IntelOwl

Release v3.0.1

Matteo Lodi

Sep 17, 2021

CONTENTS

CHAPTER

ONE

INTRODUCTION

Official first announcement: Certego News

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:

- modern Django-Python application: easy to understand and write code upon it
- it can get data from multiple sources with a single API request
- more than 100 available analyzers that you can use to generate or retrieve data about a suspicious file or observable (IP, domain, . . .)
- built-in Web Interface: **IntelOwl-ng** provides features such as dashboard, visualizations of analysis data, easy to use forms for requesting new analysis and more.
- official library and CLI client available on GitHub: PyIntelOwl
- built-in support for integration with other SIEM/SOAR projects using connectors, specifically aimed at Threat Sharing Platforms.
- easily integrable with other tools thanks to the REST API framework and to the PyIntelOwl library.
- easily and completely customizable, both the APIs and the analyzers
- early compatibility with some of the AWS services. More in the future.
- fast and reliable deploy: clone the project, set up the configuration and then you are ready to run it via docker-compose

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.

CHAPTER

TWO

INSTALLATION

2.1 Requirements

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

- docker - v1.13.0+
- docker-compose - v1.23.2+
- python - v3.6+

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.

2.2 TL;DR

Obviously we strongly suggest to read 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. Be sure to run `docker` and `python` commands with `sudo` if permissions/roles have not been set

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

# construct environment files from templates
cd docker/
cp env_file_app_template env_file_app
cp env_file_postgres_template env_file_postgres
cp env_file_integrations_template env_file_integrations

# start the app
cd ..
python3 start.py prod up

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

# now the app is running on http://localhost:80
```

2.3 Deployment Components

IntelOwl is composed of various different services, namely:

- Angular: Frontend (IntelOwl-ng)
- Django: Backend
- PostgreSQL: Database
- Rabbit-MQ: Message Broker
- Celery: Task Queue
- Nginx: Reverse proxy for the Django API and web asssets.
- Uwsgi: Application Server
- 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

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

```
cd docker/
cp env_file_app_template env_file_app
cp env_file_postgres_template env_file_postgres
cp env_file_integrations_template env_file_integrations
```

2.4.1 Environment configuration (required)

In the `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 credentals should match the ones in the `env_file_postgres`).

Strongly recommended variable to set:

- `DJANGO_SECRET`: random 50 chars key, must be unique. If you do not provide one, Intel Owl will automatically set a new secret on every run.
- `INTELOWL_WEB_CLIENT_DOMAIN` (example: `localhost/mywebsite.com`): the web domain of your instance, this is used for generating links to analysis results.

Optional variables needed to enable specific analyzers:

- `ABUSEIPDB_KEY`: AbuseIPDB API key
- `AUTH0_KEY`: Auth0 API Key
- `SECURITYTRAILS_KEY`: Securitytrails API Key
- `SHODAN_KEY`: Shodan API key
- `HUNTER_API_KEY`: Hunter.io API key
- `GSF_KEY`: Google Safe Browsing API key
- `OTX_KEY`: Alienvault OTX API key
- `CIRCL_CREDENTIALS`: CIRCL PDNS credentials in the format: `user|pass`
- `VT_KEY`: VirusTotal API key
- `HA_KEY`: HybridAnalysis API key
- `INTEZER_KEY`: Intezer API key
- `INQUEST_API_KEY`: InQuest API key
- `FIRST_MISP_API`: FIRST MISP API key
- `FIRST_MISP_URL`: FIRST MISP URL
- `MISP_KEY`: your own MISP instance key
- `MISP_URL`: your own MISP instance URL
- `DNSDB_KEY`: DNSDB API key
- `CUCKOO_URL`: your cuckoo instance URL
- `HONEYDB_API_ID` & `HONEYDB_API_KEY`: HoneyDB API credentials
- `CENSYS_API_ID` & `CENSYS_API_SECRET`: Censys credentials
- `ONYPHE_KEY`: Onyphe.io's API Key
- `GREYNOISE_API_KEY`: GreyNoise API (docs)
- `INTELX_API_KEY`: IntelligenceX API (docs)
- `UNPAC_ME_API_KEY`: UnpacMe API (docs)
- `IPINFO_KEY`: ipinfo API key
- `ZOOMEYE_KEY`: ZoomEye API Key(docs)
- `TRIAGE_KEY`: tria.ge API key(docs)
- `WIGLE_KEY`: WiGLE API Key(docs)
- `XFORCE_KEY` & `XFORCE_PASSWORD`: IBM X-Force Exchange API (docs)
- `MWDB_KEY`: API key for MWDB
- `SSAPINET_KEY`: screenshotapi.net (docs)
- `MALPEDIA_KEY`: MALPEDIA API KEY (docs)
- `OPENCTI_KEY`: your own OpenCTI instance key
- `OPENCTI_URL`: your own OpenCTI instance URL
- `YETI_KEY`: your own YETI instance key
- `YETI_URL`: your own YETI instance URL

- SPYSE_API_KEY: Spyse API key. Register here: https://spyse.com/user/registration"

Optional variables needed to work with specific connectors:

- CONNECTOR_MISP_KEY: your own MISP instance key to use with MISP connector
- CONNECTOR_MISP_URL: your own MISP instance URL to use with MISP connector
- CONNECTOR_OPENCTI_KEY: your own OpenCTI instance key to use with OpenCTI connector
- CONNECTOR_OPENCTI_URL: your own OpenCTI instance URL to use with OpenCTI connector
- CONNECTOR_YETI_KEY: your own YETI instance key to use with YETI connector
- CONNECTOR_YETI_URL: your own YETI instance API URL to use with YETI connector

Advanced additional configuration:

- OLD_JOBS_RETENTION_DAYS: Database retention for analysis results (default: 3 days). Change this if you want to keep your old analysis longer in the database.

2.4.2 Database configuration (required)

In the env_file_postgres, 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)

If you prefer to use an external PostgreSQL instance, you should just remove the relative image from the docker/default.yml file and provide the configuration to connect to your controlled instances.

2.4.3 Web server configuration (optional)

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.

  Then you should modify the nginx service configuration in docker/default.yml:

  - change http.conf with https.conf
  - in volumes add the option for mounting the directory that hosts your certificate and your certificate key.

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: Traefix doc.

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

2.4.4 Analyzers or connectors configuration (optional)

Refer to Analyzers customization and Connectors customization.

2.5 Run

You may invoke $ python3 start.py –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:

```
$ python3 start.py prod up
```

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

2.5.1 Stop

To stop the application you have to:

- if executed without `-d` parameter: press `CTRL+C`
- if executed with `-d` parameter: `python3 start.py 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

`python3 start.py 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.

2.6.2 Django Groups & Permissions settings

Refer to this section of the docs.

2.7 Extras

2.7.1 Deploy on Remnux

Remnux is a Linux Toolkit for Malware Analysis.

IntelOwl and Remnux have the same goal: save the time of people who need to perform malware analysis or info gathering.

Therefore we suggest Remnux users to install IntelOwl to leverage all the tools provided by both projects in a unique environment.

To do that, you can follow the same steps detailed above for the installation of IntelOwl.

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
$ python3 start.py prod stop # kill the currently running IntelOwl containers
$ python3 start.py prod up --build # restart the IntelOwl application
```

2.7.3 Rebuilding the project/ Creating custom docker build

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

1. `python3 start.py 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 `python3 start.py test up --build`.

2.7.4 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.

2.7.5 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!

CHAPTER

THREE

USAGE

3.1 Client

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. IntelOwl-ng (Web Interface)
    - Inbuilt Web interface with dashboard, visualizations of analysis data, easy to use forms for requesting new analysis, tags management and more features
    - Live Demo
    - Built with Angular 10+ and available on GitHub.
2. pyIntelOwl (CLI/SDK)
    - Official client that is available at: PyIntelOwl,
    - Can be used as a library for your own python projects or...
    - directly via the command line interface.

3.2 Analyzers customization

You can create new analyzers based on already existing modules by changing the configuration values inside `configuration/analyzer_config.json`. This file is mounted as a docker volume, so you won't need to rebuild the image.

You may want to change this configuration to add new analyzers or to change the configuration of some of them. The name of the analyzers can be changed at every moment based on your wishes.

The following are all the keys that you can change without touching the source code:

- `disabled`: you can choose to disable certain analyzers, then they won't appear in the dropdown list and won't run if requested.
- `leaks_info`: if set, in the case you specify via the API that a resource is sensitive, the specific analyzer won't be executed
- `external_service`: if set, in the case you specify via the API to exclude external services, the specific analyzer won't be executed
- `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`.
- `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`.
- `queue`: 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).

3.2.1 Example: add an analyzer configuration for your own Yara signatures

```
"Yara_Scan_Custom_Signatures": {
    "type": "file",
    "python_module": "yara.Yara",
    "description": "Executes Yara with custom signatures",
    "config": {
        "queue": "default",
        "soft_time_limit": 100,
    },
    "params": {
        "directories_with_rules": ["/opt/deploy/yara/custom_signatures"]
    }
}
```

3.3 Connectors customization

Connectors being optional are `disabled` by default. You can enable them by changing the configuration values inside `configuration/connector_config.json`. This file is mounted as a docker volume, so you won't need to rebuild the image.

The following are all the keys that you can change without touching the source code:

- `disabled`: *similar to analyzers*
- `soft_time_limit`: *similar to analyzers*
- `queue`: *similar to analyzers*
- `maximum_tlp` (default `WHITE`, choices `WHITE`, `GREEN`, `AMBER`, `RED`): specify with the maximum TLP of the analysis upto which the connector is allowed to run. (e.g. if `maximum_tlp` is `GREEN`, it would run for analysis with TLPs `WHITE` and `GREEN`). To learn more about TLPs see TLP Support.

3.4 Managing Analyzers and Connectors

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:**

  To 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}/analyzer/{analyzer_name}/kill` and `PATCH /api/job/{job_id}/connector/{connector_name}/kill`

- **retry:**

  To 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}/analyzer/{analyzer_name}/retry` and `PATCH /api/job/{job_id}/connector/{connector_name}/retry`

- **healthcheck:**

  To check if docker container or external platform associated with an analyzer or connector respectively are up or not:

  - GUI: Buttons on analyzers table and connectors 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/analyzer/{analyzer_name}/healthcheck` and `GET /api /connector/{connector_name}/healthcheck`

3.5 TLP Support

IntelOwl supports the **Traffic Light Protocol** (TLP) to facilitate sharing of job analysis results.

Following are the indicators available when requesting an analysis (in the order of increasing sharing restrictions):

1. `WHITE`: no restriction
2. `GREEN`: disable analyzers that could impact privacy
3. `AMBER`: 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

These indicators when used with `maximum_tlp` (option available in connectors), give you the control of what information is shared to the external platforms.

3.6 Available Analyzers

3.6.1 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,

- live demo for better UX.
- pyintelowl: `$ pyintelowl get-analyzer-config`

File analyzers:

- `File_Info`: static generic File analysis (hashes, magic and exiftool)
- `PDF_Info`: static PDF analysis (peepdf + pdfid)
- `Rtf_Info`: static RTF analysis (Oletools)
- `Doc_Info`: static generic document analysis (Oletools)
- `Xlm_Macro_Deobfuscator`: XlmMacroDeobfuscator deobfuscate xlm macros
- `Doc_Info_Experimental`: static document analysis with new features to analyze XLM macros, encrypted macros and more (combination of Oletools and XLMMacroDeobfuscator)
- `PE_Info`: static PE analysis with pefile
- `Signature_Info`: PE signature extractor with osslsigncode
- `Speakeasy`: FireEye Speakeasy binary emulation
- `Strings_Info_Classic`: strings extraction
- `Strings_Info_ML`: strings extraction plus strings ranking based on Machine Learning. Leverages Stringsifter
- `VirusTotal_v3_Get_File_And_Scan`: check file hash on VirusTotal. If not already available, send the sample and perform a scan
- `VirusTotal_v3_Get_File`: check only the file hash on VirusTotal (this analyzer is disabled by default to avoid multiple unwanted queries. You have to change that flag in the config to use it)
- `VirusTotal_v2_Get_File`: check file hash on VirusTotal using old API endpoints (this analyzer is disabled by default. You have to change that flag in the config to use it)
- `VirusTotal_v2_Scan_File`: scan a file on VirusTotal using old API endpoints (this analyzer is disabled by default. You have to change that flag in the config to use it)
- `Intezer Scan`: scan a file on Intezer. Register for a free community account here
- `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)
- `HybridAnalysis_Get_File`: check file hash on HybridAnalysis sandbox reports
- `OTX_Check_Hash`: check file hash on Alienvault OTX
- `MISP_Check_Hash`: check a file hash on a MISP instance

- `MISPFIRST_Check_Hash`: check a file hash on the FIRST MISP instance
- `Yara_Scan_Community`: scan a file with the community yara rules
- `Yara_Scan_Dail_Ioc`: scan a file with StrangerealIntel Daily IOC yara rules
- `Yara_Scan_Florian`: scan a file with Neo23x0 yara rules
- `Yara_Scan_Intezer`: scan a file with Intezer yara rules
- `Yara_Scan_Inquest`: scan a file with Inquest yara rules
- `Yara_Scan_McAfee`: scan a file with McAfee yara rules
- `Yara_Scan_Samir`: scan a file with Samir Threat Hunting yara rules
- `Yara_Scan_Stratosphere`: scan a file with Stratosphere yara rules
- `Yara_Scan_FireEye`: scan a file with FireEye yara rules
- `Yara_Scan_ReversingLabs`: scan a file with ReversingLabs yara rules
- `Yara_Scan_Custom_Signatures`: scan a file with your own added signatures
- `MalwareBazaar_Get_File`: Check if a particular malware sample is known to MalwareBazaar
- `PEframe_Scan`: Perform static analysis on Portable Executable malware and malicious MS Office documents with PeFrame
- `Cymru_Hash_Registry_Get_File`: Check if a particular file is known to be malware by Team Cymru
- `Thug_HTML_Info`: Perform hybrid dynamic/static analysis on a HTML file using Thug low-interaction honey-client
- `Capa_Info`: Capa detects capabilities in executable files
- `BoxJS_Scan_Javascript`: Box-JS is a tool for studying JavaScript malware.
- `APKiD_Scan_APK_DEX_JAR`: APKiD identifies many compilers, packers, obfuscators, and other weird stuff from an APK or DEX file.
- `Quark_Engine_APK`: Quark Engine is an Obfuscation-Neglect Android Malware Scoring System.
- `IntelX_Phonebook`: IntelligenceX is a search engine and data archive. Fetches emails, urls, domains associated with an observable.
- `UnpacMe_EXE_Unpacker`: UnpacMe is an automated malware unpacking service
- `Triage_Scan`: leverage Triage sandbox environment to scan various files
- `Manalyze`: Manalyze performs static analysis on PE executables to detect undesirable behavior.
- `MWDB_Scan`: mwdblib Retrieve malware file analysis from repository maintained by CERT Polska MWDB.
- `Qiling`: Qiling qiling binary emulation.
- `Malpedia_Scan`: scan a binary or a zip file (pwd:infected) against all the yara rules available in Malpedia
- `HashLookupServer_Get_File`: check if a md5 or sha1 is available in the database of known file hosted by CIRCL

Observable analyzers (ip, domain, url, hash)

- `VirusTotal_v3_Get_Observable`: search an observable in the VirusTotal DB
- `VirusTotal_v2_Get_Observable`: search an observable in the VirusTotal DB using the old API endpoints (this analyzer is disabled by default. You have to change that flag in the config to use it)
- `HybridAnalysis_Get_Observable`: search an observable in the HybridAnalysis sandbox reports
- `OTXQuery`: scan an observable on Alienvault OTX
- `TalosReputation`: check an IP reputation from Talos
- `Stratosphere_Blacklist`: Cross-reference an IP from blacklists maintained by Stratosphere Labs
- `Robtex_Forward_PDNS_Query`: scan a domain against the Robtex Passive DNS DB
- `Robtex_Reverse_PDNS_Query`: scan an IP against the Robtex Passive DNS DB
- `Robtex_IP_Query`: get IP info from Robtex
- `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
- `GreyNoiseCommunity`: scan an IP against the Community Greynoise API (no API key required)
- `GreyNoise`: scan an IP against the Greynoise API (requires API key)
- `CIRCLPassiveDNS`: scan an observable against the CIRCL Passive DNS DB
- `CIRCLPassiveSSL`: scan an observable against the CIRCL Passive SSL DB
- `MaxMindGeoIP`: extract GeoIP info for an observable
- `AbuseIPDB`: check if an ip was reported on AbuseIPDB
- `Fortiguard`: scan an observable with the Fortiguard URL Analyzer
- `TorProject`: check if an IP is a Tor Exit Node
- `MISP`: scan an observable on a MISP instance
- `MISPFIRST`: scan an observable on the FIRST MISP instance
- `DNSDB`: scan an observable against the Passive DNS Farsight Database (support both v1 and v2 versions)
- `Shodan_Search`: scan an IP against Shodan Search API
- `Shodan_Honeyscore`: scan an IP against Shodan Honeyscore API
- `HoneyDB_Get`: HoneyDB IP lookup service
- `HoneyDB_Scan_Twitter`: scan an IP against HoneyDB.io's Twitter Threat Feed
- `Hunter`: 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.
- `Censys_Search`: scan an IP address against Censys View API
- `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
- `ONYPHE`: search an observable in ONYPHE
- `Threatminer_PDNS`: retrieve PDNS data from Threatminer API

- `Threatminer_Reports_Tagging`: retrieve reports from Threatminer API
- `Threatminer_Subdomains`: retrieve subdomains from Threatminer API
- `URLhaus`: Query a domain or URL against URLhaus API.
- `Google_DNS`: Retrieve current domain resolution with Google DoH (DNS over HTTPS)
- `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
- `Classic_DNS`: Retrieve current domain resolution with default DNS
- `Auth0`: scan an IP against the Auth0 API
- `Securitytrails_IP_Neighbours`: scan an IP against Securitytrails API for neighbour IPs
- `Securitytrails_Details`: scan a domain against Securitytrails API for general details
- `Securitytrails_Subdomains`: scan a domain against Securitytrails API for subdomains
- `Securitytrails_Tags`: scan a domain against Securitytrails API for tags
- `Securitytrails_History_WHOIS`: scan a domain against Securitytrails API for historical WHOIS
- `Securitytrails_History_DNS`: scan a domain against Securitytrails API for historical DNS
- `Cymru_Hash_Registry_Get_Observable`: Check if a particular hash is available in the malware hash registry of Team Cymru
- `Tranco`: Check if a domain is in the latest Tranco ranking top sites list
- `Thug_URL_Info`: Perform hybrid dynamic/static analysis on a URL using Thug low-interaction honeyclient
- `Pulsedive_Active_IOC`: Scan indicators and retrieve results from Pulsedive's API.
- `CheckDMARC`: An SPF and DMARC DNS records validator for domains.
- `Whoisxmlapi`: Fetch WHOIS record data, of a domain name, an IP address, or an email address.
- `UrlScan_Search`: Search an IP/domain/url/hash against URLScan API
- `UrlScan_Submit_Result`: Submit & retrieve result of an URL against URLScan API
- `Phishtank`: Search an url against Phishtank 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
- `DNStwist`: Scan a url/domain to find potentially malicious permutations via dns fuzzing. dnstwist repo
- `IPInfo`: Location Information about an IP
- `Zoomeye`: Zoomeye Cyberspace Search Engine recording information of devices, websites, services and components etc..
- `Triage_Search`: Search for reports of observables or upload from URL on triage cloud
- `InQuest_IOCdb`: Indicators of Compromise Database by InQuest Labs
- `InQuest_REPdb`: Search in InQuest Lab's Reputation Database
- `InQuest_DFI`: Deep File Inspection by InQuest Labs
- `XForceExchange`: scan an observable on IBM X-Force Exchange
- `Renderton`: get screenshot of a web page using rendertron (puppeteer) renderton repo

- `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.
- `FireHol_IPList`: check if an IP is in FireHol's IPList
- `ThreatFox`: search for an IOC in ThreatFox's database
- `OpenCTI`: scan an observable on an OpenCTI instance
- `Intezer_Get`: check if an analysis related to a hash is available in Intezer. Register for a free community account here.
- `MWDB_Get`: mwdblib Retrieve malware file analysis by hash from repository maintained by CERT Polska MWDB.
- `YETI` (Your Everyday Threat Intelligence): scan an observable on a YETI instance.
- `HashLookupServer_Get_Observable`: check if a md5 or sha1 is available in the database of known file hosted by CIRCL
- `ClamAV`: scan a file via the ClamAV AntiVirus Engine
- `Spyse`: Scan domains, IPs, emails and CVEs using Spyse's API. Register here.

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.

- `EmailRep`: search an email address on emailrep.io
- `WiGLE`: Maps and database of 802.11 wireless networks, with statistics, submitted by wardrivers, netstumblers, and net huggers.
- `CRXcavator`: scans a chrome extension against crxcavator.io
- `Darksearch_Query`: Search a keyword against darksearch.io's search API. It's possible to make complex queries using boolean logic. For example, `OSINT AND CTI OR intelowl NOT hack` is a valid observable name.
- `Dehashed_Search`: Query any observable/keyword against https://dehashed.com's search API.

Extra analyzers

Additional analyzers that can be enabled per your wish.

3.7 Available 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.

3.7.1 Connectors list

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

- live demo for better UX.
- pyintelowl: `$ pyintelowl get-connector-config`
- `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.

To contribute to the project, see *Contribute*.

CHAPTER

FOUR

ADVANCED USAGE

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

4.1 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:

```
python3 start.py prod --all_analyzers up
```

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

```
python3 start.py prod --qiling up
```

4.2 Customize analyzer execution at time of request

Some analyzers and connectors provide the chance to customize the performed analysis based on parameters (`params` attr in the configuration file) that are different for each analyzer.

- You can set a custom default values by changing their `value` attribute directly from the configuration files.
- You can choose to provide runtime configuration when requesting an analysis that will be merged with the default overriding it. This override is done only for the specific analysis.

4.2.1 View and understand different parameters

To see the list of these parameters:

- You can view the "Analyzers Table", here.
- You can view the raw JSON configuration file, here.

4.2.2 from the GUI

You can click on "**CUSTOMIZE ANALYZERS PARAMETERS**" button and add the runtime configuration in the form of a dictionary. Example:

```
"VirusTotal_v3_Get_File": {
    "force_active_scan_if_old": true
}
```

4.2.3 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)
```

(continues on next page)

(continued from previous page)

4.3 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 copy the generated JSON key file in the directory `configuration` of the project and change its name to `service_account_keyfile.json`. This is the default configuration. If you want to customize the name or the location of the file, you can change the environment variable `GOOGLE_APPLICATION_CREDENTIALS` in the `env_file_app` file.
- `ClamAV`: this Docker-based analyzer using `clamd` daemon as it's scanner, communicating with `clamdscan` 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 and hence, can be edited by the user as per needs.

4.4 Elastic Search

Intel Owl 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.

4.4.1 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).

4.4.2 Example Configuration

1. Setup Elastic Search and Kibana and say it is running in a docker service with name `elk` on port `9200` which is exposed to the shared docker network.
2. In the `env_file_app`, we set `ELASTICSEARCH_ENABLED` to `True` and `ELASTICSEARCH_HOST` to `elk:9200`.
3. In the `Dockerfile`, set the correct version in `ELASTICSEARCH_DSL_VERSION` depending on the version of our elasticsearch server. Default value is `7.1.4`.
4. Rebuild the docker images with `docker-compose build` (required only if `ELASTICSEARCH_DSL_VERSION` was changed)
5. 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.

4.5 Django Groups & Permissions

The application makes use of Django's built-in permissions system. It provides a way to assign permissions to specific users and groups of users.

As an administrator here's what you need to know,

- Each user should belong to atleast a single group and permissions should be assigned to these groups. Please refrain from assigning user level permissions.
- When you create a first normal user, a group with name `DefaultGlobal` is created with all permissions granted. Every new user automatically gets added to this group.
  - This is done because most admins won't need to deal with user permissions and this way, they don't have to.
  - If you don't want a global group (with all permissions) but custom groups with custom permissions, just strip `DefaultGlobal` of all permissions but do *not* delete it.

The permissions work the way one would expect,

4.6 Authentication options

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

- LDAP
- GSuite (work in progress)

4.6.1 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.

   For more details on how to configure this file, check the official documentation of the django-auth-ldap library.

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`

4.7 Google Kubernetes Engine deployment

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.

4.8 Queues

4.8.1 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:

```
python3 start.py prod --multi-queue up
```

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.

4.8.2 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.

4.8.3 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:

```
python3 start.py prod --flower up
```

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.

4.9 AWS support

At the moment there's a basic support for some of the AWS services. More is coming in the future.

4.9.1 Secrets

If you would like to run this project on AWS, I'd suggest you to use the "Secrets Manager" to store your credentials. In this way your secrets would be better protected.

This project supports this kind of configuration. 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.

You can customize the AWS Region changing the environment variable `AWS_REGION`.

4.9.2 SQS

If you like, you could use AWS SQS instead of Rabbit-MQ to manage your queues. In that case, you should change the parameter `BROKER_URL` to `sqs://` and give your instances on AWS the proper permissions to access it.

Also, you need to set the environment variable `AWS_SQS` to `True` to activate the additional required settings.

4.9.3 S3

If you prefer to use S3 to store the samples, instead of a local storage, you can now 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 proper AWS bucket. Then 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_ACESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`

CHAPTER

FIVE

CONTRIBUTE

Intel Owl was designed to ease the addition of new analyzers/connectors. With a simple python script you can integrate your own engine or integrate an external service in a short time.

> Wish to contribute to the web interface ? See IntelOwl-ng.
>
> Wish to contribute to the python client ? See pyintelowl.

5.1 Code Style

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

5.2 How to start (Setup project and development instance)

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:

```
# create virtualenv to host pre-commit installation
python3 -m venv venv
source venv/bin/activate
# from the project base directory
pip install pre-commit
pre-commit install
```

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

```
python3 start.py test --django-server up
```

Every time you perform a change, you should rebuild the containers to have it reflected in the server:

```
python3 start.py test down
python3 start.py test up --build
```

5.3 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
- intezer_scan.py, if you are creating a file analyzer
- peframe.py, if you are creating a *docker based analyzer*

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. Add a new entry in the analyzer configuration following alphabetical order:

Example:

```
"Analyzer_Name": {
    "type": "file", // or "observable"
    "python_module": "<module_name>.<class_name>",
    "description": "very cool analyzer",
    "external_service": true,
    "leaks_info": true
    "run_hash": true, // required only for file analyzer
    "observable_supported": ["ip", "domain", "url", "hash", "generic"], // required only␣
↪for observable analyzer
    "supported_filetypes": ["application/javascript"], // required only for file analyzer
    "config": {
      "soft_time_limit": 100,
      "queue": "long",
    }
    "secrets": {
      "api_key_name": {
        "env_var_key": "ANALYZER_SPECIAL_KEY",
        "type": "string",
        "required": true,
        "default": null,
        "description": "API Key for the analyzer",
      }
    }
},
```

The `config` can be used in case the new analyzer uses specific configuration arguments and `secrets` can be used to declare any secrets the analyzer requires in order to run (Example: API Key, URL, etc.). In that way you can create more than one analyzer for a specific python module, each one based on different configurations. MISP and Yara Analyzers are a good example of this use case: for instance, you can use different analyzers for different MISP instances.

1. 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*
2. Ultimately, add the required secrets in the files `docker/env_file_app_template`, `docker/env_file_app_ci` and in the `docs/Installation.md`.
3. 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.

5.3.1 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".

5.4 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. Add a new entry in the connector_config.json following alphabetical order:

Example:

```
"Connector_Name": {
    "python_module": "<module_name>.<class_name>",
    "description": "very cool connector",
    "maximum_tlp": "WHITE",
    "config": {
      "soft_time_limit": 100,
      "queue": "default",
    }
    "secrets": {
         "env_var_key": "CONNECTOR_SPECIAL_KEY",
         "type": "string",
         "required": true,
         "default": null,
         "description": "API Key for the connector",
    }
},
```

Remember to set at least:

- `python_module`: name of the task that the connector must launch
- `description`: little description of the connector
- `maximum_tlp`: maximum TLP of the analysis upto which the connector is allowed to run.

Similar to analyzers, the `config` can be used in case the new connector uses specific configuration arguments and `secrets` can be used to declare any secrets the connector requires in order to run (Example: API Key).

Please see Connectors customization section to get the explanation of the other available keys.

1. Add the new connector in the lists in the docs: *Usage*. Also, if the connector provides additional optional configuration, add the available options here: *Advanced-Usage*
2. Follow steps 4-5 of How to add a new analyzer

5.5 Create a pull request

5.5.1 Install testing requirements

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

Pass linting and tests

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

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

```
$ black . --exclude "migrations|venv"
$ flake8 . --show-source --statistics
```

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:

```
$ python3 start.py ci build
$ python3 start.py ci up
```

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

```
$ docker exec -ti intelowl_uwsgi unzip -P infected tests/test_files.zip
$ 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.

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.

CHAPTER

SIX

TESTS

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.

6.1 Configuration

- In the encrypted folder `test_files.zip` (password: "infected") there are some real malware samples 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_JOB_ID`
  - `TEST_MD5`
  - `TEST_URL`
  - `TEST_IP`
  - `TEST_DOMAIN`

6.2 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.py` 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.

```
$ python3 start.py test up
$ # which corresponds to the command: docker-compose -f docker/default.yml -f docker/
↪test.override.yml up
```

Note: You may need to rebuild the docker containers on every change.

6.3 Launch tests

Now that the containers are up, we can launch the test suite. There are two helper scripts available for launching tests namely, `coverage_test.sh` and the `test_analyzers.sh`.

6.3.1 Run all tests

Examples:

```
$ docker/scripts/coverage_test.sh tests
```

6.3.2 Run tests available in a particular file

Examples:

```
$ docker/scripts/coverage_test.sh tests.test_api tests.test_auth # dotted paths
```

6.3.3 Run tests for a particular analyzer or class of analyzers

Syntax:

```
$ docker/scripts/test_analyzers.sh <analyzer_class> <comma_seperated_analyzer_names>
```

Examples:

- Observable analyzers tests:

  ```
  $ docker/scripts/test_analyzers.sh ip Shodan_Honeyscore,Darksearch_Query # run only␣
  ↪the specified analyzers
  $ docker/scripts/test_analyzers.sh domain # run all domain analyzers
  ```

  supports: `ip`, `domain`, `url`, `hash`, `generic`.

- File analyzers tests:

  ```
  $ docker/scripts/test_analyzers.sh exe File_Info,PE_Info # run only the specified␣
  ↪analyzers
  $ docker/scripts/test_analyzers.sh pdf # run all PDF analyzers
  ```

  supports: `exe`, `dll`, `doc`, `excel`, `rtf`, `html`, `pdf`, `js`, `apk`.

CHAPTER

SEVEN

INTELOWL API

POST /api/analyze_file
This endpoint allows to start a Job related to a file

Status Codes

- 200 OK –

POST /api/analyze_observable
This endpoint allows to start a Job related to an observable

Status Codes

- 200 OK –

GET /api/analyzer/{name}/healthcheck
Health Check: if instance associated with plugin is up or not

Parameters

- **name** (*string*) –

Status Codes

- 200 OK –

POST /api/ask_analysis_availability

This is useful to avoid repeating the same analysis multiple times. By default this API checks if there are existing analysis related to the md5 in status "running" or "reported_without_fails" Also, you need to specify the analyzers needed because, otherwise, it is highly probable that you won't get all the results that you expect

Status Codes

- 200 OK –

POST /api/auth/login
Durin's Login View.

This view will return a JSON response when valid `username`, `password` and (if not overwritten) `client` fields are POSTed to the view using form data or JSON.

It uses the default serializer provided by Django-Rest-Framework (`rest_framework.authtoken.serializers.AuthTokenSerializer`) to validate the user credentials.

It is possible to customize LoginView behaviour by overriding the following helper methods:

Status Codes

- 200 OK – No response body

`POST /api/auth/logout`
Durin's Logout View.

This view accepts only a post request with an empty body. It responds to Durin Token Authentication. On a successful request,

1. The token used to authenticate is deleted from the database and can no longer be used to authenticate.
2. `django.contrib.auth.signals.user_logged_out()` is called.

Returns 204 (No content)

Status Codes

- 200 OK – No response body

`GET /api/connector/{name}/healthcheck`
Health Check: if instance associated with plugin is up or not

Parameters

- **name** (*string*) –

Status Codes

- 200 OK –

`GET /api/get_analyzer_configs`

Get the uploaded analyzer configuration, can be useful if you want to choose the analyzers programmatically

Status Codes

- 200 OK –
- 500 Internal Server Error –

`GET /api/get_connector_configs`
Get the uploaded connector configuration

Status Codes

- 200 OK –
- 500 Internal Server Error –

`PATCH /api/job/{job_id}/analyzer/{name}/kill`
Kill running plugin by closing celery task and marking as killed

Parameters

- **job_id** (*integer*) –
- **name** (*string*) –

Status Codes

- 204 No Content – No response body

`PATCH /api/job/{job_id}/analyzer/{name}/retry`
Retry a plugin run if it failed/was killed previously

Parameters

- **job_id** (*integer*) –

- **name** (*string*) –

Status Codes

- 204 No Content – No response body

`PATCH /api/job/{job_id}/connector/{name}/kill`
Kill running plugin by closing celery task and marking as killed

Parameters

- **job_id** (*integer*) –
- **name** (*string*) –

Status Codes

- 204 No Content – No response body

`PATCH /api/job/{job_id}/connector/{name}/retry`
Retry a plugin run if it failed/was killed previously

Parameters

- **job_id** (*integer*) –
- **name** (*string*) –

Status Codes

- 204 No Content – No response body

`GET /api/jobs`

REST endpoint to fetch list of jobs or retrieve/delete a job with job ID. Requires authentication.

Status Codes

- 200 OK –

`GET /api/jobs/{id}`

REST endpoint to fetch list of jobs or retrieve/delete a job with job ID. Requires authentication.

Parameters

- **id** (*integer*) – A unique integer value identifying this job.

Status Codes

- 200 OK –

`DELETE /api/jobs/{id}`

REST endpoint to fetch list of jobs or retrieve/delete a job with job ID. Requires authentication.

Parameters

- **id** (*integer*) – A unique integer value identifying this job.

Status Codes

- 204 No Content – No response body

`GET /api/jobs/{id}/download_sample`
Download a sample from a given Job ID.

Parameters

- **id** (*integer*) – A unique integer value identifying this job.

Status Codes

- 200 OK –
- 400 Bad Request – No response body

`PATCH /api/jobs/{id}/kill`

Kill running job by closing celery tasks and marking as killed

Parameters

- **id** (*integer*) – A unique integer value identifying this job.

Status Codes

- 204 No Content – No response body

`GET /api/tags`

REST endpoint to pefrom CRUD operations on Job tags. Requires authentication. POST/PUT/DELETE requires model/object level permission.

Status Codes

- 200 OK –

`POST /api/tags`

REST endpoint to pefrom CRUD operations on Job tags. Requires authentication. POST/PUT/DELETE requires model/object level permission.

Status Codes

- 201 Created –

`GET /api/tags/{id}`

REST endpoint to pefrom CRUD operations on Job tags. Requires authentication. POST/PUT/DELETE requires model/object level permission.

Parameters

- **id** (*integer*) – A unique integer value identifying this tag.

Status Codes

- 200 OK –

`PUT /api/tags/{id}`

REST endpoint to pefrom CRUD operations on Job tags. Requires authentication. POST/PUT/DELETE requires model/object level permission.

Parameters

- **id** (*integer*) – A unique integer value identifying this tag.

Status Codes

- 200 OK –

`PATCH /api/tags/{id}`

REST endpoint to pefrom CRUD operations on Job tags. Requires authentication. POST/PUT/DELETE requires model/object level permission.

Parameters

- **id** (*integer*) – A unique integer value identifying this tag.

Status Codes

- 200 OK –

`DELETE /api/tags/{id}`

REST endpoint to pefrom CRUD operations on Job tags. Requires authentication. POST/PUT/DELETE requires model/object level permission.

Parameters

- **id** (*integer*) – A unique integer value identifying this tag.

Status Codes

- 204 No Content – No response body

CHAPTER

EIGHT

INTELOWL REDOC

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

HTTP ROUTING TABLE

/api