Request example The request body does not contain the `self` or `scope` properties.

```
{
  "name": "",
  "annotation": "",
  "metric": "https://ctp.ikialab.net:4443/ctp/metrics/VYk6dMy-5zr9AAAG",
  "result": {
    "value": [
      {
        "knots": 7
      }
    ],
    "updateTime": "2015-06-23T16:52:36Z",
  },
  "objective": {
    "condition": "value[0].knots>5",
    "status": "true"
  },
  "createTrigger": "",
  "userActivated": false,
  "state": "pending"
}
```

Response example The response body is a fully defined measurement resource, which includes a `self` property with the URL of the newly created object.

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/measurements/VYk6dMy-5zr9AAAH",
  "scope": "https://ctp.ikialab.net:4443/ctp/attributes/VYk6dMy-5zr9AAAF",
  "name": "",
  "annotation": "",
  "metric": "https://ctp.ikialab.net:4443/ctp/metrics/VYk6dMy-5zr9AAAG",
  "result": {
    "value": [
      {
        "knots": 7
      }
    ],
    "dateTime": "2015-06-23T16:52:36+03:00",
    "authority": "",
    "signature": ""
  },
  "objective": {
    "condition": "value[0].knots>5",
    "status": "true"
  },
  "createTrigger": "",
  "userActivated": false,
  "state": "pending"
}
```

4.26 POST /metrics

Create a new metric.

By default the metric is initialized with a set of tags containing one tag: `access:anybody`, unless tags are specified in the request body.

METHOD CHARACTERISTICS

- **Request body**: A metric resource as defined in the *client API* (section 4.2.5), with the addition of the optional `accessTags` field.
- **Query string**: none
- **Response body**: A metric resource.
- **Success status code**: 201

All properties in the request body are optional. The properties `self` and all other URL-type fields are ignored in the request body and are replaced by values constructed by the ctpd server.

Note that the response will not contain the `accessTags` field, which can only be accessed with the adequate tag related API call.

Request example The request body does not contain the `self` property.

```
{
  "name": "Nautical knots",
  "annotation": "",
  "baseMetric": "http://en.wikipedia.org/wiki/Knot_%28unit%29",
  "measurementParameters": [],
  "resultFormat": [
    {
      "name": "knots",
      "type": "number"
    }
  ]
}
```

Response example The response body is a fully specified metric resource, which includes a `self` property with the URL of the newly created object.

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/metrics/VYk6dMy-5zr9AAAG",
  "scope": "https://ctp.ikialab.net:4443/ctp/",
  "name": "Nautical knots",
  "annotation": "",
  "baseMetric": "http://en.wikipedia.org/wiki/Knot_%28unit%29",
  "measurementParameters": [],
  "resultFormat": [
    {
      "name": "knots",
      "type": "number"
    }
  ]
}
```

4.27 POST /dependencies

note: *unimplemented (June 2015)*

4.28 DELETE /views/{id}

Deletes the view identified by `{id}` and all resources that are directly dependent on the view: service-views, assets, attributes, measurements, triggers and logs.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: none
- **Success status code**: 204

Request example

```
DELETE /views/VYk6c8y-5zr9AAAB
```

There is no query body.

Response example There is no response body.

4.29 DELETE /serviceViews/{id}

Deletes the service-view identified by `{id}` and all resources that are directly dependent on the service-view: assets, attributes, measurements and triggers.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: none
- **Success status code**: 204

Request example

```
DELETE /serviceViews/VYk6c8y-5zr9AAAD
```

There is no query body.

Response example There is no response body.

4.30 DELETE /assets/{id}

Deletes the asset identified by `{id}` and all resources that are directly dependent on the asset: attributes, triggers and measurements.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: none
- **Success status code**: 204

Request example

```
DELETE /assets/VYk6c8y-5zr9AAAE
```

There is no query body.

Response example There is no response body.

4.31 DELETE /attributes/{id}

Deletes the attribute identified by `{id}` and all resources that are directly dependant on the attribute: triggers and measurements.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: none
- **Success status code**: 204

Request example

```
DELETE /attributes/VYk6dMy-5zr9AAAF
```

There is no query body.

Response example There is no response body.

4.32 DELETE /measurements/{id}

Deletes the measurement identified by `{id}` and any trigger directly dependant on the measurement.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: none
- **Success status code**: 204

Request example

```
DELETE /measurements/VYk6dMy-5zr9AAAH
```

There is no query body.

Response example There is no response body.

4.33 DELETE /metrics/{id}

Deletes the metric identified by `{id}`. This method will return an error if the metric is currently in use in a measurement (status code 409).

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: none
- **Success status code**: 204

Request example

```
DELETE /metrics/VYk6dMy-5zr9AAAG
```

There is no query body.

Response example There is no response body.

4.34 DELETE /dependencies/{id}

note: *unimplemented (June 2015).*

4.35 DELETE /logs/{id}

note: *unimplemented (June 2015).*

4.36 GET /account/{id}

Read the details of an account resource.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: a token resource
- **Success status code**: 200

Request example

```
GET /accounts/VYkyC8y-5xbBAAAC
```

Response example

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/accounts/VYkyC8y-5xbBAAAC",
  "name": "",
  "annotation": "john.smith@mail.com",
  "accessTags": [
     "views:1234",
     "access:user",
     "access:anybody"
  ],
  "key": "9rPwDVfZwdqxMiR_fGY7BxvCTPfEPqec"
}
```

4.37 POST /account

METHOD CHARACTERISTICS

- **Request body**: A token resource.
- **Query string**: none
- **Response body**: A token resource.
- **Success status code**: 201

All properties in the request body are optional. The properties `self` and all other URL-type fields are ignored in the request body and are replaced by values constructed by the ctpd server.

Request example

```
POST /tokens
```

The request body only contains the `name", `annotation` and the `accessTags` properties.

```
{
  "name": "",
  "annotation": "jsmith@gmx.com",
  "accessTags": [
     "views:1234",
     "access:user",
     "access:anybody"
  ]
}
```

Response example The response body contains a fully defined token resource, which includes a `self` property with the URL of the newly created object.

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/tokens/VYkyC8y-5xbBAAAC/",
  "scope": "https://ctp.ikialab.net:4443/ctp/",
  "name": "",
  "annotation": "jsmith@gmx.com",
  "key": "9rPwDVfZwdqxMiR_fGY7BxvCTPfEPqec"
}
```

4.38 GET /tokens

Read the collection of all existing tokens.

METHOD CHARACTERISTICS

- **Request body**: none.
- **Query string**: Optional `page` and `items` as for all collections, as defined in the *client API* (section 4.1.5).
- **Response body**: A collection of links to token resources, where `collectionType` = `tokens`.
- **Success status code**: 200

Request example

```
GET /tokens
```

There is no request body.

Response example

```
{
  "self": "https://ctp.ikialab.net:4443/ctp/tokens/",
  "scope": "https://ctp.ikialab.net:4443/ctp/"
  "collectionLength": 6,
  "returnedLength": 6,
  "collectionType": "tokens",
  "collection": [
    "https://ctp.ikialab.net:4443/ctp/tokens/VWcIa-sfTYZSm6Qs/",
    "https://ctp.ikialab.net:4443/ctp/tokens/VWcIa-sfTYZSm6Qt/",
    "https://ctp.ikialab.net:4443/ctp/tokens/VYkcKMy-5zbXAAAC/",
    "https://ctp.ikialab.net:4443/ctp/tokens/VYkcvsy-5zuVAAAC/",
    "https://ctp.ikialab.net:4443/ctp/tokens/VYkyC8y-5xbBAAAC/",
    "https://ctp.ikialab.net:4443/ctp/tokens/VYk6c8y-5zr9AAAC/"
  ]
}
```

4.39 PUT /tokens/{id}

note: *unimplemented (June 2015).*

4.40 DELETE /tokens/{id}

Delete an existing token resource.

METHOD CHARACTERISTICS

- **Request body**: none
- **Query string**: none
- **Response body**: none
- **Success status code**: 204

Request example

```
DELETE /tokens/VWcIa-sfTYZSm6Qs/
```

There is no request body.

Response example There is no response body.

Chapter 5

References

5.1 Bibliography

[1] [client_API] Cloud Security Alliance. "CTP Technical Model and API". Ed. Dr. Alain Pannetrat. Version 2.13. October 2015.

[2] [RFC_4648] S. Josefsson, "RFC 4648: The Base16, Base32, and Base64 Data Encodings", The Internet Engineering Task Force, October 2006.

[3] [RFC_6750] M. Jones, D. Hardt, RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage, Internet Engineering Task Force (IETF), October 2012.