
IntelOwl

Matteo Lodi

Apr 18, 2024

CONTENTS

1	Introduction	1
1.1	Publications and media	1
2	Installation	3
2.1	Requirements	3
2.2	TL;DR	3
2.3	Deployment Components	4
2.4	Deployment Preparation	4
2.5	Run	7
2.6	After Deployment	7
2.7	Update and Rebuild	8
3	Advanced Configuration	13
3.1	ElasticSearch	14
3.2	Authentication options	15
3.3	OpenCTI	16
3.4	Cloud Support	16
3.5	Queues	18
3.6	Manual Usage	19
4	Usage	21
4.1	How to interact with IntelOwl	21
4.2	Plugins Framework	22
4.3	Investigations Framework	39
5	Advanced Usage	43
5.1	Organizations and User management	43
5.2	Optional Analyzers	45
5.3	Customize analyzer execution	45
5.4	Analyzers with special configuration	46
5.5	Notifications	47
6	Contribute	49
6.1	Rules	49
6.2	Code Style	50
6.3	How to start (Setup project and development instance)	50
6.4	How to add a new Plugin	52
6.5	How to add a new Playbook	57
6.6	How to modify a plugin	57
6.7	Modifying functionalities of the Certego packages	59
6.8	How to test the application	59

6.9	Create a pull request	61
6.10	Debug application problems	62
7	API docs	63

INTRODUCTION

IntelOwl was designed with the intent to help the community, in particular those researchers that can not afford commercial solutions, in the generation of threat intelligence data, in a simple, scalable and reliable way.

Main features:

- Provides enrichment of Threat Intel for malware as well as observables (IP, Domain, URL, hash, etc).
- This application is built to **scale out** and to **speed up the retrieval of threat info**.
- Thanks to the official libraries [pyintelowl](#) and [go-intelowl](#), it can be integrated easily in your stack of security tools to automate common jobs usually performed, for instance, by SOC analysts manually.
- Intel Owl is composed of:
 - **analyzers** that can be run to either retrieve data from external sources (like VirusTotal or AbuseIPDB) or to generate intel from internally available tools (like Yara or Oletools)
 - **connectors** that can be run to export data to external platforms (like MISP or OpenCTI)
 - **visualizers** that can be run to create custom visualizations of analyzers results
 - **playbooks** that are meant to make analysis easily repeatable
- API REST written in Django and Python 3.9.
- Built-in frontend client written in ReactJS, with [certego-ui](#): provides features such as dashboard, visualizations of analysis data, easy to use forms for requesting new analysis, etc.

1.1 Publications and media

To know more about the project and its growth over time, you may be interested in reading the following official blog posts and/or videos:

- [Certego Blog: v6 Announcement \(in Italian\)](#)
- [HackinBo 2023 Presentation \(in Italian\)](#)
- [Certego Blog: v.5.0.0 Announcement](#)
- [Youtube demo](#)
- [Certego Blog: v.4.0.0 Announcement](#)
- [HoneyNet Blog: v3.0.0 Announcement](#)
- [Intel Owl on Daily Swig](#)
- [HoneyNet Blog: v1.0.0 Announcement](#)

- [Certego Blog: First announcement](#)

Feel free to ask everything it comes to your mind about the project to the author: Matteo Lodi ([Twitter](#)).

We also have a dedicated twitter account for the project: [@intel_owl](#).

INSTALLATION

2.1 Requirements

The project leverages `docker compose` with a custom Bash script and you need to have the following packages installed in your machine:

- `docker` - v19.03.0+
- `docker-compose` - v2.3.4+

In some systems you could find pre-installed older versions. Please check this and install a supported version before attempting the installation. Otherwise it would fail. **Note:** We've added a new Bash script `initialize.sh` that will check compatibility with your system and attempt to install the required dependencies.

Note that macOS is shipped with an older version of bash. Please ensure to upgrade before running the script.

2.2 TL;DR

Obviously we strongly suggest reading through all the page to configure IntelOwl in the most appropriate way.

However, if you feel lazy, you could just install and test IntelOwl with the following steps. `docker` will be run with `sudo` if permissions/roles have not been set.

```
# clone the IntelOwl project repository
git clone https://github.com/intelowlproject/IntelOwl
cd IntelOwl/

# verify installed dependencies and start the app
./start prod up
# now the application is running on http://localhost:80

# create a super user
sudo docker exec -ti intelowl_uwsgi python3 manage.py createsuperuser

# now you can login with the created user from http://localhost:80/login

# Have fun!
```

2.3 Deployment Components

IntelOwl is composed of various different technologies, namely:

- React: Frontend, using [CRA](#) and [certego-ui](#)
- Django: Backend
- PostgreSQL: Database
- Redis: Message Broker
- Celery: Task Queue
- Nginx: Reverse proxy for the Django API and web assets.
- Uwsgi: Application Server
- Daphne: Asgi Server for WebSockets
- Elastic Search (*optional*): Auto-sync indexing of analysis' results.
- Kibana (*optional*): GUI for Elastic Search. We provide a saved configuration with dashboards and visualizations.
- Flower (*optional*): Celery Management Web Interface

All these components are managed via `docker compose`.

2.4 Deployment Preparation

- *Environment configuration (required)*
- *Database configuration (required)*
- *Web server configuration (optional)*
- *Analyzers configuration (optional)*

Open a terminal and execute below commands to construct new environment files from provided templates.

```
./initialize.sh
```

2.4.1 Environment configuration (required)

In the `docker/env_file_app`, configure different variables as explained below.

REQUIRED variables to run the image:

- `DB_HOST`, `DB_PORT`, `DB_USER`, `DB_PASSWORD`: PostgreSQL configuration (The DB credentials should match the ones in the `env_file_postgres`). If you like, you can configure the connection to an external PostgreSQL instance in the same variables. Then, to avoid to run PostgreSQL locally, please run IntelOwl with the option `--use-external-database`. Otherwise, `DB_HOST` must be `postgres` to have the app properly communicate with the PostgreSQL container.
- `DJANGO_SECRET`: random 50 chars key, must be unique. If you do not provide one, Intel Owl will automatically set a secret key and use the same for each run. The key is generated by `initialize.sh` script.

Strongly recommended variable to set:

- `INTELOWL_WEB_CLIENT_DOMAIN` (example: `localhost/mywebsite.com`): the web domain of your instance, this is used for generating links to analysis results.

Optional configuration:

- `OLD_JOBS_RETENTION_DAYS`: Database retention for analysis results (default: 14 days). Change this if you want to keep your old analysis longer in the database.

Other optional configuration to enable specific services / features

Configuration required to enable integration with Slack:

- `SLACK_TOKEN`: Slack token of your Slack application that will be used to send/receive notifications
- `DEFAULT_SLACK_CHANNEL`: ID of the Slack channel you want to post the message to

Configuration required to enable Re-Captcha in the Login and the Registration Page: In the `docker/env_file_app`:

- `USE_RECAPTCHA`: if you want to use recaptcha on your login
- `RECAPTCHA_SECRET_KEY`: your recaptcha secret key In the `frontend/public/env.js`:
- `RECAPTCHA_SITEKEY`: Recaptcha Key for your site

Configuration required to have InteOwl sending Emails (registration requests, mail verification, password reset/change, etc)

- `DEFAULT_FROM_EMAIL`: email address used for automated correspondence from the site manager (example: `noreply@mydomain.com`)
- `DEFAULT_EMAIL`: email address used for correspondence with users (example: `info@mydomain.com`)
- `EMAIL_HOST`: the host to use for sending email with SMTP
- `EMAIL_HOST_USER`: username to use for the SMTP server defined in `EMAIL_HOST`
- `EMAIL_HOST_PASSWORD`: password to use for the SMTP server defined in `EMAIL_HOST`. This setting is used in conjunction with `EMAIL_HOST_USER` when authenticating to the SMTP server.
- `EMAIL_PORT`: port to use for the SMTP server defined in `EMAIL_HOST`.
- `EMAIL_USE_TLS`: whether to use an explicit TLS (secure) connection when talking to the SMTP server, generally used on port 587.
- `EMAIL_USE_SSL`: whether to use an implicit TLS (secure) connection when talking to the SMTP server, generally used on port 465.

2.4.2 Database configuration (required if running PostgreSQL locally)

If you use a local PostgreSQL instance (this is the default), in the `env_file_postgres` you have to configure different variables as explained below.

Required variables:

- `POSTGRES_PASSWORD` (same as `DB_PASSWORD`)
- `POSTGRES_USER` (same as `DB_USER`)
- `POSTGRES_DB` (default: `intel_owl_db`)

2.4.3 Logrotate configuration (strongly recommended)

If you want to have your logs rotated correctly, we suggest you to add the configuration for the system Logrotate. To do that you can leverage the `initialize.sh` script. Otherwise, if you have skipped that part, you can manually install logrotate by launching the following script:

```
cd ./docker/scripts
./install_logrotate.sh
```

We decided to do not leverage Django Rotation Configuration because it caused problematic concurrency issues, leading to logs that are not rotated correctly and to apps that do not log anymore. Logrotate configuration is more stable.

2.4.4 Crontab configuration (recommended for advanced deployments)

We added few Crontab configurations that could be installed in the host machine at system level to solve some possible edge-case issues:

- **Memory leaks:** Once a week it is suggested to do a full restart of the application to clean-up the memory used by the application. Practical experience suggest us to do that to solve some recurrent memory issues in Celery. A cron called `application_restart` has been added for this purpose (it uses the absolute path of `start` script in the container). This cron assumes that you have executed IntelOwl with the parameters `--all_analyzers`. If you didn't, feel free to change the cron as you wish.

This configuration is optional but strongly recommended for people who want to have a production grade deployment. To install it you need to run the following script in each deployed server:

```
cd ./docker/scripts
./install_crontab.sh
```

2.4.5 Web server configuration (required for enabling HTTPS)

Intel Owl provides basic configuration for:

- Nginx (`configuration/nginx/http.conf`)
- Uwsgi (`configuration/intel_owl.ini`)

In case you enable HTTPS, remember to set the environment variable `HTTPS_ENABLED` as “enabled” to increment the security of the application.

There are 3 options to execute the web server:

- **HTTP only (default)**

The project would use the default deployment configuration and HTTP only.

- **HTTPS with your own certificate**

The project provides a template file to configure Nginx to serve HTTPS: `configuration/nginx/https.conf`.

You should change `ssl_certificate`, `ssl_certificate_key` and `server_name` in that file and put those required files in the specified locations.

Then you should call the `./start` script with the parameter `--https` to leverage the right Docker Compose file for HTTPS.

Plus, if you use [Flower](#), you should change in the `docker/flower.override.yml` the `flower_http.conf` with `flower_https.conf`.

- **HTTPS with Let's Encrypt**

We provide a specific docker-compose file that leverages [Traefik](#) to allow fast deployments of public-faced and HTTPS-enabled applications.

Before using it, you should configure the configuration file `docker/traefik.override.yml` by changing the email address and the hostname where the application is served. For a detailed explanation follow the official documentation: [Traefik doc](#).

After the configuration is done, you can add the option `--traefik` while executing `./start`

2.5 Run

You may invoke `$./start --help` to get help and usage info.

The CLI provides the primitives to correctly build, run or stop the containers for IntelOwl. Therefore,

Now that you have completed different configurations, starting the containers is as simple as invoking:

```
$ ./start prod up
```

You can add the docker options `-d` to run the application in the background.

Example:

```
./start prod up -- -d
```

2.5.1 Stop

To stop the application you have to:

- if executed without `-d` parameter: press CTRL+C
- if executed with `-d` parameter: `./start prod down`

2.5.2 Cleanup of database and application

This is a destructive operation but can be useful to start again the project from scratch

```
./start prod down -- -v
```

2.6 After Deployment

2.6.1 Users creation

You may want to run `docker exec -ti intelowl_uwsgi python3 manage.py createsuperuser` after first run to create a superuser. Then you can add other users directly from the Django Admin Interface after having logged with the superuser account. To manage users, organizations and their visibility please refer to this [section](#)

2.7 Update and Rebuild

2.7.1 Rebuilding the project / Creating custom docker build

If you make some code changes and you like to rebuild the project, follow these steps:

1. `./start test build -- --tag=<your_tag> .` to build the new docker image.
2. Add this new image tag in the `docker/test.override.yml` file.
3. Start the containers with `./start test up -- --build`.

2.7.2 Update to the most recent version

To update the project with the most recent available code you have to follow these steps:

```
$ cd <your_intel_owl_directory> # go into the project directory
$ git pull # pull new changes
$ ./start prod down # kill and destroy the currently running IntelOwl containers
$ ./start prod up # restart the IntelOwl application
```

```
PermissionError: [Errno 13] Permission denied: '/var/log/intel_owl/django/authentication.
↪log
```

just run this:

```
sudo chown -R www-data:www-data /var/lib/docker/volumes/intel_owl_generic_logs/_data/
↪django
```

and restart IntelOwl. It should solve the permissions problem.

Updating to >=6.0.0 from a 5.x.x version

IntelOwl v6 introduced some major changes regarding how the project is started. Before upgrading, some important things should be checked by the administrator:

- Docker Compose V1 support has been dropped project-wide. If you are still using a Compose version prior to v2.3.4, please [upgrade](#) to a newer version or install Docker Compose V2.
- IntelOwl is now started with the new Bash `start` script that has the same options as the old Python `start.py` script but is more manageable and has decreased the overall project dependencies. The `start.py` script has now been removed. Please use the new `start` script instead.
- The default message broker is now Redis. We have replaced Rabbit-MQ for Redis to allow support for Websockets in the application:
 - This change is transparent if you use our `start` script to run IntelOwl. That would spawn a Redis instance instead of a Rabbit-MQ one locally.
 - If you were using an external broker like AWS SQS or a managed Rabbit-MQ, they are still supported but we suggest to move to a Redis supported service to simplify the architecture (because Redis is now mandatory for Websockets)
- Support for multiple jobs with multiple playbooks has been removed. Every Observable or File in the request will be processed by a single playbook.
- We upgraded the base PostgreSQL image from version 12 to version 16. You have 2 choice:

- remove your actual database and start from scratch with a new one
- maintain your database and do not update Postgres. This could break the application at anytime because we do not support it anymore.
- if you want to keep your old DB, follow the migration procedure you can find below

Upgrading PostgreSQL is outside the scope of the IntelOwl project so we do not guarantee that everything will work as intended.

In case of doubt, please check the official PostgreSQL documentation.

Upgrade at your own risk.

The database migration procedure is as follows:

- You have IntelOwl version 5.x.x up and running
- Bring down the application (you can use the start script or manually concatenate your docker compose configuration)
- Go inside the docker folder `cd docker`
- Bring only the postgres 12 container up `docker run -d --name intelowl_postgres_12 -v intelowl_postgres_data:/var/lib/postgresql/data/ --env-file env_file_postgres library/postgres:12-alpine`
- Dump the entire database. You need the user and the database that you configured during startup for this `docker exec -t intelowl_postgres_12 pg_dump -U <POSTGRES_USER> -d <POSTGRES_DB> --no-owner > /tmp/dump_intelowl.sql`
- Stop the container `docker container stop intelowl_postgres_12`
- Remove the backup container `docker container rm intelowl_postgres_12`
- Remove the postgres volume `docker volume rm intelowl_postgres_data`
- Start the intermediary postgres 16 container `docker run -d --name intelowl_postgres_16 -v intelowl_postgres_data:/var/lib/postgresql/data/ --env-file env_file_postgres library/postgres:16-alpine`
- Add the data to the volume `cat /tmp/dump_intelowl.sql | docker exec -i intelowl_postgres_16 psql -U <POSTGRES_USER> -d <POSTGRES_DB>`
- Remove the intermediary container `docker container rm intelowl_postgres_16`
- Update IntelOwl to the latest version
- Bring up the application back again (you can use the start script or manually concatenate your docker compose configuration)

Updating to >=5.0.0 from a 4.x.x version

IntelOwl v5 introduced some major changes regarding how the plugins and their related configuration are managed in the application. Before upgrading, some important things should be checked by the administrator:

- A lot of database migrations will need to be applied. Just be patient few minutes once you install the new major release. If you get 500 status code errors in the GUI, just wait few minutes and then refresh the page.
- We moved away from the old big `analyzer_config.json` which was storing all the base configuration of the Analyzers to a database model (we did the same for all the other plugins types too). This allows us to manage plugins creation/modification/deletion in a more reliable manner and via the Django Admin Interface. If you have created custom plugins and changed those `<plugins>_config.json` file manually, you would need to

re-create those custom plugins again from the Django Admin Interface. To do that please follow the [related new documentation](#)

- We have REMOVED all the analyzers that we deprecated during the v4 releases cycle. Please substitute them with their respective new names, in case they have a replacement.
 - REMOVED Pulsedive_Active_IOC analyzer. Please substitute it with the new Pulsedive analyzer.
 - REMOVED Fortiguard analyzer because endpoint does not work anymore. No substitute.
 - REMOVED Rendertron analyzer not working as intended. No substitute.
 - REMOVED ThreatMiner, SecurityTrails and Robtex various analyzers and substituted with new versions.
 - REMOVED Doc_Info_Experimental. Its functionality (XLM Macro parsing) is moved to Doc_Info
 - REMOVED Strings_Info_Classic. Please use Strings_Info
 - REMOVED Strings_Info_ML. Please use Strings_Info and set the parameter rank_strings to True
 - REMOVED all Yara_Scan_<repo> analyzers. They all went merged in the single Yara analyzer
 - REMOVED Darksearch_Query analyzer because the service does not exist anymore. No substitute.
 - REMOVED UnpacMe_EXE_Unpacker. Please use UnpacMe
 - REMOVED BoxJS_Scan_JavaScript. Please use BoxJS
 - REMOVED all Anomali_Threatstream_<option> analyzers. Now we have a single Anomali_Threatstream analyzer. Use the parameters to select the specific API you need.

Updating to >=5.0.0 from a 3.x.x version

This is not supported. Please perform a major upgrade once at a time.

Updating to >=4.0.0 from a 3.x.x version

IntelOwl v4 introduced some major changes regarding the permission management, allowing an easier way to manage users and visibility. But that did break the previous available DB. So, to migrate to the new major version you would need to delete your DB. To do that, you would need to delete your volumes and start the application from scratch.

```
python3 start.py prod down -v
```

Please be aware that, while this can be an important effort to manage, the v4 IntelOwl provides an easier way to add, invite and manage users from the application itself. See [the Organization section](#).

Updating to >=2.0.0 from a 1.x.x version

Users upgrading from previous versions need to manually move `env_file_app`, `env_file_postgres` and `env_file_integrations` files under the new `docker` directory.

Updating to >v1.3.x from any prior version

If you are updating to >v1.3.0 from any prior version, you need to execute a helper script so that the old data present in the database doesn't break.

1. Follow the above updation steps, once the docker containers are up and running execute the following in a new terminal

```
docker exec -ti intelowl_uwsgi bash
```

to get a shell session inside the IntelOwl's container.

2. Now just copy and paste the below command into this new session,

```
curl https://gist.githubusercontent.com/Eshaan7/b111f887fa8b860c762aa38d99ec5482/  
↪raw/758517acf87f9b45bd22f06aee57251b1f3b1bbf/update_to_v1.3.0.py | python -
```

3. If you see "Update successful!", everything went fine and now you can enjoy the new features!

ADVANCED CONFIGURATION

This page includes details about some advanced features that Intel Owl provides which can be **optionally** configured by the administrator.

- *ElasticSearch*
 - *Kibana*
 - *Example Configuration*
 - *Business Intelligence*
- *Authentication options*
 - *OAuth support*
 - *LDAP*
 - *RADIUS*
- *OpenCTI support*
- *Cloud Support*
 - *AWS support*
 - * *Secrets*
 - * *SQS*
 - * *S3*
 - * *SES*
 - *Google Kubernetes Engine*
- *Queues*
 - *Multi Queue*
 - *Queue Customization*
 - *Queue monitoring*
- *Manual usage*

3.1 ElasticSearch

Right now only ElasticSearch v8 is supported.

3.1.1 DSL

IntelOwl makes use of [django-elasticsearch-dsl](#) to index Job results into elasticsearch. The `save` and `delete` operations are auto-synced so you always have the latest data in ES.

In the `env_file_app_template`, you'd see various elasticsearch related environment variables. The user should spin their own Elastic Search instance and configure these variables.

Kibana

Intel Owl provides a Kibana's "Saved Object" configuration (with example dashboard and visualizations). It can be downloaded from [here](#) and can be imported into Kibana by going to the "Saved Objects" panel (<http://localhost:5601/app/management/kibana/objects>).

Example Configuration

1. Setup [Elastic Search and Kibana](#) and say it is running in a docker service with name `elasticsearch` on port `9200` which is exposed to the shared docker network. (Alternatively, you can spin up a local Elastic Search instance, by appending `--elastic` to the `./start` command. Note that the local Elastic Search instance consumes large amount of memory, and hence having `>=16GB` is recommended.)
2. In the `env_file_app`, we set `ELASTICSEARCH_DSL_ENABLED` to `True` and `ELASTICSEARCH_DSL_HOST` to `elasticsearch:9200`.
3. Now start the docker containers and execute

```
docker exec -ti intelowl_uwsgi python manage.py search_index --rebuild
```

This will build and populate all existing job objects into the `jobs` index.

3.1.2 Business Intelligence

IntelOwl makes use of [elasticsearch-py](#) to store data that can be used for Business Intelligence purpose. Since plugin reports are deleted periodically, this feature allows to save indefinitely small amount of data to keep track of how analyzers perform and user usage. At the moment, the following information are sent to elastic:

- application name
- timestamp
- username
- configuration used
- process_time
- status
- end_time
- parameters

Documents are saved in the `ELEASTICSEARCH_BI_INDEX-%YEAR-%MONTH`, allowing to manage the retention accordingly. To activate this feature, it is necessary to set `ELASTICSEARCH_BI_ENABLED` to `True` in the `env_file_app` and `ELASTICSEARCH_BI_HOST` to `elasticsearch:9200` or your elasticsearch server.

An [index template](#) is created after the first bulk submission of reports. If you want to use kibana to visualize your data/make dashboard, you must create an index pattern: Go to Kibana -> Discover -> Stack Management -> Index Patterns -> search for your index and use as time field `timestamp`

3.2 Authentication options

IntelOwl provides support for some of the most common authentication methods:

- *Google OAuth2*
- *LDAP*
- *RADIUS*

3.2.1 Google OAuth2

The first step is to create a [Google Cloud Platform](#) project, and then [create OAuth credentials](#) for it.

It is important to add the correct callback in the “Authorized redirect URIs” section to allow the application to redirect properly after the successful login. Add this:

```
http://<localhost|yourdomain>/api/auth/google-callback
```

After that, specify the client ID and secret as `GOOGLE_CLIENT_ID` and `GOOGLE_CLIENT_SECRET` environment variables and restart IntelOwl to see the applied changes.

3.2.2 LDAP

IntelOwl leverages [Django-auth-ldap](#) to perform authentication via LDAP.

How to configure and enable LDAP on Intel Owl?

1. Change the values with your LDAP configuration inside `configuration/ldap_config.py`. This file is mounted as a docker volume, so you won't need to rebuild the image.
1. Once you have done that, you have to set the environment variable `LDAP_ENABLED` as `True` in the environment configuration file `env_file_app`. Finally, you can restart the application with `docker-compose up`

3.2.3 RADIUS Authentication

IntelOwl leverages [Django-radius](#) to perform authentication via RADIUS server.

How to configure and enable RADIUS authentication on Intel Owl?

1. Change the values with your RADIUS auth configuration inside `configuration/radius_config.py`. This file is mounted as a docker volume, so you won't need to rebuild the image.
1. Once you have done that, you have to set the environment variable `RADIUS_AUTH_ENABLED` as `True` in the environment configuration file `env_file_app`. Finally, you can restart the application with `docker-compose up`

3.3 OpenCTI

Like many other integrations that we have, we have an [Analyzer](#) and a [Connector](#) for the *OpenCTI* platform.

This allows the users to leverage these 2 popular open source projects and frameworks together.

So why we have a section here? This is because there are various compatibility problems with the [official PyCTI library](#).

We found out (see issues in [IntelOwl](#) and [PyCTI](#)) that, most of the times, it is required that the OpenCTI version of the server you are using and the pycti version installed in IntelOwl **must** match perfectly.

Because of that, we decided to provide to the users the chance to customize the version of PyCTI installed in IntelOwl based on the OpenCTI version that they are using.

To do that, you would need to leverage the option `--pycti-version` provided by the `./start` helper:

- check the default version that would be installed by checking the description of the option `--pycti-version` with `./start -h`
- if the default version is different from your OpenCTI server version, you need to rebuild the IntelOwl Image with `./start test build --pycti-version <your_version>`
- then restart the project `./start test up -- --build`
- enjoy

3.4 Cloud Support

3.4.1 AWS support

We have support for several AWS services.

You can customize the AWS Region location of you services by changing the environment variable `AWS_REGION`. Default is `eu-central-1`

You have to add some credentials for AWS: if you have IntelOwl deployed on the AWS infrastructure, you can use IAM credentials: to allow that just set `AWS_IAM_ACCESS` to `True`. If that is not the case, you have to set both `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`

S3

If you prefer to use S3 to store the analyzed samples, instead of the local storage, you can do it.

First, you need to configure the environment variable `LOCAL_STORAGE` to `False` to enable it and set `AWS_STORAGE_BUCKET_NAME` to the AWS bucket you want to use.

Then you need to configure permission access to the chosen S3 bucket.

Message Broker

IntelOwl at the moment supports 3 different message brokers:

- Redis (default)
- RabbitMQ
- Aws SQS

The default broker, if nothing is specified, is Redis.

To use RabbitMQ, you must use the option `--rabbitmq` when launching IntelOwl with the `./start` script.

To use Aws SQS, you must use the option `--sqs` when launching IntelOwl with the `./start` script. In that case, you should create new SQS queues in AWS called `intelowl-<environment>-<queue_name>` and give your instances on AWS the proper permissions to access it. Moreover, you must populate the `AWS_USER_NUMBER`. This is required to connect in the right way to the selected SQS queues. Only FIFO queues are supported.

If you want to use a remote message broker (like an ElasticCache or AmazonMQ instance), you must populate the `BROKER_URL` environment variable.

It is possible to use `task priority` inside IntelOwl: each User has default priority of 10, and robots users (like the Ingestors) have a priority of 7. You can customize these priorities inside Django Admin, in the **Authentication. User Profiles** section.

Websockets

Redis is used for two different functions:

- message broker
- websockets

For this reason, a Redis instance is **mandatory**. You can personalize IntelOwl in two different way:

- with a local Redis instance.

This is the default behaviour.

- With a remote Redis instance.

You must use the option `--use-external-redis` when launching IntelOwl with the `./start` script. Moreover, you need to populate the `WEBSOCKETS_URL` environment variable. If you are using Redis as a message broker too, remember to populate the `BROKER_URL` environment variable

RDS

If you like, you could use AWS RDS instead of PostgreSQL for your database. In that case, you should change the database required options accordingly: `DB_HOST`, `DB_PORT`, `DB_USER`, `DB_PASSWORD` and setup your machine to access the service.

If you have IntelOwl deployed on the AWS infrastructure, you can use IAM credentials to access the Postgres DB. To allow that just set `AWS_RDS_IAM_ROLE` to `True`. In this case `DB_PASSWORD` is not required anymore.

Moreover, to avoid to run PostgreSQL locally, you would need to use the option `--use-external-database` when launching IntelOwl with the `./start` script.

SES

If you like, you could use Amazon SES for sending automated emails (password resets / registration requests, etc). You need to configure the environment variable `AWS_SES` to `True` to enable it.

Secrets

You can use the “Secrets Manager” to store your credentials. In this way your secrets would be better protected.

Instead of adding the variables to the environment file, you should just add them with the same name on the AWS Secrets Manager and Intel Owl will fetch them transparently.

Obviously, you should have created and managed the permissions in AWS in advance and accordingly to your infrastructure requirements.

Also, you need to set the environment variable `AWS_SECRETS` to `True` to enable this mode.

NFS

You can use a `Network File System` for the `shared_files` that are downloaded runtime by IntelOwl (for example Yara rules).

To use this feature, you would need to add the address of the remote file system inside the `.env` file, and you would need to use the option `--nfs` when launching IntelOwl with the `./start` script.

3.4.2 Google Kubernetes Engine

Right now there is no official support for Kubernetes deployments.

But we have an active community. Please refer to the following blog post for an example on how to deploy IntelOwl on Google Kubernetes Engine:

[Deploying Intel-Owl on GKE by Mayank Malik.](#)

3.5 Queues

3.5.1 Multi Queue

IntelOwl provides an additional `multi-queue.override.yml` compose file allowing IntelOwl users to better scale with the performance of their own architecture.

If you want to leverage it, you should add the option `--multi-queue` when starting the project. Example:

```
./start prod up --multi-queue
```

This functionality is not enabled by default because this deployment would start 2 more containers so the resource consumption is higher. We suggest to use this option only when leveraging IntelOwl massively.

3.5.2 Queue Customization

It is possible to define new celery workers: each requires the addition of a new container in the docker-compose file, as shown in the `multi-queue.override.yml`.

Moreover IntelOwl requires that the name of the workers are provided in the docker-compose file. This is done through the environment variable `CELERY_QUEUES` inside the `uwsgi` container. Each queue must be separated using the character `,`, as shown in the [example](#).

One can customize what analyzer should use what queue by specifying so in the analyzer entry in the `analyzer_config.json` configuration file. If no queue(s) are provided, the default queue will be selected.

3.5.3 Queue monitoring

IntelOwl provides an additional `flower.override.yml` compose file allowing IntelOwl users to use [Flower](#) features to monitor and manage queues and tasks

If you want to leverage it, you should add the option `--flower` when starting the project. Example:

```
./start prod up --flower
```

The flower interface is available at port 5555: to set the credentials for its access, update the environment variables

```
FLOWER_USER
FLOWER_PWD
```

or change the `.htpasswd` file that is created in the `docker` directory in the `intelowl_flower` container.

3.6 Manual Usage

The `./start` script essentially acts as a wrapper over Docker Compose, performing additional checks. IntelOwl can still be started by using the standard `docker compose` command, but all the dependencies have to be manually installed by the user.

3.6.1 Options

The `--project-directory` and `-p` options are required to run the project. Default values set by `./start` script are “docker” and “intel_owl”, respectively.

The startup is based on [chaining](#) various Docker Compose YAML files using `-f` option. All Docker Compose files are stored in `docker/` directory of the project. The default compose file, named `default.yml`, requires configuration for an external database and message broker. In their absence, the `postgres.override.yml` and `rabbitmq.override.yml` files should be chained to the default one.

The command composed, considering what is said above (using `sudo`), is

```
sudo docker compose --project-directory docker -f docker/default.yml -f docker/postgres.  
↪override.yml -f docker/rabbitmq.override.yml -p intel_owl up
```

The other most common compose file that can be used is for the testing environment. The equivalent of running `./start test up` is adding the `test.override.yml` file, resulting in:

```
sudo docker compose --project-directory docker -f docker/default.yml -f docker/postgres.  
↪ override.yml -f docker/rabbitmq.override.yml -f docker/test.override.yml -p intel_owl_↪  
↪ up
```

All other options available in the `./start` script (`./start -h` to view them) essentially chain other compose file to `docker compose` command with corresponding filenames.

3.6.2 Optional Analyzer

IntelOwl includes integrations with [some analyzer](#) that are not enabled by default. These analyzers, stored under the `integrations/` directory, are packed within Docker Compose files. The `compose.yml` file has to be chained to include the analyzer. The additional `compose-test.yml` file has to be chained for testing environment.

This page includes the most important things to know and understand when using IntelOwl.

- *How to interact with IntelOwl*
- *Plugins Framework*
 - *Analyzers*
 - *Connectors*
 - *Pivots*
 - *Visualizers*
 - *Ingestors*
 - *Playbooks*
 - *Generic Plugin Creation, Configuration and Customization*
 - *Enabling or Disabling Plugins*
 - *Special Plugins operations*
 - *TLP Support*
- *Investigations Framework*
 - *Create and populate an Investigation*

4.1 How to interact with IntelOwl

Intel Owl main objective is to provide a single API interface to query in order to retrieve threat intelligence at scale.

There are multiple ways to interact with the Intel Owl APIs,

1. Web Interface
 - Built-in Web interface with dashboard, visualizations of analysis data, easy to use forms for requesting new analysis, tags management and more features
2. pyIntelOwl (CLI/SDK)
 - Official Python client that is available at: [PyIntelOwl](#),
 - Can be used as a library for your own python projects or...
 - directly via the command line interface.
3. goIntelOwl (CLI/SDK)

- Official GO client that is available at: [go-intelowl](#)

4.2 Plugins Framework

Plugins are the core modular components of IntelOwl that can be easily added, changed and customized. There are several types of plugins:

- *Analyzers*
- *Connectors*
- *Pivots*
- *Visualizers*
- *Ingestors*
- *Playbooks*

4.2.1 Analyzers

Analyzers are the most important plugins in IntelOwl. They allow to perform data extraction on the observables and/or files that you would like to analyze.

Analyzers list

The following is the list of the available analyzers you can run out-of-the-box. You can also navigate the same list via the

- Graphical Interface: once your application is up and running, go to the “Plugins” section
- `pyintelowl`: `$ pyintelowl get-analyzer-config`

File analyzers:

Internal tools

- `APKiD`: `APKiD` identifies many compilers, packers, obfuscators, and other weird stuff from an APK or DEX file.
- `BoxJS_Scan_Javascript`: `Box-JS` is a tool for studying JavaScript malware.
- `Capa_Info`: `Capa` detects capabilities in executable files
- `Capa_Info_Shellcode`: `Capa` detects capabilities in shellcode
- `ClamAV`: scan a file via the `ClamAV AntiVirus Engine`. IntelOwl automatically keep ClamAV updated with official and `unofficial` open source signatures
- `Doc_Info`: static document analysis with new features to analyze XLM macros, encrypted macros and more (combination of `Oletools` and `XLMMacroDeobfuscator`)
- `ELF_Info`: static ELF analysis with `pyelftools` and `telfhash`
- `File_Info`: static generic File analysis (hashes, magic and `exiftool`)
- `Floss`: `Mandiant Floss` Obfuscated String Solver in files
- `Hfinger`: create fingerprints of malware HTTPS requests using `Hfinger`

- **PE_Info**: static PE analysis with [pefile](#)
- **PEframe_Scan**: Perform static analysis on Portable Executable malware and malicious MS Office documents with [PeFrame](#)
- **PDF_Info**: static PDF analysis ([peepdf](#) + [pdfid](#))
- **Qiling_Linux**: [Qiling](#) qiling linux binary emulation.
- **Qiling_Linux_Shellcode**: [Qiling](#) qiling linux shellcode emulation.
- **Qiling_Windows**: [Qiling](#) qiling windows binary emulation.
- **Qiling_Windows_Shellcode**: [Qiling](#) qiling windows shellcode emulation.
- **Quark_Engine**: [Quark Engine](#) is an Obfuscation-Neglect Android Malware Scoring System.
- **Rtf_Info**: static RTF analysis ([Oletools](#))
- **Signature_Info**: PE signature extractor with [osslsigncode](#)
- **Speakeasy**: [Mandiant Speakeasy](#) binary emulation
- **SpeakEasy_Shellcode**: [Mandiant Speakeasy](#) shellcode emulation
- **Strings_Info**: Strings extraction. Leverages Mandiant's [Stringsifter](#)
- **Suricata**: Analyze PCAPs with open IDS signatures with [Suricata engine](#)
- **Thug_HTML_Info**: Perform hybrid dynamic/static analysis on a HTML file using [Thug low-interaction honey-client](#)
- **Xlm_Macro_Deobfuscator**: [XlmMacroDeobfuscator](#) deobfuscate xlm macros
- **Yara**: scan a file with
 - [ATM malware yara rules](#)
 - [bartblaze yara rules](#)
 - [community yara rules](#)
 - [StrangerealIntel](#)
 - [Neo23x0 yara rules](#)
 - [Intezer yara rules](#)
 - [Inquest yara rules](#)
 - [McAfee yara rules](#)
 - [Samir Threat Hunting yara rules](#)
 - [Stratosphere yara rules](#)
 - [Mandiant yara rules](#)
 - [ReversingLabs yara rules](#)
 - [YARAify rules](#)
 - [SIFalcon rules](#)
 - [Elastic rules](#)
 - [JPCERTCC Yara rules](#)
 - [HuntressLab Yara rules](#)
 - [elceef Yara Rules](#)

- [dr4k0nia Yara rules](#)
- [Facebook Yara rules](#)
- [edelucia Yara rules](#)
- [LOLDrivers Yara Rules](#)
- your own added signatures. See [Advanced-Usage](#) for more details.

External services

- **CapeSandbox:** [CAPESandbox](#) automatically scans suspicious files using the CapeSandbox API. Analyzer works for private instances as well.
- **Cymru_Hash_Registry_Get_File:** Check if a particular file is known to be malware by [Team Cymru](#)
- **Cuckoo_Scan:** scan a file on Cuckoo (this analyzer is disabled by default. You have to change that flag in the [config](#) to use it)
- **DocGuard_Upload_File:** Analyze office files in seconds. [DocGuard](#).
- **Dragonfly_Emulation:** Emulate malware against [Dragonfly](#) sandbox by [Certego S.R.L.](#)
- **FileScan_Upload_File:** Upload your file to extract IoCs from executable files, documents and scripts via [FileScan.io API](#).
- **HashLookupServer_Get_File:** check if a md5 or sha1 is available in the database of [known file hosted by CIRCL](#)
- **HybridAnalysis_Get_File:** check file hash on [HybridAnalysis](#) sandbox reports
- **Intezer_Scan:** scan a file on [Intezer](#). Register for a free community account [here](#). With TLP CLEAR, in case the hash is not found, you would send the file to the service.
- **Malpedia_Scan:** scan a binary or a zip file (pwd:infected) against all the yara rules available in [Malpedia](#)
- **MalwareBazaar_Get_File:** Check if a particular malware sample is known to [MalwareBazaar](#)
- **MISPFIRST_Check_Hash:** check a file hash on the [FIRST MISP](#) instance
- **MISP_Check_Hash:** check a file hash on a MISP instance
- **MWDB_Scan:** [mwdblib](#) Retrieve malware file analysis from repository maintained by CERT Polska MWDB. With TLP CLEAR, in case the hash is not found, you would send the file to the service.
- **OTX_Check_Hash:** check file hash on [Alienvault OTX](#)
- **SublimeSecurity:** Analyze an Email with [Sublime Security](#) live flow
- **Triage_Scan:** leverage [Triage](#) sandbox environment to scan various files
- **UnpacMe:** [UnpacMe](#) is an automated malware unpacking service
- **Virushee_Scan:** Check file hash on [Virushee API](#). With TLP CLEAR, in case the hash is not found, you would send the file to the service.
- **VirusTotal_v3_File:** check the file hash on VirusTotal. With TLP CLEAR, in case the hash is not found, you would send the file to the service.
- **YARAify_File_Scan:** scan a file against public and non-public YARA and ClamAV signatures in [YARAify](#) public service
- **YARAify_File_Search:** scan an hash against [YARAify](#) database

- Zippy_scan : [Zippy](#): Fast method to classify text as AI or human-generated; takes in lzma,zlib,brotli as input based engines; ensemble being default.

Observable analyzers (ip, domain, url, hash)

Internal tools

- CheckDMARC: An SPF and DMARC DNS records validator for domains.
- DNStwist: Scan a url/domain to find potentially malicious permutations via dns fuzzing. [dnstwist repo](#)
- Thug_URL_Info: Perform hybrid dynamic/static analysis on a URL using [Thug low-interaction honeyclient](#)

External services

- AbuseIPDB: check if an ip was reported on [AbuseIPDB](#)
- Abusix: get abuse contacts of an IP address from [Abusix](#)
- BGP Ranking: [BGP-Ranking](#) provides a way to collect such malicious activities, aggregate the information per ASN and provide a ranking model to rank the ASN from the most malicious to the less malicious ASN.
- Anomali_Threatstream_PassiveDNS: Return information from passive dns of Anomali. On [Anomali Threatstream PassiveDNS Api](#).
- Auth0: scan an IP against the Auth0 API
- BinaryEdge: Details about an Host. List of recent events for the specified host, including details of exposed ports and services using [IP query](#) and return list of subdomains known from the target domains using [domain query](#)
- BitcoinAbuse : Check a BTC address against bitcoinabuse.com, a public database of BTC addresses used by hackers and criminals.
- Censys_Search: scan an IP address against [Censys View API](#)
- CheckPhish: [CheckPhish](#) can detect phishing and fraudulent sites.
- CIRCLPassiveDNS: scan an observable against the CIRCL Passive DNS DB
- CIRCLPassiveSSL: scan an observable against the CIRCL Passive SSL DB
- Classic_DNS: Retrieve current domain resolution with default DNS
- CloudFlare_DNS: Retrieve current domain resolution with CloudFlare DoH (DNS over HTTPS)
- CloudFlare_Malicious_Detector: Leverages CloudFlare DoH to check if a domain is related to malware
- Crowdsec: check if an IP was reported on [Crowdsec Smoke Dataset](#)
- Cymru_Hash_Registry_Get_Observable: Check if a particular hash is available in the malware hash registry of [Team Cymru](#)
- DNSDB: scan an observable against the [Passive DNS Farsight Database](#) (support both v1 and v2 versions)
- DNS0_EU: Retrieve current domain resolution with DNS0.eu DoH (DNS over HTTPS)
- DNS0_EU_Malicious_Detector: Check if a domain or an url is marked as malicious in DNS0.eu database ([Zero service](#))
- DNS0_names: Run advanced searches on billions of current and historical domain names. ([DNS0 /names](#))

- `DNS0_rrsets_data`: Query billions of current and historical DNS resource records sets. Performs right-hand side matching. ([DNS0 /rrsets](#))
- `DNS0_rrsets_name`: Query billions of current and historical DNS resource records sets. Performs left-hand side matching. ([DNS0 /rrsets](#))
- `DocGuard_Get`: check if an hash was analyzed on DocGuard. [DocGuard](#)
- `Feodo_Tracker`: [Feodo Tracker](#) offers various blocklists, helping network owners to protect their users from Dridex and Emotet/Heodo.
- `FileScan_Search`: Finds reports and uploaded files by various tokens, like hash, filename, verdict, IOCs etc via [FileScan.io API](#).
- `FireHol_IPList`: check if an IP is in [FireHol's IPList](#)
- `GoogleSafebrowsing`: Scan an observable against GoogleSafeBrowsing DB
- `GoogleWebRisk`: Scan an observable against WebRisk API (Commercial version of Google Safe Browsing). Check the [docs](#) to enable this properly
- `Google_DNS`: Retrieve current domain resolution with Google DoH (DNS over HTTPS)
- `GreedyBear`: scan an IP or a domain against the [GreedyBear API](#) (requires API key)
- `GreyNoise`: scan an IP against the [Greynoise API](#) (requires API key)
- `GreyNoiseCommunity`: scan an IP against the [Community Greynoise API](#) (requires API key))
- `Greynoise_Labs`: scan an IP against the [Greynoise API](#) (requires authentication token which can be obtained from cookies on Greynoise website after launching the playground from [here](#))
- `HashLookupServer_Get_Observable`: check if a md5 or sha1 is available in the database of [known file hosted by CIRCL](#)
- `HoneyDB_Get`: [HoneyDB](#) IP lookup service
- `HoneyDB_Scan_Twitter`: scan an IP against HoneyDB.io's Twitter Threat Feed
- `Hunter_How`: Scans IP and domain against [Hunter_How API](#).
- `Hunter_Io`: Scans a domain name and returns set of data about the organisation, the email address found and additional information about the people owning those email addresses.
- `HybridAnalysis_Get_Observable`: search an observable in the [HybridAnalysis](#) sandbox reports
- `IPQS_Fraud_And_Risk_Scoring`: Scan an Observable against [IPQualityscore](#)
- `InQuest_DFI`: Deep File Inspection by [InQuest Labs](#)
- `InQuest_IOCdb`: Indicators of Compromise Database by [InQuest Labs](#)
- `InQuest_REPdb`: Search in [InQuest Lab's](#) Reputation Database
- `IPapi`: Get information about IPs using [batch-endpoint](#) and DNS using [DNS-endpoint](#).
- `IPInfo`: Location Information about an IP
- `Ip2location`: [API Docs](#) IP2Location.io allows users to check IP address location in real time. (Supports both with or without key)
- `Intezer_Get`: check if an analysis related to a hash is available in [Intezer](#). Register for a free community account [here](#).
- `Koodous`: [koodous API](#) get information about android malware.
- `MalwareBazaar_Get_Observable`: Check if a particular malware hash is known to [MalwareBazaar](#)

- **MalwareBazaar_Google_Observable**: Check if a particular IP, domain or url is known to MalwareBazaar using google search
- **MaxMindGeoIP**: extract GeoIP info for an observable
- **MISP**: scan an observable on a MISP instance
- **MISPFIRST**: scan an observable on the FIRST MISP instance
- **Mmdb_server**: [Mmdb_server](#) mmdb-server is an open source fast API server to lookup IP addresses for their geographic location, AS number.
- **Mnemonic_PassiveDNS** : Look up a domain or IP using the [Mnemonic PassiveDNS public API](#).
- **MWDB_Get**: [mwdblib](#) Retrieve malware file analysis by hash from repository maintained by CERT Polska MWDB.
- **Netlas**: search an IP against [Netlas](#)
- **ONYPHE**: search an observable in [ONYPHE](#)
- **OpenCTI**: scan an observable on an [OpenCTI](#) instance
- **OTXQuery**: scan an observable on [Alienvault OTX](#)
- **Phishstats**: Search [PhishStats API](#) to determine if an IP/URL/domain is malicious.
- **Phishtank**: Search an url against [Phishtank API](#)
- **PhishingArmy**: Search an observable in the [PhishingArmy](#) blocklist
- **Pulsedive**: Scan indicators and retrieve results from [Pulsedive's API](#).
- **Quad9_DNS**: Retrieve current domain resolution with Quad9 DoH (DNS over HTTPS)
- **Quad9_Malicious_Detector**: Leverages Quad9 DoH to check if a domain is related to malware
- **Robtex**: scan a domain/IP against the Robtex Passive DNS DB
- **Securitytrails**: scan an IP/Domain against [Securitytrails API](#)
- **Shodan_Honeyscore**: scan an IP against [Shodan Honeyscore API](#)
- **Shodan_Search**: scan an IP against [Shodan Search API](#)
- **Spyse**: Scan domains, IPs, emails and CVEs using Spyse's API. Register [here](#).
- **SSAPINet**: get a screenshot of a web page using [screenshotapi.net](#) (external source); additional config options can be added to `extra_api_params` in the config.
- **Stalkphish**: Search [Stalkphish API](#) to retrieve information about a potential phishing site (IP/URL/domain/Generic).
- **Stratosphere_Blacklist**: Cross-reference an IP from blacklists maintained by [Stratosphere Labs](#)
- **TalosReputation**: check an IP reputation from [Talos](#)
- **ThreatFox**: search for an IOC in [ThreatFox's](#) database
- **Threatminer**: retrieve data from [Threatminer API](#)
- **TorNodesDanMeUk**: check if an IP is a Tor Node using a list of all Tor nodes provided by [dan.me.uk](#)
- **TorProject**: check if an IP is a Tor Exit Node
- **Triage_Search**: Search for reports of observables or upload from URL on triage cloud
- **Tranco**: Check if a domain is in the latest [Tranco](#) ranking top sites list
- **URLhaus**: Query a domain or URL against [URLhaus API](#).

- `UrlScan_Search`: Search an IP/domain/url/hash against [URLScan API](#)
- `UrlScan_Submit_Result`: Submit & retrieve result of an URL against [URLScan API](#)
- `Virushee_CheckHash`: Search for a previous analysis of a file by its hash (SHA256/SHA1/MD5) on [Virushee API](#).
- `VirusTotal_v3_Get_Observable`: search an observable in the VirusTotal DB
- `Whoisxmlapi`: Fetch WHOIS record data, of a domain name, an IP address, or an email address.
- `WhoIs_RipeDB_Search`: Fetch whois record data of an IP address from Ripe DB using their [search API](#) (no API key required)
- `XForceExchange`: scan an observable on [IBM X-Force Exchange](#)
- `YARAify_Search`: lookup a file hash in [Abuse.ch YARAify](#)
- `YETI` (Your Everyday Threat Intelligence): scan an observable on a [YETI](#) instance.
- `Zoomeye`: [Zoomeye](#) Cyberspace Search Engine recording information of devices, websites, services and components etc..
- `Validin`: [Validin](#) investigates historic and current data describing the structure and composition of the internet.
- `TweetFeed`: [TweetFeed](#) collects Indicators of Compromise (IOCs) shared by the infosec community at Twitter. Here you will find malicious URLs, domains, IPs, and SHA256/MD5 hashes.

Generic analyzers (email, phone number, etc.; anything really)

Some analyzers require details other than just IP, URL, Domain, etc. We classified them as generic Analyzers. Since the type of field is not known, there is a format for strings to be followed.

Internal tools

- `CyberChef`: Run a query on a [CyberChef server](#) using pre-defined or custom recipes.

External services

- `Anomali_Threatstream_Confidence`: Give max, average and minimum confidence of maliciousness for an observable. On [Anomali Threatstream Confidence API](#).
- `Anomali_Threatstream_Intelligence`: Search for threat intelligence information about an observable. On [Anomali Threatstream Intelligence API](#).
- `CRXcavator`: scans a chrome extension against [crxcavator.io](#)
- `Dehashed_Search`: Query any observable/keyword against <https://dehashed.com>'s search API.
- `EmailRep`: search an email address on [emailrep.io](#)
- `HaveIBeenPwned`: [HaveIBeenPwned](#) checks if an email address has been involved in a data breach
- `IntelX_Intelligent_Search`: [IntelligenceX](#) is a search engine and data archive. Fetches emails, urls, domains associated with an observable or a generic string.
- `IntelX_Phonebook`: [IntelligenceX](#) is a search engine and data archive. Fetches emails, urls, domains associated with an observable or a generic string.
- `IPQS_Fraud_And_Risk_Scoring`: Scan an Observable against [IPQualityscore](#)

- MISP: scan an observable on a MISP instance
- VirusTotal_v3_Intelligence_Search: Perform advanced queries with [VirusTotal Intelligence](#) (requires paid plan)
- WiGLE: Maps and database of 802.11 wireless networks, with statistics, submitted by wardrivers, netstumblers, and net huggers.
- YARAify_Generics: lookup a YARA rule (default), ClamAV rule, imphash, TLSH, telfhash or icon_dash in [YARAify](#)
- PhoneInfoga : [PhoneInfoga](#) is one of the most advanced tools to scan international phone numbers.

Optional analyzers

Some analyzers are [optional](#) and need to be enabled explicitly.

4.2.2 Connectors

Connectors are designed to run after every successful analysis which makes them suitable for automated threat-sharing. They support integration with other SIEM/SOAR projects, specifically aimed at Threat Sharing Platforms.

Connectors list

The following is the list of the available connectors. You can also navigate the same list via the

- Graphical Interface: once your application is up and running, go to the “Plugins” section
- [pyintelowl](#): `$ pyintelowl get-connector-config`

List of pre-built Connectors

- MISP: automatically creates an event on your MISP instance, linking the successful analysis on IntelOwl.
- OpenCTI: automatically creates an observable and a linked report on your OpenCTI instance, linking the the successful analysis on IntelOwl.
- YETI: YETI = Your Everyday Threat Intelligence. find or create observable on YETI, linking the successful analysis on IntelOwl.
- Slack: Send the analysis link to a Slack channel (useful for external notifications)

4.2.3 Pivots

With IntelOwl v5.2.0 we introduced the Pivot Plugin.

Pivots are designed to create a job from another job. This plugin allows the user to set certain conditions that trigger the execution of one or more subsequent jobs, strictly connected to the first one.

This is a “SOAR” feature that allows the users to connect multiple analysis together.

List of pre-built Pivots

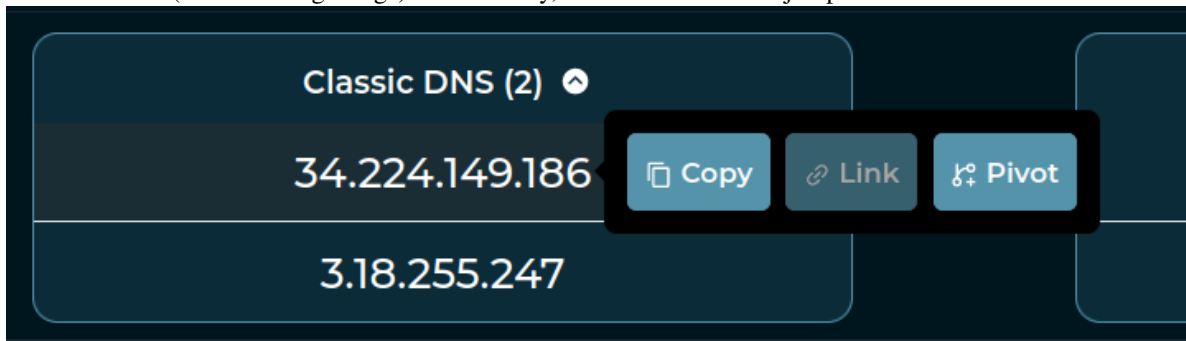
None

You can build your own custom Pivot with your custom logic with just few lines of code. See the [Contribute](#) section for more info.

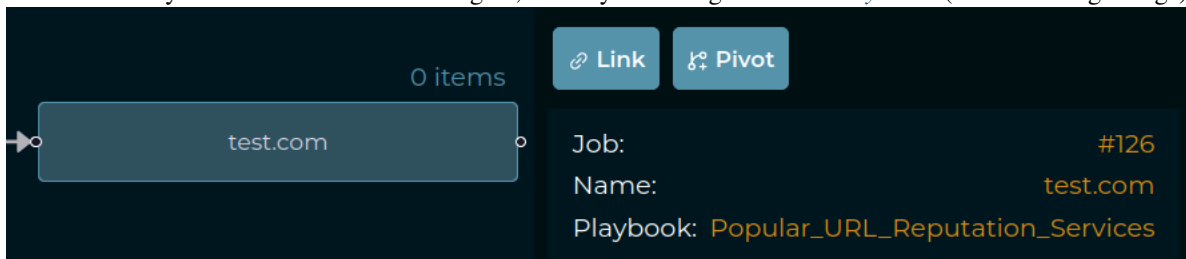
Creating Pivots from the GUI

From the GUI, the users can pivot in two ways:

- If a Job executed a *Visualizer*, it is possible to select a field extracted and analyze its value by clicking the “Pivot” button (see following image). In this way, the user is able to “jump” from one indicator to another.



- Starting from an already existing *Investigation*, it is possible to select a Job block and click the “Pivot” button to analyze the same observable again, usually choosing another *Playbook* (see following image)



In both cases, the user is redirected to the Scan Page that is precompiled with the observable selected. Then the user would be able to select the *Playbook* to execute in the new job.

Scan Observables

Month: 43 Total: 49 ⓘ

☒ observable (domain, IP, URL, HASH, etc...) ☐ file

Observable Value(s) *

[+ Add new value](#)

☒ Playbooks ☐ Analyzers/Connectors

Select Playbook

A playbook containing all free to use analyzers.

TLP ☒ CLEAR ☐ GREEN ☐ AMBER ☐ RED

use all analyzers

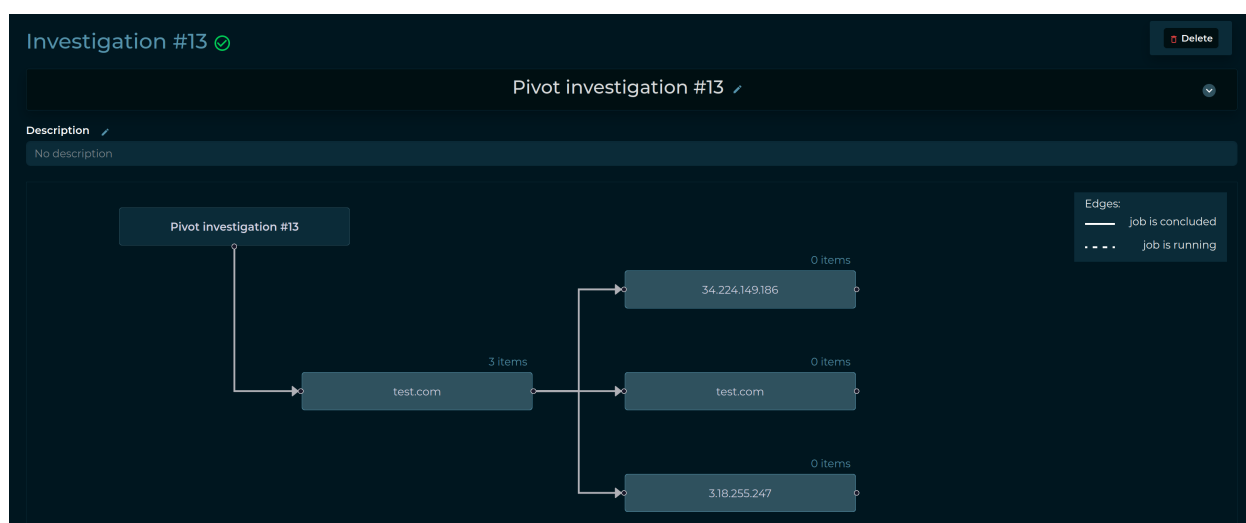
[Advanced settings](#)

[Start Scan](#)

After the new Job is started, a new *Investigation* will be created (if it does not already exist) and both the jobs will be added to the same Investigation.

In the following image you can find an example of an *Investigation* composed by 3 pivots generated manually:

- leveraging the first way to create a Pivot, the 2 Jobs that analyzed IP addresses have been generated from the first test\ .com Job
- leveraging the second way to create a Pivot, the second test\ .com analysis had been created with a different Playbook.



4.2.4 Visualizers

With IntelOwl v5 we introduced a new plugin type called **Visualizers**. You can leverage it as a framework to create *custom aggregated and simplified visualization of analyzer results*.

Visualizers are designed to run after the analyzers and the connectors. The visualizer adds logic after the computations, allowing to show the final result in a different way than merely the list of reports.

Visualizers can be executed only during Scans through the playbook that has been configured on the visualizer itself.

This framework is extremely powerful and allows every user to customize the GUI as they wish. But you know... with great power comes great responsibility. To fully leverage this framework, you would need to put some effort in place. You would need to understand which data is useful for you and then write few code lines that would create your own GUI. To simplify the process, take example from the pre-built visualizers listed below and follow the dedicated [documentation](#).

List of pre-built Visualizers

- **DNS**: displays the aggregation of every DNS analyzer report
- **Yara**: displays the aggregation of every matched rule by the Yara Analyzer
- **Domain_Reputation**: Visualizer for the Playbook “Popular_URL_Reputation_Services”
- **IP_Reputation**: Visualizer for the Playbook “Popular_IP_Reputation_Services”
- **Pivot**: Visualizer that can be used in a Playbook to show the Pivot execution result. See [Pivots](#) for more info.

4.2.5 Ingestors

With IntelOwl v5.1.0 we introduced the Ingestor Plugin.

Ingestors allow to automatically insert IOC streams from outside sources to IntelOwl itself. Each Ingestor must have a Playbook attached: this will allow to create a Job from every IOC retrieved.

Ingestors are system-wide and **disabled** by default, meaning that only the administrator are able to configure them and enable them. Ingestors can be *spammy* so be careful about enabling them.

A very powerful use case is to **combine Ingestors with Connectors** to automatically extract data from external sources, analyze them with IntelOwl and push them externally to another platform (like MISP or a SIEM)

List of pre-built Ingestors

- **ThreatFox**: Retrieves daily ioc from <https://threatfox.abuse.ch/> and analyze them.

4.2.6 Playbooks

Playbooks are designed to be easy to share sequence of running Plugins (Analyzers, Connectors, ...) on a particular kind of observable.

If you want to avoid to re-select/re-configure a particular combination of analyzers and connectors together every time, you should create a playbook out of it and use it instead. This is time saver.

This is a feature introduced since IntelOwl v4.1.0! Please provide feedback about it!

Playbooks List

The following is the list of the available pre-built playbooks. You can also navigate the same list via the

- Graphical Interface: once your application is up and running, go to the “Plugins” section
- `pyintelowl`: `$ pyintelowl get-playbook-config`

List of pre-built playbooks

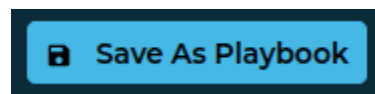
- `FREE_TO_USE_ANALYZERS`: A playbook containing all free to use analyzers.
- `Sample_Static_Analysis`: A playbook containing all analyzers that perform static analysis on files.
- `Popular_URL_Reputation_Services`: Collection of the most popular and free reputation analyzers for URLs and Domains
- `Popular_IP_Reputation_Services`: Collection of the most popular and free reputation analyzers for IP addresses
- `Dns`: A playbook containing all dns providers

Playbooks creation and customization

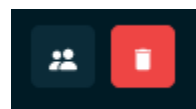
You can create new playbooks in different ways, based on the users you want to share them with:

If you want to share them to every user in IntelOwl, create them via the Django Admin interface at `/admin/playbooks_manager/playbookconfig/`.

If you want share them to yourself or your organization only, you need to leverage the “Save as Playbook” button that you can find on the top right of the Job Result Page. In this way, after you have done an analysis, you can save the configuration of the Plugins you executed for re-use with a single click.



The created Playbook would be available to yourself only. If you want either to share it with your organization or to delete it, you need to go to the “Plugins” section and enable it manually by clicking the dedicated button.



4.2.7 Generic Plugin Creation, Configuration and Customization

If you want to create completely new Plugins (not based on already existing python modules), please refer to the [Contribute](#) section. This is usually the case when you want to integrate IntelOwl with either a new tool or a new service.

On the contrary, if you would like to just customize the already existing plugins, this is the place.

SuperUser customization

If you are an IntelOwl superuser, you can create, modify, delete analyzers based on already existing modules by changing the configuration values inside the Django Admin interface at:

- for analyzers: `/admin/analyzers_manager/analyzerconfig/`.
- for connectors: `/admin/connectors_manager/connectorconfig/`.
- ...and so on for all the Plugin types.

The following are the most important fields that you can change without touching the source code:

- **Name:** Name of the analyzer
- **Description:** Description of the analyzer
- **Disabled:** you can choose to disable certain analyzers, then they won't appear in the dropdown list and won't run if requested.
- **Python Module:** Python path of the class that will be executed. This should not be changed most of the times.
- **Maximum TLP:** see *TLP Support*
- **Soft Time Limit:** this is the maximum time (in seconds) of execution for an analyzer. Once reached, the task will be killed (or managed in the code by a custom Exception). Default 300.
- **Routing Key:** this takes effects only when *multi-queue* is enabled. Choose which celery worker would execute the task: `local` (ideal for tasks that leverage local applications like Yara), `long` (ideal for long tasks) or `default` (ideal for simple webAPI-based analyzers).

For analyzers only:

- **Supported Filetypes:** can be populated as a list. If set, if you ask to analyze a file with a different mimetype from the ones you specified, it won't be executed
- **Not Supported Filetypes:** can be populated as a list. If set, if you ask to analyze a file with a mimetype from the ones you specified, it won't be executed
- **Observable Supported:** can be populated as a list. If set, if you ask to analyze an observable that is not in this list, it won't be executed. Valid values are: `ip`, `domain`, `url`, `hash`, `generic`.

For connectors only:

- **Run on Failure** (default: `true`): if they can be run even if the job has status `reported_with_fails`

For visualizers only:

- **Playbooks:** list of playbooks that trigger the specified visualizer execution.

Sometimes, it may happen that you would like to create a new analyzer very similar to an already existing one. Maybe you would like to just change the description and the default parameters. A helpful way to do that without having to copy/pasting the entire configuration, is to click on the analyzer that you want to copy, make the desired changes, and click the `save as new` button.

Other options can be added at the "Python module" level and not at the Plugin level. To do that, go to: `admin/api_app/pythonmodule/` and select the Python module used by the Plugin that you want to change. For example, the analyzer `AbuseIPDB` uses the Python module `abuseipdb.AbuseIPDB`.

AbuseIPDB

Name: AbuseIPDB

Description: check if an ip was reported on [AbuseIPDB](https://www.abuseipdb.com/)

☐ Disabled





Soft time limit: 30

Routing key: default

Type: Observable ▾

☐ Docker based

Maximum tlp: Amber ▾

Python module: abuseipdb.AbuseIPDB ▾    





Once there, you'll get this screen:





Change python module

abuseipdb.AbuseIPDB

Module: abuseipdb.AbuseIPDB

Base path: Observable Analyzer ▾

Update schedule: ----- ▾    

Health check schedule: ----- ▾    

There you can change the following values:

- **Update Schedule:** if the analyzer require some sort of update (local database, local rules, ...), you can specify the crontab schedule to update them.
- **Health Check Schedule:** if the analyzer has implemented a Health Check, you can specify the crontab sched-

ule to check whether the service works or not.

Parameters

Each Plugin could have one or more parameters available to be configured. These parameters allow the users to customize the Plugin behavior.

There are 2 types of Parameters:

- classic Parameters
- **Secrets:** these parameters usually manage sensitive data, like API keys.

To see the list of these parameters:

- You can view the “Plugin” Section in IntelOwl to have a complete and updated view of all the options available
- You can view the parameters by exploring the Django Admin Interface:
 - `admin/api_app/parameter/`
 - or at the very end of each Plugin configuration like `/admin/analyzers_manager/analyzerconfig/`

You can change the Plugin Parameters at 5 different levels:

- if you are an IntelOwl superuser, you can go in the Django Admin Interface and change the default values of the parameters for every plugin you like. This option would change the default behavior for every user in the platform.
- if you are either Owner or Admin of an org, you can customize the default values of the parameters for every member of the organization by leveraging the GUI in the “Organization Config” section. This overrides the previous option.
- if you are a normal user, you can customize the default values of the parameters for your analysis only by leveraging the GUI in the “Plugin config” section. This overrides the previous option.
- You can choose to provide runtime configuration when requesting an analysis that will override the previous options. This override is done only for the specific analysis. See Customize analyzer execution at time of request

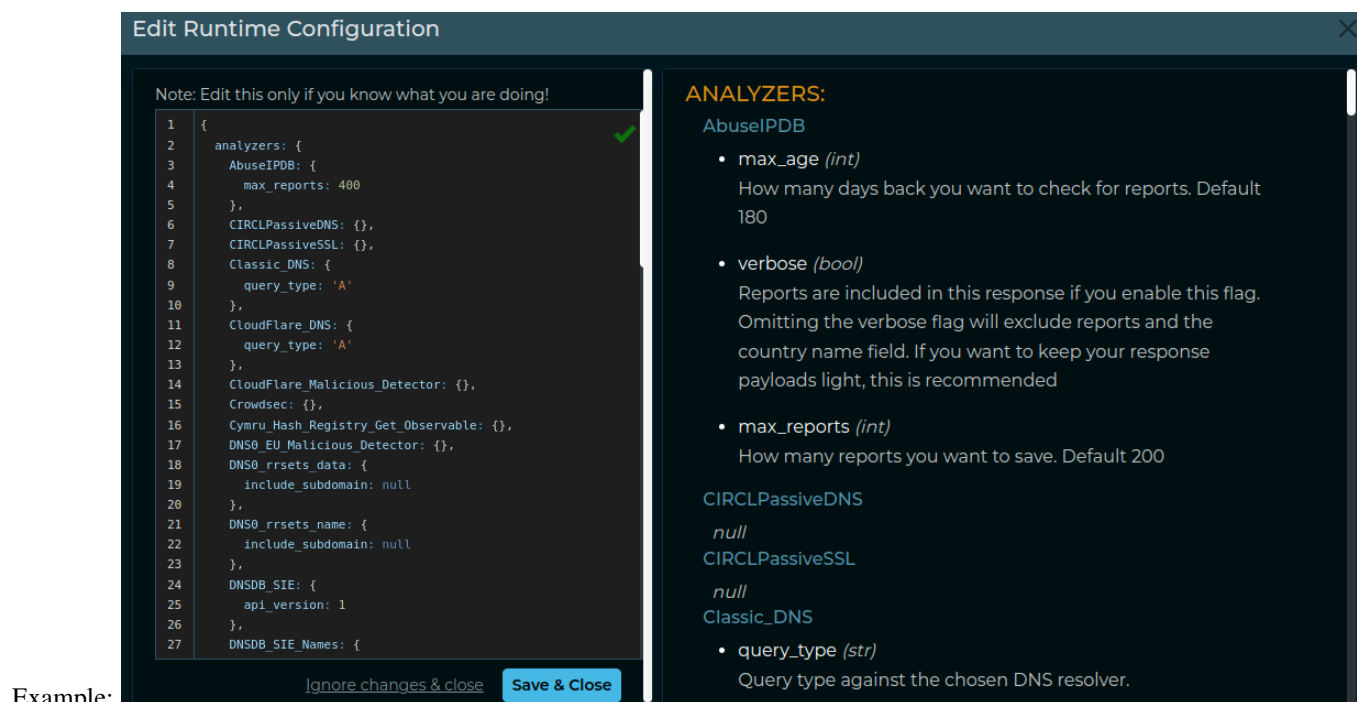
This is the order to define which values are used for the parameters, starting by the most important element:

- Runtime Configuration at Time of Request.
- Runtime Configuration of the Playbook (if a Playbook is used and the Runtime Configuration at Time of Request is empty)
- Plugin Configuration of the User
- Plugin Configuration of the Organization
- Default Plugin Configuration of the Parameter

If you are using the GUI, please remember that you can always check the Parameters before starting a “Scan” by clicking



at the “Runtime configuration” button.



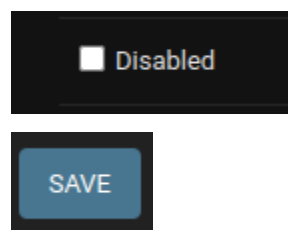
Example:

4.2.8 Enabling or Disabling Plugins

By default, each available plugin is configured as either disabled or not. The majority of them are enabled by default, while others may be disabled to avoid potential problems in the application usability for first time users.

Considering the impact that this change could have in the application, the GUI does not allow a normal user to enable/disable any plugin. On the contrary, users with specific privileges may change this configuration:

- Org Administrators may leverage the feature documented [here](#) to enable/disable plugins for their org. This can be helpful to control users' behavior.
- IntelOwl Superusers (full admin) can go to the Django Admin Interface and enable/disable them from there. This operation does overwrite the Org administrators configuration. To find the plugin to change, they'll need to first choose the section of its type ("ANALYZERS_MANAGER", "CONNECTORS_MANAGER", etc), then select the chosen plugin, change the flag on that option and save the plugin by pressing the right button.



4.2.9 Special Plugins operations

All plugins, i.e. analyzers and connectors, have `kill` and `retry` actions. In addition to that, all docker-based analyzers and connectors have a `healthcheck` action to check if their associated instances are up or not.

- **kill:**

Stop a plugin whose status is `running/pending`:

- GUI: Buttons on reports table on job result page.
- PyIntelOwl: `IntelOwl.kill_analyzer` and `IntelOwl.kill_connector` function.
- CLI: `$ pyintelowl jobs kill-analyzer <job_id> <analyzer_name>` and `$ pyintelowl jobs kill-connector <job_id> <connector_name>`
- API: `PATCH /api/job/{job_id}/{plugin_type}/{plugin_name}/kill` and `PATCH /api/job/{job_id}/connector/{connector_name}/kill`

- **retry:**

Retry a plugin whose status is `failed/killed`:

- GUI: Buttons on reports table on job result page.
- PyIntelOwl: `IntelOwl.retry_analyzer` and `IntelOwl.retry_connector` function,
- CLI: `$ pyintelowl jobs retry-analyzer <job_id> <analyzer_name>` and `$ pyintelowl jobs retry-connector <job_id> <connector_name>`
- API: `PATCH /api/job/{job_id}/{plugin_type}/{plugin_name}/retry`

- **healthcheck:**

Check if a plugin is able to connect to its provider:

- GUI: Buttons on every plugin table.
- PyIntelOwl: `IntelOwl.analyzer_healthcheck` and `IntelOwl.connector_healthcheck` methods.
- CLI: `$ pyintelowl analyzer-healthcheck <analyzer_name>` and `$ pyintelowl connector-healthcheck <connector_name>`
- API: `GET /api/{plugin_type}/{plugin_name}/healthcheck`

- **pull:**

Update a plugin with the newest rules/database:

- GUI: Buttons on every plugin table.
- API: `POST /api/{plugin_type}/{plugin_name}/pull`

4.2.10 TLP Support

The **Traffic Light Protocol (TLP)** is a standard that was created to facilitate greater sharing of potentially sensitive information and more effective collaboration.

IntelOwl is not a threat intel sharing platform, like the [MISP platform](#). However, IntelOwl is able to share analysis results to external platforms (via [Connectors](#)) and to send possible privacy related information to external services (via [Analyzers](#)).

This is why IntelOwl does support a customized version of the **Traffic Light Protocol (TLP)**: to allow the user to have a better knowledge of how their data are being shared.

Every *Analyzer* and *Connector* can be configured with a `maximum_tlp` value. Based on that value, IntelOwl understands if the specific plugin is allowed or not to run (e.g. if `maximum_tlp` is `GREEN`, it would run for analysis with TLPs `WHITE` and `GREEN` only)

These is how every available TLP value behaves once selected for an analysis execution:

1. **CLEAR**: no restriction (`WHITE` was replaced by `CLEAR` in TLP v2.0, but `WHITE` is supported for retrocompatibility)
2. **GREEN**: disable analyzers that could impact privacy
3. **AMBER** (default): disable analyzers that could impact privacy and limit view permissions to my group
4. **RED**: disable analyzers that could impact privacy, limit view permissions to my group and do not use any external service

4.3 Investigations Framework

Investigations are a new framework introduced in IntelOwl v6 with the goal to allow the users to connect the analysis they do with each other.

In this way the analysts can use IntelOwl as the starting point of their “Investigations”, register their findings, correlate the information found, and collaborate... all in a single place.

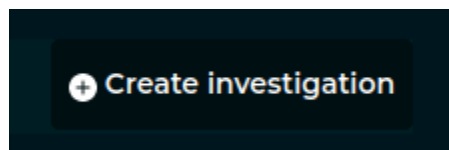
Things to know about the framework:

- an *Investigation* is a superset of IntelOwl Jobs. It can have attached one or more existing IntelOwl Jobs
- an *Investigation* contains a “description” section that can be changed and updated at anytime with new information from the analysts.
- modification to the Investigation (description, jobs, etc) can be done by every member of the Organization where the creator of the Investigation belongs. However only they creator can delete an Investigation.

4.3.1 Create and populate an investigation

Investigations are created in 2 ways:

- automatically:
 - if you scan multiple observables at the same time, a new investigation will be created by default and all the observables they will be automatically connected to the same investigation.
 - if you run a Job with a Playbook which contains a *Pivot* that triggers another Job, a new investigation will be created and both the Jobs will be added to the same investigation. See how you can create a new *Pivot* manually from the GUI.
- manually: by clicking on the button in the “History” section you can create an Investigation from scratch without any job attached (see following image)



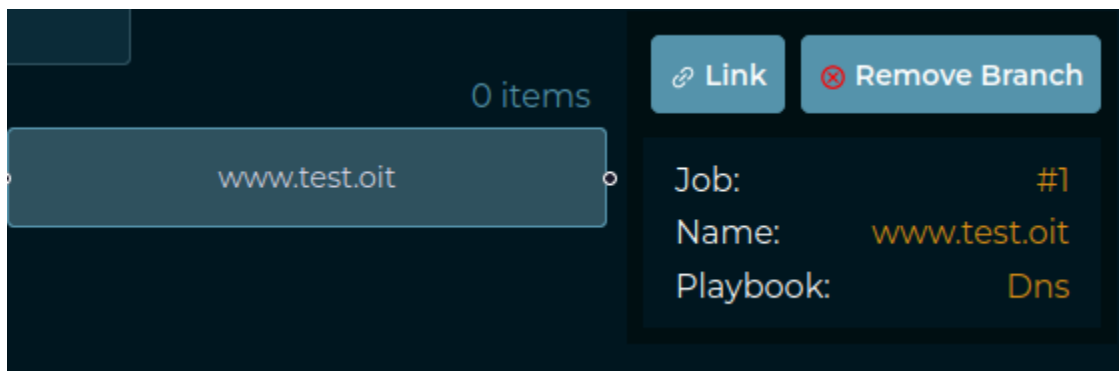
If you want to add a job to an Investigation, you should click to the root block of the Investigation (see following image):



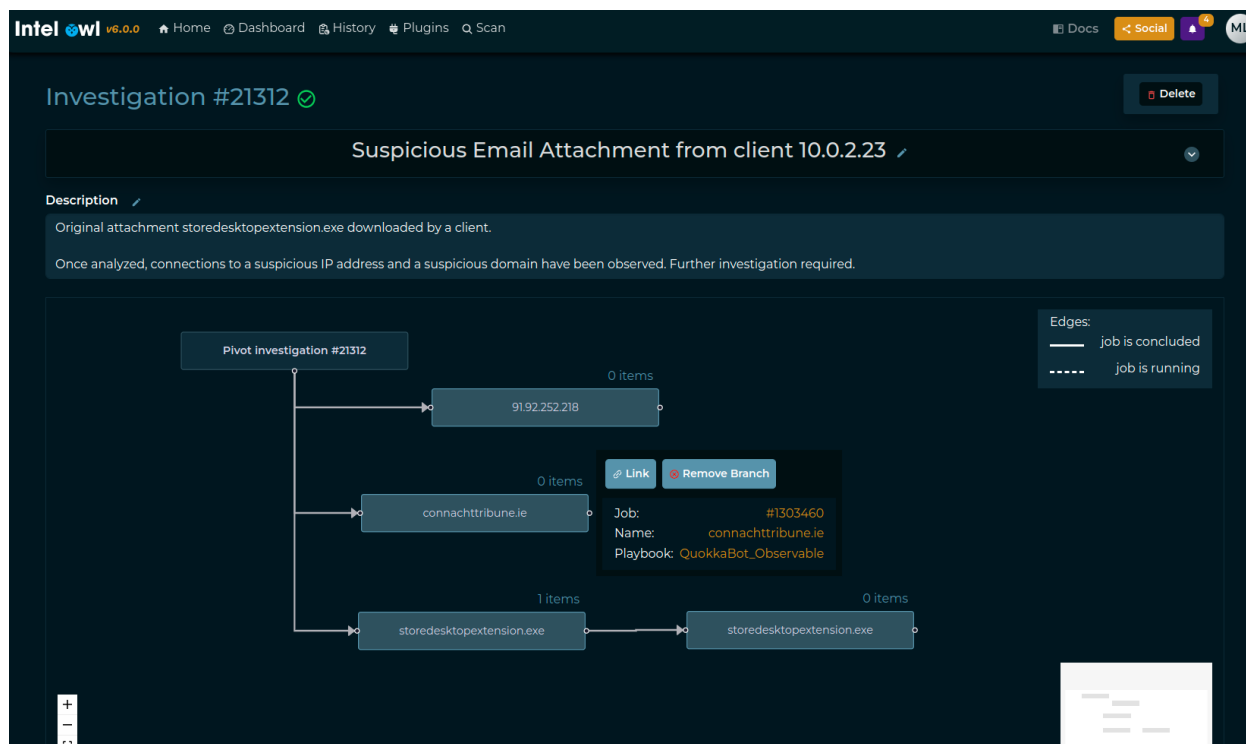
Once a job has been added, you'll have something like this:



If you want to remove a Job, you can click on the Job block and click “Remove branch”. On the contrary, if you just want to see Job Results, you can click in the “Link” button. (check next image)



4.3.2 Example output of a complex investigation



ADVANCED USAGE

This page includes details about some advanced features that Intel Owl provides which can be **optionally** enabled. Namely,

- *Advanced Usage*
 - *Organizations and User Management*
 - * *Multi Tenancy*
 - * *Registration*
 - *Optional Analyzers*
 - *Customize analyzer execution*
 - * *from the GUI*
 - * *from Pyintelowl*
 - * *CyberChef*
 - * *PhoneInfoga*
 - *Analyzers with special configuration*
 - *Notifications*

5.1 Organizations and User management

Starting from IntelOwl v4, a new “Organization” section is available on the GUI. This section substitute the previous permission management via Django Admin and aims to provide an easier way to manage users and visibility.

5.1.1 Multi Tenancy

Thanks to the “Organization” feature, IntelOwl can be used by multiple SOC's, companies, etc... very easily. Right now it works very simply: only users in the same organization can see analysis of one another. An user can belong to an organization only.

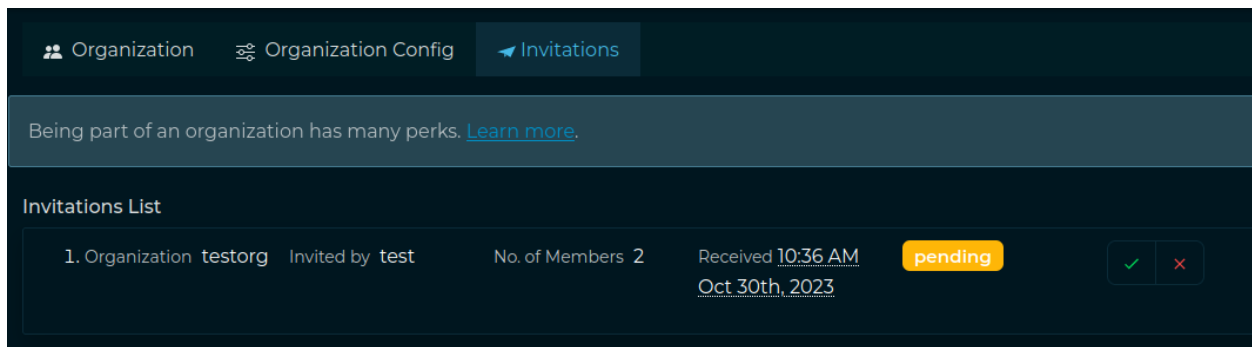
Manage organizations

You can create a new organization by going to the “Organization” section, available under the Dropdown menu you can find under the username.

Once you create an organization, you are the unique “Owner” of that organization. So you are the only one who can delete the organization and promote/demote/kick users. Another role, which is called “Admin”, can be set to a user (via the Django Admin interface only for now). Owners and admins share the following powers: they can manage invitations and the organization’s plugin configuration.

Accept Invites

Once an invite has sent, the invited user has to login, go to the “Organization” section and accept the invite there. Afterwards the Administrator will be able to see the user in his “Organization” section.

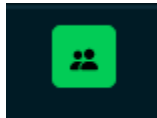


Plugins Params and Secrets

From IntelOwl v4.1.0, Plugin Parameters and Secrets can be defined at the organization level, in the dedicated section. This allows to share configurations between users of the same org while allowing complete multi-tenancy of the application. Only Owners and Admins of the organization can set, change and delete them.

Disable Plugins at Org level

The org admin can disable a specific plugin for all the users in a specific org. To do that, Org Admins needs to go in the “Plugins” section and click the button “Enabled for organization” of the plugin that they want to disable.



5.1.2 Registration

Since IntelOwl v4.2.0 we added a Registration Page that can be used to manage Registration requests when providing IntelOwl as a Service.

After a user registration has been made, an email is sent to the user to verify their email address. If necessary, there are buttons on the login page to resend the verification email and to reset the password.

Once the user has verified their email, they would be manually vetted before being allowed to use the IntelOwl platform. The registration requests would be handled in the Django Admin page by admins. If you have IntelOwl deployed on an AWS instance with an IAM role you can use the [SES](#) service.

To have the “Registration” page to work correctly, you must configure some variables before starting IntelOwl. See [Optional Environment Configuration](#)

In a development environment the emails that would be sent are written to the standard output.

Recaptcha configuration

The Registration Page contains a Recaptcha form from Google. By default, that Recaptcha is not configured and is not shown.

If your intention is to publish IntelOwl as a Service you should first remember to comply to the [AGPL License](#).

Then you need to add the generated Recaptcha Secret in the `RECAPTCHA_SECRET_KEY` value in the `env_file_app` file.

Afterwards you should configure the Recaptcha Key for your site and add that value in the `RECAPTCHA_SITEKEY` in the `frontend/public/env.js` file. In that case, you would need to [re-build](#) the application to have the changes properly reflected.

5.2 Optional Analyzers

Some analyzers which run in their own Docker containers are kept disabled by default. They are disabled by default to prevent accidentally starting too many containers and making your computer unresponsive.

To enable all the optional analyzers you can add the option `--all_analyzers` when starting the project. Example:

```
./start prod up --all_analyzers
```

Otherwise you can enable just one of the cited integration by using the related option. Example:

```
./start prod up --tor_analyzers
```

5.3 Customize analyzer execution

Some analyzers provide the chance to customize the performed analysis based on parameters that are different for each analyzer.

5.3.1 from the GUI



You can click on “**Runtime Configuration**” button in the “Scan” page and add the runtime configuration in the form of a dictionary. Example:

```
"VirusTotal_v3_File": {
  "force_active_scan_if_old": true
}
```

5.3.2 from Pyintelowl

While using `send_observable_analysis_request` or `send_file_analysis_request` endpoints, you can pass the parameter `runtime_configuration` with the optional values. Example:

```
runtime_configuration = {
    "Doc_Info": {
        "additional_passwords_to_check": ["passwd", "2020"]
    }
}
pyintelowl_client.send_file_analysis_request(..., runtime_configuration=runtime_
↪ configuration)
```

5.3.3 PhoneInfoga

PhoneInfoga provides several [Scanners](#) to extract as much information as possible from a given phone number. Those scanners may require authentication, so they're automatically skipped when no authentication credentials are found.

By default the scanner used is `local`. Go through this [guide](#) to initiate other required API keys related to this analyzer.

5.3.4 CyberChef

You can either use pre-defined recipes or create your own as explained [here](#).

To use a pre-defined recipe, set the `predefined_recipe_name` argument to the name of the recipe as defined [here](#). Else, leave the `predefined_recipe_name` argument empty and set the `custom_recipe` argument to the contents of the [recipe](#) you want to use.

Additionally, you can also (optionally) set the `output_type` argument.

Pre-defined recipes

- "to decimal": [{"op": "To Decimal", "args": ["Space", False]}]

5.4 Analyzers with special configuration

Some analyzers could require a special configuration:

- **GoogleWebRisk**: this analyzer needs a service account key with the Google Cloud credentials to work properly. You should follow the [official guide](#) for creating the key. Then you can populate the `secret_service_account_json` for that analyzer with the JSON of the service account file.
- **ClamAV**: this Docker-based analyzer uses `clamd` daemon as its scanner and is communicating with `clamscan` utility to scan files. The daemon requires 2 different configuration files: `clamd.conf` (daemon's config) and `freshclam.conf` (virus database updater's config). These files are mounted as docker volumes in `/integrations/malware_tools_analyzers/clamav` and hence, can be edited by the user as per needs, without restarting the application. Moreover ClamAV is integrated with unofficial open source signatures extracted with [Fangfrisch](#). The configuration file `fangfrisch.conf` is mounted in the same directory and can be customized on your wish. For instance, you should change it if you want to integrate open source signatures from [SecuriteInfo](#)
- **Suricata**: you can customize the behavior of Suricata:

- `/integrations/pcap_analyzers/config/suricata/rules`: here there are Suricata rules. You can change the `custom.rules` files to add your own rules at any time. Once you made this change, you need to either restart IntelOwl or (this is faster) run a new analysis with the Suricata analyzer and set the parameter `reload_rules` to `true`.
- `/integrations/pcap_analyzers/config/suricata/etc`: here there are Suricata configuration files. Change it based on your wish. Restart IntelOwl to see the changes applied.
- Yara:
 - You can customize both the `repositories` parameter and `private_repositories` secret to download and use different rules from the default that IntelOwl currently support.
 - * The `repositories` values is what will be used to actually run the analysis: if you have added private repositories, remember to add the url in `repositories` too!
 - You can add local rules inside the directory at `/opt/deploy/files_required/yara/YOUR_USERNAME/custom_rules/`. Please remember that these rules are not synced in a cluster deploy: for this reason is advised to upload them on GitHub and use the `repositories` or `private_repositories` attributes.
- `DNS0_rrsets_name` and `DNS0_rrsets_data` ([DNS0 API](#)):
 - Both these analyzers have a default parameter named `direction` that is used to dispatch the type of query to run.
 - * The value `right` for this parameter runs the query using `data API` parameter. Otherwise, if the parameter value is `left` it runs the query using the `name API` parameter.
 - This parameter should not be changed from default value.

5.5 Notifications

Since v4, IntelOwl integrated the notification system from the `certego_saas` package, allowing the admins to create notification that every user will be able to see.

The user would find the Notifications button on the top right of the page:

There the user can read notifications provided by either the administrators or the IntelOwl Maintainers.

As an Admin, if you want to add a notification to have it sent to all the users, you have to login to the Django Admin interface, go to the “Notifications” section and add it there. While adding a new notification, in the body section it is possible to even use HTML syntax, allowing to embed images, links, etc; in the `app_name` field, please remember to use `intelowl` as the app name.

Everytime a new release is installed, once the backend goes up it will automatically create a new notification, having as content the latest changes described in the [CHANGELOG.md](#), allowing the users to keep track of the changes inside intelowl itself.

CONTRIBUTE

There are a lot of different ways you could choose to contribute to the IntelOwl Project:

- main repository: [IntelOwl](#)
- official Python client: [pyintelowl](#).
- official GO client: [go-intelowl](#).
- official IntelOwl Site: [intelowlproject.github.io](#).
- honeypots project: [Greedybear](#)

6.1 Rules

Intel Owl welcomes contributors from anywhere and from any kind of education or skill level. We strive to create a community of developers that is welcoming, friendly and right.

For this reason it is important to follow some easy rules based on a simple but important concept: **Respect**.

- Before asking any questions regarding how the project works, please read *through all the documentation* and [install](#) the project on your own local machine to try it and understand how it basically works. This is a form of respect to the maintainers.
- DO NOT contact the maintainers with direct messages unless it is an urgent request. We don't have much time and cannot just answer to all the questions that we receive like "Guide me please! Help me understand how the project work". There is plenty of documentation and a lot of people in the community that can help you and would benefit from your questions. Share your problems and your knowledge. Please ask your questions in open channels (Github and Slack). This is a form of respect to the maintainers and to the community.
- Before starting to work on an issue, you need to get the approval of one of the maintainers. Therefore please ask to be assigned to an issue. If you do not that but you still raise a PR for that issue, your PR can be rejected. This is a form of respect for both the maintainers and the other contributors who could have already started to work on the same problem.
- When you ask to be assigned to an issue, it means that you are ready to work on it. When you get assigned, take the lock and then you disappear, you are not respecting the maintainers and the other contributors who could be able to work on that. So, after having been assigned, you have a week of time to deliver your first *draft* PR. After that time has passed without any notice, you will be unassigned.
- Once you started working on an issue and you have some work to share and discuss with us, please raise a draft PR early with incomplete changes. This way you can continue working on the same and we can track your progress and actively review and help. This is a form of respect to you and to the maintainers.
- When creating a PR, please read through the sections that you will find in the PR template and compile it appropriately. If you do not, your PR can be rejected. This is a form of respect to the maintainers.

6.2 Code Style

Keeping to a consistent code style throughout the project makes it easier to contribute and collaborate. We make use of `psf/black` and `isort` for code formatting and `flake8` for style guides.

6.3 How to start (Setup project and development instance)

This guide assumes that you have already performed the steps required to install the project. If not, please do it ([Installation Guide](#)).

Create a personal fork of the project on Github. Then, please create a new branch based on the **develop** branch that contains the most recent changes. This is mandatory.

```
git checkout -b myfeature develop
```

Then we strongly suggest to configure `pre-commit` to force linters on every commits you perform

```
# From the project directory
python3 -m venv venv
source venv/bin/activate
# from the project base directory
pip install pre-commit
pre-commit install

# create .env file for controlling repo_downloader.sh
# (to speed up image builds during development: it avoid downloading some repos)
cp docker/.env.start.test.template docker/.env.start.test

# set STAGE env variable to "local"
sed -i "s/STAGE=\"production\"/STAGE=\"local\"/g" docker/env_file_app
```

6.3.1 Backend

Now, you can execute IntelOwl in development mode by selecting the mode `test` while launching the startup script:

```
./start test up
```

Every time you perform a change, you should perform an operation to reflect the changes into the application:

- if you changed the python requirements, restart the application and re-build the images. This is the slowest process. You can always choose this way but it would waste a lot of time.

```
./start test down && ./start test up -- --build
```

- if you changed either analyzers, connectors, playbooks or anything that is executed asynchronously by the “celery” containers, you just need to restart the application because we leverage Docker bind volumes that will reflect the changes to the containers. This saves the time of the build

```
./start test down && ./start test up
```

- if you made changes to either the API or anything that is executed only by the application server, changes will be instantly reflected and you don't need to do anything. This is thanks to the Django Development server that is executed instead of uwsgi while using the `test` mode

NOTE about documentation:

If you made any changes to an existing model/serializer/view, please run the following command to generate a new version of the API schema and docs:

```
docker exec -it intelowl_uwsgi python manage.py spectacular --file docs/source/schema.  
→ yml && make html
```

6.3.2 Frontend

To start the frontend in “develop” mode, you can execute the startup npm script within the folder **frontend**:

```
cd frontend/  
# Install  
npm i  
# Start  
DANGEROUSLY_DISABLE_HOST_CHECK=true npm start  
# See https://create-react-app.dev/docs/proxying-api-requests-in-development/#invalid-host-header-errors-after-configuring-proxy for why we use that flag in development mode
```

Most of the time you would need to test the changes you made together with the backend. In that case, you would need to run the backend locally too:

```
./start prod up
```

Certego-UI

The IntelOwl Frontend is tightly linked to the **certego-ui** library. Most of the React components are imported from there. Because of this, it may happen that, during development, you would need to work on that library too. To install the **certego-ui** library, please take a look to [npm link](#) and remember to start **certego-ui** without installing peer dependencies (to avoid conflicts with IntelOwl dependencies):

```
git clone https://github.com/certego/certego-ui.git  
# change directory to the folder where you have the cloned the library  
cd certego-ui/  
# install, without peer deps (to use packages of IntelOwl)  
npm i --legacy-peer-deps  
# create link to the project (this will globally install this package)  
sudo npm link  
# compile the library  
npm start
```

Then, open another command line tab, create a link in the **frontend** to the **certego-ui** and re-install and re-start the frontend application (see previous section):

```
cd frontend/  
npm link @certego/certego-ui
```

This trick will allow you to see reflected every changes you make in the **certego-ui** directly in the running frontend application.

Example application

The `certego-ui` application comes with an example project that showcases the components that you can re-use and import to other projects, like IntelOwl:

```
# To have the Example application working correctly, be sure to have installed `certego-
↪ui` *without* the `--legacy-peer-deps` option and having it started in another command.
↪line
cd certego-ui/
npm i
npm start
# go to another tab
cd certego-ui/example/
npm i
npm start
```

6.4 How to add a new Plugin

IntelOwl was designed to ease the addition of new plugins. With a simple python script you can integrate your own engine or integrate an external service in a short time.

There are two possible cases:

1. You are creating an entirely new Plugin, meaning that you actually wrote python code
2. You are creating a new Configuration for some code that already exists.

If you are doing the step number 2, you can skip this paragraph.

First, you need to create the python code that will be actually executed. You can easily take other plugins as example to write this. Then, you have to create a `Python Module` model. You can do this in the `Django Admin` page: You have to specify which type of Plugin you wrote, and its python module. Again, you can use as an example an already configured `Python Module`.

Some `Python Module` requires to update some part of its code in a **schedule way**: for example Yara requires to update the rule repositories, QuarkEngine to update its database and so on. If the `Python Module` that you define need this type of behaviour, you have to configure two things:

- In the python code, you have to override a method called `update` and put the updating logic (see other plugins for examples) there.
- In the model class, you have to add the `update_schedule` (crontab syntax) that define when the update should be executed.

Some `Python Module` requires further check to see if the service provider is able to answer requests; for example if you have done too many requests, or the website is currently down for maintenance and so on. If the `Python Module` that you define need this type of behaviour, you have to configure two things:

- In the python code, you can override a method called `health_check` and put there the custom health check logic. As default, plugins will try to make an HTTP HEAD request to the configured url.
- In the model class, you have to add the `health_check_schedule` (crontab syntax) that define when the health check should be executed.

Press `Save` and `continue editing` to, at the moment, manually add the `Parameters` that the python code requires (the class attributes that you needed):

1. `*name`: Name of the parameter that will be dynamically added to the python class (if is a secret, in the python code a `_` will be prepended to the name)
2. `*type`: data type, `string`, `list`, `dict`, `integer`, `boolean`, `float`
3. `*description`
4. `*required`: `true` or `false`, meaning that a value is necessary to allow the run of the analyzer
5. `*is_secret`: `true` or `false`

At this point, you can follow the specific guide for each plugin

6.4.1 How to add a new Analyzer

You may want to look at a few existing examples to start to build a new one, such as:

- `shodan.py`, if you are creating an observable analyzer
- `malpedia_scan.py`, if you are creating a file analyzer
- `peframe.py`, if you are creating a *docker based analyzer*
- **Please note:** If the new analyzer that you are adding is free for the user to use, please add it in the `FREE_TO_USE_ANALYZERS` playbook. To do this you have to make a migration file; you can use `0026_add_mmdb_analyzer_free_to_use` as a template.

After having written the new python module, you have to remember to:

1. Put the module in the `file_analyzers` or `observable_analyzers` directory based on what it can analyze
2. Remember to use `_monkeypatch()` in its class to create automated tests for the new analyzer. This is a trick to have tests in the same class of its analyzer.
3. Create the configuration inside django admin in `Analyzers_manager/AnalyzerConfigs` (* = mandatory, ~ = mandatory on conditions)
 1. `*Name`: specific name of the configuration
 2. `*Python module`: `<module_name>.<class_name>`
 3. `*Description`: description of the configuration
 4. `*Routing key`: celery queue that will be used
 5. `*Soft_time_limit`: maximum time for the task execution
 6. `*Type`: `observable` or `file`
 7. `*Docker based`: if the analyzer run through a docker instance
 8. `*Maximum tlp`: maximum tlp to allow the run on the connector
 9. `~Observable supported`: required if `type` is `observable`
 10. `~Supported filetypes`: required if `type` is `file` and `not supported filetypes` is empty
 11. `Run hash`: if the analyzer supports hash as inputs
 12. `~Run hash type`: required if `run hash` is `True`
 13. `~Not supported filetypes`: required if `type` is `file` and `supported filetypes` is empty

Integrating a docker based analyzer

If the analyzer you wish to integrate doesn't exist as a public API or python package, it should be integrated with its own docker image which can be queried from the main Django app.

- It should follow the same design principle as the [other such existing integrations](#), unless there's very good reason not to.
- The dockerfile should be placed at `./integrations/<analyzer_name>/Dockerfile`.
- Two docker-compose files `compose.yml` for production and `compose-tests.yml` for testing should be placed under `./integrations/<analyzer_name>`.
- If your docker-image uses any environment variables, add them in the `docker/env_file_integrations_template`.
- Rest of the steps remain same as given under "How to add a new analyzer".

6.4.2 How to add a new Connector

You may want to look at a few existing examples to start to build a new one:

- [misp.py](#)
- [opencti.py](#)

After having written the new python module, you have to remember to:

1. Put the module in the `connectors` directory
2. Remember to use `_monkeypatch()` in its class to create automated tests for the new connector. This is a trick to have tests in the same class of its connector.
3. Create the configuration inside django admin in `Connectors_manager/ConnectorConfigs` (* = mandatory, ~ = mandatory on conditions)
 1. *Name: specific name of the configuration
 2. *Python module: `<module_name>.<class_name>`
 3. *Description: description of the configuration
 4. *Routing key: celery queue that will be used
 5. *Soft_time_limit: maximum time for the task execution
 6. *Maximum tlp: maximum tlp to allow the run on the connector
 7. *Run on failure: if the connector should be run even if the job fails

6.4.3 How to add a new Ingestor

1. Put the module in the `ingestors` directory
2. Remember to use `_monkeypatch()` in its class to create automated tests for the new ingestor. This is a trick to have tests in the same class of its ingestor.
3. Create the configuration inside django admin in `Ingestors_manager/IngestorConfigs` (* = mandatory, ~ = mandatory on conditions)
 1. *Name: specific name of the configuration
 2. *Python module: `<module_name>.<class_name>`

3. *Description: description of the configuration
4. *Routing key: celery queue that will be used
5. *Soft_time_limit: maximum time for the task execution
6. *Playbook to Execute: Playbook that **will** be executed on every IOC retrieved
7. *Schedule: Crontab object that describes the schedule of the ingestor. You are able to create a new clicking the plus symbol.

6.4.4 How to add a new Pivot

1. Put the module in the `pivots` directory
2. Remember to use `_monkeypatch()` in its class to create automated tests for the new pivot. This is a trick to have tests in the same class of its pivot.
3. Create the configuration inside django admin in `Pivots_manager/PivotConfigs` (* = mandatory, ~ = mandatory on conditions)
 1. *Name: specific name of the configuration
 2. *Python module: `<module_name>.<class_name>`
 3. *Description: description of the configuration
 4. *Routing key: celery queue that will be used
 5. *Soft_time_limit: maximum time for the task execution
 6. *Playbook to Execute: Playbook that **will** be executed in the Job generated by the Pivot

6.4.5 How to add a new Visualizer

Configuration

1. Put the module in the `visualizers` directory
2. Remember to use `_monkeypatch()` in its class to create automated tests for the new visualizer. This is a trick to have tests in the same class of its visualizer.
3. Create the configuration inside django admin in `Visualizers_manager/VisualizerConfigs` (* = mandatory, ~ = mandatory on conditions)
 1. *Name: specific name of the configuration
 2. *Python module: `<module_name>.<class_name>`
 3. *Description: description of the configuration
 4. *Config:
 1. *Queue: celery queue that will be used
 2. *Soft_time_limit: maximum time for the task execution
 5. *Playbook: Playbook that **must** have run to execute the visualizer

Python class

The visualizers' python code could be not immediate, so a small digression on *how* it works is necessary. Visualizers have as goal to create a data structure inside the **Report** that the frontend is able to parse and correctly *visualize* on the page. To do so, some utility classes have been made:

The best way to create a visualizer is to define several methods, one for each **Visualizable** you want to show in the UI, in your new visualizer and decore them with `visualizable_error_handler_with_params`. This decorator handles exceptions: in case there is a bug during the generation of a **Visualizable** element, it will be show an error instead of this component and all the other **Visualizable** are safe and will render correctly. Be careful using it because is a function returning a decorator! This means you need to use a syntax like this:

```
@visualizable_error_handler_with_params(error_name="custom visualizable", error_
↪size=VisualizableSize.S_2)
def custom_visualizable(self):
    . . .
```

instead of the syntax of other decorators that doesn't need the function call.

You may want to look at a few existing examples to start to build a new one:

- [dns.py](#)
- [yara.py](#)

6.4.6 How to share your plugin with the community

To allow other people to use your configuration, that is now stored in your local database, you have to export it and create a data migration

1. You can use the django management command `dumpplugin` to automatically create the migration file for your new analyzer (you will find it under `api_app/YOUR_PLUGIN_manager/migrations`). The script will create the following models:
 1. `PythonModule`
 2. `AnalyzerConfig`
 3. `Parameter`
 4. `PluginConfig`
2. Example: `docker exec -ti intelowl_uwsgi python3 manage.py dumpplugin AnalyzerConfig <new_analyzer_name>`

Add the new analyzer in the lists in the docs: [Usage](#). Also, if the analyzer provides additional optional configuration, add the available options here: [Advanced-Usage](#)

In the Pull Request remember to provide some real world examples (screenshots and raw JSON results) of some successful executions of the analyzer to let us understand how it would work.

6.5 How to add a new Playbook

1. Create the configuration inside django admin in Playbooks_manager/PlaybookConfigs (* = mandatory, ~ = mandatory on conditions)
 1. *Name: specific name of the configuration
 2. *Description: description of the configuration
 3. *Type: list of types that are supported by the playbook
 4. *Analyzers: List of analyzers that will be run
 5. *Connectors: List of connectors that will be run

6.5.1 How to share your playbook with the community

To allow other people to use your configuration, that is now stored in your local database, you have to export it and create a data migration. You can use the django management command `dumpplugin` to automatically create the migration file for your new analyzer (you will find it under `api_app/playbook_manager/migrations`).

Example: `docker exec -ti intelowl_uwsgi python3 manage.py dumpplugin PluginConfig <new_analyzer_name>`

6.6 How to modify a plugin

If the changes that you have to make should stay local, you can just change the configuration inside the Django admin page.

But if, instead, you want your changes to be usable by every IntelOwl user, you have to create a new migration.

To do so, you can use the following snippets as an example:

1. You have to create a new migration file
2. Add as dependency the previous last migration of the package
3. You have to create a `forward` and a `reverse` function
4. You have to make the proper changes of the configuration inside these functions (change parameters, secrets, or even delete the configuration)
 1. If changes are made, you have to validate the instance calling `.full_clean()` and then you can save the instance with `.save()`

6.6.1 Example: how to add a new parameter in the configuration with a default value

```
def migrate(apps, schema_editor):
    PythonModule = apps.get_model("api_app", "PythonModule")
    Parameter = apps.get_model("api_app", "Parameter")
    PluginConfig = apps.get_model("api_app", "PluginConfig")
    pm = PythonModule.objects.get(module="test.Test", base_path="api_app.connectors_
↪manager.connectors")
    p = Parameter(name="mynewfield", type="str", description="Test field", is_
```

(continues on next page)

(continued from previous page)

```

↪ secret=False, required=True, python_module=pm)
    p.full_clean()
    p.save()
    for connector in pm.connectorconfigs.all():
        pc = PluginConfig(value="test", connector_config=connector, python_module=pm, for_
↪ organization=False, owner=None, parameter=p)
        pc.full_clean()
        pc.save()

```

6.6.2 Example: how to add a new secret in the configuration

```

def migrate(apps, schema_editor):
    PythonModule = apps.get_model("api_app", "PythonModule")
    Parameter = apps.get_model("api_app", "Parameter")
    pm = PythonModule.objects.get(module="test.Test", base_path="api_app.connectors_
↪ manager.connectors")
    p = Parameter(name="mynewsecret", type="str", description="Test field", is_
↪ secret=True, required=True, python_module=pm)
    p.full_clean()
    p.save()

```

6.6.3 Example: how to delete a parameter

```

def migrate(apps, schema_editor):
    PythonModule = apps.get_model("api_app", "PythonModule")
    Parameter = apps.get_model("api_app", "Parameter")
    pm = PythonModule.objects.get(module="test.Test", base_path="api_app.connectors_
↪ manager.connectors")
    Parameter.objects.get(name="myoldfield", python_module=pm).delete()

```

6.6.4 Example: how to change the default value of a parameter

```

def migrate(apps, schema_editor):
    PythonModule = apps.get_model("api_app", "PythonModule")
    Parameter = apps.get_model("api_app", "Parameter")
    PluginConfig = apps.get_model("api_app", "PluginConfig")
    pm = PythonModule.objects.get(module="test.Test", base_path="api_app.connectors_
↪ manager.connectors")
    p = Parameter.objects.get(name="myfield", python_module=pm)
    PluginConfig.objects.filter(parameter=p, python_module=pm, for_organization=False,
↪ owner=None).update(value="newvalue")

```

6.7 Modifying functionalities of the Certego packages

Since v4, IntelOwl leverages some packages from Certego:

- `certego-saas` that integrates some common reusable Django applications and tools that can be used for generic services.
- `certego-ui` that contains reusable React components for the UI.

If you need to modify the behavior or add feature to those packages, please follow the same rules for IntelOwl and request a Pull Request there. The same maintainers of IntelOwl will answer to you.

Follow these guides to understand how to start to contribute to them while developing for IntelOwl:

- `certego-saas`: create a fork, commit your changes in your local repo, then change the commit hash to the last one you made in the [requirements file](#). Ultimately re-build the project
- `certego-ui`: [Frontend doc](#)

6.8 How to test the application

IntelOwl makes use of the django testing framework and the `unittest` library for unit testing of the API endpoints and End-to-End testing of the analyzers and connectors.

6.8.1 Configuration

- In the encrypted folder `tests/test_files.zip` (password: “intelowl”) there are some files that you can use for testing purposes.
- With the following environment variables you can customize your tests:
 - `DISABLE_LOGGING_TEST` -> disable logging to get a clear output
 - `MOCK_CONNECTIONS` -> mock connections to external API to test the analyzers without a real connection or a valid API key
- If you prefer to use custom inputs for tests, you can change the following environment variables in the environment file based on the data you would like to test:
 - `TEST_MD5`
 - `TEST_URL`
 - `TEST_IP`
 - `TEST_DOMAIN`

6.8.2 Setup containers

The point here is to launch the code in your environment and not the last official image in Docker Hub. For this, use the `test` or the `ci` option when launching the containers with the `./start` script.

- Use the `test` option to *actually* execute tests that simulate a real world environment without mocking connections.
- Use the `ci` option to execute tests in a CI environment where connections are mocked.

```
$ ./start test up
$ # which corresponds to the command: docker-compose -f docker/default.yml -f docker/
  ↳ test.override.yml up
```

6.8.3 Launch tests

Now that the containers are up, we can launch the test suite.

Backend

Run all tests

Examples:

```
$ docker exec intelowl_uwsgi python3 manage.py test
```

Run tests for a particular plugin

To test a plugin in real environment, i.e. without mocked data, we suggest that you use the GUI of IntelOwl directly. Meaning that you have your plugin configured, you have selected a correct observable/file to analyze, and the final report shown in the GUI of IntelOwl is exactly what you wanted.

Run tests available in a particular file

Examples:

```
$ docker exec intelowl_uwsgi python3 manage.py test tests.api_app tests.test_crons #
  ↳ dotted paths
```

Frontend

All the frontend tests must be run from the folder frontend. The tests can contain log messages, you can suppress them with the environment variable `SUPPRESS_JEST_LOG=True`.

Run all tests

```
npm test
```


Run a specific component tests

```
npm test -- -t <componentPath>
// example
npm test tests/components/auth/Login.test.jsx
```

Run a specific test

```
npm test -- -t '<describeString> <testString>'
// example
npm test -- -t "Login component User login"
```

6.9 Create a pull request

6.9.1 Remember!!!

Please create pull requests only for the branch **develop**. That code will be pushed to master only on a new release.

Also remember to pull the most recent changes available in the **develop** branch before submitting your PR. If your PR has merge conflicts caused by this behavior, it won't be accepted.

6.9.2 Install testing requirements

Run `pip install -r requirements/test-requirements.txt` to install the requirements to validate your code.

Pass linting and tests

1. Run `psf/black` to lint the files automatically, then `flake8` to check and `isort`:

(if you installed `pre-commit` this is performed automatically at every commit)

```
$ black . --exclude "migrations|venv"
$ flake8 . --show-source --statistics
$ isort . --profile black --filter-files --skip venv
```

if `flake8` shows any errors, fix them.

1. Run the build and start the app using the `docker-compose` test file. In this way, you would launch the code in your environment and not the last official image in Docker Hub:

```
$ ./start ci build
$ ./start ci up
```

1. Here, we simulate the GitHub CI tests locally by running the following 3 tests:

```
$ docker exec -ti intelowl_uwsgi unzip -P intelowl tests/test_files.zip -d test_files
$ docker exec -ti intelowl_uwsgi python manage.py test tests
```

Note: IntelOwl has dynamic testing suite. This means that no explicit analyzers/connector tests are required after the addition of a new analyzer or connector.

If everything is working, before submitting your pull request, please squash your commits into a single one!

How to squash commits to a single one

- Run `git rebase -i HEAD~[NUMBER OF COMMITS]`
- You should see a list of commits, each commit starting with the word “pick”.
- Make sure the first commit says “pick” and change the rest from “pick” to “squash”. – This will squash each commit into the previous commit, which will continue until every commit is squashed into the first commit.
- Save and close the editor.
- It will give you the opportunity to change the commit message.
- Save and close the editor again.
- Then you have to force push the final, squashed commit: `git push --force-with-lease origin`.

Squashing commits can be a tricky process but once you figure it out, it’s really helpful and keeps our repo concise and clean.

6.10 Debug application problems

Keep in mind that, if any errors arise during development, you would need to check the application logs to better understand what is happening so you can easily address the problem.

This is the reason why it is important to add tons of logs in the application... if they are not available in time of needs you would cry really a lot.

Where are IntelOwl logs? With a default installation of IntelOwl, you would be able to get the application data from the following paths in your OS:

- `/var/lib/docker/volumes/intel_owl_generic_logs/_data/django`: Django Application logs
- `/var/lib/docker/volumes/intel_owl_generic_logs/_data/uwsgi`: Uwsgi application server logs
- `/var/lib/docker/volumes/intel_owl_nginx_logs/_data/`: Nginx Web Server Logs

