
mwdb-core

Release 2.12.0

CERT Polska

Apr 19, 2024

CONTENTS:

1	Features	3
1.1	What's changed?	3
1.2	Setup and configuration	17
1.3	User guide	27
1.4	Integration guide	81
1.5	Extra features	90
1.6	Developer guide	90
1.7	Remote instances guide	92
1.8	Karton integration guide	95
1.9	OpenID Connect authentication (Single Sign-On)	101
1.10	Rich attributes guide	102
1.11	Prometheus metrics	105
2	Indices and tables	107

Malware repository for automated malware collection and analysis systems. You can use it to index and share your collection of malware and extracted configurations, providing convenient, unified interface for your malware analysis pipeline.

Under the hood of mwdb.cert.pl service hosted by CERT.pl.

FEATURES

- Storage for malware binaries and configurations
- Tracking and visualizing relationships between objects
- Quick search using Lucene-based syntax
- Data sharing and user management mechanism
- Integration capabilities via webhooks and plugin system

1.1 What's changed?

This page describes the most significant changes in the following versions, that may be interesting especially for integration and plugin developers. We usually don't break API compatibility within major version, but plugins may have compatibility problems after minor mwdb-core upgrade.

For upgrade instructions, see [Upgrading mwdb-core to latest version](#).

1.1.1 v2.12.0

This release contains major changes in search mechanism and drops usage of Flask-RESTful, which might break plugin compatibility.

Complete changelog can be found here: [v2.12.0 changelog](#).

[Important change] Refactor of search engine

Search engine uses @> and @? PostgreSQL operators and can utilize GIN index on JSONB and ARRAY columns. It means that queries like:

- `file.name:"sample.exe"`
- `cfg.url:"https://example.com"`
- `attribute.url:"https://example.com"`

will work much faster. In addition, new search engine comes with various improvements:

- Both inclusive and exclusive ranges are allowed for datetime-columns
- Range boundaries can be reversed and are automatically sorted
- Escape parser is no longer regex-based and should be much more consistent

As search engine uses a bit different approach, some things may work a bit different:

- `>=`, `>`, `<`, `<=` are regular range operators, so they need to be put outside term
correct form: `size:>="5kB"`
incorrect form: `size:">=5kB"`

Complete description of changes can be found here: <https://github.com/CERT-Polska/mwdb-core/pull/906>

[Important change] Replaced Flask-RESTful with own lightweight implementation

Flask-RESTful is nice framework for creating REST API Flask services, but it's a bit abandoned which blocks us from further development and causes issues with newer Flask version. We switched to our own implementation of Resource classes.

For API users this change is completely transparent, but if you develop your own plugins, you should change imports:

```
- from flask_restful import Resource
+ from mwdb.core.service import Resource
```

Complete description of changes can be found here: <https://github.com/CERT-Polska/mwdb-core/pull/916>

[Important change] Changes in logging, introduced Prometheus metrics

We're buried in tons of non-meaningful logs without any option to set the verbosity. Examples of non-meaningful logs:

- information about usage of deprecated endpoints
- `'before_request'/'request'` logs which are useful for debugging hanging or slow endpoints, but only for debugging
- `threadName`, `moduleName`, `lineNo` which doesn't carry any useful information that can't be read from the log itself

That's why in v2.12.0 debug verbosity level was introduced and is turned off by default. If you want to turn on debug logs, you can enable it via `enable_debug_log=1` option in configuration.

As a replacement for flooding our disk space with log files, we introduced Prometheus metrics that can be tracked using Grafana platform. Read more about setup in [Prometheus metrics](#).

1.1.2 v2.11.0

Minor release with small QoL improvement: added forms to upload configs and blobs directly from eb UI.

Complete changelog can be found here: [v2.11.0 changelog](#).

1.1.3 v2.10.1

In v2.9.0 we switched from native ssdeep implementation to Python-based ppdeep library. Unfortunately, we have not taken into account the large impact on performance. This bugfix release goes one step backwards and requires **libfuzzy2** native library to be installed on server.

Complete changelog can be found here: [v2.10.0 changelog](#).

1.1.4 v2.10.0

Small release that includes minor improvements of existing features.

Complete changelog can be found here: [v2.10.0 changelog](#).

1.1.5 v2.9.0

This release contains changes in sharing mechanism and uses slightly different web build engine, which breaks web plugin compatibility.

Complete changelog can be found here: [v2.9.0 changelog](#).

[Important change] Opt-in counting of search results

In previous versions, click on “Search” button was counting all the results at once before showing the first part of them, which was time-consuming and heavy task for database. The actual count of results is usually not that useful for users to wait that much, unless they’re checking it on purpose.

In v2.9.0 there is extra “Count” button on search bar which switches between counting and non-counting search mode. We decided to make it non-counting by default for better search experience.

[Important change] Changes in sharing model

When user uploads sample, they can use **Share with** field to choose with whom this sample should be shared. This action is noted with **Added** reason type which is set both for uploader and all groups that have got permission to see the sample.

On the other hand, explicit shares are noted by different reason type: **Shared**. That difference between **Added** and **Shared** was not very clear, especially when inheritance comes into play so we decided to unify it.

v2.9.0 sets **Added** reason type only for uploader. All groups being part of **Share with** are noted with **Shared** just like other explicit shares. To make it even more visible: uploaders, groups and inherited shares are shown in separate sections.

Shares	
Added by:	
psrok1	Thu, 16 Mar 2023 18:07:56 GMT
Shared by psrok1:	
certpl-admins	Thu, 16 Mar 2023 18:07:56 GMT
certpl	Thu, 16 Mar 2023 18:07:56 GMT
everything	Thu, 16 Mar 2023 18:07:56 GMT
Share with group	Add
Inherited shares	
No shares to display	

All objects are migrated to the new scheme automatically after upgrade.

[Important change] Changed behavior of access_all_objects capability

Since v2.9.0, MWDB doesn't check permission table for users with access_all_objects and additional permission entries are not created.

Before that change, MWDB was adding explicit access permission for every new object and every group with enabled access_all_objects. Extra entries for groups with access_all_objects are removed during migration.

Initial everything group is no longer created on first configuration.

[Important change] Changes in web plugins engine

MWDB Core switched from [Create React App](#) to [Vite](#) which uses Rollup instead of Webpack.

1. First change you need to apply in plugin code is to rename all .js files to .jsx extension.

Remember to change all references in package.json as well.

```
- "main": "frontend/index.js",  
+ "main": "frontend/index.jsx",
```

2. @mwdb-web/commons is virtual package that is injected by plugin, so it's no longer installed into node_modules and should be removed from peerDependency section in package.json
3. If possible, don't use subpaths of @mwdb-web/commons/<module>, all required things should be imported from main package.

```
- import { APIContext } from "@mwdb-web/commons/api/context";  
+ import { APIContext } from "@mwdb-web/commons/api";
```

4. @mwdb-web/commons/api no longer serves api as default export. Use named import instead.

```
- import api from "@mwdb-web/commons/api";  
+ import { api } from "@mwdb-web/commons/api";
```

5. Finally, your main plugin file (index.jsx) should export function that returns plugin specification instead of exporting plugin specification directly.

```
- export default {  
+ export default () => ({  
  routes: [  
    <Route path='terms/:lang' element={<TermsOfUse />} />  
  ],  
  navdropdownAbout: [  
    <Link className="dropdown-item" to={'/terms/en'}>Terms of use</Link>  
  ],  
- }  
+ })
```

That function is called at very early stage of web application initialization. Plugins are imported before first render, so you don't have access to any useful context values though.

Plugin modules are imported dynamically (using [import\(\)](#) syntax). Check for any runtime errors in DevTools, especially noting messages like Plugin \${pluginName} failed to load.

[Important change] Replaced uWSGI with Gunicorn

certpl/mwdb Docker image uses [Gunicorn](#) instead of [uWSGI](#) for serving Python WSGI application. If you have uWSGI-dependent configuration customized via environment variables, you need to change it to Gunicorn equivalent.

Docker image by default spawns 4 sync workers and that number can be set via `GUNICORN_WORKERS` environment variable.

In addition, application code is no longer loaded lazily by default. If you want to keep that behavior, set `PRELOAD_APP` environment variable to 1.

For more information about configuring Gunicorn, check [Settings page in Gunicorn documentation](#).

1.1.6 v2.8.0

Release includes few improvements of performance, integration and search capabilities.

Complete changelog can be found here: [v2.8.0 changelog](#).

[Important change] Changes in database model

This release contains few model optimizations to improve query time, especially for tag queries.

- Relationship between Object and Tag was converted from many-to-many to one-to-many. Tag is represented by (object_id, tag_string) association instead of (object_id, tag_id) with tag in separate Table.
- Inheritance model is single-table based instead of join-based. All information is contained in single table Object instead of using separate tables for specialized fields, joined with common primary key.

Database migration may take a while during upgrade and requires extra space (~70% more) because major data must be copied from one table to another.

It's also recommended to **make a database backup before upgrade**.

[New feature] Rich attributes rendering

Starting from v2.8.0, MWDB Core supports rich attribute value rendering. For more information, see [Rich attributes guide](#).

[Important change] Upgrade to Karton v5.0.0

Changed name of `karton.ini` section that contains S3 client configuration from `[minio]` to `[s3]`.

In addition to this, you need to add a URI scheme to the address field and remove the secure field. If secure was 0, correct scheme is `http://`. If secure was 1, use `https://`.

```

- [minio]
+ [s3]
  access_key = karton-test-access
  secret_key = karton-test-key
- address = localhost:9000
+ address = http://localhost:9000
  bucket = karton
- secure = 0

```

v5.0.0 maps [minio] configuration to correct [s3] configuration internally, but [minio] scheme is considered deprecated and can be removed in further major release.

1.1.7 v2.7.0

Release includes few improvements of security, integration and search capabilities.

Complete changelog can be found here: [v2.7.0 changelog](#).

[Important change] Changed API key generation and handling

MWDB Core uses JWT tokens for various resources that require special authorization. One of them is managed directly by the end user: API keys. In this release, we slightly changed the implementation to improve security and make them more compliant with [RFC7519](#).

That's why it's recommended to regenerate your API keys at some point after upgrade. All previously generated API keys will be honored by further 2.x.x releases of MWDB Core, but should be considered deprecated.

The next important change is that API key token is shown **only just after creation** and token can't be regenerated for existing API key.

[New feature] Configurable rate limits

From now, you doesn't have to rely on arbitrary hardcoded rate limits like before 2.7.0. Now, you're open to configure it depending on your needs. You can use different limits for specific endpoints and HTTP methods.

For more information, read [Rate limit configuration](#) section.

[New feature] Relative date-time ranges in search

v2.7.0 comes with the next improvements in search. The new thing is support for relative date-time ranges.

```
upload_time:>=2h or upload_time:[2h TO *]
```

For more information, read [Query syntax: relative timestamps](#).

[Improvement] New object hooks accessible for plugins

In previous versions, MWDB Core was able to notify your plugins only of limited set of simple actions like creation of the new object, added tag or comment. From v2.7.0 you are able to integrate with much broader set of actions including object removals, changes in attributes and even administrative actions like creation of new user account.

Complete list of hooks can be found in [Available hooks](#) section.

1.1.8 v2.6.0

This release implements multiple feature requests and improvements. The most noteworthy are support for OpenID Connect authentication and new Attribute API that allows to store whole JSON objects as attribute values.

Another noticeable change is redesigned Shares box. In addition, we swapped the positions of Attributes box and Shares box, so main part of view contains the most important information about object. In future, we plan to enrich attributes with extended rendering features, so you can place and visualize complete analysis report just by using Attributes feature. If you have any ideas regarding that, [let us know by creating an issue!](#)

Complete changelog can be found here: [v2.6.0 changelog](#).

[New feature] Support for OpenID Connect authentication

Users can bind their MWDB accounts with external identity provider, so they can authenticate via corporate Single Sign-On.

Feature was tested on Keycloak, but feature should support other OpenID Providers as well.

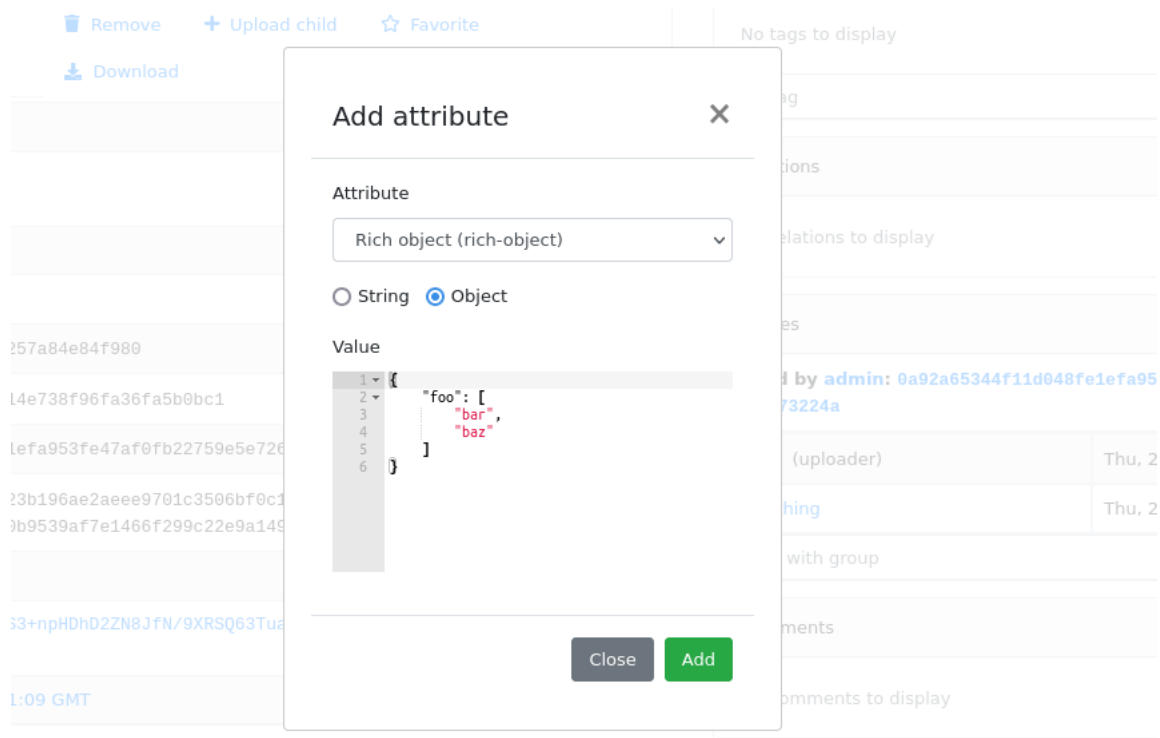
For more instructions, read *[OpenID Connect authentication \(Single Sign-On\)](#)*.

[New feature] New Attribute API - support for JSON values

Before 2.6.0, attributes supported only relatively short key-value string pairs and there were no good place for complex structures like:

- enrichments from other services
- file static analysis information like code signing, sections, list of resources
- information about produced dumps from sandbox
- [apivectors](#)

That's why we decided to migrate from plain strings to [JSONB type](#) in internal attribute value representation. We also designed a new Attribute API to operate on JSON objects rather than simple values.



Upload time	Thu, 23 Dec 2021 08:51:09 GMT
Attributes	
Rich object	<pre>(object) { "foo": ["bar", "baz"] }</pre>

Attribute API is the new set of endpoints and request fields. You can easily recognize them as we name them *attributes* instead of *meta(keys)*.

attribute			
POST	/api/{type}/{identifier}/attribute	Add object attribute	✓ 🔒
GET	/api/{type}/{identifier}/attribute	Get object attributes	✓ 🔒
DELETE	/api/{type}/{identifier}/attribute/{attribute_id}	Delete object attribute	✓ 🔒
POST	/api/attribute	Create attribute key	✓ 🔒
GET	/api/attribute	Get list of attribute keys	✓ 🔒
DELETE	/api/attribute/{key}	Delete attribute key	✓ 🔒
GET	/api/attribute/{key}	Get attribute key details	✓ 🔒

For compatibility reasons: deprecated Metakey API just coerces object values to strings. Keep in mind that strings

'{"foo": "bar"}' and objects {"foo": "bar"} are indistinguishable after type coercion, so don't use that API for attribute keys that are intended to contain JSON objects.

Because of used representation, JSON dictionaries are not ordered. Attribute key still behaves as set: all values under the same attribute key are guaranteed to be unique and when we try to add the same value twice, the second one won't be added.

Attribute API exposes attribute value identifier that can be used for removing the specific attribute value. Metakeys were identified directly by *key*, *value* tuple but it wasn't convenient for objects because these values can be pretty huge.

Response body

```
{
  "attributes": [
    {
      "id": 1,
      "key": "rich-object",
      "value": {
        "foo": [
          "bar",
          "baz"
        ]
      }
    }
  ]
}
```

More information can be found in [#413 feature draft on Github](#). At the time of 2.6.0 release, not all planned Attribute API extensions are implemented, but we're going to deliver them in future.

[New feature] Configurable timeouts in MWDB Core

Before 2.6.0, all MWDB Core timeouts were hardcoded directly in Web client code:

- 8 seconds timeout for API endpoints
- 60 seconds timeout for file upload

Timeout only interrupted HTTP request processing, but all SQL statements were still processed on the backend. In addition, it wasn't enforced for other REST API clients.

In 2.6.0, we introduced set of timeouts that are configured on backend side:

- `statement_timeout` (integer) - If set, database server aborts any SQL statement that takes more than the specified number of milliseconds.
- `file_upload_timeout` (integer) - File upload process will be terminated by Web client if it takes more than this parameter value in milliseconds. Default value is 60000 ms.
- `request_timeout` (integer) - HTTP request will be terminated by Web client if it takes more than this parameter value in milliseconds. Default value is 20000 ms.

If you want to enforce effective timeout on the backend, set `statement_timeout` to non-zero value, but keep in mind that it may interrupt some long-running operations. Other timeouts are suggestions for REST API client (exposed via `/api/server`) and are set on Web client level.

Default Web timeout is now a bit longer and set to 20 seconds instead of 8 seconds.

[New feature] Storing alternative names for sample

MWDB stores all unique names for sample that it was uploaded with. They are exposed via “Variant file names” field in Web UI object view.

File details	
<div> <div>Details</div> <div>Relations</div> <div>Remove</div> <div>+ Upload child</div> <div>Favorite</div> </div> <div> <div>Preview</div> <div>Download</div> </div>	
File name	debug_dumps.json
Variant file names	dumps_info.json dumps.json
File size	3.44 kB
File type	JSON data

[New feature] Transactional tag adding along with object upload

From 2.6.0 you can include tags as additional upload arguments. Previously that feature was supported only for attributes.

In that way, new object will appear in repository with all tags set via single database transaction, so you can avoid race-conditions when tags are required immediately after object is spawned.

```
from mwdblib import MWDB # >= 4.0.0

mwdb = MWDB()
...
mwdb.upload_file("sample", contents, tags=["vt:unknown"])
```

[New feature] New search features

2.6.0 release comes with new handful search fields:

- ``comment_author:<login>`` search field that allows to search for objects commented by selected user
- ``upload_count:<number>`` search field that allows to search for objects related with more than N different user uploads.
- ``multi:`` search field that allows to search for multiple hashes separated by spaces

The last one is used by Web client to automatically transform copy-pasted hashes, placed in search field.

1.1.9 v2.5.0

Small release that includes minor improvements on Karton integrations and other existing features.

Complete changelog can be found here: [v2.5.0 changelog](#).

1.1.10 v2.4.0

Small release that includes minor improvements of existing features.

Complete changelog can be found here: [v2.4.0 changelog](#).

1.1.11 v2.3.0

This release is focused mainly on MWDB administration improvements and further UI refactoring. In addition, Karton integration is now available out-of-the-box, without need of extra plugins.

Complete changelog can be found here: [v2.3.0 changelog](#).

[New feature] Built-in Karton integration

Karton integration is now included as a built-in part of MWDB Core. In addition, MWDB-Core 2.3.0 includes automatic migration spawned on `mwdb-core configure` for `mwdb-plugin-karton` users.

If you use `mwdb-plugin-karton` in your setup: remove the plugin before upgrade. For more instructions, read [Karton integration guide](#).

[New feature] registered group

Before v2.3.0, it was difficult to setup guest accounts. To implement that, we added new capabilities:

- `adding_files` which is required for file upload
- `manage_profile` which is required for changes in user authentication (API keys, reset password)
- `personalize` that enables personalization features like Favorites or Quick queries.

But it was still painful to manage having only `public` group, which defines capabilities for all users in MWDB. That's why we created new predefined group called `registered`. Within migration, all capabilities are moved to `registered` group (with new one enabled) and all existing users are added to that group.

`registered` group behavior is similar to `public`: new users are added by default and don't see each other within the group. The only difference is that `registered` group is mutable, so any user can be easily removed from `registered`.

By removing `registered` membership, you can make guest account with disabled file upload and personalization features!

If you don't like the split between `public` and `registered` in your instance, you can just remove the `registered` group and manually recover capabilities settings in `public`.

[API] Plugin information is no longer available for non-admin users

Plugin information was moved from `/api/server` endpoint to `/api/server/admin`. Information was also moved from `/about` to the new `/settings` view in UI.

In addition `/api/docs` also requires authentication.

[API] Removed `managing_attributes` capability

`managing_attributes` behavior was inconsistent, because `manage_users` was still required e.g. to set up permissions for attribute key. From now, `manage_users` is required for all administration tasks, including setting up new attribute keys.

1.1.12 v2.2.0

In 2.2.0 frontend part was heavily refactored, so some Web plugins may stop working properly without proper upgrade. Follow the sections below to learn about the most important changes.

Complete changelog can be found here: [v2.2.0 changelog](<https://github.com/CERT-Polska/mwdb-core/releases/tag/v2.2.0>)

[New feature] Remote API feature

There is new feature that allows to connect directly to the other MWDB Core instance (e.g. `mwdb.cert.pl`). This allows us to pull or push objects and discover new objects in the remote repository. At the time of release, feature is considered **beta** so don't rely too much on it. If you want to test it, we'll be glad for feedback!

Read *Remote instances guide* to learn more.

[API] New file download endpoint

Requests to MWDB API are mostly authenticated via Authorization header (instead of Cookie which is managed by browser), so there is no easy way to let a browser download a file. That's why download process looked like below:

1. POST `/request/sample/{identifier}` is used to get partial download URL with generated token
2. GET `/api/download/{access_token}` is used to download the actual file

So we had always two HTTP requests to download the file contents. That's why in 2.2.0 you can download a file without intermediate token via new `/file/{identifier}/download` endpoint.

- GET `/file/<identifier>/download` returns file contents for Authorization: Bearer requests
- GET `/file/<identifier>/download?token=<token>` for download token authorization that doesn't require Authorization header.
- POST `/file/<identifier>/download` that generates download token.

Old endpoints are considered obsolete and may be removed in further major release.

[Backend] Typed-Config is no longer embedded in mwdb package

typedconfig is no longer embedded in mwdb.core package, because it's used as external dependency.

For plugin compatibility, change

```
from mwdb.core.typedconfig import ...
```

to

```
from typedconfig import ...
```

[Web] React Context is used instead of Redux

That's the most breaking change, because we no longer use React-Redux for handling the global state. Instead we use bunch of React Context providers that are available also for plugins.

So if you use code presented below to check if current user has required capability:

```
import {connect} from 'react-redux';

...

function mapStateToProps(state, ownProps)
{
  return {
    ...ownProps,
    isKartonManager: state.auth.loggedUser.capabilities.includes("karton_manage"),
  }
}

export default connect(mapStateToProps)(KartonAttributeRenderer);
```

rewrite it like below:

```
import React, { useContext } from 'react';
import { AuthContext } from "@mwdb-web/commons/auth";

export default function KartonAttributeRenderer(props) {
  const auth = useContext(AuthContext);
  const isKartonManager = auth.hasCapability("karton_manage");

  ...
}
```

Learn more about React Context in [React documentation](#).

[Web] Extra routes must be passed as instantiated components

This is specific for `Switch` component from `React-Router`. Component must be instantiated when passed as a children of `Switch`, instead it doesn't work correctly.

It worked before 2.2.0 because default route wasn't handled. From 2.2.0 incorrectly defined routes will be unreachable.

Instead of:

```
export default {
  routes: [
    (props) => (
      <ProtectedRoute
        condition={
          props.isAuthenticated &&
          props.capabilities &&
          props.capabilities.includes("mquery_access")
        }
        exact
        path="/mquery"
        component={MQuerySearchView}
      />
    )
  ]
}
```

use:

```
function MQueryRoute(props) {
  const auth = useContext(AuthContext);
  return (
    <ProtectedRoute
      condition={auth.hasCapability("mquery_access")}
      {...props}
    />
  )
}

export default {
  routes: [
    <MQueryRoute exact path="/mquery" component={MQuerySearchView}/>,
  ],
}
```

[Web] `props.object` may be undefined for `ShowObject` extensions. Use `ObjectContext` instead

`ShowObject` components use `ObjectContext` natively which may affect some plugins that extend parts of this view

Instead of

```
export function MTrackerStatusBanner(props) {
  const objectType = props.object.type;
  const objectId = props.object.id;
```

(continues on next page)

(continued from previous page)

```

    ...
}

export default {
  showObjectPresenterBefore: [MTrackerStatusBanner],

```

use

```

import React, { useContext } from "react";

import { ObjectContext } from "@mwdb-web/commons/context";

export function MTrackerStatusBanner(props) {
  const objectState = useContext(ObjectContext);
  const objectType = objectState.object.type;
  const objectId = objectState.object.id;

  ...
}

export default {
  showObjectPresenterBefore: [MTrackerStatusBanner],

```

1.2 Setup and configuration

1.2.1 Installation and configuration with Docker Compose

The quickest way setup MWDB is to just clone the repository and use Docker-Compose with all batteries included.

```
$ git clone https://github.com/CERT-Polska/mwdb-core.git
```

After cloning repository, the first step is to go to the mwdb-core directory and generate configuration using `./gen_vars.sh` script. Generated variables can be found in `mwdb-vars.env`.

```
$ ./gen_vars.sh
Credentials for initial mwdb account:

-----
Admin login: admin
Admin password: la/Z7MsmKA3UxW8Psrk1Opap
-----
```

Please be aware that initial account will be only set up on the first run. If you
 → already have a database with at least one user, then this setting will be ignored for
 → security reasons. You can always create an admin account manually by executing a
 → command. See "flask create_admin --help" for reference.

Then build images via `docker-compose build` and run MWDB via `docker-compose up -d`.

Your MWDB instance will be available on default HTTP port (80): <http://127.0.0.1/>

If you want to use Docker Compose for MWDB development, check out [Developer guide](#).

1.2.2 Standalone installation

Step 1.: Prerequisites

MWDB was tested on Debian-based systems, but should work as well on other Linux distributions.

For production environments, you need to install:

- **PostgreSQL database** (minimum supported version: 12, <https://www.postgresql.org/download/linux/debian/>)
- libfuzzy2 for ssdeep evaluation
- other native dependencies listed below

```
$ apt install gcc libfuzzy-dev python3-dev python3-venv postgresql-client postgresql-  
common
```

Optionally you can install:

- **Docker engine and Docker-Compose** if you want to use the Docker-based setup (<https://docs.docker.com/engine/install/>)
- **Redis database** needed by extra features like rate-limiting (<https://redis.io/topics/quickstart>)

It's highly recommended to create a fresh `virtualenv` for local MWDB installation:

```
# Create virtual environment  
~/mwdb$ python3 -m venv venv  
  
# Activate virtual environment  
~/mwdb$ source ./venv/bin/activate  
  
(venv) ~/mwdb$
```

Note: If you are a bit overwhelmed by setting up PostgreSQL database and you are looking for quick setup method just for testing: first make sure you have Docker and Docker-Compose installed and go to the *Alternative setup with Docker-Compose*.

You can also setup temporary PostgreSQL database container using Docker image:

```
$ docker run -d --name mwdb-postgres -e POSTGRES_DB=mwdb -e POSTGRES_USER=mwdb -e  
POSTGRES_PASSWORD=mwdb -p 127.0.0.1:54322:5432 postgres
```

The connection string is: `postgresql://mwdb:mwdb@127.0.0.1:54322/mwdb`

Step 2.: Installation and configuration

The recommended installation method is pip:

```
$ pip install mwdb-core
```

After installing `mwdb-core` package, let's start with `mwdb-core` command:

```
$ mwdb-core

[!] Wrong MWDB configuration.

Use 'mwdb-core configure' to setup your MWDB instance.

Usage: mwdb-core [OPTIONS] COMMAND [ARGS]...

    MWDB malware database

Options:
  --help  Show this message and exit.

Commands:
  configure  Configure MWDB instance
  db         Perform database migrations.
  routes     Show the routes for the app.
  run        Run a development server.
  shell      Run a shell in the app context.
```

Then, use `mwdb-core configure` to provide first configuration for your MWDB server.

```
$ mwdb-core configure

Where do you want to place MWDB local files?

1) Global directories (/var/lib/mwdb, /etc/mwdb)
2) Local user directory (/home/steve/.mwdb)
3) Current directory
: 3
```

For first installation we recommend to install everything in current folder via 3 option. If you want to install MWDB system-wide or locally for user: choose 1 or 2.

Then, input the connection string for PostgreSQL database. The database must be online and reachable at the time of configuration. After that, you will be asked for path for uploads and instance base URL. If the default value is ok, press Enter:

```
PostgreSQL database connection string [postgresql://localhost/mwdb]:
Uploads storage path [./uploads]:
Base public URL of Malwarecage service [http://127.0.0.1]:
```

Depending on the installation type, your configuration will be stored in `mwdb.ini` file and can be changed any time you want:

```
[+] Configuration stored in ./mwdb.ini file!
```

After storing the configuration, the `configure` command will initialize database schema:

```
[+] Configuration already initialized... skipping
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade -> 2e692ea445a1, Initial version
...
```

(continues on next page)

(continued from previous page)

```
Provide password for Malwarecage 'admin' account:
Repeat password:
```

Finally, you will be asked for the admin account password that will be used as the management account.

```
MWDB configured successfully!
```

```
Use 'mwdb-core run' to run the server.
```

And you are done! `run` command will start the Flask server:

```
$ mwdb-core run
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Your MWDB instance will be available on port 5000 (use `--port` to change that): <http://127.0.0.1:5000/>

Keep in mind that Flask server is meant to be used as development server and **is not suitable for production**. See also: <https://flask.palletsprojects.com/en/2.2.x/server/>

Warning: In standalone setup, remember to run `mwdb-core configure` after each version upgrade to apply database migrations.

Step 3.: Setting up gunicorn and nginx

It's recommended to deploy Flask applications using dedicated WSGI server. We highly recommend Gunicorn as it's used in our Docker images and combine it with Nginx serving as proxy server for best security and performance

See also:

<https://flask.palletsprojects.com/en/2.2.x/deploying/gunicorn/>

<https://docs.gunicorn.org/en/latest/deploy.html#deploying-gunicorn>

Proper configuration files and templates used in our Docker images can be found in [docker directory on our Github repository](#)

1.2.3 Upgrading mwdb-core to latest version

For standalone installation (pip-based), upgrade `mwdb-core` package to the latest version.

```
$ pip install -U mwdb-core
```

Then apply required database migrations using `mwdb-core configure`.

```
$ mwdb-core configure
```

If you use Docker-based environment, just pull the latest changes from repository and rebuild the images. Database migrations will be applied as a part of container startup.

1.2.4 Storing files in S3 Compatible storage (MinIO, AWS S3)

New in version 2.1.0.

By default, all files uploaded to mwdb-core are stored in the local file system (under path provided in `uploads_folder` configuration key). It's the most universal and simplest way, but not sufficient if our scale requires distributed storage or cloud-based infrastructure. In that case we can use solutions like [MinIO](#) or another S3-compatible object-based storage.

If you want to store files using object storage, open the `mwdb.ini` file and set the following keys:

```
storage_provider = s3
hash_pathing = 0
s3_storage_endpoint = <storage endpoint>
s3_storage_access_key = <storage access key>
s3_storage_secret_key = <storage secret key>
s3_storage_bucket_name = <storage bucket name>

# optional (for AWS S3)
s3_storage_region_name = <AWS S3 region name>
# optional (for TLS)
s3_storage_secure = 1
# optional (for AWS IAM role authentication)
s3_storage_iam_auth = 1
```

If you use Docker-based setup, all the configuration can be set using environment variables (e.g. `MWDB_STORAGE_PROVIDER=s3`).

Note: If you are using `karton`, we highly recommend creating a separate bucket for it. Failing to do so might result in data loss caused by `karton`'s garbage collector.

1.2.5 Setting higher upload size limit in Docker

mwdb-core allows to set maximum upload size via `max_upload_size` parameter in configuration (see also [Advanced configuration](#) section).

mwdb-core package doesn't enforce any limitation by default, but if you use Docker images: nginx configuration in mwdb-web image have set 50M limit using `client_max_body_size` option. If you want to set different limitation for Docker environment, use `NGINX_MAX_UPLOAD_SIZE` environment variable to set `client_max_body_size` option.

If you want to customize other nginx settings (e.g. timeouts), you can also provide your own `nginx.conf.template` by building your own image based on `certpl/mwdb-web`. More information can be found in [#927 issue discussion on Github](#).

1.2.6 Advanced configuration

mwdb-core can be configured using several methods. Configuration is read from the following sources, ordered by priority:

- Environment variables (MWDB_XXX)
- ./mwdb.ini configuration file in current directory
- ~/.mwdb-core/mwdb.ini in home directory
- /etc/mwdb-core/mwdb.ini as global configuration

Sources are overriding each other depending on the priority, which means that environment value MWDB_ENABLE_PLUGINS=0 will override the enable_plugins = 1 entry in mwdb.ini file.

The format for environment variable is <SECTION>_<KEY> uppercase. The default section for all mwdb-core settings is mwdb. Plugins can also be configured by mwdb.ini file using their own sections. Check appropriate section name in plugin's documentation.

Basic settings:

- **postgres_uri** (string, required) - PostgreSQL database connection string
- **secret_key** (string, required) - Secret key used by Flask application and to sign authentication tokens. Change of that value will invalidate all sessions and all registered API keys.
- **uploads_folder** (string, required) - Path where MWDB stores uploaded files
- **base_url** (string) - Base URL of MWDB web application, used for registration mail templates. Default is `http://127.0.0.1`
- **file_upload_timeout** (integer) - File upload process will be terminated if it takes more than this parameter value in milliseconds. Default value is 60000 ms.
- **statement_timeout** (integer) - Database statement_timeout parameter. Database server aborts any statement that takes more than the specified number of milliseconds.
- **request_timeout** (integer) - HTTP request will be terminated if it takes more than this parameter value in milliseconds. Default value is 20000 ms.
- **instance_name** - (string) - custom name for local MWDB instance. Default value is "mwdb".

Web application settings:

- **serve_web** (0 or 1) - By default, web application is served by mwdb-core application (1). If you want mwdb-core to serve only API and host web application by yourself (e.g. using nginx), you can turn off serving static files by setting this option to 0.
- **web_folder** (string) - Path to web application static files. By default, web application files are served from pre-compiled bundle embedded to Python package. If you want to use plugins that are incorporating additional frontend features, you need to rebuild the web application and serve them from your own path.
- **flask_config_file** (string) - additional file containing Flask configuration (.py)
- **admin_login** (string) - administrator account name
- **admin_password** (string) - initial password for administrator account
- **use_x_forwarded_for** (0 or 1) - Set this to 1 if MWDB backend is behind reverse proxy, so X-Forwarded-For header is correctly translated to request.remote_addr value. Set by default to 1 in certpl/mwdb Docker image.

Plugin settings:

- **enable_plugins** (0 or 1) - If you want to turn off all plugins, set this option to 0. Default is 1.

- `plugins` (list of strings, separated by commas) - List of installed plugin module names to be loaded, separated by commas
- `local_plugins_folder` (string) - Directory that will be added to `sys.path` for plugin imports. Useful if you want to import local plugins that are not redistributable packages.
- `local_plugins_autodiscover` (0 or 1) - Autodiscover plugins contained in `local_plugins_folder` so you don't need to list them all manually in `plugins`. Default is 0.

Storage settings:

- `max_upload_size` (integer) - Maximum upload size in bytes. Keep in mind that this value refers to whole upload request (`Content-Length` from request header), so the maximum file size is smaller than that by +/- 500B (because of additional payload with metadata). Default is `None`, which means there is no limit.
- `storage_provider` (disk or s3) - If you want to use S3-compatible object storage instead of local file system, set this option to `s3`. Default is `disk`.
- `hash_pathing` (0 or 1) - Should we break up the uploads into different folders. If you use S3-compatible storage, recommended option is 0 (default: 1).
- `s3_storage_endpoint` (string) - S3 API endpoint for object storage. Required if you use S3-compatible storage.
- `s3_storage_access_key` (string) - S3 API access key for object storage. Required if you use S3-compatible storage.
- `s3_storage_secret_key` (string) - S3 API secret key for object storage. Required if you use S3-compatible storage.
- `s3_storage_bucket_name` (string) - S3 API bucket name for object storage. Required if you use S3-compatible storage.
- `s3_storage_region_name` (string, optional) - S3 API storage region name. Used mainly with AWS S3 storage (default is `None`).
- `s3_storage_secure` (0 or 1) - Use TLS for S3 API connection (default is 0).
- `s3_storage_iam_auth` (0 or 1) - Use AWS IAM role for S3 authentication (default is 0). If 1, then `s3_storage_access_key` and `s3_storage_secret_key` aren't required.

Extra features:

- `enable_rate_limit` (0 or 1) - Turns on rate limiting. Requires Redis database and `redis_uri` to be set. Default is 0.
- `enable_registration` (0 or 1) - Turns on user registration features. Requires additional configuration. Default is 0.
- `enable_maintenance` (0 or 1) - Turns on maintenance mode, making MWDB unavailable for users other than admin. Default is 0.
- `enable_json_logger` (0 or 1) - Enables JSON logging which may be more convenient for automated log processing. Default is 0.
- `enable_prometheus_metrics` (0 or 1) - Enables Prometheus metrics (`/api/varz` endpoint)
- `enable_debug_log` (0 or 1) - Enables debug logging
- `redis_uri` (string) - Redis database connection string, required by rate limiter.
- `remotes` (comma separated strings) - list of MWDB remotes (experimental)
- `enable_hooks` (0 or 1) - enable plugin hooks
- `enable_oidc` (0 or 1) - enable OIDC (experimental)

- `listing_endpoints_count_limit` (integer) - Limits number of objects returned by listing endpoints. Default is 1000.

Registration feature settings:

- `mail_smtp` (string) - SMTP connection string (`host:port`)
- `mail_from` (string) - From field value used in e-mails sent by MWDB
- `mail_username` (string) - SMTP user name
- `mail_password` (string) - SMTP user password
- `mail_tls` (0 or 1) - Enable STARTTLS
- `mail_templates_folder` (string) - Path to the directory containing custom mail templates
- `recaptcha_site_key` (string) - ReCAPTCHA site key. If not set - ReCAPTCHA won't be required for registration.
- `recaptcha_secret` (string) - ReCAPTCHA secret key. If not set - ReCAPTCHA won't be required for registration.

1.2.7 Rate limit configuration

New in version 2.7.0.

mwdb-core service has implemented rate limiting feature. Each limit for HTTP method can contain a few conditions (space separated).

Default limits were applied for HTTP methods. The default values are as below:

- GET method: 1000/10second 2000/minute 6000/5minute 10000/15minute
- POST method: 100/10second 1000/minute 3000/5minute 6000/15minute
- PUT method: 100/10second 1000/minute 3000/5minute 6000/15minute
- DELETE method: 100/10second 1000/minute 3000/5minute 6000/15minute

User can override these limits for individual endpoints by placing new limits in `mwdb.ini` - in section `[mwdb_limiter]`. Each line in `[mwdb_limiter]` section should have a pattern - `<resourcename>_<httpmethod> = limit_values_space_separated`.

Example rate-limit records in `mwdb.ini` file are as below

```
[mwdb_limiter]
file_get = 100/10second
textblob_post = 10/second 1000/minute 3000/15minute
attributedefinition_delete = 10/minute 100/hour
```

Above records establish request rate limits for endpoints:

- GET `/api/file` to value: 100 per 10 seconds
- POST `/api/blob` to values: 10 per second, 1000 per minute and 3000 per 15 minutes
- DELETE `/api/attribute/<key>` to values: 10 per minute and 100 per hour

Other endpoints are limited by default limits.

Note: Complete list of possible rate-limit parameters is placed in `mwdb-core\mwdb\core\templates\mwdb.ini.tpl` file - section `mwdb_limiter`.

If your MWDB instance uses standalone installation and MWDB backend is behind reverse proxy, make sure that `use_x_forwarded_for` is set to 1 and your reverse proxy correctly sets X-Forwarded-For header with real remote IP.

1.2.8 Using MWDB in Kubernetes environment

Here are examples of YAML specifications for k8s deployments:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mwdb
  namespace: mwdb-prod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mwdb
  template:
    metadata:
      labels:
        app: mwdb
    spec:
      initContainers:
        # Init container that performs database migration on upgrade
        - env:
            # Provide secrets and first configuration admin password via
            ↪environment vars
          - name: MWDB_SECRET_KEY
            valueFrom:
              secretKeyRef:
                key: key
                name: secret-mwdb-secret-key
          - name: MWDB_POSTGRES_URI
            valueFrom:
              secretKeyRef:
                key: uri
                name: secret-mwdb-database-uri
          - name: MWDB_ADMIN_PASSWORD
            valueFrom:
              secretKeyRef:
                key: password
                name: secret-mwdb-admin-password
          - name: MWDB_BASE_URL
            value: https://mwdb.cert.pl
      image: certpl/mwdb:v2.9.0
      imagePullPolicy: Always
      name: mwdb-migration-container
      command: [ 'sh', '-c', '/app/venv/bin/mwdb-core configure -q' ]
      containers:
        - env:
            - name: GUNICORN_WORKERS
            - name: MWDB_SECRET_KEY
```

(continues on next page)

(continued from previous page)

```

      valueFrom:
        secretKeyRef:
          key: key
          name: secret-mwdb-secret-key
    - name: MWDB_POSTGRES_URI
      valueFrom:
        secretKeyRef:
          key: uri
          name: secret-mwdb-database-uri-key
    - name: MWDB_BASE_URL
      value: https://mwdb.cert.pl
    - name: MWDB_ENABLE_KARTON
      value: '1'
    - name: MWDB_S3_STORAGE_ENDPOINT
      value: s3.cert.pl
    # ... more configuration
  image: certpl/mwdb:v2.9.0
  imagePullPolicy: Always
  livenessProbe:
    httpGet:
      path: /api/ping
      port: 8080
  readinessProbe:
    httpGet:
      path: /api/ping
      port: 8080
  name: mwdb-container
  volumeMounts:
    - mountPath: /etc/karton/karton.ini
      name: karton-config-volume
      subPath: config.ini
  volumes:
    - name: karton-config-volume
      secret:
        defaultMode: 420
        items:
          - key: config-file
            path: config.ini
        secretName: karton-config

```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mwdb-web
  namespace: mwdb-prod
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mwdb-web
  template:
    metadata:

```

(continues on next page)

(continued from previous page)

```

labels:
  app: mwdb-web
spec:
  containers:
  - env:
    # Provide internal URI to backend service for nginx reverse proxy
    - name: PROXY_BACKEND_URL
      value: http://mwdb-service:8080/
    image: certpl/mwdb-web:v2.9.0
    livenessProbe:
      httpGet:
        path: /
        port: 80
    name: mwdb-web-container
    readinessProbe:
      httpGet:
        path: /
        port: 80

```

1.3 User guide

This guide will help you to learn basic MWDB concepts step by step. It is not only limited to the potential use-cases of your own mwdb-core instance, but also recommended for mwdb.cert.pl service users who want improve their skills in exploring MWDB database.

1.3.1 1. Introduction to MWDB

A brief history (Why do I need MWDB?)

It's just a matter of time before researcher faces a problem of maintaining the constantly-increasing set of malware samples. It's not only about the binaries, but also all of the information that comes as a result of malware analysis like static configurations, associations between files and configs, metadata etc.

In CERT.pl we tried to bring the order in our dataset with [VxCage project](#) authored by Claudio “nex” Guarnieri. It was then heavily modified by Maciej “mak” Kotowicz and extended with simple web interface. That extension allowed to cooperate on building the MWDB (**Malware Database**) with other colleagues and researchers.

Over time, we've decided to rewrite everything from scratch deriving lessons learned from the first version. The goal was to create a highly-efficient solution that could be shared with broad malware research community, available both as a service and open-source software.

MWDB is not just a simple database. It has several roles in mwdb.cert.pl service:

- Aggregates various feeds and malware collections.
- Comes up with a well-organized model that allows to discover relations between samples, families and campaigns.
- Provides unified interface for malware analysis pipeline.
- Allows to share our malware insights and makes them available for malware research community.

Main views

MWDB UI is quite rich and can be intimidating at first glance. Most things can be reached directly from the navbar:



Let's go through them shortly:



- **Samples:** list of recently uploaded files. The files are tagged and can be queried easily.
- **Configs:** configs are structured data (in a JSON format), usually statically extracted from the samples. You can store things like encryption keys or C&C servers used by samples there.
- **Blobs:** blobs are unstructured text files, usually configuration or webinjects downloaded from C&C server (sometimes also plaintext data from the binary, like ransom notes).
- **Upload:** can be used to upload your own samples to the system
- **Admin:** accessible only for administrators, allows to manage your MWDB instance (e.g. create new users and groups)
- **Search:** untyped search, which returns all types of objects at once, but is hard-limited to 10.000 records. For most cases: we recommend Quick search field placed above Recent views.
- **Statistics:** a summary of the malware configurations that were uploaded recently. You can use this view in mwdb.cert.pl service to determine if we support a specific malware family or when did we last see a specimen of the family you're interested in.
- **About:** various links about MWDB version and REST API documentation.

Recent objects view

MWDB welcomes us with a list of recently uploaded samples.

Search (Lucene query or hash)...

Quick query: Only uploaded by me Exclude public Exclude feed:* Only ripped:* Add +

Name/Hash	Size/Type	Tags	First seen
 Name: 431dfcf9e1e0876288f00484744197463... SHA256: 431dfcf9e1e0876288f004_3bde11e6e38f MD5: f816cb5a107fd66329004f96819dccc26	Size: 151552 Type: Composite Document File V2 Docum...	document:win32:xls	Thu, 08 Oct 2020 14:33:37 GMT
 Name: 30b54ec5603aa0b31b5351c9b664bdf65... SHA256: f60d46232c9d94913dc6a77_6e443f9864fd MD5: b30ed3b7a54b2faace5b663492fd0ee9	Size: 474624 Type: PE32 executable (GUI) Intel 80386, f...	emotet ripped:emotet	Thu, 08 Oct 2020 14:33:32 GMT

Recent views allow to see basic information about latest objects in the repository and interactively explore the dataset using Lucene-based queries and clickable fields. If sample was uploaded within last 72 hours, it is additionally marked with yellowish background. Yellow color is a bit more intense if file was uploaded at least 24 hours ago.

If you simply want to find a sample with specific SHA256 hash: just paste this hash into a search box and press ENTER.

sha256: has been added automatically to your query!

Interactive search is even more powerful. For example: you can filter out samples with specific tags just by clicking on them. Reset your query and click on tag to include only these samples that are currently in your interest.

Sample view

Let's click on one of the hashes to navigate to the **detailed sample view**.

Here you can see all the details about file. Left side contains four tabs. The first one called **Details** presents basic file information like original file name, size, file type and hash values.

On the right side of view you can see tags, relations with other objects, attributes containing some extra information and the comments section. All of these things will be described in details in further chapters of this guide.

Let's come back to the tabs. The second tab is **Relations** that shows graph representation of relations with other objects. You can expand next levels of graph by clicking on nodes.

The third tab is **Preview** where you can quickly view the file contents, both in text mode or hex view.

Finally you can download the sample file just by clicking **Download** button or mark it as **Favorite** so that you can easily find it in the search engine.

The fourth tab called **Static config** is optional and shows only if there are some configurations related with sample.

This is only an overview of the most basic MWDB functions. Follow this guide to learn in details how to effectively use the repository, upload new objects and define relationships between them.

1.3.2 2. Storing malware samples

Objects in MWDB are divided into three types: files, configurations and blobs.

File is the primary object type in MWDB, used for storing malware samples and artifacts in various forms e.g. binaries, memory dumps, archives, scripts, .eml/.msg mail files etc.

File attributes

Files are described by the following attributes:

- **File name:** original sample name
- **Variant file names:** other names that sample was uploaded with
- **File size:** file size in bytes
- **File type:** file type returned by Unix `file` command
- **MD5, SHA1, SHA256, SHA512** - hash digest from file contents that can be used as object reference
- **CRC32** - CRC32 file checksum
- **ssdeep** - ssdeep fuzzy hash value
- **Upload time** - timestamp of first file upload

Objects in MWDB are mostly identified by SHA256 hash. The special case are files that can be referenced by other hashes like MD5, SHA1 and SHA512.

The screenshot shows the MWDB web interface. The browser address bar displays the URL: `https://mwdb.cert.pl/sample/a1670f1417e221c45e154b08ec36c8a2`. The page header includes the CERT.PL logo and navigation links: Samples, Configs, Blobs, Upload, Yara search, Search, Statistics, and About.

The main content area is titled "File details" and contains a table with the following attributes:

Attribute	Value
File name	33e4000_52f0ad333a614735
File size	181760
File type	data
md5	a1670f1417e221c45e154b08ec36c8a2
sha1	4ebd8971992435cd5f4ec0e613f7f4d38a3aa3b2
sha256	52f0ad333a61473583ae712d5a8664ee740f40cf786f108d84f25767e2eb0f60
sha512	93743d1e082027cea3b25c6696bbb15ed951d7ce70a1a4717eacacbe53fe826b7b5ec8a75ef1636e03610a30808775f198918a49b1843e7a0dee32bc2791e675
crc32	c8892e13
ssdeep	3072:zLU66mt0BwYc7VhvYuNr1wfrGoXBj9w+Any3i3VX6AHvqR2pMGC9:nLssPvvDcrGoRjAdFoyMG
Upload time	Thu, 08 Oct 2020 14:17:33 GMT

On the right side of the interface, there are sections for Tags (showing "dump:win32:exe"), Add tag, Related samples (listing parent and child hashes), Related configs (listing child hash), Attributes, and Karton analysis.

Warning: The main hash used for identifying objects in MWDB is **SHA256**. MD5 works only for files and may not work in all contexts.

Uploading a file

The simplest way to upload new file to MWDB is to use the **Upload** view accessible from the navbar.

The above view shows how to upload file along with additional options:

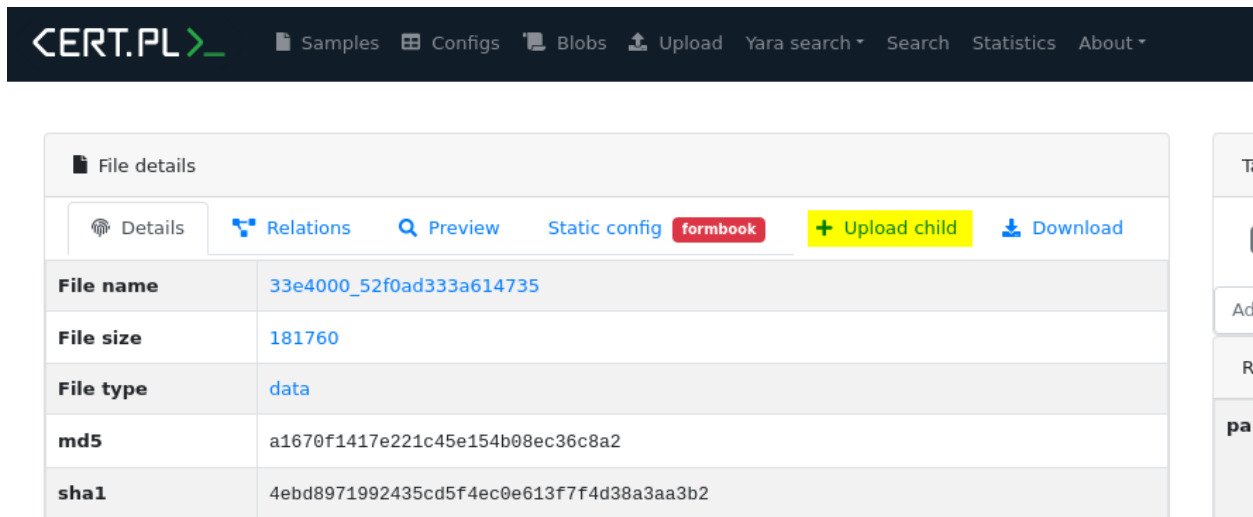
- **Parent:** SHA256 parent identifier
- **Share with:** Groups that we can share our object and all descendants with
 - **All my groups** - default option, share objects with all our groups
 - **Single group...** - shares objects with specified group by name
 - **Everybody** - adds objects to public feed (sharing with public group)
 - **Only me** - shares objects only with user's private group
- **Add attributes:** Additional attributes that will be added to the uploaded sample

All of these parameters are optional and will be described in further chapters. If you are already intrigued what are attributes and how sharing model works, you can jump to [6. Object attributes](#) and [9. Sharing objects with other collaborators](#).

After upload, we're redirected to the detailed file view. It can be already filled up with data if the uploaded sample was already present in the database.

Uploading child file

Sometimes we want to add the file that is related to the object added previously e.g. after adding .eml file we want to add the attachment contained there. In that case, we can **Upload child** button.

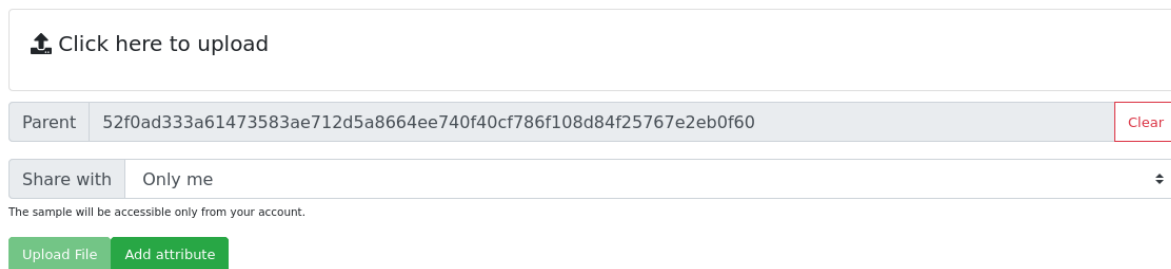


The screenshot shows the CERT.PL web interface. At the top is a navigation bar with links: Samples, Configs, Blobs, Upload, Yara search, Search, Statistics, and About. Below this is a 'File details' section for a file with ID 33e4000_52f0ad333a614735. The details table shows:

File name	33e4000_52f0ad333a614735
File size	181760
File type	data
md5	a1670f1417e221c45e154b08ec36c8a2
sha1	4ebd8971992435cd5f4ec0e613f7f4d38a3aa3b2

Navigation tabs include Details, Relations, Preview, Static config, formbook, **+ Upload child** (highlighted in yellow), and Download.

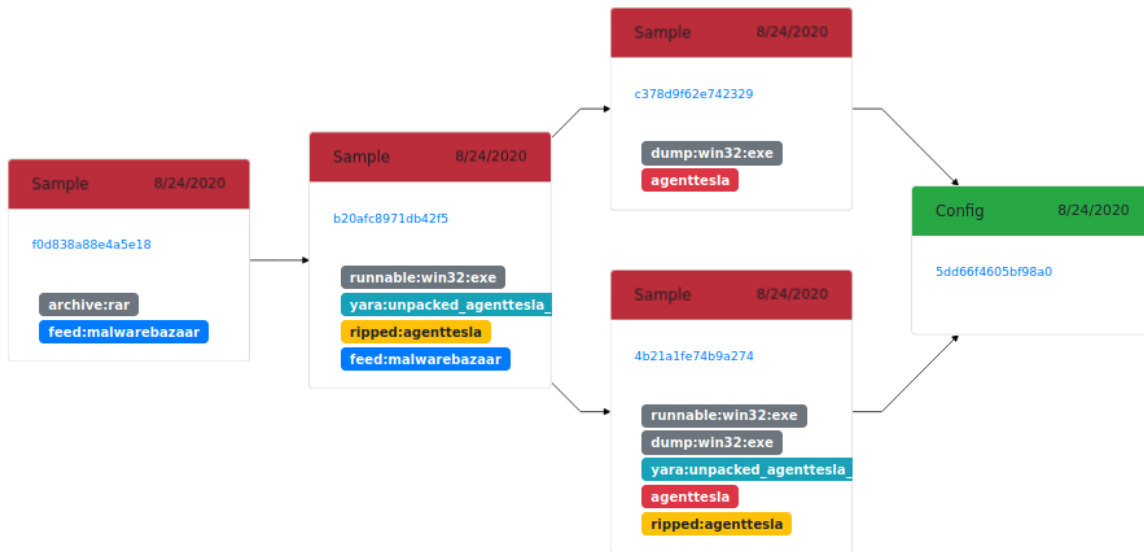
The **Upload child** button navigates to the **Upload** page. Notice that the **Parent** field is already filled with an object identifier, which become a parent of the uploaded object.



The 'Upload' page form includes:

- A button: Click here to upload
- A 'Parent' field containing the identifier: 52f0ad333a61473583ae712d5a8664ee740f40cf786f108d84f25767e2eb0f60, with a 'Clear' button.
- A 'Share with' dropdown menu set to 'Only me'.
- A note: 'The sample will be accessible only from your account.'
- Two green buttons: 'Upload File' and 'Add attribute'.

After uploading a sample as a child, we're redirected to the uploaded sample view. Our parent has been added to Related samples section.



Using that feature we can chain together various files extracted during the analysis. The example above shows the relationships between three objects:

- the original archive
- executable file contained there (agenttesla malicious binary)
- two memory dumps with unpacked code

Finally, from both of these dumps we got a malware configuration.

1.3.3 3. Storing malware configurations

Configuration objects are intended to hold a malware **static configuration** extracted from binaries or a **dynamic configuration** fetched from C&C.

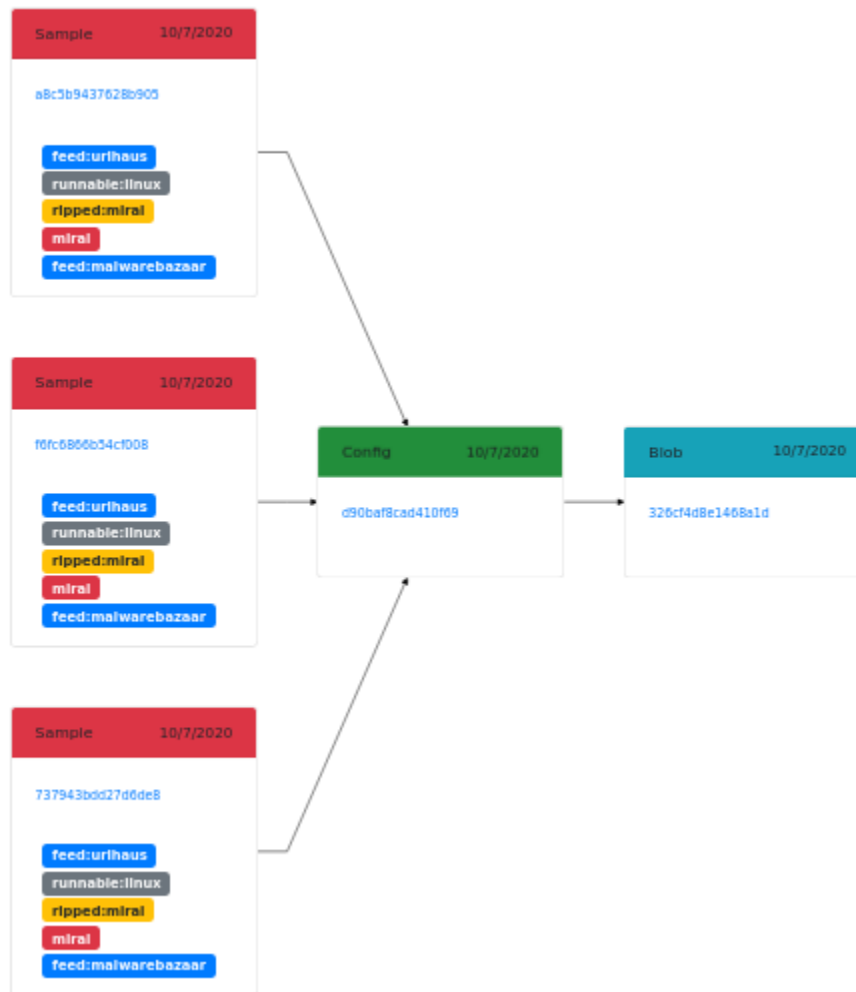
Config 64f777a73a67e0b4090c4b35e24025a2795c88ebd02245abee4ea8ad14325037	
<div>  Details  Relations  Preview  Download </div>	
Family	mirai
Config type	static
— cncs	[{ "host": "45.95.168.88", "port": 16339 }]
— 0	{ "host": "45.95.168.88", "port": 16339 }
+ host	45.95.168.88
+ port	16339
+ table_key	0xdeadbeef
+ type	mirai
+ variant	Esegovic
Upload time	Mon, 24 Aug 2020 10:09:01 GMT

What is malware configuration (or what we think it is)?

In general, configurations are structured data represented in MWDB by JSON objects. Malware samples usually contain embedded “configuration”, called **static configuration** that determines the:

- malware family
- addresses of C&C servers or initial peers
- DGA seeds
- encryption keys used for communication
- malware version
- botnet/campaign ID
- module type etc.

It means that static configuration determines the operations performed by malware sample and how they are parametrized. Various samples unpacking to the same configuration usually have **the same, but differently packed core**, which allows us to determine the similarity between these files.



The format of configuration depends on malware family, usually deriving from the structure “proposed” by the malware author.

On the other hand, malware operations are also parametrized externally by data fetched from Command & Control servers. This data is called **dynamic configuration** and can be parsed as well into the structured form.

Dynamic configuration consists of:

- commands to be executed
- webinjects (for banking malware)
- new C&C IP addresses / peer IP addresses
- mail templates etc.

Configuration attributes

Configurations are described using the following attributes:

- **Family**: describes the malware family related to its configuration
- **Config type**: configuration type. Default is `static` but you can use any string you want. In `mwdb.cert.pl` we're using `static` and `dynamic` to distinguish between static and dynamic configurations.
- **Contents (cfg)**: dictionary with configuration contents
- **Upload time** - timestamp of first configuration upload

Configuration contents are stored as JSON object:

Config f508c76b96d239b8d461d0c6c1c795420072a35448ebe42818febe1990348e1a

Details Relations Preview Download

```

1 {
2   "variant": "KURC",
3   "cnscs": [
4     {
5       "host": "185.63.253.157",
6       "port": 6281
7     }
8   ],
9   "table_key": "0xdedefbaf",
10  "type": "mirai"
11 }

```

Note: It's a good practice to **keep the same configuration structure per malware family** including keys schema and value types.

How to upload configuration?

Configuration is intended to be uploaded by automated systems or scripts. That's why you can't add it directly from MWDB UI. Nevertheless, it's still possible to add configuration using `mwdblib` CLI or REST API.

Warning: Configuration can be uploaded only if a user has the `adding_configs` capability turned on. Check your capabilities in the **Profile** view.

In `mwdb.cert.pl` configuration upload is turned off for external users. If you want to share your own configuration, feel free to contact someone from CERT.pl on Slack or via e-mail (info@cert.pl).

First install the `mwdblib` library including CLI extra dependencies:

```
$ pip install mwdblib[cli]
```

Then you can upload a new configuration using the `mwdb upload config` command:

```
$ mwdb --api-url http://127.0.0.1:3000/api/ login
Username: admin
```






(continues on next page)

(continued from previous page)

Password:

```
$ mwdb --api-url http://127.0.0.1:3000/api/ upload config evil -
{"cnc": ["127.0.0.1"], "key": "asdf"} <CTRL+D>
Uploaded config 64efd0b1f964ad48aadd849a2242ebd1bb803d9e3309ee3d154b15d0dc2c5336
```

Then you can find the configuration in MWDB instance:

Config 64efd0b1f964ad48aadd849a2242ebd1bb803d9e3309ee3d154b15d0dc2c5336	
 Details	 Relations
 Preview	 Download
	 Remove
Family	evil
Config type	static
+ cnc	["127.0.0.1"]
+ key	asdf
Upload time	Mon, 24 Aug 2020 19:37:30 GMT

A new configuration can be also uploaded using a Python script:

```
from mwdblib import MWDB

# Omit api_url if you want to use mwdb.cert.pl API
mwdb = MWDB(api_key=..., api_url=...)
config = {
    "cnc": [
        "127.0.0.1"
    ],
    "key": "asdf"
}
config_object = mwdb.upload_config("evil", config)
# <mwdblib.config.MWDBConfig>
```

Note: If you want to experiment with mwdblib, you don't need to create the API key. Just use the `mwdb.login()` method and you'll be asked for login and password.

More information about automating things is described in the chapter 8. *Automating things using REST API and mwdblib*.

How configurations are deduplicated?

MWDB generates unique SHA256-like hash value for all objects in repository, including configurations. For files and blobs, we just use the SHA256 function to hash the content.

The hashing algorithm is a bit more complicated for structured data like configuration. The main idea is to avoid duplications occurring due to slightly different order of list elements or dictionary keys in uploaded JSON.

That's why our hashing function follows few assumptions:

- Keys in dictionaries are hashed non-orderwise

- Values can have all types supported by JSON, but they're all stringified during hashing e.g. False and "False" are the same. It's not a big deal if you avoid mixing value types under the same key:

```
from mwdblib import config_dhash

config_dhash({"value": "1"})
# 141767ab98a062fcd5bbfb48ddd5d5c2bb3556d64006d774372f15d045d0ba89

config_dhash({"value": 1})
# 141767ab98a062fcd5bbfb48ddd5d5c2bb3556d64006d774372f15d045d0ba89
```

- Lists are treated more like multisets. They're stored orderwise, but hashed non-orderwise.

```
from mwdblib import config_dhash

config_dhash({"domains": ["google.com", "spamhaus.com"]})
# '93b6befcc25bb339eb449d6aa7db47bc3a661f20026e4cb4124388b539336d81'

config_dhash({"domains": ["spamhaus.com", "google.com"]})
# '93b6befcc25bb339eb449d6aa7db47bc3a661f20026e4cb4124388b539336d81'
```

Configuration dictionaries are hashed recursively:

- simple values are stringified and UTF-8-encoded and then hashed using SHA256
- lists are evaluated into the lists of hashes, then sorted and hashed in a stringified form
- dictionaries are converted into the list of tuples (key, hash(value)), sorted by the first element (key) and then hashed in a stringified form

If you want to pre-evaluate hash for configuration, you can use the `config_dhash` function in `mwdblib`.

Searching configuration parts

The most simple way to search for similar configurations is to use interactive search. You can generate the appropriate query just by clicking on the config fields:

Configurations can be also queried manually using the following syntax:

```
config.cfg.field_1.field_2:value
```

which finds configs that contain structure shown below:

```
{
  "field_1": {
    "field_2": "value"
  }
}
```

Note: You can search for configurations only in Recent configs or Search. In Recent configs view config prefix is optional, because the view already makes assumption about the type.

Sometimes you may want to find a specific string in configuration e.g. IP address. In that case, you can use wildcards and search them regardless of the JSON structure:

```
config.cfg:*127.0.0.1*
```

or if you want to be more strict

```
config.cfg:*"127.0.0.1"*
```

For more information see [7. Advanced search based on Lucene queries](#).

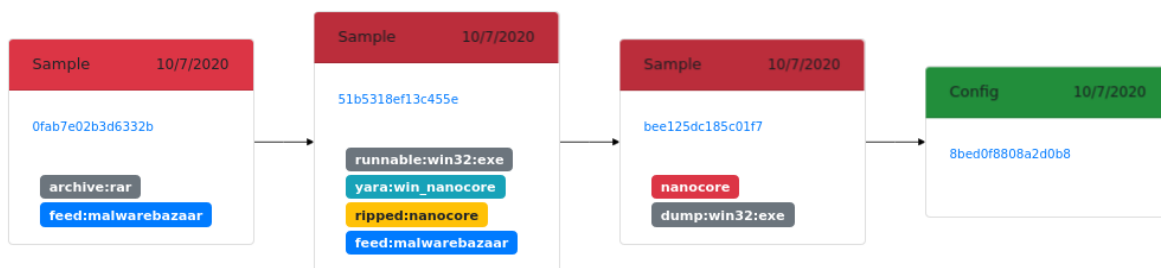
Relationships with files

Configuration semantics is defined not only by the dictionary itself, but also by the relations with other objects. In mwdb.cert.pl service we're following few specific conventions that have special support in mwdb-core.

File → Config relationship

File → Config relationship determines the association between malware sample and static configuration. Configuration parents are the direct source of configuration, which means that configuration is contained in these files and can be extracted directly from them.

That's why the common relationship pattern in MWDB is Executable (packed) → Dump (with unpacked code) → Static configuration.



In addition, the original sample is tagged as `ripped:<family name>` and dump is tagged as `<family name>`.

MWDB has special support for File → Config relationship and presents **the latest configuration** along with basic file information. Relationships returned by API are ordered from the latest one, so hash of the most recent configuration is the first element in the list.

Request URL

```
https://mwdb.cert.pl/api/file/bee125dc185c01f747e18927b301bdd59474888699d6a53fd4b37bb1c15edf7b
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "sha256": "bee125dc185c01f747e18927b301bdd59474888699d6a53fd4b37bb1c15edf7b", "type": "file", "file_type": "PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows", "upload_time": "2020-10-07T10:59:29.465610+00:00", "id": "bee125dc185c01f747e18927b301bdd59474888699d6a53fd4b37bb1c15edf7b", "file_size": 229376, "latest_config": { "type": "static_config", "upload_time": "2020-10-07T10:59:30.121485+00:00", "id": "8bed0f8808a2d0b88f482b3c726b55ee4158fb934efbaa17baf293b2ff41a2f3", "family": "nanocore", "parents": [{ "upload_time": "2020-10-07T10:59:29.465610+00:00", "type": "file", "id": "bee125dc185c01f747e18927b301bdd59474888699d6a53fd4b37bb1c15edf7b", "tags": [{ "tag": "nanocore" }] }] } }</pre> <p>Response headers</p>

Latest configuration is also presented in the UI by the separate `Static config` tab, appearing in the detailed file view.

File details

Details Relations Preview **Static config** nanocore + Upload child Download

File name	400000_bee125dc185c01f7
-----------	-------------------------

Config → File relationship

Config → File relationship represents files dropped by malware from the C&C. These files can be:

- modules (for modular malware)
- next stage malware
- updates
- tasks

Static configuration is required to fetch these files from the server. It can contain distribution URLs where file is placed or the encryption key needed to decrypt the payload.

Thus we bind these files to the configuration instead of making relationships with all malware samples that drop them.

1.3.4 4. Storing human-readable data (blobs)

Blob object is a representation of unstructured, human-readable (text) data. Simply everything that is non-binary and unparsed.



What is blob in MWDB?

Usually “Blob” stands for “Binary Large Object”, but in MWDB the meaning is different. Blob represents **various data stored in unstructured form** that are only simply formatted to be human-readable.

Blobs can also store data that are part of configuration, but are **too big to be conveniently represented** by JSON value.

The good examples are:

- bunch of extracted strings
- injects fetched from C&C for banking malware
- mail templates for spam bots
- lists of peers fetched from C&C for P2P botnets
- raw static configurations if malware keeps them structured (e.g. in XML format)

Note: Before storing a blob, filter out things that are originating from specific execution in sandbox and are not specific for malware operations themselves e.g. **random nonces, memory pointers and other noise**. This will help you to avoid duplications.

Blob attributes

Blobs are described using the following attributes:

- **Blob name:** name of blob file
- **Blob size:** size of blob contents
- **Blob type:** blob type, classifying type of data (e.g. peers, raw_cfg, strings etc.)
- **Contents:** blob contents
- **First seen** - timestamp of first blob upload
- **Last seen** - timestamp of last blob upload

Blob details	
Details	Relations
Preview	Diff with
Download	
Blob name	mirai_dyn_cfg_9d98f85ea50522fe09462d6496fa1fc5
Blob size	203
Blob type	dyn_cfg
First seen	Wed, 07 Oct 2020 23:51:24 GMT
Last seen	Wed, 07 Oct 2020 23:51:24 GMT

How to upload blobs?

Just like configurations, blobs are intended to be uploaded by automated systems or scripts, so they can't be added directly from MWDB UI. Similarly to the configuration upload described in previous chapter (*How to upload configuration?*), you can push them using `mwdb upload blob` command or `MWDB.upload_blob` function or via REST API.

More information about mwdblib can be found in chapter 8. *Automating things using REST API and mwdblib*.

Embedded blobs

Blobs can be embedded as a part of configuration using `in-blob` key. This feature is especially useful for config values that are too big to be nicely stored in JSON.

Warning: `in-blob` keys are supported only at the top level keys and can't be used for values in nested dictionaries.

Embedding new blob

Let's assume we have a key called `raw_cfg` that contains long string with whole malware configuration.

```
{
  "family": "malwarex",
  "...": "...",
  "raw_cfg": "<really long string with contents>"
}
```

If you want to upload the `raw_cfg` contents as a blob object, wrap contents with `in-blob` structure like below. Blob reference will be placed under the original (`raw_cfg`) key.

```
{
  "family": "malwarex",
  "...": "...",
  "raw_cfg": {
    "in-blob": {
      "blob_name": "malwarex-raw-configuration",
      "blob_type": "raw_cfg",
      "content": "<really long string with contents>"
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

    }
  }
}

```

in-blob objects are processed as follows:

- MWDB creates blob objects for all in-blob objects
- in-blob values are transformed into blob object references. MWDB puts string with object identifier in place of dictionary with blob specification
- Transformed configuration object is added to MWDB repository.
- Relationships are added, where uploaded configuration is parent and blobs are children.

The screenshot displays the 'Config details' interface. The top section shows a JSON configuration with fields like 'variant', 'table_key_effective_encr', 'table_entries', and 'cncs'. The 'table_entries' field contains an 'in-blob' reference. Below this, a table view summarizes the configuration details.

Config details	
Family	mirai
Config type	static
+ cncs	...
+ table_entries	5bbac727c051c559263870d423e6c4627bb2bb33bdf66a60fd9953b41583752c
+ table_key	0xdeadbeef

All upload options passed to the upload request (excluding parent) apply to the added blobs. If you want to set attribute along with configuration upload, attribute will be added to all blob objects as well.

Embedding already uploaded blob

If you want to embed already uploaded blob in the configuration, you can upload configuration with already transformed in-blob objects into references:

```

{
  "family": "malwarex",
  "...": "...",
  "raw_cfg": {
    "in-blob": "<blob identifier>"
  }
}

```

Searching blob files

The best way to search blob contents is to use *blob.content* field with wildcards e.g.:

```
blob.content:*<gtag>kas33</gtag>*
```

For more information see [7. Advanced search based on Lucene queries](#).

Blob diffing

Sometimes you may want to visualize content differences between two blobs.

To show blob diff, go to the Blob details and click “Diff with”.

The screenshot shows the mwdb web interface. The top navigation bar includes links for Samples, Configs, Blobs, Upload, Admin, Search, Statistics, and About. The main content area is titled 'Details' and shows a JSON blob. The JSON content is as follows:

```
{
  "duration": 35,
  "flags": {
    "7": "25"
  },
  "targets": [
    {
      "netmask": 32,
      "address": "54.39.29.147"
    }
  ],
  "attack_vector": 7
}
```

On the right side of the details view, there are buttons for 'Diff with' and 'Download'.

Then you have to choose a blob you want to compare:

The screenshot shows the 'Diff with' selection process. A search bar is used to find a blob by ID. The search results are displayed in a table:

Blob name	Blob ID	Blob type
mirai_dyn_cfg_78...	1b38d69a65a6f85550043eb3b39a890f65cd14129dfd34b42a3337dbec8a155d	dyn_cfg
mirai_dyn_cfg_78...	c0228918b4a5ae4b104172c5afb66a32c5bc1bc4c31262aad72cdd26d0b6c81	dyn_cfg

After that you will see the difference between two blobs:

The screenshot shows the 'Blob diff' view. Two JSON blobs are compared side-by-side, highlighting differences in the 'duration', 'flags', 'targets', and 'attack_vector' fields. The left blob (ID: c0228918b4a5ae4b104172c5afb66a32c5bc1bc4c31262aad72cdd26d0b6c81) has a duration of 30, flags of 21, and an attack vector of 5. The right blob (ID: 1b38d69a65a6f85550043eb3b39a890f65cd14129dfd34b42a3337dbec8a155d) has a duration of 35, flags of 25, and an attack vector of 7.

Relationships with configurations

Just like configurations, relationships between blobs and other object types have special meaning:

Config → Blob relations

Config → Blob relationship usually represent data that are:

- dynamically fetched based on static configuration
- part of static configuration (previously described in “Embedded blobs” section)

Bankers are parametrized by dynamic configuration containing injects. Spam botnets fetch spam templates. All of these text-like things can be stored as text blobs bound to related static configuration.

Blob → Config relations

Sometimes we are able to parse the dynamically fetched content from C&C, but the process is lossy and we want to store it in both forms:

- unstructured, but more complete blobs
- easier to process (structured), but limited configurations

In that case we can upload configuration that represents parsed blob as a child of that blob.

Just like File → Config relations, MWDB has special support for that type of relationship and presents **the latest configuration** for a given blob along with other blob information.

1.3.5 5. Tagging objects

Tags are used for basic object classification, allowing to quickly search for interesting samples in a malware feed.

Warning: Tags can be added only if user has the `adding_tags` capability turned on. Check your capabilities in Profile view.

In `mwdb.cert.pl` you are allowed to add tags, but **you can't remove them** (`removing_tags` is required). If you see that malware family has not been correctly recognized by `mwdb.cert.pl`, you can left a comment or set *maybe:<family>* tag to help us track these issues.

How to use tags?

You can add new tag to the object using `Add tag` field on the right side of detailed view. Click `Add` to submit a new tag. Autocompletion will help you explore existing tags to avoid typos or other misspelled duplications.

If you want to remove a tag - click on the `x` placed on the right side of the tag you want to remove.

To explore other objects with the same tag, just click on the tag to navigate to the search page. There you can filter in/out other tags to make your search more specific for what you are looking for.

Tags can be added as well using `mwdblib` library.

```
from mwdblib import MWDB

FILE_SHA256 = "f60d462..."

mwdb = MWDB(api_key=..., api_url=...)
file_object = mwdb.query_file(FILE_SHA256)
file_object.add_tag("chthonic")

print(file_object.tags)
# ["chthonic"]
```

Built-in tag conventions

Probably you have already noticed that tags are differently colored depending on the prefix. That's because MWDB has few tag conventions built-in to highlight certain groups of tags.

Simple tags are **red**. In mwdb.cert.pl they're mostly used for marking identified malware name.

formbook ⓘ

Tags describing the source are **blue** (src:, uploader:, feed:). feed: tags are most special, because you can easily filter out all external feed by choosing built-in Exclude feed:* option in Quick query bar.

feed:sample ⓘ

Tags indicating matched malware are **yellow** (ripped:, contains:). In mwdb.cert.pl we mark the original sample with ripped:<family> tag. Unpacked samples or dumps originating from ripped samples are added as a child and tagged red with malware family name.

ripped:formbook ⓘ

Note: If you want to get only samples that are marked as malicious with high confidence - use Only ripped:* button in Quick query bar.

File types can be additionally classified with another group of **gray** tags (runnable:, archive:, dump:, script:).

runnable:win32:exe ⓘ

Generic tags containing : are **cyan**. We use them to add some secondary tags fetched from feed or indicating classification result by other systems (e.g. yara:, et:)

yara:win_formbook ⓘ

1.3.6 6. Object attributes

Attributes (formerly called “metakeys”) are used to store associations with external systems and other extra information about an object.

List of object attributes can be found in detailed object view in the **Attributes** box.

Attributes + Add	
From	http://51.89.213.132/powerpc
Karton analysis	<div> ✓ done 443b67cb-398a-46e1-9c5f-7a54c9535cd1 ▾ </div>

How attributes can be used?

Attributes are set of unique key-value pairs, ordered from the earliest added. Keys are not arbitrary and **must be defined first** in attribute configuration. Each attribute key can have special properties depending on the key semantics.

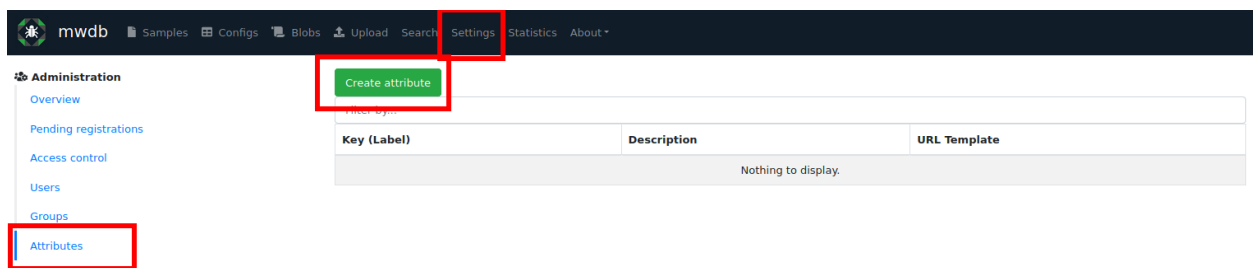
Attribute feature was initially designed to store the analysis identifier. Attribute values **can be rendered as URLs** to the external webpage with analysis details.

Plugins can apply custom rendering of attribute values to include additional information from external systems. Good example is Karton plugin that uses custom component to show information about analysis status.

Declaring new attribute

Attributes must be declared first by administrator (having `manage_users` capability). To declare a new attribute, go to the **Settings** → **Attributes** view using navigation bar.

Then, click on **Create attribute** button.



Attribute key has few basic properties described below:

- **Key** - the main attribute key name. Identifies the key and is used in API. Must contain only lowercase letters and digits and maximum 32 characters are allowed.
- **Label** (optional) - user-friendly label for attribute, visible for MWDB users instead of key. Can be changed any time you want.
- **Description** (optional) - description to explain users the semantics of attribute, visible in MWDB UI.

- **URL template** (optional) - URL template used to map the values to the URLs. Template can contain \$value placeholder, where MWDB UI will put the attribute value and make clickable link.
- **Hidden attribute** (optional) - Actually it should be called “protected attribute”. These attributes have special purpose and are described further.

After defining new attribute, click **Submit**. The most important thing at this step is to choose good attribute key. Don't worry much about the optional fields, they can be changed any time you want.

When attribute is added, you will be redirected to the attribute key settings page. It looks almost the same, but have additional box at the bottom with ACL (access control list) settings for groups.

Hidden attribute ☐

Hidden attributes have protected values. Attribute values are not visible for users without reading_all_attributes capability and explicit request for reading them. Also only exact search is allowed. User still must have permission to read key to use it in query.

Submit

Group name	Permissions	Actions
public	<input checked="" type="checkbox"/> Can read <input type="checkbox"/> Can set	Update Remove group
<input type="text" value="public"/>		Add group

Attributes without ACLs are accessible only for administrators or users with special capabilities like `Has access to all attributes of object` and `Can add all attributes to object`. Regular users can't see and set the attributes without explicit permission.

If you want to give permission to the specific group or user, type the group name (or user login) and click **Add group**. Then you can use `Can read` and `Can set` switches to set the permissions. Apply changes for group using **Update** button.

If attribute should be available to all users in MWDB instance, add `public` group with `Can read` permission. If you want all users to add new values with defined attribute key - additionally enable `Can set` permission.

Adding attributes to objects

If you have permission to set the attribute key, you can add a new attribute value using **Add +** button placed in **Attributes** box header.

Attributes can be added during object upload. It's really useful if you want to use attributes as plugin arguments e.g. to provide password to the encrypted file/archive. Attributes passed that way will be set just before the plugin hook fires.

[Configs](#)
[Blobs](#)
[Upload](#)
[Admin](#)
[Search](#)
[Statistics](#)
[About](#)
Logg

38492.7z - 41158125 bytes

Parent (Optional) Type parent identifier... Clear

Share with All my groups

The sample and all related artifacts will be shared with all your workgroups

Attributes

password	infected	Dismiss
----------	----------	---------

Upload File
Add attribute

Another use case is passing source feed identifier with the uploaded file.

The same operations can be performed using mwdblib. Let's explore the attributes in mwdb.cert.pl:

```
from mwdblib import MWDB

mwdb = MWDB(api_key=..., api_url=...)
file_object = mwdb.query_file("d9f00cc51e1c107881ebeaa0fd4f022a")
print(file_object.attributes)

# {'from': ['http://45.95.168.97/bins/sora.x86'],
#  'karton': ['da83d95a-0564-4d32-8fea-ff61ac7396d1']}
```

To add a new attribute to the existing object, use `MWDBObject.add_attribute` function. Example below will fail in mwdb.cert.pl because attribute `test-key` is probably undefined, but you can try it on your own instance!

```
file_object.add_attribute("test-key", "test-value")
print(file_object.attributes)

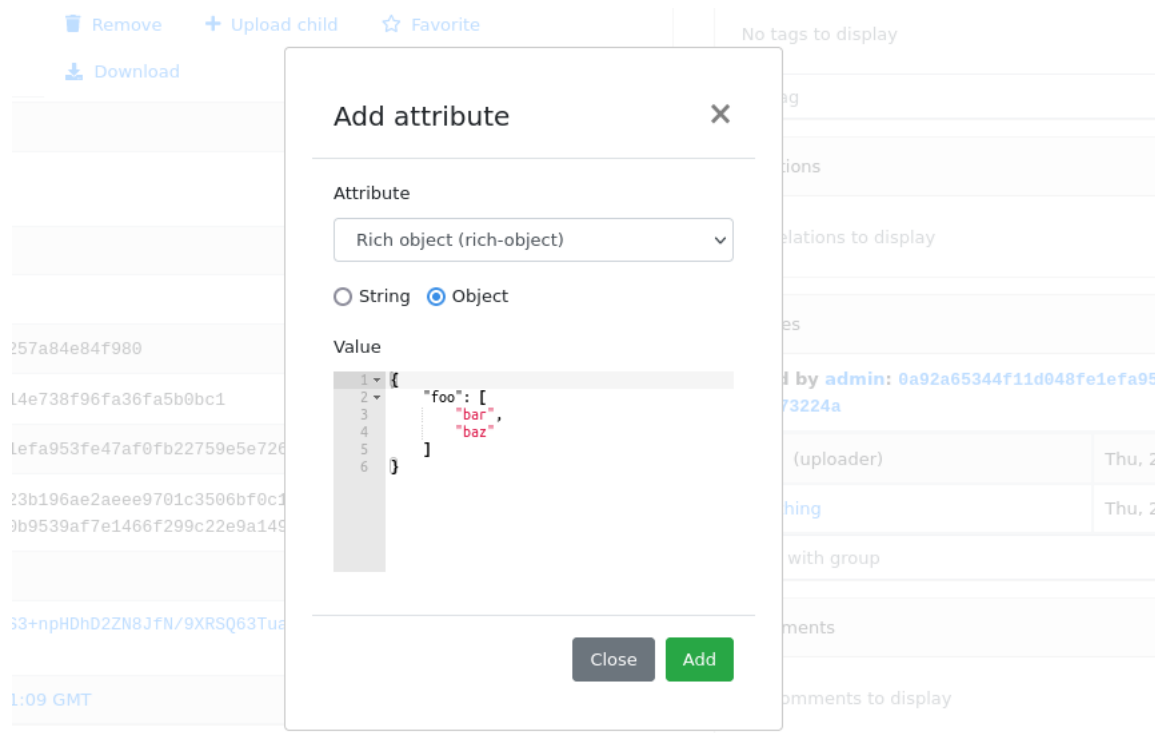
# {'from': ['http://45.95.168.97/bins/sora.x86'],
#  'karton': ['da83d95a-0564-4d32-8fea-ff61ac7396d1'],
#  'test-key': ['test-value']}
```

To pass an attribute value during upload, you can set it via `attributes` argument using `MWDB.upload_file` routine.

```
mwdb.upload_file("infected.zip",
                 zip_contents,
                 attributes={
                     "test-key": "test-value"
                 })
```

JSON-like attribute values

Starting from 2.6.0, MWDB Core attribute values are not limited to be plain strings, but also JSON-like objects.



Upload time	Thu, 23 Dec 2021 08:51:09 GMT
Attributes	
Rich object	(object) { "foo": ["bar", "baz"] }

Using JSON attributes, you can easily store complex data like:

- enrichments from other services
- file static analysis information like code signing, sections, list of resources
- information about produced dumps from sandbox

```
file_object.add_attribute(
    "dumps", {
        "fffb0000_1908f226280751a1": {
            "apivector":
↪ "A45oA36CA7BA15iAgA6gA3CABQAEA3IAAQQA5CAGBkACA4QA6JIAAICISIMhY]Us",
            "families": [
                "isfb"
            ],
            "similarity": 0.35330578512396693
        }
    },
```

(continues on next page)

(continued from previous page)

```

    "fffa0000_568d29558bebb4d8": {
      "apivector": "A119CABQAEACA4QAAG4BEA21QwMEhYQUM",
      "families": [
        "isfb"
      ],
      "similarity": 0.3490675028882654
    }
  }
)

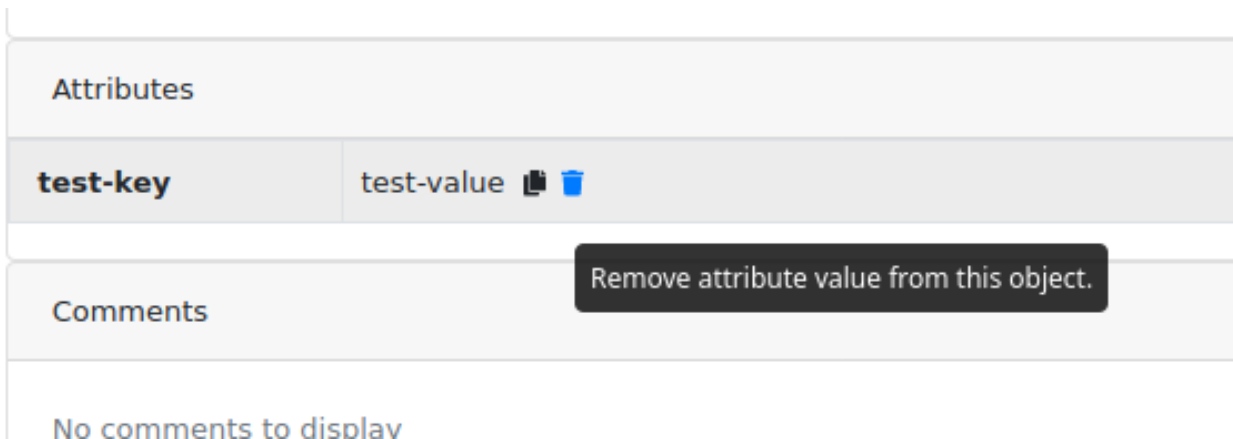
```

You can also search for parts of object value using JSON queries, like for configuration contents:

```
attribute.dumps.fffb0000_1908f226280751a1.families*:isfb
```

Removing attributes from objects

To remove attribute value, hover over that value and click the remove button.



Attribute can be removed only if user has permission to set the attribute key and has `removing_attributes` capability turned on.

Hidden (protected) attributes

Attributes marked as “hidden” can be only queried using exact search (without wildcards). They can’t be directly read in object view, unless you have special capability (`reading_all_attributes`).

Hidden attributes can be used if you want to share the job identifier and allow to search associated objects without giving access to other job references for these objects.

A good example is the `mquery` plugin. In case of `mquery`, we want to ensure that user will have access only to the results of jobs runned by themselves or shared by other users. Unfortunately, attribute key permissions are “all or nothing” and we can’t define ACLs on the value level.

Rich templates

Starting from v2.8.0, MWDB Core supports rich attribute value rendering. For more information, see [Rich attributes guide](#).

1.3.7 7. Advanced search based on Lucene queries

MWDB comes with a powerful search engine based on Lucene query syntax subset.

Query syntax: fields

A query is broken up into fields and operators.

You can search any field by typing the field name followed by “:” and then the term value you are looking for.

```
tag:emotet
```

Multi-word terms separated by spaces or containing a special character (e.g. colon or parentheses) are called “phrases” and must be surrounded by double quotes

```
type:"PE32 executable (GUI) Intel 80386, for MS Windows"
```

Most fields support **wildcard search**. Symbol “?” represents a single character wildcard and “*” represents multiple characters (0 or more). If you want to include all PE executables in your results - use:

```
type:PE32*
```

Query syntax support escaping if you want to include “*” as a character. Query presented below looks for all type values that are containing asterisk:

```
type:"*\**"
```

Query syntax: operators

MWDB supports three boolean operators: AND, OR and NOT.

Warning: Boolean operators must be UPPERCASE.

If you want to search all samples that are tagged with anything that contains “emotet” word (e.g emotet_drop, ripped:emotet or just emotet) and exclude samples tagged as feed:spam - use:

```
tag:*emotet* AND NOT tag:"feed:spam"
```

Query syntax supports using parentheses to group logic expressions:

```
name:emotet* OR (tag:*emotet* AND NOT tag:"feed:spam")
```


Query syntax: ranges

Integer and date fields support range search. Range queries can be inclusive or exclusive of the upper and lower bounds.

Warning: TO operator must be UPPERCASE.

Query written below will find all files that have size between 50 and 50000 bytes inclusive:

```
size:[50 TO 50000]
```

If you want to exclude one of the range sides, replace “[” character with “{”.

```
size:{50 TO 50000]
```

This will find files between 51 and 50000 bytes in size. Inclusive range queries are denoted by square brackets. Exclusive are denoted by curly brackets.

In most cases we want to search for that are only one-side bounded e.g. all files bigger than 50000 bytes. In that case, we can use single wildcard character to denote the infinity:

```
size:[50000 TO *]
```

This syntax is still not very convenient, so we have introduced shorter syntax incorporating >, <, >= and <= operators. To use them, just add appropriate operator to the beginning of a term.

```
size:>=500000
size:>="500000"
```

MWDB-Core supports human-readable file size so instead of specifying the number of bytes, we can refer to larger units like kB, MB and GB.

```
size:>=500kB
size:>=0.5MB
```

Warning: Remember that converting a file size from bytes to human-readable form does not always match with the conversion the other way around.

For example 1 kB equals 1024 bytes, rounding 1026 bytes to the second decimal number 1026 bytes will also give 1 kB (1.002 kB).

So do not be surprised if you enter `size:1kB` in the search engine and a sample of this size is not found, because in bytes this size may differ slightly.

For this reason, searching for a size from an object view always redirects to the query in bytes.

Query syntax: timestamps

With timestamps you can search for objects within certain time range.

If you want to find objects that were uploaded from the beginning of September till the 28th:

```
upload_time:[2020-09-01 TO 2020-09-28]
```

If you want to find objects that were uploaded from the beginning of September:

```
upload_time:[2020-09-01 TO *]
```

Alternatively:

```
upload_time:>=2020-09-01
```

If you want to search for objects within time certain range:

```
upload_time:["2020-09-28 08:00" TO "2020-09-28 09:00"]
```

If you want to search for objects uploaded after certain hour:

```
upload_time:>="2020-09-28 08:00"
```

If you want to search for objects uploaded at certain minute:

```
upload_time:"2020-09-28 15:32"
```

Remember that exclusive range is not allowed for date-time field so this is not allowed:

```
upload_time:{2020-09-01 TO *}]
```

```
upload_time:>2020-09-01
```

Query syntax: relative timestamps

New in version 2.7.0.

It is also possible to use `upload_time` in relation to current time. For example, if you want to search for objects uploaded during last 2 hours:

```
upload_time:>=2h or upload_time:[2h TO *]
```

This way of time definition contains value and relative time format.

Below are listed acceptable relative time symbols:

- **y** or **Y** : years
- **m** : months
- **w** or **W** : weeks
- **d** or **D** : days
- **h** or **H** : hours
- **M** : minutes

- **s** or **S** : seconds

Warning: Bring awareness to symbols **m** and **M**. They means quite different period time. Other symbols can be used as uppercase or lowercase letters.

During defining relative time you can combine different time symbols. For example, if you want to search for objects uploaded earlier then 1 month and 5 days ago:

```
upload_time:<=1m5d or upload_time:[* TO 1m5d]
```

It is possible to use relative timestamps and timestamps.

```
upload_time:[2022-02-01 TO 1m10d]
```

Basic search fields

Fields represent the object properties and can be **typed** (specific for object type) or **untyped** (generic, used by all object types).

Usage depends on the search context. If you're querying `Recent files` tab, query engine assumes that object type is `file`. If you're using `Search` tab, you need to add appropriate type prefix to the typed fields.

In simple words: *name:* field in *Recent files* must be replaced by `file.name:field inSearch`.

Untyped fields

- `dhash:<string>` - Object identifier (SHA256)
- `tag:<string>` - Object tag
- `comment:<string>` - Object comment contents
- `meta.<attribute>:<string>` - Object attribute value
- `upload_time:<datetime>` - Object first upload timestamp
- `karton:<uuid>` - Karton analysis artifacts

Typed fields (file)

- `file.name:<string>` - Name of file
- `file.type:<string>` - Type of file, returned by `file` Unix command
- `file.size:<integer>` - Size of file in bytes
- `file.md5:`, `file.sha1:`, `file.sha256:`, `file.sha512:`, `file.ssdeep:`, `file.crc32:` - File contents hashes and checksums

Typed fields (config)

- `config.type:<string>` - Type of configuration
- `config.family:<string>` - Malware family name
- `config.cfg[<.path>]:<string>` - JSON field with configuration contents

Typed fields (blob)

- `blob.name:<string>` - Name of blob
- `blob.size:<integer>` - Size of blob
- `blob.type:<string>` - Type of blob
- `blob.content:<string>` - Blob contents
- `blob.first_seen:<datetime>` - Alias for `upload_time`
- `blob.last_seen:<datetime>` - Timestamp when blob was last uploaded

Special fields

There are also other fields that have special meaning. They will be described in further sections of this chapter.

- `favorites:<string>`
- `sharer:<string>`
- `shared:<string>`
- `uploader:<string>`
- `parent:<subquery>`
- `child:<subquery>`

JSON fields (`config.cfg:`)

Configurations can be searched using path queries:

```
config.cfg.field_1.field_2:value
```

which would find configs that contain structure below:

```
{
  "field_1": {
    "field_2": "value"
  }
}
```

Configurations are stored as JSON objects. The most simple way to search something inside configuration is to treat them as simple text fields and use wildcards.

Assuming we are in `Recent configs` tab:

```
cfg:*google.com*
```

If we want to be more specific and look for `google.com` only inside “urls” key, we can add a field name to `cfg` field using dot:

```
cfg.urls:*google.com*
```

If you want to search for elements contained in an array, simply use `*` at the end of the field where it is nested. For example, let’s use the following configuration.

```
{
  "field": {
    "array": [1, 2, 3]
  }
}
```

In this case, to find the object, we can use array search to check if the nested array contains a specific value:

```
cfg.field.array*:1
```

Searching in this way applies to both numbers and strings contained in the array.

If you search by more than one value contained in an array, just type:

```
cfg.field.array*:"*1, 2*"
```

Starting from v2.8.0, you can also use range queries for numbers, for example:

```
cfg.field.array*:[1 TO 3]
```

looks for any configuration that contains *field.array* with at least one number element within range 1 to 3 inclusively.

Favorites field (favorites:)

Typing the field `favorites:` you can search for your objects marked as favorite in object view.

```
favorites:<user login>
```

The above query returns the favorite objects of specific user.

Warning: Remember that you can only search for your own favorites objects.

Only system administrator with “manage_users” capabilities can search for other users favorites.

Comment author field (`comment_author:`)

Typing the field `comment_author:` you can search for objects commented by selected user.

```
comment_author:<user login>
```

The above query returns the objects commented by user `<user login>`.

Warning: Comment authors are kept only for existing users, so you can't search for comments from deleted accounts.

Wildcards are not allowed for field `comment_author:`.

Upload count field (`upload_count:`)

Typing the field `upload_count:` you can search for objects using upload information that are related with **n** different users uploads. Using that query, you can spot objects that might be most interesting within your dataset.

```
upload_count:n
```

The wildcards ranges are supported for field `upload_count:`.

For example: If you want to search for samples which were uploaded by 5 and more different users, use undermentioned search statement. It's also highly recommended to combine `upload_count` with `upload_time` query, so you can select only samples that were uploaded within the specific period of time:

```
upload_count:["5 TO *"] AND upload_time:[2021-06-01 TO 2021-06-30]
```

Group access queries (`sharer:`, `shared:` and `uploader:`)

Search engine supports `sharer:`, `shared:` and `uploader:` special fields that are useful for filtering out specific user or group uploads.

- `sharer:` checks if object is explicitly shared with current user by specific user
- `shared:` checks if object is explicitly shared with specific group or user
- `uploader:` checks if object was uploaded by specified user or any user from specified group

If you want to exclude objects shared with everyone (public group):

```
NOT shared:public
```

If you want to include only objects that are uploaded by yourself:

```
uploader:<your login>
```

If you want to see objects that are uploaded by somebody from your group excluding your own uploads:

```
uploader:<group name> AND NOT uploader:<your login>
```

Keep in mind that you can query only for objects uploaded by you or members of your own groups (excluding members of the public group). This limitation doesn't apply to administrators (`manage_users` capability).

Read more about MWDB sharing model and capabilities in chapter 9. *Sharing objects with other collaborators*.

Parent/child subqueries

MWDB allows to use parent/child subqueries.

If you want to search for samples that have ripped configuration for Emotet family as their child, go to Samples and type:

```
child:(config.family:emotet)
```

If you want to search for configs that have a sample as their parent with file size greater than 1000, go to Configs and type:

```
parent:(file.size:>1000)
```

Nested searches can be performed as well. If you want to find object which is parent of object tagged as `emotet` and grandparent of config object for Emotet family:

```
child:(tag:emotet AND child:(config.family:emotet))
```

Multi field (multi:)

Search engine supports `multi:` special field that is useful for filtering out objects using multiple type of object attributes.

Depending on type of object, we can use field `multi:` applying various object attributes separated by spaces. Types of attributes are automatically recognised. Below allowable attributes were listed for different type of objects, which can be used in query building.

- File (sample)
 - all hashes values
- Config
 - dhash values
 - extract of configuration content
- Blob
 - dhash values
 - extract of Blob content

If you want to search for samples that have `0cb988d042a7f28dd5fe2b55b3f5ac7a` md5 value or `3b0ee981` crc32 value use below query.

```
multi:"0cb988d042a7f28dd5fe2b55b3f5ac7a 3b0ee981"
```

If query contains only hashes, field `multi:` can be omitted.

```
0cb988d042a7f28dd5fe2b55b3f5ac7a_
↪eb1c78d4994f7a107f78dec529988900e3601852ae0bfdefb3e15967c6d8f127
```

If you want to search for Configs that have configuration content which contain strings “abcd” or “xyz” or dhash value `eb1c78d4994f7a107f78dec529988900e3601852ae0bfdefb3e15967c6d8f127` - use below query

```
multi:"abcd xyz eb1c78d4994f7a107f78dec529988900e3601852ae0bfdefb3e15967c6d8f127"
```

Warning: Multi-query terms containing extract of Config configuration or Blob content must be separated by spaces and surrounded by double quotes. Field `multi`: also has to be explicitly used.

Wildcards are not allowed for field `multi`:. They are automatically used for Config configuration and Blob content.

Escaping special characters

Field labels may contain characters that need to be escaped in query. Let's consider a config field with space character:

```
{
  "decoy domains": [ "evil.com" ]
}
```

Special characters (like space) need to be escaped using backslash, so correct query looks like below:

```
config.cfg.decoy\ domains*:"evil.com"
```

Other special characters that also need to be escaped are:

- dots (\.)
- colons (\:)
- asterisks (*)
- backslashes (\\)

Quick queries



Quick queries can be found just under the search field.

You can use quick query by clicking on one of the badges. First four queries are built-in:

- **Only uploaded by me** is `uploader:<my login>` query that can be used to filter only samples uploaded by ourselves
- **Exclude public** is `NOT shared:public` and filters out public objects
- **Exclude feed:*** is `NOT tag:"feed:"` and excludes all the external feeds
- **Only ripped:*** is `tag:"ripped:"` and includes only original samples recognized as malware and with successfully ripped configuration.

You can also add your own quick query by first typing the query in search field and then clicking on **Add +**

Afterwards, you can see your newly added query as another black-coloured badge. You can click it any time and even the most complex query will be performed!

×
uploader:"admin" AND NOT shared:"public" AND NOT tag:"feed:*"

Quick query:
Only uploaded by me
Exclude public
Exclude feed:*
Only ripped:*
my_query ×
Add +

1.3.8 8. Automating things using REST API and mwdblib

The previous chapters are guide over MWDB features mostly from the Web UI perspective. Actually, Web UI is only a client for rich REST API, which brings out the full potential of the MWDB server.

The most recommended way to use MWDB API is [mwdblib](#), which is dedicated binding with few high-level utilities for scripts written in Python. If you want to use other language to interact with API, any HTTP client will be fine for that.

Introduction to mwdblib

Let's start with mwdblib installation using pip.

```
pip install mwdblib
```

The main interface is `MWDB` object that provides various methods to interact with MWDB. Let's start with log in to `mwdb.cert.pl` service.

```
>>> from mwdblib import MWDB
>>> mwdb = MWDB()
>>> mwdb.login()
Username: jkowalski
Password:
```

If you want to use your local instance instead of `mwdb.cert.pl`, pass the appropriate API url to the `MWDB` constructor:

```
>>> mwdb = MWDB(api_url="http://127.0.0.1:3000/api")
>>> mwdb.login("admin")
Password:
```

After successful login, let's begin with `recent_files` to get recently uploaded file from API.

```
>>> mwdb.recent_files()
<generator object MWDB._recent at ...>
```

`recent_files` function returns generator which does the same job as scrolling down the `Samples` view to view the older entries. Let's use `next` function to get the most recent file:

```
>>> files = mwdb.recent_files()
>>> file = next(files)
>>> file
<mwdblib.file.MWDBFile at ...>
```

...and we got the file! To get the next 10 files, we can use `itertools.islice` method:

```
>>> import itertools
>>> recent_10 = list(itertools.islice(files, 10))
```

(continues on next page)

(continued from previous page)

```
[<mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>]
```

If you want to get a file by hash, you can use `query_file` method.

```
>>> file = mwdb.query_file("780e8fb254e0b8c299f834f61dc80809")
>>> file
<mwdblib.file.MWDBFile at ...>]
```

Using retrieved MWDBFile object we can get some details about the file e.g. name, tags, child objects or download its contents:

```
>>> file.name
'400000_1973838fc27536e6'
>>> file.tags
['dump:win32:exe', 'avemaria']
>>> file.children
[<mwdblib.file.MWDBConfig at ...>]
>>> file.download()[:16]
b'MZ\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00'
```

As you can see, there is a configuration attached to the file. We can get it by index operator or use `config` attribute to get the latest configuration object. Let's see what has been ripped:

```
>>> file.children[0].config
{'c2': [{'host': '172.111.210.207'}], 'type': 'avemaria'}
>>> file.config
<mwdblib.file.MWDBConfig at ...>
>>> file.config.config
{'c2': [{'host': '172.111.210.207'}], 'type': 'avemaria'}
```

Few malware samples can share the same configuration. Let's explore them:

```
>>> avemaria = file.config
>>> avemaria.parents
[<mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>,
 <mwdblib.file.MWDBFile at ...>]
>>> [parent.name for parent in avemaria.parents]
['400000_1973838fc27536e6',
 '400000_2bf452f7796153ef',
 '400000_3539b9d228df73c6']
```

More methods can be found in [mwdblib documentation](#). After short introduction and fun with exploring MWDB, let's dive into more advanced concepts.

Using mwdblib for automation

mwdblib library is especially useful for writing external integrations.

Feeding MWDB service

Uploading a sample to the MWDB service allows you to access all the descendant objects and the results of automatic analysis. New sample can be uploaded using `upload_file` function.

```
from mwdblib import MWDB

file_name = sys.argv[1]

with open(file_name, "rb") as f:
    contents = f.read()

mwdb = MWDB(api_key=...)
file_object = mwdb.upload_file(file_name, contents)
```

If you want to serve your files as a public feed, it's recommended to:

- set a tag `feed:<feed name>` for each uploaded object, which allow people to filter samples coming from your feed.
- if additional insights about your sample are available in external service, it's nice to expose additional reference to that service as an attribute e.g. `<feed name>: <sample internal id>`
- finally, set public flag to share your sample with all users in MWDB

```
file_object = mwdb.upload_file(
    file_name,
    contents,
    metakeys={"maldb": file_id},
    public=True
)
file_object.add_tag("feed:maldb")
```

but remember that all of these things are **optional**. If you want to share additional insights about sample, you can **share them using comments**:

```
file_object.add_comment(f"Sample downloaded from {url}")
```

Note: If you want to serve a public feed for `mwdb.cert.pl`, contact people from CERT.pl (via Slack or info@cert.pl) to setup all the things e.g. to register attribute key if you want to share external link to your resource.

Using MWDB service as a feed

The next common use-case is to use a script for listening for recently added files or new configurations. MWDB does not support any kind of notifications, so you need to periodically ask for recently uploaded objects and retrieve if there is anything new.

For that type of tasks, mwdblib includes a group of helper methods called `listen_for_objects`. One of them is `listen_for_files` used in example presented below that downloads each new Portable Executable file from MWDB.

```
from mwdblib import MWDB

mwdb = MWDB(api_key="<secret>")

def report_new_sample(sample):
    print(f"Found new sample {sample.name} ({sample.sha256})")
    if "PE32" in sample.type:
        with open(sample.id, "wb") as f:
            f.write(sample.download())
        print("[+] PE32 downloaded successfully!")

for sample in mwdb.listen_for_files():
    report_new_sample(sample)
```

Sometimes you may want to keep track of the latest reported sample between script executions. In that case, mwdblib doesn't concern itself with persistence - you need to store the latest reported object ID on your own.

```
from mwdblib import MWDB

mwdb = MWDB(api_key="<secret>")

def store_last(last_id):
    """Stores last identifier in last_id file"""
    with open("last_id", "w") as f:
        f.write(last_id)

def load_last():
    """Recovers last identifier from last_id file"""
    try:
        with open("last_id", "r") as f:
            return f.read()
    except IOError:
        return None

def report_new_sample(sample):
    print(f"Found new sample {sample.name} ({sample.sha256})")
    if "PE32" in sample.type:
        with open(sample.id, "wb") as f:
            f.write(sample.download())
        print("[+] PE32 downloaded successfully!")

last_id = load_last()
```

(continues on next page)

(continued from previous page)

```
# Recovered last_id is passed to the listen_for_files function
for sample in mwdb.listen_for_files(last_id):
    report_new_sample(sample)
    store_last(sample.id)
```

Retrieving Karton analysis status

Our mwdb.cert.pl service offers automatic configuration extraction for malware samples. Analysis backend is based on Karton project, malware processing framework developed by CERT.pl that orchestrates whole analysis process.

Karton integration is not fully supported yet in mwdblib. Nevertheless, you can still retrieve the analysis status using APIClient interface, used by mwdblib to perform requests.

Warning: This part of documentation applies to the feature that is currently in **release candidate stage**. Things may work a bit different in stable release.

```
import time
from mwdblib import MWDB

...

mwdb = MWDB(api_key=...)

# Upload new file to the MWDB
file_object = mwdb.upload_file(file_name, contents)
karton_id = file_object.metakeys["karton"][0]

while True:
    time.sleep(5)
    # Get analysis status
    analysis_status = mwdb.api.get(f"object/{file_object.id}/karton/{karton_id}")["status"]
    ↪ print("Current status is '{analysis_status}'...")
    # If analysis is finished: break
    if analysis_status == "finished":
        break

# Get all configurations ripped during the analysis
configs = list(mwdb.search_configs(f'karton:"{karton_id}"'))
```

Optimizing API usage

During extensive usage of an API, you may hit the rate limit in MWDB service or performance limitations on your own instance. mwdblib retrieves all the information at the time of use (lazy-loading) and caches them in the object instance. This section will describe how to minimize the amount of requests and still get the required information.

First of all, you should turn on logging API requests performed by mwdblib. The most simple way is just to use logging.basicConfig with DEBUG level enabled.

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG)

>>> from mwdblib import MWDB
>>> mwdb = MWDB()
>>> mwdb.login()
Username: jkowalski
Password:
<redacted>/mwdblib/api.py:91: UserWarning: Password-authenticated sessions are short-
↳lived, so password needs to be stored in APIClient object. Ask MWDB instance-
↳administrator for an API key (send e-mail to info@cert.pl if you use mwdb.cert.pl)
warnings.warn("Password-authenticated sessions are short lived, so password needs to be-
↳stored "
DEBUG:urllib3.connectionpool:Starting new HTTPS connection (1): mwdb.cert.pl:443
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "POST /api/auth/login HTTP/1.1"
↳200 722
```

As you can see, login() method sends POST /api/auth/login request to authenticate and get the authorization token.

How lazy loading works?

MWDB API offers wide range of methods that are serving different portions of data. Object are represented by MWDBObject class hierarchy, storing cached property values fetched from API.

Cached values can be read using undocumented field MWDBObject.data. Let's try to check the difference in cache after fetching the same object using recent_objects() and recent_files()

```
>>> object = next(mwdb.recent_objects())
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "GET /api/object HTTP/1.1" 200 2034
>>> object.data
{'upload_time': '2020-10-12T14:15:09.719741+00:00',
 'tags': [{'tag': 'runnable:win32:exe'}, {'tag': 'feed:malwarebazaar'}],
 'id': 'cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71',
 'type': 'file'}

>>> file = next(mwdb.recent_files())
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "GET /api/file HTTP/1.1" 200 4698
>>> file.data
{'upload_time': '2020-10-12T14:15:09.719741+00:00',
 'file_name': '3e424264572d0d986fa3ae49c98f566ba7d8e2d7',
 'tags': [{'tag': 'runnable:win32:exe'}, {'tag': 'feed:malwarebazaar'}],
 'file_size': 211456,
 'sha256': 'cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71',
 'id': 'cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71',
```

(continues on next page)

(continued from previous page)

```
'file_type': 'PE32 executable (GUI) Intel 80386, for MS Windows',
'md5': 'e883226589b32952d07e057c468ffbb8',
'type': 'file'}
```

As you can see, the closer we are to the actual object type the more data are fetched using single request. The same thing we can observe comparing `query()` method fetching objects by SHA256 in general and `query_file()` which looks only for files.

```
>>> object = mwdb.query("cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71")
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "GET /api/object/
→ cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71 HTTP/1.1" 200 245
>>> object.data
{'parents': [],
 'upload_time': '2020-10-12T14:15:09.719741+00:00',
 'tags': [{'tag': 'runnable:win32:exe'}, {'tag': 'feed:malwarebazaar'}],
 'id': 'cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71',
 'children': [],
 'type': 'file'}

>>> file = mwdb.query_file(
→ "cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71")
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "GET /api/file/
→ cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71 HTTP/1.1" 200 850
>>> file.data
{'crc32': 'efb41668',
 'parents': [],
 'upload_time': '2020-10-12T14:15:09.719741+00:00',
 'ssdeep':
→ '3072:9ofJySUG7zULD7B4QutZ02L3Zt0OWBuQUNgRnJONVGMVKzJ7sAjf:WRye7ALD7B4QEZNwOWBdUNkJ0swAjf
→ ',
 'tags': [{'tag': 'runnable:win32:exe'}, {'tag': 'feed:malwarebazaar'}],
 'file_name': '3e424264572d0d986fa3ae49c98f566ba7d8e2d7',
 'file_size': 211456,
 'sha512':
→ '9520111d2cab4c760ee6a91148265dc3fbd65f37688ed8a9aeed543fe99a565c4fe47f22abbf067d2d81ddd4cc69106a9fdb
→ ',
 'latest_config': None,
 'sha256': 'cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71',
 'md5': 'e883226589b32952d07e057c468ffbb8',
 'id': 'cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71',
 'sha1': '3e424264572d0d986fa3ae49c98f566ba7d8e2d7',
 'file_type': 'PE32 executable (GUI) Intel 80386, for MS Windows',
 'children': [],
 'type': 'file'}
```

If required property value is not cached, mwdblib will choose the best API endpoint to fetch the property

```
>>> object = mwdb.query("cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71")
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "GET /api/object/
→ cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71 HTTP/1.1" 200 245
```

(continues on next page)

(continued from previous page)

```
>>> object.name
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "GET /api/file/
→ cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71 HTTP/1.1" 200 850
'3e424264572d0d986fa3ae49c98f566ba7d8e2d7'
>>> object.type
'PE32 executable (GUI) Intel 80386, for MS Windows'
>>> object.size
211456
>>> object.comments
DEBUG:urllib3.connectionpool:https://mwdb.cert.pl:443 "GET /api/object/
→ cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71/comment HTTP/1.1" 200
→ 287
[<mwdblib.comment.MWDBComment at ...>]
```

Sometimes you may need to flush the cache to fetch the refreshed state of object. In that case, use `flush()` method.

```
>>> file.flush()
>>> file.data
{'id': 'cea813cbef6581e0c95aacb2e747f5951325444b941e801164154917a17bfe71'}
```

Command-line interface (CLI)

MWDB library provides optional command line interface, which can be used to interact with MWDB repository.

Command-line interface requires extra `mwdblib[cli]` dependencies that can be installed using `pip`

```
$ mwdb version

[!] It seems that you haven't installed extra dependencies needed by CLI extension.
Best way to install them is to use `pip install mwdblib[cli]` command.
If it doesn't help, let us know and create an issue: https://github.com/CERT-Polska/mwdblib/issues
→ mwdblib/issues
Missing dependency: click

$ pip install mwdblib[cli]
...

$ mwdb version
3.3.0
```

User authentication

Before we start, we need to setup credentials. If you don't do that, you will be asked for them on each access to MWDB API.

```
$ mwdb login
Username: user
Password:
```

If you want, you can also provide your API key instead of storing your password in keyring.


```
$ mwdb login -A
Provide your API key token:
```

Just copy your API token from /profile view and paste into the CLI. All secrets will be stored in keyring and your username will be saved in ~/.mwdb file.

Looking for recent data

Let's start with listing the latest samples in your workspace. After typing

```
$ mwdb list
```

you will see the list of samples similar to the main MWDB webapp view.

Name/SHA256	Size	Type/Tags	Creation time
word1.tmp d5c95eae3316aa7a730c0397e307bfa0113d1e35c8b76b1adec8e22a6f484791	421.9 kB	PE32 executable (GUI) Intel 80386, for MS Windows feed:urlhaus runnable:win32.exe urlhaus:exe urlhaus:bur an	today
emotet.a22732be1da7ae878bdc01f7e2431030c616a071a56d5324f1771ef94 2a57e82.exe a22732be1da7ae878bdc01f7e2431030c616a071a56d5324f1771ef942a57e82	536.6 kB	PE32 executable (GUI) Intel 80386, for MS Windows runnable:win32.exe emotet update	today
400000_25390ea181bb808b 25390ea181bb808bf9b0c9e7a94a1a8aef92f775724c6e0cd522758831efd604	389.1 kB	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows netwire dump:win32.exe	today
400000_bb85f6c5d139bde5 bb85f6c5d139bde57b4ac27b96179b4e8cd626ae46892d8b6c02d6d6c7b88cd4	389.1 kB	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows netwire dump:win32.exe	today
22INC_67994550347393334_09242019.doc22	137.2 kB	Composite Document File V2 Document, Little Endian, Os: Windows, Version 6.1, Code page: 1252, Author: Dannie Konopelski, Template: Normal.dotm, Revision Number: 1,	today

If you don't like pager or coloring, you can use `nocolor` and `nopager` modifiers.

```
$ mwdb list -o nopager,nocolor --limit 5
```

Recent lists are limited by default to 200 entries. If you want to find more or less, you can use `--limit` option but be careful not to exceed the requests limit or your session will be temporarily throttled.

If you want only to get IDs of recent files, you need to use `short` modifier

```
$ mwdb list -o short -n 1
aad0d64af8363c58e9eada461dd0adace02569c508fb9979f080181e4a9f6b26
```

Gathering information about objects

If you want to get detailed information about specific object, use `get` subcommand.

```
$ mwdb get aad0d64af8363c58e9eada461dd0adace02569c508fb9979f080181e4a9f6b26
File name: 1fa0000_aad0d64af8363c58
File size: 1.4 MB
File type: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
MD5: f0c24d5c400ffa16bd34619cdec2f7c9
SHA1: 539be225e92d670f802c676ff9aa17513eeb0bcc
SHA256: aad0d64af8363c58e9eada461dd0adace02569c508fb9979f080181e4a9f6b26
SHA512: 3de184bf04b6e7dc82eb3c074352c4fb86770ce73488de2d06b8cd6bad4dfe71bc87971c82472be2
CRC32: f8a46646
SSDEEP: 24576:MZnluZRvQ/qpyr0krswnokR6AkE8YV4+8bXSRJAKf7/0VPHxtIdzAb+tUiu:jXvQ/qpyr0koWF
Upload time: today
Tags: emotet_spam dump:win32:exe
Parent tags: runnable:win32:exe emotet_update ripped:emotet ripped:emotet_spam
Child tags: <none>
```

Then file can be download using `fetch` command.

```
$ mwdb fetch aad0d64af8363c58e9eada461dd0adace02569c508fb9979f080181e4a9f6b26 --keep-name
$ ls
1fa0000_aad0d64af8363c58
```

If you'd like to store file under its original name, you can use `keep-name` option as presented above. File will be stored in current working directory.

In case your file is already stored in your local filesystem, you can just **provide the path instead of providing SHA256** - hash will evaluated automatically. For example, getting list of comments for locally stored `sample.exe` looks like below:

```
$ mwdb get comments ./sample.exe
```

Uploading files

Let's assume you want to upload `dropper.js`. Just type:

```
$ mwdb upload dropper.js
```

If you want to upload a drop called `drop.exe` and add relation to previously uploaded `dropper.js` you can specify parent:

```
$ mwdb upload dropper.js --parent drop.exe
```

... and if you want to suggest the family, add appropriate tag:

```
$ mwdb tag drop.exe maybe:netwire
```

Use `mwdb --help` to get the complete list CLI options and `mwdb <subcommand> --help` for details.

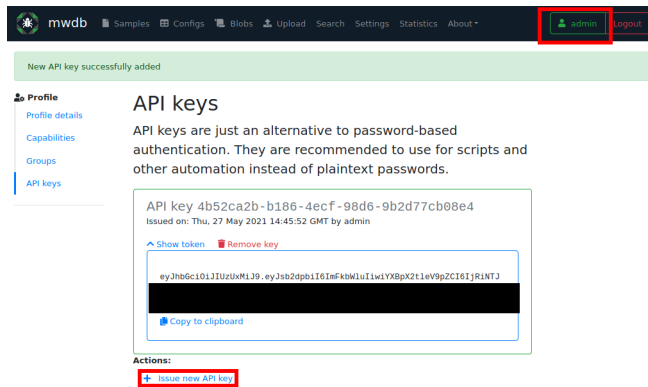
How to use API keys?

If you use `mwdb.login()` method, you may notice a warning that is shown by this method:

UserWarning: Password-authenticated sessions are short lived, so password needs to be stored in APIClient object. Ask MWDB instance administrator for an API key (send e-mail to info@cert.pl if you use `mwdb.cert.pl`)

Recommended authentication way to be used in scripts is to use API key token instead of password. List of created API keys can be found on the bottom of **Profile** view (next to the Logout in navbar) in **API keys** section.

To create a new API key, click on your nickname in navigation bar and go to the API keys section. Then click on + Issue new API key action:



Authorization token can be used in `mwdblib` via `api_key` argument:

```
mwdb = MWDB(api_key="eyJ...")
```

Using REST API directly (Non-Python integration)

You can use REST API directly via any HTTP client e.g. `curl`.

```
$ curl https://mwdb.cert.pl/api/ping
{"status": "ok"}

$ curl https://mwdb.cert.pl/api/file
{"message": "Not authenticated."}
```

Most API endpoints require authentication. Authorization token must be passed using **Authorization: Bearer** header.

```
$ curl https://mwdb.cert.pl/api/file -H "Authorization: Bearer <token>"
{"files": [{"upload_time": ...}
```

Complete API documentation can be found under the `/docs` endpoint. API documentation is based on [Swagger client](#) and you can interact with API directly from that page.

After logging in, click on **About** in the navbar and go to the **Docs** page.

[nfigs](#) [Blobs](#) [Upload](#) [Admin](#) [Search](#) [Statistics](#) [About](#)

About mwdb
Docs

MWDB

2.0.0-alpha4 OAS3

MWDB API documentation.

If you want to automate things, we recommend using [mwdblib library](#)

Servers

http://127.0.0.1/ - MWDB API endpoint

Authorize

server

GET /api/ping Ping server

GET /api/server Get server information

GET /api/docs Get server API documentation

Use **Try it out** button to send the API request.

GET /api/file/{identifier} Get file information

⛔

Returns information about file.

Parameters

Try it out

Name	Description
identifier * required	File identifier (SHA256/SHA512/SHA1/MD5)
string (path)	<div>47441974633c2b4fc925dc7e77673bde11e6e38f</div>

Code	Details
200	<p>Response body</p> <pre>{ "upload_time": "2020-10-08T14:33:37.802873+00:00", "sha512": "b46bbeec37483e1451794c6873314edecb3bfc2f149f180e60cbd1bf06a349f3d03c048d41e181130072bad-beb1e6f27b90cbbf88ba6d2d52f0a6b28abefb7c", "parents": [{ "upload_time": "2020-10-08T14:33:32.484343+00:00", "id": "f60d46232c9d94913dc6a775956a6d1b173c023965360eab48fb6e443f9864fd", "tags": [{ "tag": "emotet" }, { "tag": "ripped:emotet" }], "type": "file" }], "file_name": "431dfc0f9e1e0876288f004847441974633c2b4fc925dc7e77673bde11e6e38f", "sha256": "431dfc0f9e1e0876288f004847441974633c2b4fc925dc7e77673bde11e6e38f", "ssdeep": "3072:7hk3hbdlylKsqgopeJBWhZFGkE+cL2NdANxWqCDeu/XKV/E2b8RwBvXT08:lk3hbdlylKsqgopeJBWhZFVE+W2NdAN6" }</pre> <p>Download</p>

1.3.9 9. Sharing objects with other collaborators

Access control to the objects and features in MWDB is based on **groups**.

For MWDB users: group is a workspace, that allows to share the same view of uploaded objects across the various users. For MWDB administrators: group is also a way to give a specific set of permissions, depending on the role and level of trust.

In `mwdb.cert.pl`, we use them to group the users collaborating within the same organisation, to allow them to share their uploads and insights with other workmates or keep the information to be accessible only within trusted group of people.

Every user account is a member of at least two groups:

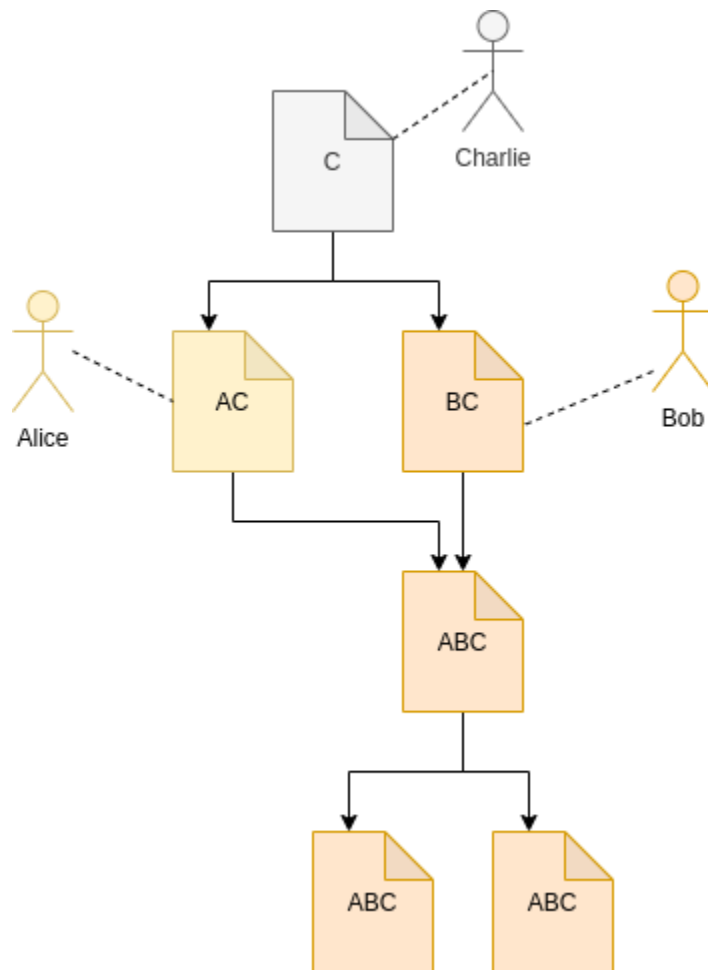
- user's private group, named the same as user login, which represents the exclusive user permissions.
- public group with permissions that apply to all users in MWDB.

Additionally, user can be member of registered group with permissions that apply only to non-guest users in MWDB.

Users' permissions are the sum of permissions of their groups.

Object access rules

The basic rule of sharing model in MWDB is that **group sees only their own uploads and all descendant objects**. In that model, you have permission to see the configuration for your sample and all the information derived from that configuration. Access to parent object implies the access to its children, but not the other way around.



As an user, you need to choose which groups you represent uploading the sample. There are four options:

- **All my groups** - the default option, which shares uploaded object with all groups you belong to, excluding public. That will share the object with all your workspaces to and your own private group. It means that even if you lose access to the workspace, you will still have access to your own uploads.
- **Single group...** - allows to share an object exclusively with chosen group and your own private group.
- **Everybody** - in that case, object is shared with public group, which means that everybody will have access to the uploaded object and all its descendants.
- **Only me** - your object will be shared only with your group.

Note that all options share the uploaded object with your private group.

Click here to upload

Parent (Optional) Type parent identifier... Clear

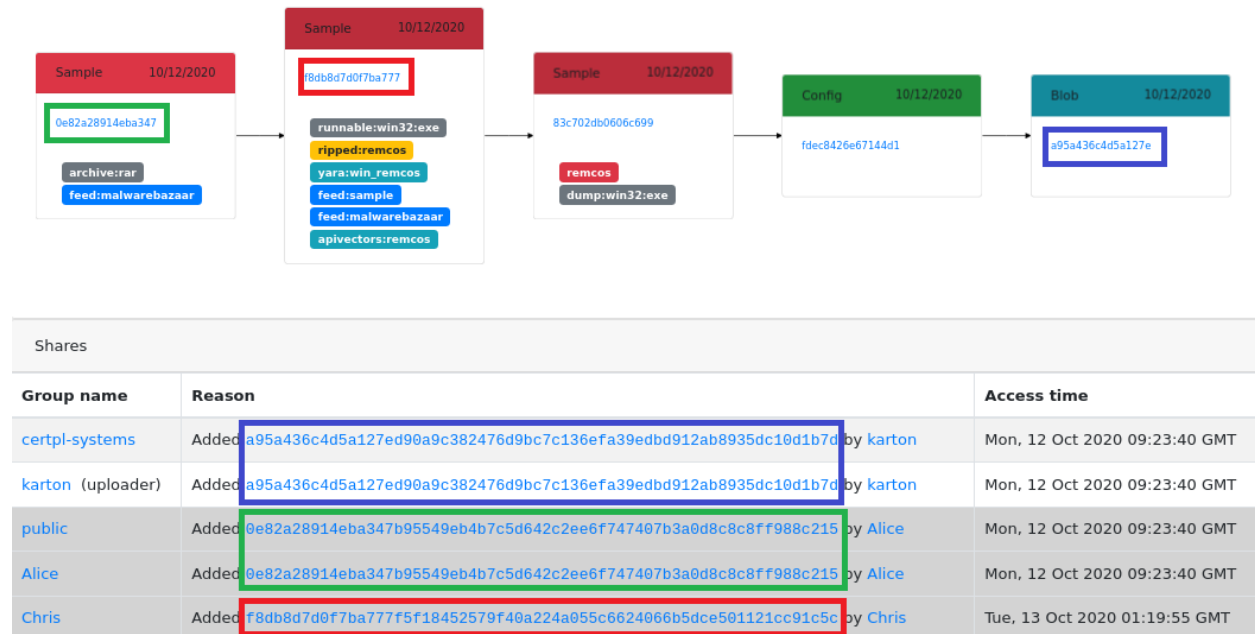
Share with

All my groups
 All my groups
 Single group...
 Everybody
 Only me

Upload File

Note: If you are not a member of any additional group, you will see only **Everybody** and **Only me** (default) options

Current object access rules are visible in Shares box. Entries with the same identifier as currently watched object are originating from the upload of that object. Others are marked with gray background and they are inherited from the parent objects.



In example presented above:

- **karton** added blob (blue colored) with config as a parent
- **certpl-systems** group has access to blob because Karton is a member of that group. Uploader (Karton) decided to share uploaded object with all groups.
- **Alice** has access to blob because she added the original archive (green colored) before
- **public** group has access because Alice decided that she want to share archive with everybody
- **Chris** added Remcos sample (red colored) directly one day later, so he has got additional exclusive access to the blob. If the archive was not added to the *public* group he would still have access.

Who is who? User visibility rules

Regular users are able to see only their own groups. It means that they're able to only see these users that are members of their groups.

Joining the group, you are allowed to:

- share objects with its members
- search for their uploads using **shared:** and **uploader:** queries (but only within groups common for both users)
- see their profiles and membership in other common groups

Rules above doesn't apply to groups marked as **Role groups** e.g. **public** or **registered**.

But there are few exclusions for that, when your login may be visible for all users in MWDB regardless of group membership:

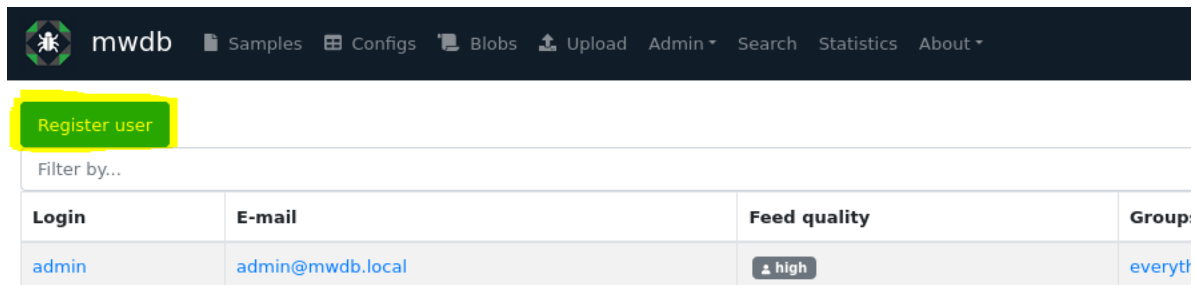
- when you are the first one sharing an object with `public` group
- when you add a comment (login of author)

How to add new user/group?

Users and groups can be managed by administrator using Users and Groups views in Settings menu.

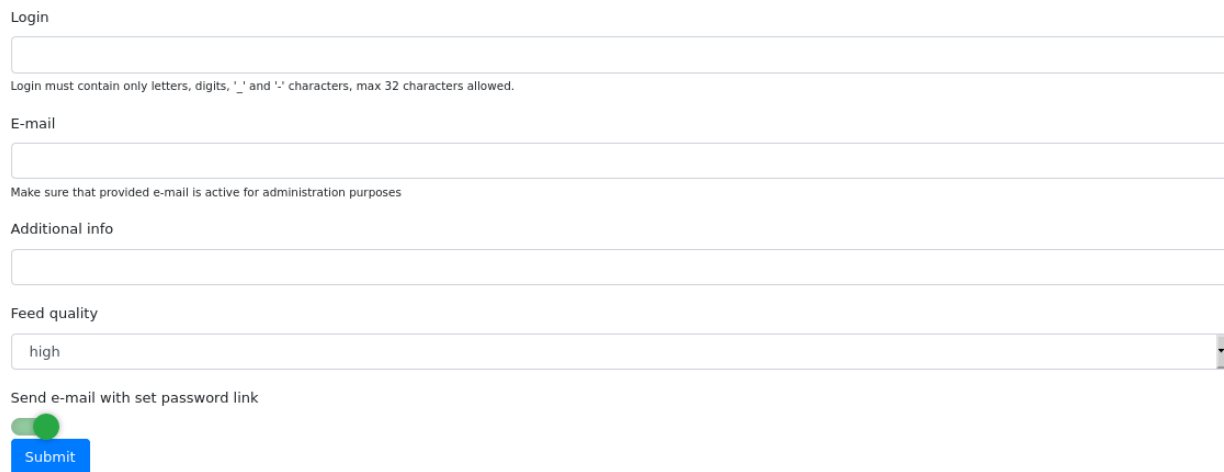
Create a new user

To create a new user, go to Settings/Users and click on Register user button.



Filter by...	Login	E-mail	Feed quality	Group
	admin	admin@mwdb.local	high	everyt

Create new user



Login

Login must contain only letters, digits, '_' and '-' characters, max 32 characters allowed.

E-mail

Make sure that provided e-mail is active for administration purposes

Additional info

Feed quality

high

Send e-mail with set password link

☒

Submit

Then fill the form with the following information:

- **Login** and **E-mail** that will be used for authentication, password recovery etc.
- **Additional info** (optional) to store additional description of user account
- **Feed quality** which is useful for plugins to determine if user account is associated with automatic feed (low) or human user (high, default).

By default - MWDB sends an e-mail to the new user with set password link, but if you have not configured SMTP service: disable **Send e-mail with set password link** first.

After clicking on **Submit**, you will be redirected to user settings.

Using user settings, you can add user to additional groups and generate set password link. Go to the bottom of the page and click on the **Change password** action.

Pass the reset password link to the user to let them set a new password for an account.

Create a new group

To create a new group, go to **Settings/Groups** and click on **Create group** button.

Create group	
Filter by...	
Name	Members
everything	admin
public	(Group is public and contains all members)

Create group


Name

Group name must contain only letters, digits, '-' and '.' characters, max 32 characters allowed.

Submit

Set name for a new group. After clicking on **Submit**, you will be redirected to group settings.

Group details: new_group


Group name	new_group	Edit 
Members		

Group features:

Workgroup Enabled
 Converts group to the workgroup, so users will see each other within this group. Enabled by default for user-defined groups. Disable

Default group
 Automatically adds new users to this group. Enable

Actions:

- [Search for uploads](#)
- [Search for shared files](#)
- [Show group members](#)
- [Check group capabilities](#)
-  [Remove group](#)

In group settings view, you can add members to the new group. Go to **Access control** if you want to set additional capabilities for group.

Group capabilities (superpowers)

All groups can have additional permissions that apply to all members. MWDB by default is quite restrictive and regular user accounts are allowed only to upload samples and access the object information. That default prevents breaking the existing conventions or making potentially irreversible actions, but even in CERT.pl we don't apply such limitations for users.

You can change the capabilities for group and users, using `Access control` view.

Access control

Use a form below to enable/disable capabilities for specific user or group.

Provide group name first

<div> everything </div> Applies to <code>admin</code>	
<input checked="" type="checkbox"/>	<div>access_all_objects</div> Has access to all new uploaded objects into system
<div> registered </div> Applies to <code>karton</code> , <code>admin</code>	
<input checked="" type="checkbox"/>	<div>adding_files</div> Can upload files
<input checked="" type="checkbox"/>	<div>manage_profile</div> Can manage own profile
<input checked="" type="checkbox"/>	<div>personalize</div> Can mark favorites and manage own quick queries
<div> admin </div>	
<input checked="" type="checkbox"/>	<div>manage_users</div> Managing users and groups (system administration)
<input checked="" type="checkbox"/>	<div>share_queried_objects</div> Query for all objects in system
<input checked="" type="checkbox"/>	<div>access_all_objects</div> Has access to all new uploaded objects into system
<input checked="" type="checkbox"/>	<div>sharing_objects</div> Can share objects with all groups in system

By default, `admin` private group has enabled all capabilities. All other groups are created with all disabled.

Each capability has its own name and scope:

- **manage_users - Managing users and groups (system administration)**

Allows to access all users and groups in MWDB. Rules described in *Who is who?* don't apply to users with that permission. Enables user to create new user accounts, new groups and change their capabilities and membership. Allows to manage attribute keys, define new ones, delete and set the group permissions for them.

- **share_queried_objects - Query for all objects in system**

That one is a bit tricky and will be possibly deprecated. MWDB will automatically share object and all descendants with group if member directly accessed it via identifier (knows the hash e.g. have direct link to the object). It can be used for bot accounts, so they have access only to these objects that are intended to be processed by them. Internally, we abandoned that idea, so that capability may not be stable.

- **access_all_objects - Has access to all uploaded objects into system**

Grants access to all uploaded objects in MWDB.

- **sharing_with_all - Can share objects with all groups in system**

Implies the access to the list of all group names, but without access to the membership information and management features. Allows to share object with arbitrary group in MWDB. It also allows the user to view full history

of sharing an object (if the user has access to the object).

- **access_uploader_info - Can view who uploaded object and filter by uploader**

Can view who uploaded object and filter by uploader. Without this capability users can filter by / see only users in their workspaces.

- **adding_tags - Can add tags**

Allows to tag objects. This feature is disabled by default, as you may want to have only tags from automated analyses.

- **removing_tags - Can remove tags**

Allows to remove tags. Tag doesn't have "owner", so user will be able to remove all tags from the object.

- **adding_comments - Can add comments**

Allows to add comments to the objects. Keep in mind that comments are public.

- **removing_comments - Can remove (all) comments**

Allows to remove **all** comments, not only these authored by the user.

- **adding_parents - Can add parents**

Allows to add new relationships by specifying object parent during upload or adding new relationship between existing objects.

- **removing_parents - Can remove parent of object and inherited permissions from that relation**

Allows to remove relationships along with all inherited permissions.

- **adding_files - Can upload files**

Enables upload of files. Enabled by default for registered group.

- **adding_configs - Can upload configs**

Enables upload of configurations. Configurations are intended to be uploaded by automated systems or trusted entities that follow the conventions.

- **adding_blobs - Can upload text blobs**

Enables upload of blobs. Blobs may have similar meaning as configurations in terms of user roles.

- **reading_all_attributes - Has access to all attributes of object (including hidden)**

With that capability, you can read all the attributes, even if you don't have read permission for that attribute key. It allows to list hidden attribute values.

- **adding_all_attributes - Can add all attributes to object**

Enables group to add all the attributes, even if it doesn't have set permission for that attribute key.

- **removing_attributes - Can remove attribute from objects**

Allows to remove attribute from object. To remove attribute, you need to have set permission for key. Combined with adding_all_attributes, allows to remove all attributes.

- **unlimited_requests - API requests are not rate-limited for this group**

Disables rate limiting for users from that group, if rate limiting feature is enabled.

- **removing_objects - Can remove objects**

Can remove all accessible objects from the MWDB. May be quite destructive, we suggest to keep that capability enabled only for admin account.

- **manage_profile - Can manage profile**

Allows to change personal authentication settings like issuing/deleting own API keys and resetting password.

- **personalize - Can mark favorites and manage own quick queries**

Allows to use personalization features like favorites or quick queries.

- **karton_assign - Can assign existing analysis to the object**

Allows to assign Karton analysis to the object by setting `karton` attribute or using dedicated API.

- **karton_reanalyze - Can resubmit any object for analysis**

Can manually resubmit object to Karton.

- **modify_3rd_party_sharing - Can mark objects as shareable with 3rd parties**

Can manually mark object as shareable with 3rd parties - it can be done only for objects, which are visible for this user.

User capabilities are the sum of all group capabilities. If you want to enable capability system-wide (e.g. enable all users to add tags), enable that capability for `registered` group or `public` group if you want to include guests.

In `mwdb.cert.pl` service - `registered` group is allowed to:

- add new tags
- add new comments
- add relationships (parents)
- have access to extended features provided by internal plugins

You can easily check your capabilities in `Profile` view.

Plugins are allowed to extend the set of capabilities in case MWDB administrator wants to require additional permission for using them.

Sharing with third parties

Files, configs and blobs uploaded to `mwdb.cert.pl` may be shared with our partners. Sharing occurs when the object is analyzed by *karton* - some pipelines may share data with third parties. If you want your upload not to be shared with third parties, you can specify it during upload (with MWDB 2.9.0 or newer).

After upload, marking objects as shareable is possible only for users with `modify_3rd_party_sharing` capability.

If object was shareable at any time, we reserve right to share it:

- If one object was uploaded multiple times and at least one of them was marked as shareable with third parties,

the object will be marked as shareable - Marking objects as shareable is irreversible

1.4 Integration guide

MWDB comes with advanced plugin engine, which allows to add new API features, integrate MWDB with other systems using webhooks and extend MWDB UI functionality.

Plugins are used by `mwdb.cert.pl` to:

- integrate MWDB with malware analysis backend and reporting systems
- provide new features like mquery search
- customize `mwdb.cert.pl` instance

Note: This chapter describes only the most basic features of plugin system, allowing to write simple integrations. More plugin system features will be documented in the near future.

1.4.1 Getting started with local plugins

Backend plugins are just Python packages imported by MWDB from specified location. Let's check the plugin settings in `mwdb.ini`:

```
### Plugin settings

# Set enable_plugins to 0 to turn off plugins (default: 1)
# enable_plugins = 0

# List of plugin module names to be loaded, separated by commas
# plugins =

# Directory that will be added to sys.path for plugin imports
# Allows to load local plugins without installing them in site-packages
# local_plugins_folder = ./plugins

# Autodiscover plugins contained in local_plugins_folder (default: 0)
# local_plugins_autodiscover = 1
```

Plugins can be loaded from installed packages or imported from `local_plugins_folder`.

Let's create a simple, local `hello_world` plugin:

```
plugins
├─ hello_world
│   └─ __init__.py
```

and put short description in `__init__.py` file:

```
__author__ = "just me"
__version__ = "1.0.0"
__doc__ = "Simple hello world plugin"
```

Then, set `mwdb.ini` file to load your `hello_world` plugin. If you configured `mwdb-core` to use current directory, you should find that file there. If not, you can still overwrite the `mwdb.ini` settings by creating another `mwdb.ini` file in the current working directory, where `mwdb-core` run is invoked.

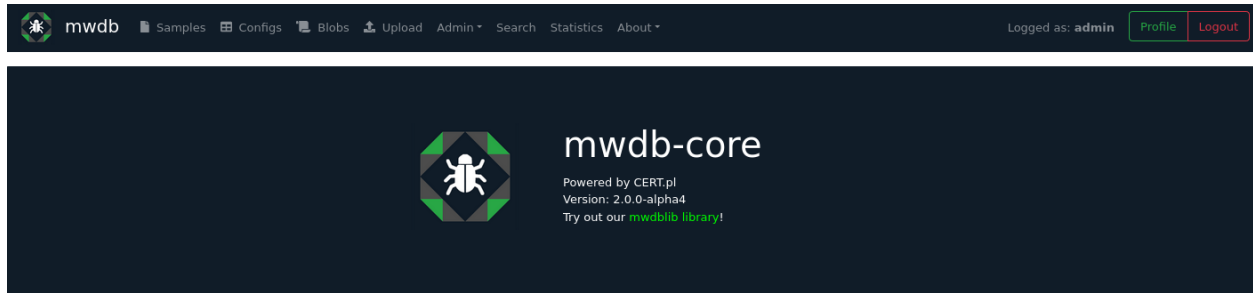
```
[mwdb]
...

plugins = hello_world
local_plugins_folder = ./plugins
```

Let's run the mwdb-core:

```
$ mwdb-core run
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
[INFO] MainThread - plugins.load_plugins:141 - Loaded plugin 'hello_world'
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

As you can see in logs, your plugin has been loaded successfully. You can additionally check that using /about endpoint in UI.



Plugin name	Description	Version
hello_world Active	Simple hello world plugin	1.0.0

1.4.2 Adding webhook

Now, let's make it a bit more useful and add the actual webhook. When plugin module is loaded by MWDB, it calls the entrypoint function named `__plugin_entrypoint__`.

Modify the `__init__.py` file to implement simple entrypoint saying "Hello world!".

```
import logging

from mwdb.core.plugins import PluginAppContext

__author__ = "just me"
__version__ = "1.0.0"
__doc__ = "Simple hello world plugin"

logger = logging.getLogger("mwdb.plugin.hello_world")

def entrypoint(app_context: PluginAppContext):
```

(continues on next page)

(continued from previous page)

```
logger.info("Hello world!")

__plugin_entrypoint__ = entrypoint
```

The expected result is:

```
$ mwdb-core run
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
[INFO] MainThread - __init__.entrypoint:14 - Hello world!
[INFO] MainThread - plugins.load_plugins:141 - Loaded plugin 'hello_world'
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

PluginAppContext object allows to provide extension for MWDB like adding webhook handler and extending the API.

Webhook handler is implemented by providing a new class that inherits from PluginHookHandler. New handler class can be then registered using app_context.register_hook_handler method.

```
import logging

from mwdb.core.plugins import PluginAppContext, PluginHookHandler
from mwdb.model import File

__author__ = "just me"
__version__ = "1.0.0"
__doc__ = "Simple hello world plugin"

logger = logging.getLogger("mwdb.plugin.hello_world")

class HelloHookHandler(PluginHookHandler):
    def on_created_file(self, file: File):
        logger.info("Nice to meet you %s", file.file_name)

    def on_reuploaded_file(self, file: File):
        logger.info("Hello again %s", file.file_name)

def entrypoint(app_context: PluginAppContext):
    logger.info("Hello world!")
    app_context.register_hook_handler(HelloHookHandler)

__plugin_entrypoint__ = entrypoint
```

After applying above modifications to __init__.py, let's restart mwdb-core and add a new to file and check if it works.

```
[INFO] Thread-3 - __init__.on_created_file:16 - Nice to meet you evil.exe
[INFO] Thread-3 - object.create_object:88 - File added -_
↳dhash:9e302844386835ef50bec3017e2c60705ab6bf33e4849e58e3af19a605b46d00 - is_new:True
...
[INFO] Thread-12 - __init__.on_reuploaded_file:19 - Hello again evil.exe
[INFO] Thread-12 - object.create_object:88 - File added -_
↳dhash:9e302844386835ef50bec3017e2c60705ab6bf33e4849e58e3af19a605b46d00 - is_new:False
```

Webhooks can be used to automatically analyze the uploaded file in sandbox. The good example is [mwdb-plugin-drakvuf](#) which implements webhook that sends all uploaded files to the [Drakvuf Sandbox](#) for analysis.

Check out [mwdb-plugin-drakvuf](#) on Github!

1.4.3 Available hooks

A lot of hooks have been implemented in MWDB. Each of these hooks is triggered when particular event occurs in system.

List of available hooks and events triggering these hooks.

- `on_created_object(self, object: Object)` - object was uploaded (file, blob or config) or pulled from remotd resource
- `on_reuploaded_object(self, object: Object)` - object was again uploaded or pulled from remote resource
- `on_removed_object(self, object: Object)` - object was deleted
- `on_created_file(self, file: File)` - file was uploaded or pulled from remotd resource
- `on_reuploaded_file(self, file: File):` - file was again uploaded or pulled from remote resource
- `on_removed_file(self, file: File)` - file was deleted
- `on_created_config(self, config: Config)` - config was uploaded or pulled from remotd resource
- `on_reuploaded_config(self, config: Config)` - config was again uploaded or pulled from remote resource
- `on_removed_config(self, config: Config)` - config was deleted
- `on_created_text_blob(self, blob: TextBlob)` - text blob was uploaded or pulled from remotd resource
- `on_reuploaded_text_blob(self, blob: TextBlob)` - text blob was again uploaded or pulled from remote resource
- `on_removed_text_blob(self, blob: TextBlob)` - text blob was deleted
- `on_created_tag(self, object: Object, tag: Tag)` - a new tag was created and assigned to object
- `on_reuploaded_tag(self, object: Object, tag: Tag)` - tag was again assigned to object
- `on_removed_tag(self, object: Object, tag: Tag)` - tag was removed from object
- `on_created_comment(self, object: Object, comment: Comment)` - a new comment was created and assigned to object
- `on_removed_comment(self, object: Object, comment: Comment)` - comment was removed from object

- `on_created_relation(self, parent: Object, child: Object)` - relation between parent and child objects was added
- `on_removed_relation(self, parent: Object, child: Object)` - relation between parent and child objects was removed
- `on_created_attribute_key(self, attribute_def: AttributeDefinition)` - attribute definition was created
- `on_updated_attribute_key(self, attribute_def: AttributeDefinition)` - attribute definition was updated
- `on_removed_attribute_key(self, attribute_def: AttributeDefinition)` - attribute definition was removed
- `on_created_attribute(self, object: Object, attribute: Attribute)` - attribute was assigned to object
- `on_removed_attribute(self, object: Object, attribute: Attribute)` - attribute was removed from object
- `on_created_user(self, user: User)` - a new user account was created (also using OpenID Provider)
- `on_removed_user(self, user: User)` - user account was removed
- `on_updated_user(self, user: User)` - user account was updated
- `on_created_group(self, group: Group)` - a new group was created. Also when a new user is registered and his private group is created
- `on_removed_group(self, group: Group)` - group was removed. Also when a user is deleted and his private group is removed
- `on_updated_group(self, group: Group)` - group attributes were updated
- `on_created_membership(self, group: Group, user: User)` - user was added to the group
- `on_removed_membership(self, group: Group, user: User)` - user was removed from the group
- `on_updated_membership(self, group: Group, user: User)` - membership was updated
- `on_changed_object(self, object: Object)` - this hook is triggered when one of the undermentioned events takes place:
 - a new tag was created and assigned to object
 - tag was removed from object
 - a new comment was created and assigned to object
 - comment was removed from object
 - relation between parent and child objects was added
 - relation between parent and child objects was removed
 - attribute was assigned to object
 - attribute was removed from object

1.4.4 Creating web plugins

MWDB Core comes with powerful web plugin engine which allows to extend almost any component within MWDB UI. For a long time it was an undocumented feature used mainly by mwdb.cert.pl service and built on top of Create React App hacks and overrides.

Starting from v2.9.0 release, we're using joined powers of [Vite](#) and [Rollup](#) to make it a real thing.

Web plugins: getting started

Frontend plugins are a very different animal from Python backend plugins and you may need a bit more knowledge about build-time mechanisms.

Let's go to the `docker/plugins` directory within `mwdb-core` repository and extend our `hello_world` plugin:

```
docker
├── plugins
│   └── hello_world
│       ├── __init__.py
│       ├── index.jsx
+       └── package.json
+       
```

First is `package.json` that contains short specification of our plugin. Name must contain `@mwdb-web/plugin-` prefix.

```
{
  "name": "@mwdb-web/plugin-hello-world",
  "version": "0.0.1",
  "main": "./index.jsx"
}
```

Finally we can write simple plugin that adds new `Hello world` page. Let's check `index.jsx` contents:

```
// Imports from React and Font Awesome libraries
import React from 'react';
import { Route, Link } from 'react-router-dom';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faHandHoldingHeart } from '@fortawesome/free-solid-svg-icons';

// Import from MWDB Core commons and components
import { View } from '@mwdb-web/commons/ui';
import { AboutView } from '@mwdb-web/components/Views/AboutView';

function HelloWorld() {
  return (
    <View>
      <h1>Hello world!</h1>
      Nice to see you!
      <hr />
      <About/>
    </View>
  )
}

export default () => ({
```

(continues on next page)

(continued from previous page)

```

routes: [
  <Route path='hello' element={<HelloWorld />} />
],
navbarAfter: [
  () => (
    <li className="nav-item">
      <Link className="nav-link" to={"/hello"}>
        <FontAwesomeIcon
          className="navbar-icon"
          icon={faHandHoldingHeart}
        />
        Hello there!
      </Link>
    </li>
  )
],
})

```

After setting up all of the things, run `docker-compose -f docker-compose-dev.yml build` and `docker-compose -f docker-compose-dev.yml up` to run the application. If everything is OK, you should see the results like below:

<show the result>

But what actually happened in that `index.jsx` file? there are lots of things going there!

Let's focus on most important ones:

- Line starting with `export default` is actually an endpoint of our plugin. It exports callback that is called after plugin is loaded.
- Endpoint callback is expected to return an object that contains specification of extensions provided by plugin.
- Our plugin contains two extensions: * `routes` that implement React Router routes to be included in web application * `navbarAfter` being a list of React component functions that will be rendered after navbar
- Plugins adds new navbar button `Hello there!` and `/hello` route rendering `HelloWorld` component. Our new component uses `View` from `@mwdb-web/commons/ui` which is common wrapper for main views used within application. In addition, it renders `About` view imported from `@mwdb-web/components` just under our greeting.

But where is actual list of possible extensions defined? They're defined in core application code and can be found by references to few methods and wrappers from `common/plugins` :

- `fromPlugins` collects specific type of extension from all loaded plugins and returns a list of them. For example: new routes to be added.
- `Extension` does the same but treats all collected objects as components and renders them.
- `Extendable` wraps object with `<name>Before`, `<name>Replace` and `<name>After` extensions, so we can add extra things within main views.

So navbar is one of `Extendable` wrappers that can be found within application and that's why we can add extra navbar item.

Web plugins: how it works internally?

There are two requirements to be fulfilled by the plugin engine:

- Plugin code needs to be loaded and executed along with the core application
- Plugin must be allowed to reuse and extend the core application parts

MWDB uses Rollup import aliases and Vite virtual modules to make a link between plugin code and the core.

1. Vite build runtime looks for `@mwdb-web/plugin-*` packages that are installed in `node_modules`

They can be regular packages or just links (see also [npm-link](#))

2. `@mwdb-web/plugins` virtual module defines dynamic imports that are further resolved by Vite to create separate bundles for plugins that can be asynchronously loaded.

Virtual module code looks like below:

```
export default {  
  "plugin-example": import("@mwdb-web/plugin-example"),  
  ...  
}
```

3. `@mwdb-web/plugins` package is then resolved at runtime by `commons/plugins` loader that resolves dynamic imports and collects hook specification. Plugins are loaded before first render occurs.

When plugin loader finishes its job, initial render kicks in and plugin is finally able to extend the application. MWDB uses Rollup capabilities to make plugins able to use components from web source root (`mwdb/web/src`) and expose them as `@mwdb-web/*` aliases:

- **`@mwdb-web/commons` contains core parts of application that are expected to be used by plugins as well.**
Main packages are:
 - `api` module serving backend REST API bindings built on top of Axios
 - `auth` module serving `AuthContext` with information about currently authenticated user
 - `config` module with current server configuration and useful globals
 - `helpers` with useful helper methods
 - `ui` with UI components and utilities
- `@mwdb-web/components` contains implementation of all application views and there is higher chance that something will break across the versions if you use them directly.

1.4.5 Building customized images

If you want to extend MWDB with new features using the plugin system, it's always useful to be able to build your own customized Docker images.

There are two ways to do that:

1. Simple way: clone <https://github.com/CERT-Polska/mwdb-core> repository. Then place your plugins in `docker/plugins` and use Dockerfiles from `deploy/docker` to build everything from scratch.
2. More extensible way: use `certpl/mwdb` and `certpl/mwdb-web-source` as base images and make your own Dockerfiles. This method enables you to install additional dependencies and provide custom plugin-specific overrides.

Building custom backend image is simple as in *Dockerfile* below:

```
# It's recommended to pin to specific version
ARG MWDB_VERSION=v2.9.0
FROM certpl/mwdb:$MWDB_VERSION

# Install any Alpine dependencies you need
RUN apk add p7zip
# Install any Python dependencies you need (certpl/mwdb image uses venv internally)
RUN /app/venv/bin/pip install malduck

# Copy arbitrary backend plugins and mail templates
COPY mail_templates /app/mail_templates
COPY plugins /app/plugins
```

Backend plugins are linked in runtime, so that part is pretty easy to extend. A bit more complicated thing is frontend part:

```
ARG MWDB_VERSION=v2.9.0
FROM certpl/mwdb-web-source:$MWDB_VERSION AS build

# Copy web plugins code
COPY plugins /app/plugins

# Set workdir to /app, install plugins to ``/app/node_modules`` and rebuild everything
WORKDIR /app
RUN npm install --unsafe-perm $(find /app/plugins -name 'package.json' -exec dirname {} \
  ↪; | sort -u) \
  && CI=true npm run build

# Then next stage is copied from https://github.com/CERT-Polska/mwdb-core/blob/master/
  ↪deploy/docker/Dockerfile-web
# You need to copy start-web.sh and nginx.conf.template as well, or adapt them according
  ↪to your needs
FROM nginx:stable

LABEL maintainer="admin@example.org"

ENV PROXY_BACKEND_URL http://mwdb.:8080

COPY nginx.conf.template /etc/nginx/conf.d/default.conf.template
COPY start-web.sh /start-web.sh
COPY --from=build /app/dist /usr/share/nginx/html

# Give +r to everything in /usr/share/nginx/html and +x for directories
RUN chmod u=rX,go=-R /usr/share/nginx/html

# By default everything is owned by root - change owner to nginx
RUN chown nginx:nginx -R /usr/share/nginx/html

CMD ["/bin/sh", "/start-web.sh"]
```

1.4.6 Room for improvement

Plugin system was created mainly for mwdb.cert.pl, so not everything may fit your needs. Also things may break from time to time, but as we maintain our internal plugins ourselves, most important changes will be noted in changelog. You can also find broader explanation and migration recipes in What's changed chapter.

So if you need another `Extendable` place within UI or yet another hook within backend: feel free to [create issue](#) on our GitHub repository.

1.5 Extra features

MWDB provides additional features such as:

- User registration features (hosting public services with vetting)
- Rate limiting
- Family statistics

These will be documented in further releases.

1.6 Developer guide

1.6.1 Setting up development environment

Generate configuration using `./gen_vars.sh` as for production installation.

Then build images using

```
docker-compose -f docker-compose-dev.yml build
```

and run MWDB via

```
docker-compose -f docker-compose-dev.yml up -d
```

After a minute - MWDB should be accessible via `http://127.0.0.1` with enabled hot-reload and debug facilities.

All changes in code are automatically reloaded excluding:

- Changes in database model, which need migrations
- Changes in configuration
- Registering new plugins or adding frontend extension to existing plugins without that feature

In cases mentioned above - Docker images need to be rebuilt.

Password for administration account is available in `mwdb-vars.env` file in `MWDB_ADMIN_PASSWORD` field.

1.6.2 Testing mail-related features

Development environment is configured to use [Mailhog](#) as SMTP server.

Mailhog provides very convenient webmail collecting all outgoing e-mails which are available here: <http://127.0.0.1:8025>

You may also add any external SMTP server using the following environment variables:

```
MWDB_MAIL_SMTP = "smtp_server:port"
MWDB_MAIL_FROM = "name@example.com"
MWDB_MAIL_USERNAME = "your_username" # optional
MWDB_MAIL_PASSWORD = "your_password" # optional
MWDB_MAIL_TLS = 1 # Enables StartTLS, optional, defaults to 0
```

1.6.3 Auto generating Alembic migrations

Let's say you made changes in model (e.g. added some table) but your feature still doesn't work and MWDB reports that something is wrong with database. That's because you need to provide appropriate migration script, that will apply your model changes to database. Fortunately, [Alembic](#) is very helpful when we deal with simple changes like providing new nullable columns or dropping the tables.

```
class Comment(db.Model):
    __tablename__ = 'comment'
    id = db.Column(db.Integer, primary_key=True)
    comment = db.Column(db.String, nullable=False, info=ColumnToSearchInDict)
+   likes = db.Column(db.Integer)
    timestamp = db.Column(db.DateTime, nullable=False, default=datetime.datetime.utcnow)
```

In development version of Docker Compose file, migrations are mounted as volume, so we can use Flask CLI features directly from the container.

Make sure that development Docker Compose is up. Then spawn interactive shell:

```
$ docker-compose -f docker-compose-dev.yml exec mwdb /bin/sh
/app #
```

Then use `flask db migrate` command to generate migration

```
/app # flask db migrate
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.ddl.postgresql] Detected sequence named 'tag_id_seq' as owned by integer_
↳column 'tag(id)', assuming SERIAL and omitting
INFO [alembic.ddl.postgresql] Detected sequence named 'object_id_seq' as owned by_
↳integer column 'object(id)', assuming SERIAL and omitting
INFO [alembic.ddl.postgresql] Detected sequence named 'comment_id_seq' as owned by_
↳integer column 'comment(id)', assuming SERIAL and omitting
INFO [alembic.ddl.postgresql] Detected sequence named 'metakey_id_seq' as owned by_
↳integer column 'metakey(id)', assuming SERIAL and omitting
INFO [alembic.autogenerate.compare] Detected added column 'comment.likes'
Generating /app/migrations/versions/c22b64a416e9.py ... done
```

In this case we can find our migration script `./migrations/versions/c22b64a416e9.py`. Check if everything is alright and apply migration.

```
/app # flask db upgrade
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade 426a27b8e97c -> c22b64a416e9, empty_
↪message
```

Now, your feature is ready for tests. If you restart your Docker environment, all migrations will be applied automatically.

If you need to write change on your own e.g. because you need to migrate data instead of schema: use `flask db revision`

```
/app # flask db revision
Generating /app/migrations/versions/6819021086c5_.py ... done
```

Note: Alembic migrations generated inside container will be owned by root. If you have problem with permissions, use `chown` inside container to change the owner to your local UID.

1.7 Remote instances guide

New in version 2.2.0.

Warning: This is **experimental** feature and it may significantly change in the future versions.

Remote instance feature allows us to connect our local repository with other, external MWDB servers. This gives us the ability to view samples located in the remote instance, as well as exchange the objects between the remote and the local instance in both directions.

Using that feature, you can easily synchronize samples between your local MWDB Core repository and our `mwdb.cert.pl` service.

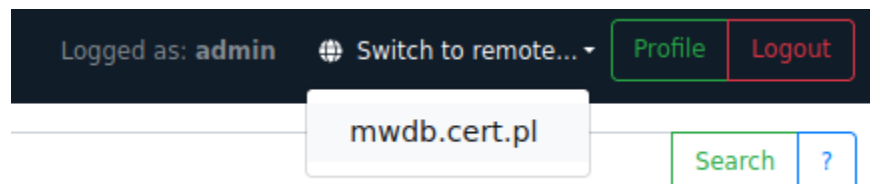
Depending on the given permissions on the remote instance, the following things can be performed:

- viewing samples, configs and blobs
- pulling an object from a remote instance
- performing a push action to send a local object to the remote instance

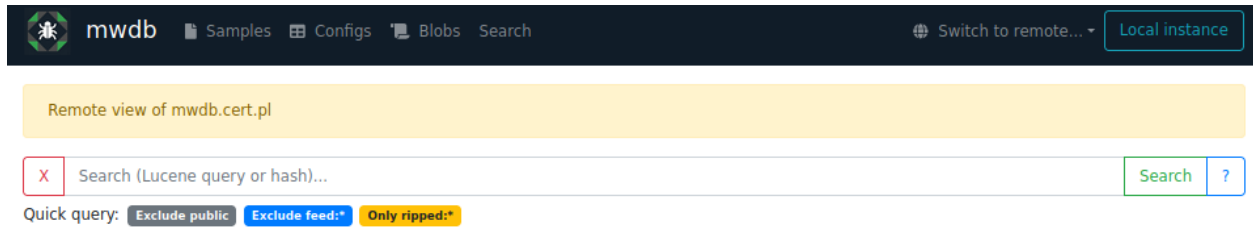
Remote instance UI is read-only and currently limited to the basic MWDB views.

1.7.1 Remote instance features

We can switch to the remote MWDB instance using the new dropdown in the right side of the navigation bar. After clicking on it, a list of defined remotes will appear. Selecting a given instance will take you to its remote view.



To go back to your local instance, just click on the button `Local instance`. You can also change the current remote instance by selecting a different one from the list under `Switch to remote...`.

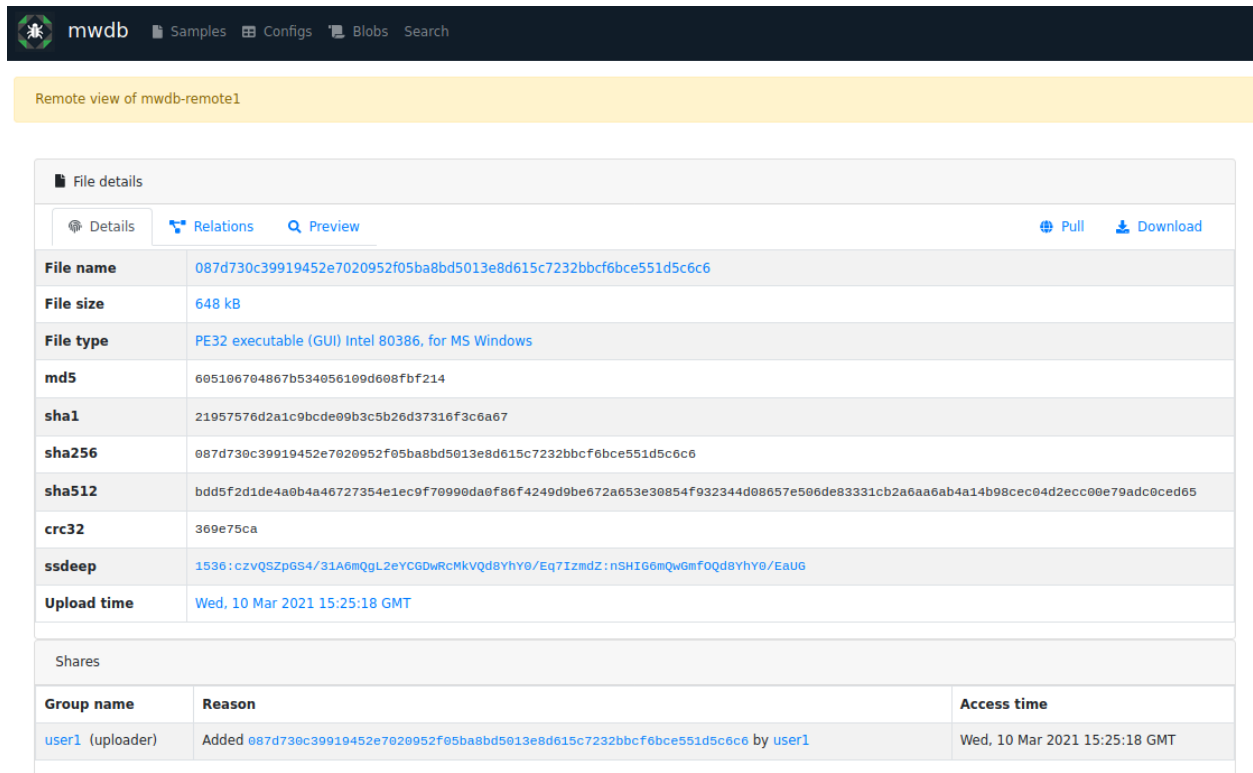


Remote instance view allows you to view remote samples, configs and blobs along with the detailed information about them.

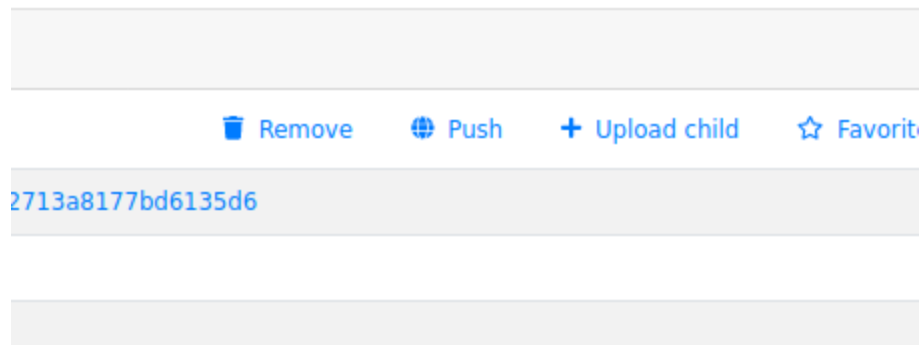
By going to the view of a specific object on a remote instance, we can read detailed information about this object and we're able to perform two possible actions on it:

- **Download** to download the object from the remote instance directly to your computer
- **Pull** to pull the object into the local MWDB database

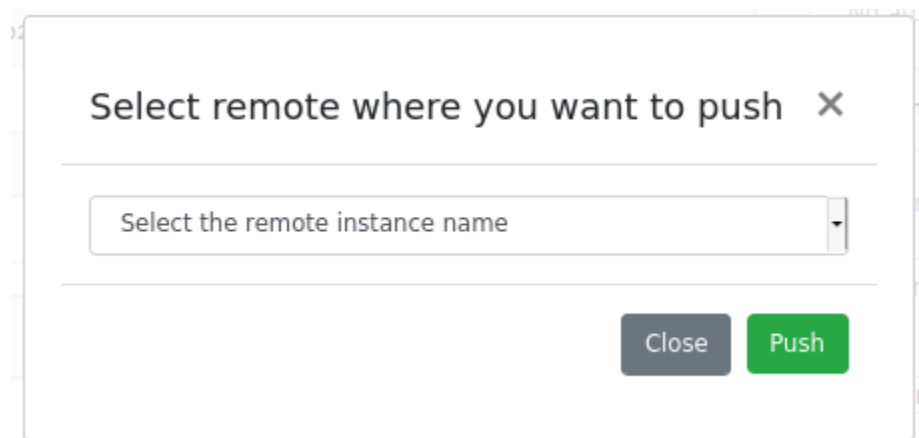
Although remote view is almost identical with the local one, it's limited to the read-only actions.



Similar to the pull object action in remote instance, we can also push an object to the remote instance by clicking the Push button in the local object view.



After clicking on that button, a window will appear where you can select the remote instance to which you want to push the object.



1.7.2 Setting up remote instance

In order to use the remote instance views, you need the API key associated with the authorized account on the remote instance. If you don't have the API key generated or don't know how to get it, check [How to use API keys?](#) for more information.

After setting up API key, open configuration file `mwdb.ini`. In this file you should add the URL of the remote instance along with the API key. Example below shows how to enable Remote instances to connect our repository with `mwdb.cert.pl` service.

```
...
remotes = mwdb.cert.pl

[remote:mwdb.cert.pl]
url = https://mwdb.cert.pl
api_key=ey...
```

The `remotes` variable is comma-separated list of MWDB remote instance names. Parameters for each instance can be set in separate `[remote:<remote name>]` sections.

```
remotes = mwdb.cert.pl, other-remote

[remote:mwdb.cert.pl]
```

(continues on next page)

(continued from previous page)

```
url = https://mwdb.cert.pl
api_key=ey...

[remote:other-remote]
url = http://other-remote.example.com
api_key=ey...
```

This way you can access multiple MWDB Core instances to synchronize objects with your own repository

Warning: All users on the local instance have the same identity and rights as the remote user associated with the API key that has been used to setup the remote instance.

1.7.3 Known issues and limitations

Remote instances are just a proof-of-concept, that enables us to implement more complex synchronization mechanism in the future. That's why it comes with few significant limitations:

- Automatic synchronization of tags, attributes, relationships is not implemented yet. Push/pull feature is limited to the object itself.

1.8 Karton integration guide

New in version 2.3.0.

Karton is distributed malware processing framework, that integrates all malware analysis services behind the mwdb.cert.pl. It's ready to use straight out of the box, so you can use it to easily introduce some background tasks into your MWDB Core setup.

If you are looking for quick integration example to play with MWDB+Karton setup, look at our [Karton-Playground project](#). Using playground you can easily experiment with Karton pipeline before making a production-grade setup.

See also our [Karton Gems series of blogposts](#) to learn more about Karton.

Warning: If you already use Karton plugin, check out *Migration from unofficial plugin setup*. Plugin is not fully compatible with built-in implementation.

1.8.1 How does it work?

Karton integration does the following things:

- Automatically spawns Karton tasks for all **new** files, configurations and blobs to analyze them;
- Allows user to track the Karton analysis status and manually trigger a reanalysis from UI;
- Aggregates all analyzed artifacts under the common analysis identifier;

MWDB Core itself just produces the initial tasks and provides a repository for data from analysis. For effective processing of these tasks and sending artifacts back to MWDB, you need to incorporate few essential Karton services into your pipeline:

- [karton-classifier](#) (entry-point) that labels the type of sample for further routing

- `karton-mwdb-reporter` (exit-point) that uploads all the artifacts back to MWDB under the common identifier

1.8.2 How to setup MWDB with Karton?

Before you start reading this chapter, setup the Karton. Instructions can be found in [Karton documentation](#).

The integration itself is easy to enable:

1. Enable MWDB integration by setting `enable_karton = 1` option in `mwdb.ini` or `MWDB_ENABLE_KARTON=1` environment variable.
2. Provide a `karton.ini` file in known location e.g. `/etc/karton/karton.ini` or `mwdb-core` application root.

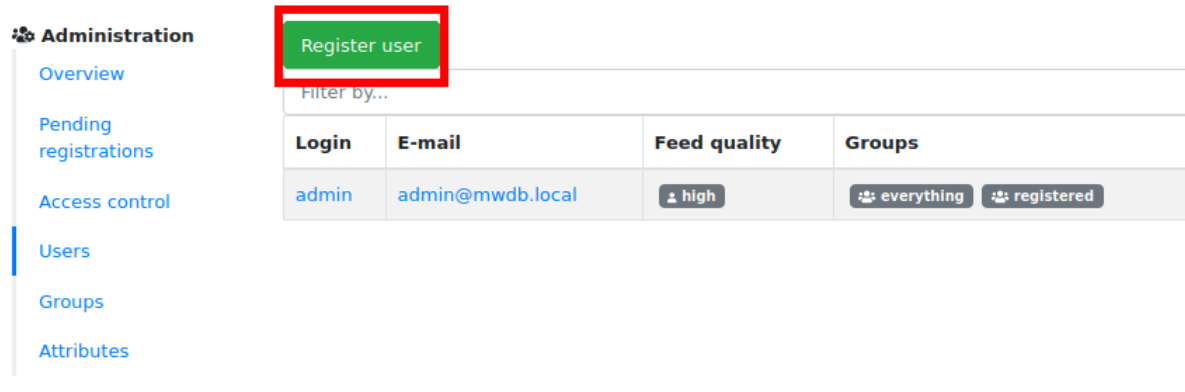
If you are using alternative location for Karton configuration file, use additional section in `mwdb.ini` to configure it:

```
[mwdb]
...
enable_karton = 1

[karton]
config_path = /opt/karton/karton.ini
```

Then you should setup the `karton-classifier` and `karton-mwdb-reporter` Karton services. The important thing here is that `karton-mwdb-reporter` requires API credentials to upload artifacts back to MWDB. Just for experiments you can use default `admin` account, but we encourage you to create the separate account for Karton like below.

First, create karton account in Settings tab:



The screenshot shows the 'Administration' tab in the MWDB interface. On the left, there is a sidebar with links: Overview, Pending registrations, Access control, Users, Groups, and Attributes. The 'Users' link is currently selected. In the main content area, there is a 'Register user' button highlighted with a red rectangle. Below this button is a table with columns: Login, E-mail, Feed quality, and Groups. The table contains one row for the 'admin' user. The 'Groups' column shows two buttons: 'everything' and 'registered'.

Login	E-mail	Feed quality	Groups
admin	admin@mwdb.local	high	everything registered

Administration[Overview](#)[Pending registrations](#)[Access control](#)[Users](#)[Groups](#)[Attributes](#)

Create new user

Login

Login must contain only letters, digits, '_' and '-' characters, max 32 characters allowed.

E-mail

Make sure that provided e-mail is active for administration purposes

Additional info

Feed quality

Send e-mail with set password link



Then go to the Access control to give `karton` all the required capabilities:

- `adding_tags`
- `adding_comments`
- `adding_parents`
- `adding_all_attributes` (if you don't mind to setup separate ACLs for each attribute Karton manages)
- `adding_files`
- `adding_configs`
- `adding_blobs`
- `unlimited_requests` (if you have rate limits enabled)
- `karton_assign`

Administration

Overview

Pending registrations

Access control

Users

Groups

Attributes

Access control

Use a form below to enable/disable capabilities for specific user or group.

karton

adding_tags

adding_comments

adding_parents

adding_all_attributes

adding

manage_users

Managing users and groups (system administration)

share_queried_objects

Query for all objects in system

access_all_objects

Has access to all new uploaded objects into system

sharing_objects

Can share objects with all groups in system

adding_tags

Can add tags

removing_tags

Can remove tags

adding_comments

Can add comments

removing_comments

Can remove (all) comments

adding_parents

Can specify parent of uploaded object

removing_parents

Can remove parent of object and inherited permissions from that relation

If you just use admin account, make sure that `karton_assign` is enabled for admin as well.

Finally go to the karton account details and click on `Manage API keys` action to create an API key for this account. Click `Issue new API key` to create the key.

Administration[Overview](#)[Pending registrations](#)[Access control](#)[Users](#)[Groups](#)[Attributes](#)**Account details: karton**

E-mail	karton@mwdb.local	Edit
Additional info	karton-mwdb-reporter account	Edit
Feed quality	high	Edit
Requested on	(never)	
Registered on	Thu, 27 May 2021 13:44:10 GMT	
Last login	(never)	
Last password set	(never)	
Groups	public registered	

Actions:[Search for uploads](#)[Show user groups](#)[Check user capabilities](#)[Manage API keys](#)[Change password](#)[Block user](#)[Remove user](#)

Include the following lines in `karton.ini` file used by `karton-mwdb-reporter`:

```
[mwdb]
api_url = http://<mwdb host here>/api/
api_key = ey...
```

After getting done with the steps above, run `mwdb-core` and upload a new file to check if Karton integration works correctly:

Relations

+ Add

No relations to display

Attributes

+ Add

Karton analysis

processing

54b63ef1-007d-42f1-be5f-f6c1e54f8b67 ▾

Last update: 2 minutes ago

Currently processed by:

- karton.mwdb-reporter

Started

got from karton.classifier

Search artifacts

+ reanalyze

1.8.3 Resubmitting analysis

Let's say that you have recently improved your pipeline. You probably want to resubmit some files for analysis to check if you get better results. Everything you need is + reanalyze button

If you don't see it, you probably need to turn on `karton_reanalyze` capability. Use Admin -> Access control panel to give appropriate permission for your account.

1.8.4 Migration from unofficial plugin setup

This section is dedicated for users who built MWDB+Karton setup using pre-2.3.0 plugin from <https://github.com/CERT-Polska/karton-playground/>

MWDB-Core 2.3.0 includes automatic migration spawned on `mwdb-core configure` which:

- automatically converts `karton` attributes to built-in analysis associations
- removes the `karton` attribute key definition

Before upgrade to 2.3.0:

- remove `mwdb-plugin-karton` from `plugins` directory.

After upgrade:

- enable `enable_karton = 1` setting in MWDB configuration as described in this chapter.
- enable `karton_assign` capability for account used by `karton-mwdb-reporter`.
- enable `karton_reanalyze` for all groups having `karton_manage` capability before.

Built-in integration emulates the original `karton` attribute behavior and still exposes and accepts the values provided that way.

1.9 OpenID Connect authentication (Single Sign-On)

New in version 2.6.0.

MWDB starting from version 2.6.0 allows you to log in using an external identity provider. The OpenID Connect protocol is used for this and it was integrated with the MWDB authentication system.

Warning: The current implementation of this authentication method is still under development and should be considered to be beta feature.

To be able to use this method of logging in, you must configure MWDB with special environment variable: `MWDB_ENABLE_OIDC=1` or via `mwdb.ini` configuration field: ``enable_oidc = 1``

The easiest way to play with this functionality is to set up the environment with a `docker-compose-oidc-dev.yml` file which sets the appropriate environment variable and sets up a test external [Keycloak server](#)

Automatic integration of MWDB with the test server is described in `dev/oidc/README.md`.

OpenID Connect integration gives us the opportunity to perform new actions:

- identity provider registration
- logging in with an external identity provider
- register new user using external identity

1.9.1 Setting up new OpenID Provider

If you want to add a new OpenID Provider, go to Settings and click on OpenID Connect tab.

The required information is:

- name
- client_id
- client_secret (if symmetric key is used to sign the JWT e.g. HS256)
- authorization_endpoint
- userinfo_endpoint
- token_endpoint
- jwks_endpoint (if asymmetric key is used to sign the JWT e.g. RS256)

1.9.2 OpenID client setup on OpenID Provider

During client registration, you may need information listed below (example for Keycloak):

- Standard Flow Enabled - MWDB-Core uses authorization code flow only
- Valid Redirect URLs: `https://<mwdb core url>/oauth/callback`
- Minimal required scope: `email, profile`

Note: Using the OpenID Connect protocol requires ``base_url`` to be set in configuration. This value is used for generating `redirect_uri` therefore it is essential to authenticate that way.

Current configuration is pretty minimal. In future versions it may be extended with roles for automatic group/permission assignment or Single Logout parameters.

As feature is still during development, keep in mind that these parameters may change a bit in future versions.

1.9.3 Bind MWDB account with OpenID Provider

After successfully registering the correct information about the appropriate provider, you can try to authenticate with it.

To do this go to the **Profile** and click on the **OpenID Connect** section. There you can link your local account with an external identity.

Now to test authentication with external provider, please log out and go to the **OAuth authentication** tab where you can log in with OpenID Connect protocol.

Note: It is also possible to create a new MWDB account by authorizing through external identity provider.

For users who already have MWDB accounts it is recommended to bind the account with external identity in **Profile** section.

1.10 Rich attributes guide

New in version 2.8.0.

Warning: This feature is still **in development** and it may significantly change in the future versions.

Attributes can be used to store complex information in JSON format. In addition, values can be queried just like configurations with the use of `attribute.<key>:<value>` syntax.

Rich attribute templates combine forces of Mustache and Markdown languages that give you the next level of flexibility in rendering attribute values in UI without the need of writing additional plugins. Templates allow you to render value objects as custom links, lists, tables or combinations of them. You can also combine your representation with other context values like sample name or hash.

Rich attributes enable us to produce similar reports like the Details section in VirusTotal (e.g. <https://www.virustotal.com/gui/file/32fae9922417d6405bf60144d819d5e02b44060fa8f07e5e71c824725f44307f/details>) based on data contained in attribute objects. All templates are rendered client-side.

1.10.1 Getting started

Go to **Settings -> Attributes** and choose the attribute key for editing.

Attribute details: apivectors

Label		Edit
Description		Edit
URL template		Edit
Rich template		Edit

Attribute features:

Hidden attribute
Hidden attributes have protected values. Attribute values are not visible for users without reading_all_attributes capability and explicit request for reading them. Also only exact search is allowed. User still must have permission to read key to use it in query. Enable

Actions:

[Edit attribute permissions](#)

Remove attribute

Then click on the Edit button next to the Rich template field to open the interactive template editor.

The view consists of two edit fields. The first one is **Template** that can be filled with Mustache&Markdown template used for rendering attribute value. The second one is **Example value** which contains an example attribute value appropriate for the attribute key in JSON format.

Preview field shows the rendered representation based on template and example value.

Rich templates combine forces of [Mustache](#) and [Markdown](#) languages to give you next level of flexibility in rendering attribute values in UI without the need of writing additional plugins. Templates allow you to render value objects as custom links, lists, tables or combinations of them. You can also combine your representation with other context values like sample name or hash.

Simple URL

Template <pre>[[{{value}}]](https://example.com/analysis/{{value}})</pre>	Example value <pre>"123456"</pre>
-------------------------------------------------------------------------------------	---------------------------------------------

Preview

My attribute	123456
--------------	--------

Store Cancel Delete

After filling fields with the appropriate template, use **Store** to save the template or **Cancel** to leave it unchanged. If you want to clear the template and show only raw representation of attribute value, click on **Clear**.

The interactive editor contains few sample templates that can be used for making its own definitions.

1.10.2 Mustache basics

Mustache is a basic logic-less template language. Mustache substitution is the first stage of template processing.

The most basic Mustache tag is `{{value}}` which will be substituted by Markdown-escaped attribute value.

If the value is represented by a JSON object, we can reference the appropriate subkey with the use of a dot as a key separator. For example: `{{value.id}}` will be substituted by value under `id` key in JSON object

Another type of tag is **section**. Sections render blocks of text one or more times and are mostly used for iterating over lists. The template inside a section is relative to the current object. If we want to reference a value in iterated list, we need to use a single dot `{{.}}`.

```
{{#value.functions}}
- {{name}}
**Offsets:**
```

(continues on next page)

(continued from previous page)

```

{{#offsets}}
- {{.}}
{{/offsets}}
{{/value.functions}}

```

Sections can be also used for conditional rendering: false values or empty lists won't be rendered.

Substituted values are Markdown-escaped by default. If somehow escaping interferes with your template, you can turn it off by using triple-mustache tag `{{{value}}}` or `{{&value}}` syntax.

Read more about supported Mustache syntax in Mustache.js project README (<https://github.com/janl/mustache.js/>)

1.10.3 Markdown basics

Markdown is a popular, lightweight markup language for creating formatted text. Markdown processor is used as a second stage of template processing when Markdown markups are converted to React objects.

MWDB Core uses client-side markedjs project (<https://github.com/markedjs/marked>) that was customized to emit React objects instead of HTML.

Supported Markdown markups are:

- **bold text**
- *italics*
- ~~strikethrough~~
- [links](http://example.com)
- > blockquotes
- `inline`
- lists

```

- element 1
- element 2
- element 3

```

- tables

```

|Name|Virtual Address|Virtual Size|Raw Size|MD5|
|----|-----|-----|-----|---|
{{#value.pe-sections}}
|{{name}}|`{{vaddr}}`|`{{vsize}}`|`{{psize}}`|`{{md5}}`|
{{/value.pe-sections}}

```

1.10.4 Known issues

- Current implementation is based on libraries that output HTML target instead of Markdown/React. Missing essential Markdown support and improper escaping (e.g. HTML entities) may happen. If you notice any problems like that: [create an issue](#).
- Some features for plain values are still missing for rich templates e.g. interactive searching. Work is in progress.

1.11 Prometheus metrics

New in version 2.12.0.

MWDB allows to enable Prometheus metrics to grab information about API usage by users.

Available metrics:

- `mwdb_api_requests` (method, endpoint, user, status_code) that tracks usage of specific endpoints by users and status codes.
- `mwdb_deprecated_usage` (feature, method, endpoint, user) that tracks usage of deprecated API endpoints

Useful PromQL queries (in most cases we use 5-minute window):

- `sum(increase(mwdb_api_requests{}[5m])/5) by (user)` - counts general API usage by MWDB users in requests per minute
- `sum(increase(mwdb_api_requests{status_code != "200"}[5m])) by (user, status_code)` - API error rate grouped by user and status_code
- `sum(increase(mwdb_api_requests{endpoint=~"api.filedownloadresource|api.requestsdownloadresource|api.downloadresource|api.filedownloadzipresource", method="GET"}[5m])/5) by (user)` - file download rate
- `sum(increase(mwdb_api_requests{endpoint=~"api.fileresource|api.fileitemresource", method="POST"}[5m])/5) by (user)` - file upload rate

1.11.1 Setup guide

1. First, you need to configure Redis server where metric counters are stored. Redis instance can be configured in MWDB via `redis_uri` option in `mwdb.ini` file or `MWDB_REDIS_URI` environment variable.
2. Then you need to enable metrics by setting `enable_prometheus_metrics=1` in configuration or `MWDB_ENABLE_PROMETHEUS_METRICS=1` in env vars
3. Log in as admin in MWDB and go to <http://<mwdb>/settings/users>. Create user account that will be used to access prometheus metrics.
4. Go to Check user capabilities or <http://<mwdb>/settings/user/<username>/capabilities> to set `access_prometheus_metrics` ACL for created account.
5. Then generate API key on <http://<mwdb>/settings/user/<username>/api-keys> that will be used to scrape metrics by Prometheus

After setup, metrics can be scraped using `http://<mwdb-api>/api/varz` endpoint with API key provided in Authorization: Bearer <api_key> HTTP header.

You can test it before setting up Prometheus using curl:

```
$ curl http://<mwdb-api>/api/varz -H 'Authorization: Bearer <api_key>'
```

Detailed instructions about Prometheus configuration can be found in [Prometheus docs](#).

INDICES AND TABLES

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