Yeti Documentation, Release 1.0

Useful links

- Project website & blog
- Yeti Users mailing list
- Code repository
- Installation
- Getting started

Yeti is a platform meant to organize observables, indicators of compromise, TTPs, and knowledge on threats in a single, unified repository. Yeti will also automatically enrich observables (e.g. resolve domains, geolocate IPs) so that you don’t have to. Yeti provides an interface for humans (shiny Bootstrap-based UI) and one for machines (web API) so that your other tools can talk nicely to it.

Yeti was born out of frustration of having to answer the question “where have I seen this artifact before?” or Googling shady domains to tie them to a malware family.

In a nutshell, Yeti allows you to:

- Submit observables and get a pretty good guess on the nature of the threat.
- Inversely, focus on a threat and quickly list all TTPs, Observables, and associated malware.
- Let responders skip the “Google the artifact” stage of incident response.
- Let analysts focus on adding intelligence rather than worrying about machine-readable export formats.
- Visualize relationship graphs between different threats.

This is done by:

- Collecting and processing observables from a wide array of different sources (MISP instances, malware trackers, XML feeds, JSON feeds . . . )
- Providing a web API to automate queries (think incident management platform) and enrichment (think malware sandbox).
- Export the data in user-defined formats so that they can be ingested by third-party applications (think blocklists, SIEM).
1.1 Installation

Installing Yeti is pretty straightforward. This procedure was tested on Ubuntu 18.04, but YMMV.

Install dependencies:

```
$ sudo apt-get install build-essential git python-dev mongodb redis-server libxml2-dev libxslt-dev zlib1g-dev python-virtualenv wkhtmltopdf
```

Install Yarn:

```
$ curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
$ echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
$ sudo apt-get update && sudo apt-get install yarn
```

Download Yeti:

```
$ git clone https://github.com/yeti-platform/yeti.git
```

Activate virtualenv if you want to, then install requirements:

```
$ cd yeti
$ [sudo] pip install -r requirements.txt
$ yarn install
```

Create the logging directory:

```
$ [sudo] mkdir /var/log/yeti
$ [sudo] chown <user> /var/log/yeti
```
1.1.1 Quick & dirty

Start the web UI (will spawn a HTTP server on http://localhost:5000):

```
$ ./yeti.py webserver
```

This will only enable the web interface - if you want to use Feeds and Analytics, you’ll be better off starting the workers as well:

```
$ celery -A core.config.celeryctl.celery_app worker --loglevel=ERROR -Q exports -n exports -Ofair -c 2 --purge
$ celery -A core.config.celeryctl.celery_app worker --loglevel=ERROR -Q feeds -n feeds -Ofair -c 2 --purge
$ celery -A core.config.celeryctl.celery_app worker --loglevel=ERROR -Q analytics -n analytics -Ofair -c 2 --purge
$ celery -A core.config.celeryctl.celery_app worker --loglevel=ERROR -Q oneshot -n oneshot -c 2 --purge
$ celery -A core.config.celeryctl.celery_app worker --loglevel=ERROR -Q analytics -n analytics -Ofair -c 2 --purge
```

Or, to bootstrap a production use instance of Yeti on Ubuntu 16.04 (without the Redis tweaks), everyone’s favorite command:

```
$ curl https://raw.githubusercontent.com/yeti-platform/yeti/master/extras/ubuntu_bootstrap.sh | sudo /bin/bash
```

There is also support for other platforms:

- **CentOS**: extras/centos_bootstrap.sh

1.1.2 Production use

For production use, it may be better to daemonize Yeti and tweak redis for performance.

Install nginx and uwsgi:

```
$ sudo apt-get install nginx uwsgi
```

Optimize redis

Some optimizations for redis (taken from here):

Add the following lines in /etc/sysctl.conf:

```
# redis tweak
vm.overcommit_memory = 1
```

Add the following lines in /etc/rc.local:

```
# disable transparent huge pages (redis tweak)
See here for details: https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/
# increase max connections
echo 65535 > /proc/sys/net/core/somaxconn or (sysctl -w net.core.somaxconn=65535)
exit 0
```
Install systemd services

Copy all files in `extras/systemd/*` to `/lib/systemd/system/`. If you’d rather have the web content served through nginx (recommended for production), copy `yeti_uwsgi.service`, otherwise you’ll be fine with `yeti_web.service`.

Enable the scripts with:

```
$ sudo systemctl enable yeti_<SERVICENAME>.service
```

And start with:

```
$ sudo systemctl start yeti_<SERVICENAME>.service
```

systemd protips:

```
$ sudo service yeti_web start|stop|restart
or
$ sudo systemctl start|status|stop yeti_web
```

To enable the systemd scripts once you’ve installed them:

```
sudo systemctl enable yeti_web
```

If you’re running nginx, add the following configuration to one of the nginx server directives:

```
server {
    listen 80;
    server_name yeti;

    location / {
        include uwsgi_params;
        uwsgi_pass 127.0.0.1:8000;
    }
}
```

Replace the `listen` and `server_name` directives as you see fit.

1.1.3 Keeping Yeti up-to-date

Keeping Yeti up-to-date is not that hard. You just need to `cd` into the directory and `git pull`. In some cases, the database schema might change a bit, and we always recommend running:

```
$ ./yeti.py syncdb
```

before restarting Yeti. Note that Yeti will automatically check if its code and databases are not synchronized and will apply migrations automatically when the webserver is launched.

1.2 Use-cases

So Yeti sounds cool, but how do I actually use it? Glad you asked. Even though great efforts have been made to “build the manual into the UI”, it’s natural that some questions arise.

You will need to have a running install of Yeti for this to make any sense. Please refer to the `Installation` section of the documentation.
What is it all about?

Yeti is about organizing observables, indicators of compromise, TTPs, and knowledge on threat actors in a single, unified repository. Ideally, this repository should be queryable in an automated way by other tools (spoiler: it is!)

1.2.1 Documenting malware intelligence

Your sandbox just spat out its analysis of the latest Dridex sample and you figured out that it’s using a subdirectory in the user’s Roaming directory to store the data it steals. You’d like to document this so that another analyst can quickly know that this is typical Dridex behavior.

Creating a Malware Entity

You start by adding a new Entity of type Malware called Dridex. Navigate to New > Malware, and populate the fields.

Most fields are self-explanatory. As for the others:

- Tags that link to this entity - Observables tagged with the Tags specified here will be linked to this Entity. See note below for more information.
- Bind to entities - Allows you to bind this entity to another one. Malware to TTPs, actors to malware, etc.
- **Aliases** - Other known names for this entity. This field will be indexed in search too.

**Note:** Tags are a way to quickly assess an Observable’s context. Yeti also uses tags to link observables to Entities (TTPs, Actors, Exploit Kits, etc.) When you tag an observable, Yeti will look for any Entities that have declared these tags and link the observable to them. You can also tag observables even if no Entity has declared it; it will just not be linked to anything. This is may be useful for tags like *sinkhole, whitelist, internal*.

Clicking on **Save** will take you to the following page:

![Screen capture of YETI interface showing the Entities page for Dridex malware](image)

This will be your go-to page for all information on Dridex. You can get back to it by navigating to “Entities > Malware”.

**Note:** The **Files** control box allows you to upload files (external reports, samples, etc.) and attach them to this Entity. *(No further processing is done on these files, for the moment).*

Any other linked entities will appear in their corresponding tab under the **Description** section.

---

### 1.2. Use-cases
Adding Observables

Time to add our observable! Head to Observables > Search & Add. This page allows you to add or try to match one or several Observables. The “write” behavior is toggled by the Add unknown observables to database checkbox. When adding Observables, you can specify one or more Tags for your observable as well as the observable type. Guessing should be fine for most observable types (URLs, IP addresses, hostnames) but make sure you specify it if it gets ambiguous (e.g. Bitcoin address vs. Hash).

Let’s add the following observable:

C:\Users\admin\AppData\Roaming\Adada\stolen.dat

Click on Launch to launch the Search / Add procedure.
The results page shows the newly added Observable(s) as well as any entities associated linked to them, if any. In this case, we’re “cheating” since tagging our Observable “Dridex” associated the observable to the malware. But if another analyst searches for the same observable, they will leverage Yeti’s database and see the newly created association (you can try this by just searching for your Observable again, with the “Add to database” checkbox off.).

You can navigate back to the Dridex page by clicking on the link under Related Entities. Notice how the “Observables” tab shows the link was successful. You can always break the link between Entities and Observables by hovering on the line and clicking the “Unlink” icon, as can be seen below:
Adding indicators

The following week, you run into a new sample of Dridex which seems to have changed. It now stores its data in a random subdirectory under Roaming. You’ve observed the following paths:

- C:\Users\admin\AppData\Roaming\Faas
- C:\Users\admin\AppData\Roaming\Thsk
- C:\Users\admin\AppData\Roaming\Lzoo

At this point, it seems pretty clear that the subdirectory name consists of 1 uppercase letter followed by 3 lowercase letters, all chosen randomly. You add the three observables to the database as seen before, tagging them dridex. But what if an analyst searches for C:\Users\admin\AppData\Roaming\Enxa or C:\Users\admin\AppData\Roaming\Xiwa? This is precisely what indicators are for.

Think of indicators as an “enhanced Observable”. Indicators come in different flavors, simple regular expressions or Yara rules (other indicator types will be included in the future). When submitting Observables in “Search & Add”, Yeti will run all your indicators on them and any matches will be put forward.

Head over to New > Regular expression and complete the Pattern tab as follows:

```
[A-Z]:\Users\[a-z]+\AppData\Roaming ([A-Z][a-z]{3})
```

Make sure you add “Dridex” in the “Link with entities” field.
Short explanation on fields:

- **Location** - This helps the analyst know where they can find this indicator. This is free-text and useful values could be *Filesystem, HTTP headers, HTTP URI, Registry*, etc.

- **Diamond edge** - Corresponding Diamond Model edge.

- **Link with entities** - Link this Indicator to other entities. In this case we want to link it to Dridex.

- **Pattern** - Complete this with your regular expression or Yara rule.
Note how the **Malware** tab in the indicator page on the screenshot above now shows a link to the Dridex malware. Opening the Dridex entity and selecting the **Indicators** tab also reflects this relationship:
Good! Now any analysts who wonder if a weirdly-named directory under Roaming is actually malware, they can look it up in the **Observables > Search & Add** page with or without checking the “Add unknown” checkbox.
In our example, we haven’t checked the “Add” checkbox, so we can add the observable and tag it directly from this page.

**Note:** The “suggested tags” are generated by the Entity name and any associated tags the entity has. In this case, Dridex is tagged with “dridex” and “banker”.

**Automation**

All this information can be queried from other software (think incident management platforms, forensic frameworks…) using Yeti’s API. Let’s search for all observables with a value matching the regular expression "Roaming":

```
$ http -vv --json POST localhost:5000/api/observablesearch/ filter:='{"value": "Roaming"}' params:='{"regex": "true"}'
```

POST /api/observablesearch/ HTTP/1.1
Accept: application/json, */*
Accept-Encoding: gzip, deflate
Connection: keep-alive
Content-Length: 61
Content-Type: application/json
Host: localhost:5000

(continues on next page)
User-Agent: HTTPie/0.9.8
{
  "filter": {
    "value": "Roaming"
  },
  "params": {
    "regex": "true"
  }
}

HTTP/1.0 200 OK
Content-Length: 2033
Content-Type: application/json
Date: Mon, 06 Mar 2017 17:38:50 GMT
Server: Werkzeug/0.11.15 Python/2.7.13

[
{
  "context": [],
  "created": "2017-03-06T17:35:07.614000",
  "human_url": "http://localhost:5000/observable/58bd9dcb10c553738521480e",
  "id": "58bd9dcb10c553738521480e",
  "last_analyses": {},
  "sources": [],
  "tags": [
    {
      "first_seen": "2017-03-06T17:35:07.627000",
      "fresh": true,
      "last_seen": "2017-03-06T17:35:07.627000",
      "name": "dridex"
    }
  ],
  "type": "Path",
  "url": "http://localhost:5000/api/observable/58bd9dcb10c553738521480e",
  "value": "C:\Users\admin\AppData\Roaming\Lzoo"
},
{
  "context": [],
  "created": "2017-03-06T17:35:07.592000",
  "human_url": "http://localhost:5000/observable/58bd9dcb10c553738521480b",
  "id": "58bd9dcb10c553738521480b",
  "last_analyses": {},
  "sources": [],
  "tags": [
    {
      "first_seen": "2017-03-06T17:35:07.606000",
      "fresh": true,
      "last_seen": "2017-03-06T17:35:07.606000",
      "name": "dridex"
    }
  ],
  "type": "Path",
  "url": "http://localhost:5000/api/observable/58bd9dcb10c553738521480b",
  "value": "C:\Users\admin\AppData\Roaming\Thsk"
},
{

}  
]  

1.2. Use-cases
We can also match observables against indicators, just like in the “Search & Add” page. This is typically what you’d want to implement in your sandbox’s code, so that it can automatically query Yeti with any observables it finds and return any interesting results based on your intelligence!

Try matching a new, unknown observable C:\Users\admin\AppData\Roaming\Ijhz:

```bash
$ http -jvv POST http://localhost:5000/api/analysis/match observables:='[ "C:\\Users\\admin\\AppData\\Roaming\\Ijhz"]'
```

(continues on next page)
1.2. Use-cases
1.2.2 Ingesting and enriching a third-party report

An abundant source (but of uneven quality) of intelligence on adversaries can be found in the different reports that are published by vendors, CERTs, etc. Yeti provides an easy way to parse information from a variety of sources (blogposts, PDF files, raw text) and integrate them in Yeti.

Starting an Investigation

The “ingestion” of a report in Yeti parlance typically means the creation of an Investigation, from which you will be able to add observables and tie them to existing or new Entities like Actors, Malware, or TTPs.

Let’s try creating an investigation out of this blogpost. Head to Investigations > Import:

Investigations can be started on your own, from a single Observable or Entity, using the New Investigation button on their corresponding page.

Clicking on Start Import will lead you to the Import page where all the magic happens. On the left pane is the processed document (a generated PDF or Markdown, depending on whether wkhtmltopdf is installed) corresponding to the document you just imported. On the right pane you’ll see and be able to tag all the Observables that were parsed from it. Clicking on one of those will scroll the left pane until you see the observable in its original context. Once you know what it is, you can tag the observable by clicking on the right of the + sign.
You can always discard or edit an observable by hovering and clicking on the corresponding symbol.

Once you have tagged all your observables accordingly, it’s time to link them together so that you can better represent the attack flow. Click on **Import**.
This is a mess! Now it’s up to you to link the entities with each other. You do that by Clicking on the **Add link** button in the toolbar, then by drawing the link you want between entities. You’ll have something like this.
Once you have a clear picture of what’s happening in that blogpost and the links between observables, you can start creating or adding Indicators, TTPs, Actors, and linking them to each other.

**Note:** Links made during an investigation exist only in the investigation and have no effect outside of it. This is to separate every analyst’s work. We’re still thinking of the best way to include this information (once confirmed? vetted?) in the largest Yeti corpus. Stay tuned!

### 1.2.3 Creating a blocklist

An export is a recurring task that will render a (usually large) selection of tagged Observables according to a specific Template. In this scenario we’re going to create a list of all Path observables involving the Dridex malware.

**Creating an export template**

First thing you want to do when creating an Export is to create a matching Template. Templates are a way for Exports to know how to render the data they’re cycling through. The templating engine is Jinja2, the same one Flask uses for rendering webpages; it is therefore extremely flexible.

**Note:** Templates can be re-used across exports. For instance, you can have one CSV template rendering value, tags, and you may want to create separate CSV exports for URLs, Hostnames, and IPs: they can all re-use the same CSV template. In this case, since it’s your first export, you’ll need to create the template first.
Head to Settings > Dataflows and click on the Templates tab. The form is pretty straightforward, a Name and a Body. The body must be valid Jinja2 (a very succinct example is provided in the placeholder). The template is fed a generator of Observables called elements.

For instance, if we want to create a value,created_date CSV, we need to write something along the lines of:

```
value,tags
{%for obs in elements%},{%obs.value}{{obs.created}}
{%endfor%}
```

**Warning:** Since this will be rendered as text and not as HTML, whitespace is taken into account (this is why the loop does not span multiple lines)

Now back to the export tab. Click on the New button. Noteworthy fields in this form include:

- **Ignore tags** - These are tags that should be ignored. They will disappear from the exported data. Also, if an Observable’s only tag is an ignored tag, it won’t be exported. E.g.: You have a feed that tags Alexa’s top 1M domains with Alexa. You may want to export them if they are tagged with malware or C2, but not if they are only tagged alexa.

- **Include tags** - By default Exports will select all tagged Observables. Here you can refine this selection by specifying which tags the Observables must have to be exported. E.g.: You want to create a separate, high-confidence blocklist for ransomware URLs, you could include ransomware and high here.

- **Exclude tags** - Same rationale, sometimes you may want some Observables never to be exported, even if they are tagged with other stuff. E.g. Google is often tagged malware since it happens that malicious domains are sometimes “parked” on Google IPs. If google.com is tagged whitelist, you might want to include whitelist here.

1.2. Use-cases
Once this is set, click “Save” and your export is ready to go!

Provided the Celery Scheduler is launched, you should be generating your feeds soon enough. You can toggle or manually generate them using the icons on the corresponding line. The link icon contains the external link where the exported data will be available from. You can try downloading it:
1.3 Yeti objects

Yeti relies on several objects, which are defined here.

1.3.1 Observables

class core.observables.Observable(*args, **values)

Base class for Observables in Yeti

Observables describe elements that can be seen in investigations, incidents, reports, intelligence, etc. They are usually technical data about specific threats or actors.

value
The observable’s technical value (the observed URL, hostname, IP address…)

sources
An array of strings that define how the observable was inserted

description
A free-text description of the observable

context
A JSON object providing extra information as to why the observable was added. Context can be added through the API or through Feeds
tags
An array of core.observables.tag.ObservableTag objects

last_analyses
An array of JSON objects indicating the last analysis time for a particular analytics

created
Creation date

last_tagged
Date when a given observable was last tagged

exclude_fields
Fields to be excluded from automatic form creation

exception DoesNotExist
exception MultipleObjectsReturned

add_context (context, replace_source=None, dedup_list=[])
Adds context to an Observable.

“Context” is represented by a JSON object (or Python dict()) that will be added to the Observable’s context set. Context should provide information on why the Observable has been added to the database.

Context can be any information, but it needs to have a source key that can point the analyst to the source of the context.

Parameters

• context – a JSON object representing the context to be added.

• replace_source – If defined, contexts having a source attribute set to replace_source will be deleted before insert

• dedup_list – takes a list of fields to ignore during dedup comparison. i.e. date/count type fields. Empty list will skip the partial dedup as dedup for the exact same context is already builtin.

Returns A fresh instance of the Observable as it exists in the database.

add_source (source)
Adds a source to the observable instance

Parameters source – a string to add to the array of sources.

classmethod add_text (text, tags=[], force_type=None)
Adds and returns an observable for a given string.

Parameters text – the text that will be used to add an Observable from.

Returns A saved Observable instance.

static change_all_tags (old_tags, new_tag)
Changes tags on all observables

Parameters

• old_tags – A string or array of strings representing tag names to change

• new_tag – The new tag name by which all old_tags should be replaced

clean()
Hook for doing document level data cleaning before validation is run.
Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

```
classmethod get_form()
    Gets the appropriate form for a given observable

get_tags (fresh=True)
    Returns an array of strings containing an observables’ fresh tags names.
    Parameters fresh – set to False to also include non-fresh tags in the result
    Returns Array of strings containing an observables’ fresh tags names.

static guess_type (string)
    Tries to guess the type of observable given a string.
    Parameters string – The string that will be used to guess the observable type from.
    Returns An observable Class.
    Raises ObservableValidationError if no type could be guessed.

remove_context (context)
    Removes Context from an observable.
    Parameters context – a JSON object representing the context to be removed.
    Returns A fresh instance of the Observable as it exists in the database.

tag (new_tags, strict=False, expiration=None)
    Tags an observable.
    An observable can be tagged to add more information as to what it represents.
    Parameters
        • new_tags – An array of strings to tag the observable with.
        • strict – Set to True to replace all existing tags with the new_tags.
        • expiration – Timedelta field after which the Tag will not be considered fresh anymore.
    Returns A fresh Observable instance as reloaded from the database.
```

### 1.3.2 Feeds

```
class core.feed.Feed(*args, **values)
    Base class for Feeds. All feeds must inherit from this.

    Feeds describe the way Yeti automatically collects and processes data.

    frequency
        A timedelta variable defining the frequency at which a feed is to be ran. Example: timedelta(hours=1)

    name
        Required. The feed’s name. Must be the same as the class name. Example: "ZeusTrackerConfigs"

    source
        If working with helpers. This designates URL on which to fetch the data. Example: "https://zeustracker.abuse.ch/monitor.php?urlfeed=configs"

    description
        Bref feed description. Example: "This feed shows the latest 50 ZeuS config URLs."
```
Note: These attributes must be defined in every class inheriting from `Feed` as the key-value items of a `default_values` attribute. See `Creating feeds` for more details.

```python
exception DoesNotExist
exception MultipleObjectsReturned

analyze(item)
Function responsible for processing the item passed on by the `update` function.

Raises `NotImplementedError` if no function has been implemented.

parse_commit(item, headers, verify=True)
Helper function used to parse github commit and extract content. See `core.feed.Feed.update_github()` for details

Parameters

  - `item` – All details about an github commit
  - `headers` – Used for correct github auth or empty

Returns Yields all new content for the commit and filename of the original file

parse_xml(data, main_node, children)
Helper function used to parse XML. See `core.feed.Feed.update_xml()` for details

update()
Function responsible for retrieving the data for a feed and calling the `analyze` function on its data, typically one line at a time.

Helper functions may be called to facilitate parsing of common data formats.

Raises `NotImplementedError` if no function has been implemented.

update_csv(delimiter=",", headers=None, auth=None, verify=True, comment=",", filter_row=None, names=None, header=0, compare=False, date_parser=None)
Helper function. Performs an HTTP request on `source` and treats the response as an CSV file, yielding a dict for each parsed line.

Parameters

  - `delimiter` – A string delimiting fields in the CSV. Default is ",".
  - `headers` – Optional headers to be added to the HTTP request.
  - `auth` – Username / password tuple to be sent along with the HTTP request.
  - `verify` – Force ssl verification.
  - `comment` – Comment char in csv data for panda.
  - `filter_row` – name of columns to filter rows
  - `names` – names of columns of the dataframe
  - `header` – number of the if the name of columns is specified in csv data.
  - `compare` – if the filtering must be made by the last run
  - `date_parser` – function to parse the date

Returns return a dataframe pandas filtered by date of the last run
**update_github** *(headers=None, auth=None, params=None, verify=True)*

Helper function. Grabs data about latest commits iterates them.

**Parameters**

- **headers** – Optional headers to be added to the HTTP request.
- **auth** – Username / password tuple to be sent along with the HTTP request.
- **params** – Optional param to be added to the HTTP request.

**Returns**

Python `dict` object representing the response JSON. .. rubric:: Example

https://api.github.com/repos/eset/malware-ioc/commits/2602f02a1b0ff6d4cfcefecf93f3b4320d8b4207

**update_json** *(method=\'get\', data=None, headers=None, auth=None, params=None, verify=True, filter_row=\'u\', key=None)*

Helper function. Performs an HTTP request on source and parses the response JSON, returning a Python `dict` object.

**Parameters**

- **method** – Optional HTTP method to use GET/POST/etc lowercase
- **headers** – Optional headers to be added to the HTTP request.
- **data** – Dictionary containing POST data to send.
- **auth** – Username / password tuple to be sent along with the HTTP request.
- **params** – Optional param to be added to the HTTP request.
- **verify** – Force SSL verification.
- **filter_row** – Name of columns to filter rows.
- **key** – Key in JSON response to return data.

**Returns**

Python `dict` object representing the response JSON.

**update_lines** *(headers=None, auth=None, verify=True)*

Helper function. Performs an HTTP request on source and treats each line of the response separately.

**Parameters**

- **headers** – Optional headers to be added to the HTTP request.
- **auth** – Username / password tuple to be sent along with the HTTP request.
- **verify** – Force ssl verification.

**Returns**

Yields string lines from the HTTP response.

**update_xml** *(main_node, children, headers=None, auth=None, verify=True)*

Helper function. Performs an HTTP request on source and treats the response as an XML object, yielding a `dict` for each parsed element.

The XML must have a `main_node`, and an array of `children`. For example:

```xml
<main_node>
  <child1></child1>
  <child1></child2>
  <child1></child3>
</main_node>
```

1.3. Yeti objects
Parameters

- **main_node** – A string defining the parent node that delimitates a dict to be yielded.
- **children** – An array of strings defining the children of the parent node. These will be the keys of the dict.
- **headers** – Optional headers to be added to the HTTP request.
- **auth** – Username / password tuple to be sent along with the HTTP request.
- **verify** – Force ssl verification.

Returns

Yields Python dictionary objects. The dictionary keys are the strings specified in the children array.

### 1.4 Extending Yeti

Yeti can be extended to suit most use-cases. Extension is usually made via three vectors:

- **Feeds**: Your basic data source. Feeds can be customized to automatically collect and processes external data.
- **Analytics**: Analytics are meant to enrich data that is already present in the database. Extract hostnames from URLs, resolve hostnames, etc.
- **Exports**: The way Yeti disseminates data. Exports use templates to format and export data that may be consumed by proxy appliances, scripts, etc.

#### 1.4.1 Feeds

Feeds are Yeti’s main way of automatically collecting and parsing data into searchable objects. These can be Observables (URLs, IP addresses, hashes, etc.), Indicators (regular expressions, Yara rules), or Entities (Exploit Kits, Malware, etc.).

The most common use-case for Feeds is to quickly and regularly import large amounts of Observables or Indicatos from various sources.

##### Creating feeds

Creating a feed is pretty straightforward. Yeti will recursively load all core.feed.Feed objects defined in the /plugins/feed/ directory.

New feeds need to have default_values attribute which sets some of the necessary fields (see the class attributes for details).

- **frequency**: A timedelta field designating the delta between runs.
- **name**: The name of the feed.
- **source**: The URL which the feed will use to query data.
- **description**: A short description of the feed.

For example, in the :class:ZeusTrackerConfigs feed, the class is defined as follows:
**class ZeusTrackerConfigs(Feed):**

```python
defal t_values = {
    "frequency": timedelta(hours=1),
    "name": "ZeusTrackerConfigs",
    "description": "This feed shows the latest 50 ZeuS config URLs."
}
```

After that, two functions need to be created: `core.feed.Feed.update()` and `core.feed.Feed.analyze()`. The goal of the update function is to fetch the remote data, and the goal of analyze is to parse it and translate it into Observables, Indicators, or Entities that Yeti can store and later analyze.

ZeusTrackerConfigs’s update function looks like this:

```python
def update(self):
    for d in self.update_xml('item', ["title", "link", "description", "guid"])::
        self.analyze(d)
```

See how the `core.feed.Feed.update_xml()` helper is used. Since the source URL returns XML data, `update_xml` will know how to parse it and produce python dictionaries that can then be passed to the `core.feed.Feed.analyze()` function:

```python
def analyze(self, item):
    url_string = re.search(r"URL: (?P<url>\S+),", item['description']).group('url')
    context = {}
    date_string = re.search(r"\((?P<date>[0-9\-]+)\)\", item['title']).group('date')
    context['date_added'] = datetime.strptime(date_string, "%Y-%m-%d")
    context['status'] = re.search(r"status: (?P<status>[^,]+)\", item['description']).group('status')
    context['version'] = int(re.search(r"version: (?P<version>[^,]+)\", item['description']).group('version'))
    context['guid'] = item['guid']
    context['source'] = self.name
    try:
        context['md5'] = re.search(r"MD5 hash: (?P<md5>[a-f0-9]+)\", item['description']).group('md5')
    except AttributeError as e:
        pass
    try:
        n = Url.get_or_create(value=url_string)
        n.add_context(context)
        n.add_source("feed")
        n.tag(['zeus', 'c2', 'banker', 'crimeware', 'malware'])
    except ObservableValidationError as e:
        logging.error(e)
```

Here some pretty basic parsing using regular expressions is being done. Since the parsing is done using Python code, feeds can parse virtually any data in any format.

To avoid having to deal with duplicate elements, the use of `core.observables.Observable.get_or_create()`, `core.indicators.Indicator.get_or_create()` or `core.entities.Entity.get_or_create()` is recommended.

Context, tags, and sources can also be added to Observables. To do so, use the `core.observables.Observable.add_context()`, `core.observables.Observable.tag()`, or `core.observables.```
Observable.add_source() accordingly.

Testing feeds

Before pushing a feed into production, it is recommended to test them with the simple script tests/testfeeds.py:

```bash
$ python testfeeds.py ZeusTrackerConfigs
Running ZeusTrackerConfigs...
ZeusTrackerConfigs: success!
```

Any raised exception will be displayed.

How to add plugins

Plugins to work on data(observables)

- Plugins should be placed in: plugins/analytics/(public/private)/
- The observable types can be found in core/observables/
- Example: MacAddress, Hash, Url, Ip, Hash, Hostname, Email, Bitcoint, etc:

Can be imported using:

```python
from core.observables import Hash, Url, Hostname, Ip, MacAddress, Email
```

How to check observable type, for example Ip:

```python
if isinstance(observable, Ip):
```

How to extract iocs/observables from text:

```python
from core.observables import Observable
observables = Observable.from_string(text)
```

How to access config data:

```python
from core.config.config import yeti_config
example: yeti_config.redis.host
```

Extend web api

Place your .py, use use redis_api.py as example in:

```python
core/web/api/redis_api.py
```

Add import and register to: core/web/api/api.py:

```python
from core.web.api.redis_api import ManageRedisData
ManageRedisData.register(api)
# you can use render to render html or render_json, for raw responses
```
How to check if all services running correctly

- Service state should be **running** not loaded:

```
"systemctl status yeti_*"
```

Logging

All the logging by default can be found in `/var/log/syslog`:

```
tail -f /var/log/syslog
```

You can modify some of the systemd services to change **Celery** logging to file, if you need that:

```
-f PATH_TO_LOGFILE
```

Pushing into production

Once the feed is in its corresponding directory, it will show up in the URL `/dataflows`. Any errors raised by the feeds will show up here. Feeds can also be individually refreshed or toggled. A green row confirms that your feed is up and running!

1.4.2 Contributing

Want to contribute? Awesome! Please follow the instructions in contrib to make sure everything goes smoothly.

1.5 The API

This part details the various API endpoints that can be used within YETI.

API bindings:

- Python: pyeti

1.5.1 Observables, Indicators and Entities

Observables

**POST /api/observable/bulk**

Bulk-add observables

Bulk-add Observables from an array of strings.

Request JSON Object

- `string – observable, tags: [string]]` observables: Array of Strings representing observables (URLs, IPs, hostnames, etc.)

- `refang (boolean)` – If set, the observables will be refanged before being added to the database
DELETE /api/observable/ (id)
Deletes the corresponding entry from the database

Query Parameters
- id (ObjectId) – Element ID

Response JSON Object
- deleted (string) – The deleted element’s ObjectID

GET /api/observable/ (id)
Get details on a specific element

Query Parameters
- id (ObjectId) – Element ID

GET /api/observable/
List all corresponding entries in the database. Do not use on large datasets!

POST /api/observable/
Create a new Observable
Create a new Observable from the JSON object passed in the POST data.

Request JSON Object
- params (object) – JSON object containing fields to set
- refang (boolean) – If set, the observable will be refanged before being added to the database

DELETE /api/tag/ (id)
Deletes a Tag
Also remove the tag from any tagged elements.

Query Parameters
- id (ObjectId) – Element ID

Response JSON Object
- deleted (string) – The deleted element’s ObjectID

GET /api/tag/ (id)
Get details on a specific element

Query Parameters
- id (ObjectId) – Element ID

GET /api/tag/
List all corresponding entries in the database. Do not use on large datasets!

POST /api/tag/merge
Merge one or more tags
Merge one or more tags into a single tag. This is useful for replacing one or several tags with other tags.

Request JSON Object
- merge ([String]) – Array of Strings (tag names) representing tags to be merged.
- merge_into (String) – The tag to merge into
• **make_dict** *(boolean)* – Create a Tag dictionary out of this merge. In the future, tags included in the `merge` object will be automatically replaced by the tag specified in `merge_into`.

**POST /api/tag/**
Create a new element

Create a new element from the JSON object passed in the `POST` data.

Request JSON Object

• **params**(object) – JSON object containing fields to set

**POST /api/tag/** *(id)*
Create a new Tag

Create a new Tag

Edit an existing Tag according to the JSON object passed in the `POST` data. If the name of a tag is changed, it will repeat the change in all Observables associated with this tag.

Query Parameters

• **id**(ObjectID) – Element ID

Request JSON Object

• **params**(object) – JSON object containing fields to set

### Indicators

**DELETE /api/indicator/** *(id)*

Deletes the corresponding entry from the database

Query Parameters

• **id**(ObjectID) – Element ID

Response JSON Object

• **deleted**(string) – The deleted element’s ObjectID

**GET /api/indicator/** *(id)*

Get details on a specific element

Query Parameters

• **id**(ObjectID) – Element ID

**GET /api/indicator/**

List all corresponding entries in the database. **Do not use on large datasets!**

**POST /api/indicator/**
Create a new element

Create a new element from the JSON object passed in the `POST` data.

Request JSON Object

• **params**(object) – JSON object containing fields to set

**POST /api/indicator/** *(id)*
Modify an element

Edit an existing element according to the JSON object passed in the `POST` data.

Query Parameters
• **id** (*ObjectID*) – Element ID

Request JSON Object

• **params** (*object*) – JSON object containing fields to set

Entities

GET /api/entity/
List all corresponding entries in the database. **Do not use on large datasets!**

GET /api/entity/(*id*)
Get details on a specific element

Query Parameters

• **id** (*ObjectID*) – Element ID

DELETE /api/entity/(*id*)
Deletes the corresponding entry from the database

Query Parameters

• **id** (*ObjectID*) – Element ID

Response JSON Object

• **deleted** (*string*) – The deleted element’s *ObjectID*

POST /api/entity/
Create a new element

Create a new element from the JSON object passed in the *POST* data.

Request JSON Object

• **params** (*object*) – JSON object containing fields to set

POST /api/entity/(*id*)
Modify an element

Edit an existing element according to the JSON object passed in the *POST* data.

Query Parameters

• **id** (*ObjectID*) – Element ID

Request JSON Object

• **params** (*object*) – JSON object containing fields to set

Searching

POST /api/observablesearch/
Launches a simple search against the database

This endpoint is mostly used by paginators in Yeti.

Request JSON Object

• **params** (*object*) – JSON object specifying the *page*, *range* and *regex* variables.
  
  • **params.page** (*integer*) – Page or results to return (default: 1)
  
  • **params.range** (*integer*) – How many results to return (default: 50)
• **params.regex** (*boolean*) – Set to true if the arrays in **filter** are to be treated as regular expressions (default: false)

• **filter** (*object*) – JSON object specifying keys to be matched in the database. Each key must contain an array of OR-matched values.

**Request Headers**
- **Accept** – must be set to **application/json**
- **Content-Type** – must be set to **application/json**

**POST /api/indicatorsearch/**
Launches a simple search against the database
This endpoint is mostly used by paginators in Yeti.

**Request JSON Object**
- **params** (*object*) – JSON object specifying the page, range and regex variables.
- **params.page** (*integer*) – Page or results to return (default: 1)
- **params.range** (*integer*) – How many results to return (default: 50)
- **params.regex** (*boolean*) – Set to true if the arrays in **filter** are to be treated as regular expressions (default: false)
- **filter** (*object*) – JSON object specifying keys to be matched in the database. Each key must contain an array of OR-matched values.

**Request Headers**
- **Accept** – must be set to **application/json**
- **Content-Type** – must be set to **application/json**

**POST /api/entitysearch/**
Launches a simple search against the database
This endpoint is mostly used by paginators in Yeti.

**Request JSON Object**
- **params** (*object*) – JSON object specifying the page, range and regex variables.
- **params.page** (*integer*) – Page or results to return (default: 1)
- **params.range** (*integer*) – How many results to return (default: 50)
- **params.regex** (*boolean*) – Set to true if the arrays in **filter** are to be treated as regular expressions (default: false)
- **filter** (*object*) – JSON object specifying keys to be matched in the database. Each key must contain an array of OR-matched values.

**Request Headers**
- **Accept** – must be set to **application/json**
- **Content-Type** – must be set to **application/json**
1.5.2 Feeds and Exports

GET /api/export/(string: id)/content
Return export content

Returns a given export’s content.

Query Parameters

• **id** *(ObjectID)* – Export ID

Response Headers

• **X-Yeti-Export-MD5** – The MD5 hash of the exported content. Use it to check the export’s integrity

DELETE /api/export/(id)
Deletes the corresponding entry from the database

Query Parameters

• **id** *(ObjectID)* – Element ID

Response JSON Object

• **deleted** *(string)* – The deleted element’s ObjectID

GET /api/export/(id)
Get details on a specific element

Query Parameters

• **id** *(ObjectID)* – Element ID

GET /api/export/
List all corresponding entries in the database. **Do not use on large datasets!**

POST /api/export/
Create a new element

Create a new element from the JSON object passed in the POST data.

Request JSON Object

• **params** *(object)* – JSON object containing fields to set

POST /api/export/(id)
Modify an element

Edit an existing element according to the JSON object passed in the POST data.

Query Parameters

• **id** *(ObjectID)* – Element ID

Request JSON Object

• **params** *(object)* – JSON object containing fields to set

POST /api/export/(string: id)/refresh
Refresh an export

Manually executes an export if it is not already exporting.

Query Parameters

• **id** *(ObjectID)* – Export ID
Response JSON Object

- **id (ObjectID)** – The export’s ObjectID

**POST /api/export/ (string: id) /toggle**
Toggles an export

Toggles an export. A deactivated export will not execute when called (manually or scheduled)

Query Parameters

- **id (ObjectID)** – Export ID

Response JSON Object

- **id (ObjectID)** – The export’s ObjectID
- **status (boolean)** – The result of the toggle operation (true means the export has been enabled, false means it has been disabled)

**DELETE /api/exporttemplate/ (id)**
Deletes the corresponding entry from the database

Query Parameters

- **id (ObjectID)** – Element ID

Response JSON Object

- **deleted (string)** – The deleted element’s ObjectID

**GET /api/exporttemplate/ (id)**
Get details on a specific element

Query Parameters

- **id (ObjectID)** – Element ID

**GET /api/exporttemplate/**
List all corresponding entries in the database. Do not use on large datasets!

**POST /api/exporttemplate/**
Create a new element

Create a new element from the JSON object passed in the POST data.

Request JSON Object

- **params (object)** – JSON object containing fields to set

**POST /api/exporttemplate/ (id)**
Modify an element

Edit an existing element according to the JSON object passed in the POST data.

Query Parameters

- **id (ObjectID)** – Element ID

Request JSON Object

- **params (object)** – JSON object containing fields to set

**DELETE /api/feed/ (id)**
Deletes the corresponding entry from the database

Query Parameters

- **id (ObjectID)** – Element ID
**Response JSON Object**

- **deleted** *(string)* – The deleted element’s ObjectID

**GET /api/feed/ (id)**

Get details on a specific element

**Query Parameters**

- **id** *(ObjectID)* – Element ID

**GET /api/feed/**

List all corresponding entries in the database. **Do not use on large datasets!**

**POST /api/feed/**

Create a new element

Create a new element from the JSON object passed in the **POST** data.

**Request JSON Object**

- **params** *(object)* – JSON object containing fields to set

**POST /api/feed/ (id)**

Modify an element

Edit an existing element according to the JSON object passed in the **POST** data.

**Query Parameters**

- **id** *(ObjectID)* – Element ID

**Request JSON Object**

- **params** *(object)* – JSON object containing fields to set

**POST /api/feed/ (id) /refresh**

Runs a Feed

**Query Parameters**

- **id** *(ObjectID)* – Feed ID

**Response JSON Object**

- **id** *(ObjectID)* – Feed ID

**POST /api/feed/ (id) /toggle**

Toggles a Feed

Feeds can be individually disabled using this endpoint.

**Query Parameters**

- **id** *(ObjectID)* – Analytics ID

**Response JSON Object**

- **id** *(ObjectID)* – The Analytics’s ObjectID
- **status** *(boolean)* – The result of the toggle operation (true means the export has been enabled, false means it has been disabled)
1.5.3 Analysis

POST /api/analysis/match
Match observables against Yeti’s intelligence repository.

Takes an array of observables, expands them and tries to match them against specific indicators or known observables.

To “expand” an observable means to enrich the query. For instance, if the arrays of observables contains the URL http://google.com, the “expanded” observable array will also include the hostname google.com.

Request JSON Object

- observables ([string]) – An array of observables to be analyzed

Response JSON Object

- entities ([Entity]) – Related Entity objects
- known ([Observable]) – Observable objects that are already present in database
- matches ([Indicator]) – Indicators that matched observables
- matches[].observable (Observable) – The Observable object that matched the Indicator
- unknown (string) – Array of observable strings that didn’t match any Indicators and are unknown to Yeti

GET /api/analysis/
List all corresponding entries in the database. Do not use on large datasets!

GET /api/analysis/ (id)
Get details on a specific element

Query Parameters

- id (ObjectID) – Element ID

DELETE /api/analysis/ (id)
Deletes the corresponding entry from the database

Query Parameters

- id (ObjectID) – Element ID

Response JSON Object

- deleted (string) – The deleted element’s ObjectID

POST /api/analysis/
Create a new element

Create a new element from the JSON object passed in the POST data.

Request JSON Object

- params (object) – JSON object containing fields to set

POST /api/analysis/ (id)
Modify an element

Edit an existing element according to the JSON object passed in the POST data.

Query Parameters

- id (ObjectID) – Element ID
Request JSON Object

- `params` (*object*) – JSON object containing fields to set

**DELETE /api/analytics/oneshot/** (/id)

Deletes the corresponding entry from the database

Query Parameters

- `id` (**ObjectID**) – Element ID

Response JSON Object

- `deleted` (**string**) – The deleted element’s ObjectID

**GET /api/analytics/oneshot/** (/id)

Get details on a specific element

Query Parameters

- `id` (**ObjectID**) – Element ID

**POST /api/analytics/oneshot/**

Create a new element

Create a new element from the JSON object passed in the POST data.

Request JSON Object

- `params` (*object*) – JSON object containing fields to set

**POST /api/analytics/oneshot/** (/id)

Modify an element

Edit an existing element according to the JSON object passed in the POST data.

Query Parameters

- `id` (**ObjectID**) – Element ID

Request JSON Object

- `params` (*object*) – JSON object containing fields to set

**POST /api/analytics/oneshot/** (/id)/**/run

Runs a One-Shot Analytics

Asynchronously runs a One-Shot Analytics against a given observable. Returns an `AnalyticsResults` instance, which can then be used to fetch the analytics results

Query Parameters

- `id` (**ObjectID**) – Analytics ID

Form Parameters

- **ObjectID** id – Observable ID

Response JSON Object

- `object` – JSON object representing the `AnalyticsResults` instance

**POST /api/analytics/oneshot/** (/id)/**/toggle

Toggles a One-shot Analytics

One-Shot Analytics can be individually disabled using this endpoint.

Query Parameters
• **id** (*ObjectID*) – Analytics ID

**Response JSON Object**
• **id** (*ObjectID*) – The Analytics’s ObjectID
• **status (boolean)** – The result of the toggle operation (*true* means the export has been enabled, *false* means it has been disabled)

**DELETE /api/analytics/scheduled/ (id)**
Deletes the corresponding entry from the database

**Query Parameters**
• **id** (*ObjectID*) – Element ID

**Response JSON Object**
• **deleted (string)** – The deleted element’s ObjectID

**GET /api/analytics/scheduled/ (id)**
Get details on a specific element

**Query Parameters**
• **id** (*ObjectID*) – Element ID

**GET /api/analytics/scheduled/**
List all corresponding entries in the database. **Do not use on large datasets!**

**POST /api/analytics/scheduled/**
Create a new element

Create a new element from the JSON object passed in the POST data.

**Request JSON Object**
• **params (object)** – JSON object containing fields to set

**POST /api/analytics/scheduled/ (id)**
Modify an element

Edit an existing element according to the JSON object passed in the POST data.

**Query Parameters**
• **id** (*ObjectID*) – Element ID

**Request JSON Object**
• **params (object)** – JSON object containing fields to set

**POST /api/analytics/scheduled/ (id) /refresh**
Runs a Scheduled Analytics

**Query Parameters**
• **id** (*ObjectID*) – Scheduled Analytics ObjectID

**Response JSON Object**
• **id** (*ObjectID*) – ID of refreshed Scheduled Analytics

**POST /api/analytics/scheduled/ (id) /toggle**
Toggles a Scheduled Analytics

Scheduled Analytics can be individually disabled using this endpoint.

**Query Parameters**
• id (ObjectID) – Analytics ID

Response JSON Object

• id (ObjectID) – The Analytics’s ObjectID
• status (boolean) – The result of the toggle operation (true means the export has been enabled, false means it has been disabled)

1.5.4 Investigation

DELETE /api/investigation/(id)
Deletes the corresponding entry from the database

Query Parameters

• id (ObjectID) – Element ID

Response JSON Object

• deleted (string) – The deleted element’s ObjectID

GET /api/investigation/(id)
Get details on a specific element

Query Parameters

• id (ObjectID) – Element ID

GET /api/investigation/
List all corresponding entries in the database. Do not use on large datasets!

POST /api/investigation/
Create a new element

Create a new element from the JSON object passed in the POST data.

Request JSON Object

• params (object) – JSON object containing fields to set

POST /api/investigation/(id)
Modify an element

Edit an existing element according to the JSON object passed in the POST data.

Query Parameters

• id (ObjectID) – Element ID

Request JSON Object

• params (object) – JSON object containing fields to set

DELETE /api/neighbors/(id)
Deletes the corresponding entry from the database

Query Parameters

• id (ObjectID) – Element ID

Response JSON Object

• deleted (string) – The deleted element’s ObjectID

GET /api/neighbors/
List all corresponding entries in the database. Do not use on large datasets!
**POST /api/neighbors/**

Create a new element

Create a new element from the JSON object passed in the `POST` data.

**Request JSON Object**

- `params(object)` – JSON object containing fields to set

**POST /api/neighbors/ (id)**

Modify an element

Edit an existing element according to the JSON object passed in the `POST` data.

**Query Parameters**

- `id(Object ID)` – Element ID

**Request JSON Object**

- `params(object)` – JSON object containing fields to set
<table>
<thead>
<tr>
<th>URL Path</th>
<th>Method</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>/api/analysis</td>
<td>GET</td>
<td>41</td>
</tr>
<tr>
<td>/api/analysis/(id)</td>
<td>GET</td>
<td>41</td>
</tr>
<tr>
<td>/api/analytics/oneshot</td>
<td>GET</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/scheduled</td>
<td>GET</td>
<td>43</td>
</tr>
<tr>
<td>/api/analytics/scheduled/(id)</td>
<td>GET</td>
<td>43</td>
</tr>
<tr>
<td>/api/entity</td>
<td>GET</td>
<td>36</td>
</tr>
<tr>
<td>/api/entity/(id)</td>
<td>GET</td>
<td>36</td>
</tr>
<tr>
<td>/api/export</td>
<td>GET</td>
<td>38</td>
</tr>
<tr>
<td>/api/export/(id)</td>
<td>GET</td>
<td>38</td>
</tr>
<tr>
<td>/api/export/(string:id)/content</td>
<td>GET</td>
<td>38</td>
</tr>
<tr>
<td>/api/exporttemplate</td>
<td>GET</td>
<td>39</td>
</tr>
<tr>
<td>/api/exporttemplate/(id)</td>
<td>GET</td>
<td>39</td>
</tr>
<tr>
<td>/api/feed</td>
<td>GET</td>
<td>40</td>
</tr>
<tr>
<td>/api/feed/(id)</td>
<td>GET</td>
<td>40</td>
</tr>
<tr>
<td>/api/indicator</td>
<td>GET</td>
<td>35</td>
</tr>
<tr>
<td>/api/indicator/(id)</td>
<td>GET</td>
<td>35</td>
</tr>
<tr>
<td>/api/investigation</td>
<td>GET</td>
<td>44</td>
</tr>
<tr>
<td>/api/investigation/(id)</td>
<td>GET</td>
<td>44</td>
</tr>
<tr>
<td>/api/neighbors</td>
<td>GET</td>
<td>44</td>
</tr>
<tr>
<td>/api/neighbors/(id)</td>
<td>GET</td>
<td>44</td>
</tr>
<tr>
<td>/api/observable</td>
<td>GET</td>
<td>34</td>
</tr>
<tr>
<td>/api/observable/(id)</td>
<td>GET</td>
<td>34</td>
</tr>
<tr>
<td>/api/tag</td>
<td>GET</td>
<td>34</td>
</tr>
<tr>
<td>/api/tag/(id)</td>
<td>GET</td>
<td>34</td>
</tr>
<tr>
<td>/api/analysis</td>
<td>POST</td>
<td>41</td>
</tr>
<tr>
<td>/api/analysis/(id)</td>
<td>POST</td>
<td>41</td>
</tr>
<tr>
<td>/api/analysis/match</td>
<td>POST</td>
<td>41</td>
</tr>
<tr>
<td>/api/analytics/oneshot</td>
<td>POST</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/oneshot/(id)</td>
<td>POST</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/oneshot/(id)/run,</td>
<td>POST</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/oneshot/(id)/toggle,</td>
<td>POST</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/scheduled</td>
<td>POST</td>
<td>43</td>
</tr>
<tr>
<td>/api/analytics/scheduled/(id)</td>
<td>POST</td>
<td>43</td>
</tr>
<tr>
<td>/api/analytics/scheduled/(id)/refresh,</td>
<td>POST</td>
<td>43</td>
</tr>
<tr>
<td>/api/analytics/scheduled/(id)/toggle,</td>
<td>POST</td>
<td>43</td>
</tr>
<tr>
<td>/api/entity</td>
<td>POST</td>
<td>36</td>
</tr>
<tr>
<td>/api/entity/(id)</td>
<td>POST</td>
<td>36</td>
</tr>
<tr>
<td>/api/export</td>
<td>POST</td>
<td>38</td>
</tr>
<tr>
<td>/api/export/(id)</td>
<td>POST</td>
<td>38</td>
</tr>
<tr>
<td>/api/export/(string:id)/refresh,</td>
<td>POST</td>
<td>38</td>
</tr>
<tr>
<td>/api/export/(string:id)/toggle,</td>
<td>POST</td>
<td>39</td>
</tr>
<tr>
<td>/api/exporttemplate</td>
<td>POST</td>
<td>39</td>
</tr>
<tr>
<td>/api/exporttemplate/(id)</td>
<td>POST</td>
<td>39</td>
</tr>
<tr>
<td>/api/feed</td>
<td>POST</td>
<td>40</td>
</tr>
<tr>
<td>/api/feed/(id)</td>
<td>POST</td>
<td>40</td>
</tr>
<tr>
<td>/api/feed/(id)/refresh,</td>
<td>POST</td>
<td>40</td>
</tr>
<tr>
<td>/api/feed/(id)/toggle,</td>
<td>POST</td>
<td>40</td>
</tr>
<tr>
<td>/api/indicator</td>
<td>POST</td>
<td>35</td>
</tr>
<tr>
<td>/api/indicator/(id)</td>
<td>POST</td>
<td>35</td>
</tr>
<tr>
<td>/api/investigation</td>
<td>POST</td>
<td>44</td>
</tr>
<tr>
<td>/api/investigation/(id)</td>
<td>POST</td>
<td>44</td>
</tr>
<tr>
<td>/api/neighbors</td>
<td>POST</td>
<td>45</td>
</tr>
<tr>
<td>/api/neighbors/(id)</td>
<td>POST</td>
<td>45</td>
</tr>
<tr>
<td>/api/observable</td>
<td>POST</td>
<td>34</td>
</tr>
<tr>
<td>/api/observable/bulk,</td>
<td>POST</td>
<td>33</td>
</tr>
<tr>
<td>/api/observablesearch</td>
<td>POST</td>
<td>36</td>
</tr>
<tr>
<td>/api/tag</td>
<td>POST</td>
<td>35</td>
</tr>
<tr>
<td>/api/tag/(id)</td>
<td>POST</td>
<td>35</td>
</tr>
<tr>
<td>/api/tag/merge</td>
<td>POST</td>
<td>34</td>
</tr>
<tr>
<td>/api/analysis</td>
<td>DELETE</td>
<td>41</td>
</tr>
<tr>
<td>/api/analysis/(id)</td>
<td>DELETE</td>
<td>41</td>
</tr>
<tr>
<td>/api/analytics/oneshot</td>
<td>DELETE</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/oneshot/(id)</td>
<td>DELETE</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/oneshot/(id)/run,</td>
<td>DELETE</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/oneshot/(id)/toggle,</td>
<td>DELETE</td>
<td>42</td>
</tr>
<tr>
<td>/api/analytics/scheduled</td>
<td>DELETE</td>
<td>43</td>
</tr>
<tr>
<td>/api/analytics/scheduled/(id)</td>
<td>DELETE</td>
<td>43</td>
</tr>
<tr>
<td>/api/analytics/scheduled/(id)/refresh,</td>
<td>DELETE</td>
<td>43</td>
</tr>
<tr>
<td>/api/analytics/scheduled/(id)/toggle,</td>
<td>DELETE</td>
<td>43</td>
</tr>
<tr>
<td>/api/entity</td>
<td>DELETE</td>
<td>36</td>
</tr>
<tr>
<td>/api/export</td>
<td>DELETE</td>
<td>38</td>
</tr>
<tr>
<td>/api/exporttemplate</td>
<td>DELETE</td>
<td>39</td>
</tr>
<tr>
<td>/api/feed</td>
<td>DELETE</td>
<td>40</td>
</tr>
<tr>
<td>/api/indicator</td>
<td>DELETE</td>
<td>35</td>
</tr>
<tr>
<td>/api/investigation</td>
<td>DELETE</td>
<td>44</td>
</tr>
<tr>
<td>/api/neighbors</td>
<td>DELETE</td>
<td>44</td>
</tr>
<tr>
<td>/api/observable</td>
<td>DELETE</td>
<td>44</td>
</tr>
<tr>
<td>/api/tag</td>
<td>DELETE</td>
<td>34</td>
</tr>
<tr>
<td>/api/tag/(id)</td>
<td>DELETE</td>
<td>34</td>
</tr>
<tr>
<td>A</td>
<td>add_context() (core.observables.Observable method), 26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add_source() (core.observables.Observable method), 26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add_text() (core.observables.Observable class method), 26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>analyze() (core.feed.Feed method), 28</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>change_all_tags() (core.observables.Observable static method), 26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>clean() (core.observables.Observable method), 26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>context (core.observables.Observable attribute), 25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>created (core.observables.Observable attribute), 26</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>description (core.feed.Feed attribute), 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>description (core.observables.Observable attribute), 25</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>exclude_fields (core.observables.Observable attribute), 26</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>Feed (class in core.feed), 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feed.DoesNotExist, 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Feed.MultipleObjectsReturned, 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>frequency (core.feed.Feed attribute), 27</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>get_form() (core.observables.Observable class method), 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>get_tags() (core.observables.Observable method), 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>guess_type() (core.observables.Observable static method), 27</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>last_analyses (core.observables.Observable attribute), 26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>last_tagged (core.observables.Observable attribute), 26</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>name (core.feed.Feed attribute), 27</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>Observable (class in core.observables), 25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Observable.DoesNotExist, 26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Observable.MultipleObjectsReturned, 26</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>parse_commit() (core.feed.Feed method), 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>parse_xml() (core.feed.Feed method), 28</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>remove_context() (core.observables.Observable method), 27</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>source (core.feed.Feed attribute), 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sources (core.observables.Observable attribute), 25</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>tag() (core.observables.Observable method), 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td>tags (core.observables.Observable attribute), 25</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>update() (core.feed.Feed method), 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>update_csv() (core.feed.Feed method), 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>update_github() (core.feed.Feed method), 28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>update_json() (core.feed.Feed method), 29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>update_lines() (core.feed.Feed method), 29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>update_xml() (core.feed.Feed method), 29</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>value (core.observables.Observable attribute), 25</td>
<td></td>
</tr>
</tbody>
</table>