MISP

INTELLIGENCE SHARING

PyMISP with MISP 2.5

Build models, query context, automate deliberate changes

CIRCL / MISP Project

https://www.misp-project.org/

Reference snapshot: 2 November 2026

From a notebook to a deliberate integration

- Build and inspect models without contacting a server.
- Map Python operations to the MISP 2.5 REST API.
- Query the right scope and recognise incomplete results.
- Create, verify and remove a synthetic training event.

Client, models and API

Pin the client and identify the server

- Reviewed: MISP 2.5.50 and PyMISP 2.5.34.4.
- PyMISP requires Python 3.10 or later, below 4.0.
- PyPI and GitHub release checked on 2 November 2026.
- Server and client version numbers need not match.

Two complementary interfaces

- `PyMISP`: authenticated requests to the server.
- `MISPEvent`, `MISPAttribute`, `MISPObject`: local models.
- Model changes and JSON conversion do not save remotely.
- Use the public client methods to persist intended changes.

The Python methods still use REST routes

Python operation	REST route in this snapshot
`search_index(...)`	`POST events/index`
`search(...)` (attributes)	`POST attributes/restSearch`
`get_event(uuid)`	`GET events/view/{uuid}`
`add_event(event)`	`POST events/add`
`update_attribute(...)`	`POST attributes/edit/{uuid}`

Authentication and visibility remain server decisions

- The raw auth key goes in the Authorization header.
- An API-enabled role and visible data are still required.
- Advanced keys can restrict IPs, expiry and read-only use.
- Read-only keys can POST searches; they cannot create events.

Install the reviewed release in a virtual environment

```
python3 -m venv .venv
. .venv/bin/activate
python -m pip install -r examples/requirements.txt
python -m unittest examples.test_examples -v
python -m examples.models
```

Connect with trusted TLS and bounded request timeouts

```
from pymisp import PyMISP
from examples.common import Config

config = Config.from_env()
misp = PyMISP(config.url, config.key,
              ssl=config.verify, timeout=(5, 60),
              debug=False)
```

Check the response before using it

```
from pymisp import MISPEvent
from examples.common import expect_model

saved = expect_model(
    misp.get_event(event_uuid, pythonify=True),
    MISPEvent,
)
```

Read the right data

Choose the search scope first

- `search_index`: event summaries from the index.
- `search(controller="events")`: matching events.
- `search(controller="attributes")`: matching attribute rows.
- `get_event(uuid)`: one known event and its context.

One typed attribute page

```
rows = misp.search(
    controller="attributes", return_format="json",
    type_attribute="domain", value="%.example",
    include_event_uuid=True, include_event_tags=True,
    order="Attribute.id ASC", limit=10, page=1,
    pythonify=True,
)
```

Raw JSON, normalised JSON and typed models differ

Interface	Attribute-search result
Raw HTTP JSON	`response.Attribute` array
PyMISP JSON	Dictionary containing `Attribute`
PyMISP with `pythonify=True`	List of `MISPAttribute` objects

Do not reuse the raw-envelope parser on typed results.

Translate parameter names and states deliberately

REST JSON	PyMISP keyword
`type`	`type_attribute`
`returnFormat`	`return_format`
`includeEventUuid`	`include_event_uuid`
`includeEventTags`	`include_event_tags`
`enforceWarninglist`	`enforce_warninglist`

A matching event can contain nonmatching attributes

- Event search selects events using the supplied filters.
- A full event result can include broader incident context.
- Attribute search selects the rows required by a consumer.
- Retain event UUID and inherited-tag context when needed.

Traverse bounded pages without claiming false completion

- Start at page 1 and use a stable order.
- A role cap can make every nonempty page short.
- For this ordinary JSON query, continue to an empty page.
- Fail on repeated UUIDs or the maximum-page bound.

Keep polling clocks and checkpoints explicit

```
from time import time
upper = int(time())
lower = previous_checkpoint - overlap_seconds
# Use the same bounds for every page:
rows = misp.search(controller="attributes",
                   timestamp=[lower, upper],
                   limit=100, page=1, pythonify=True)
```

Export format is also a selection policy

- Ordinary JSON can include unpublished contextual attributes.
- Detection formats can default to published and to_ids records.
- Set publication, to_ids and warninglist policy explicitly.
- A text export is a string, not a list of typed attributes.

Build, save and verify

Build a private synthetic event offline

```
from pymisp import MISPEvent

event = MISPEvent()
event.info = "PyMISP 2.5 training / synthetic"
event.distribution = 0
event.threat_level_id = 4
event.analysis = 0
event.published = False
```

Add a contextual attribute to the local model

```
attribute = event.add_attribute(
    "domain", "observer.example", to_ids=False,
    comment="Synthetic observation only.",
)
attribute.comment = "Reviewed synthetic observation."
print(attribute.uuid, attribute.type)
```

Group related observations in a template-backed object

```
from pymisp import MISPObject
obj = MISPObject("domain-ip", strict=True,
                 standalone=False)
obj.add_attribute("domain", value="host.example",
                  to_ids=False)
obj.add_attribute("ip", value="192.0.2.10",
                  to_ids=False)
event.add_object(obj)
```

Inspect JSON and round-trip the local model

```
encoded = event.to_json(indent=2)
restored = MISPEvent()
restored.load(encoded)
assert restored.uuid == event.uuid
print(len(restored.attributes), len(restored.objects))
```

Persist once, then use the server response

```
from examples.common import expect_model

created = expect_model(
    misp.add_event(event, pythonify=True), MISPEvent)
verified = expect_model(
    misp.get_event(created.uuid, pythonify=True), MISPEvent
    )
```

Update a known attribute and verify its identity

```
from pymisp import MISPAttribute
attribute = verified.attributes[0]
attribute.comment = "Reviewed in the workshop."
updated = expect_model(
    misp.update_attribute(attribute, pythonify=True),
    MISPAttribute,
)
assert updated.uuid == attribute.uuid
```

Reference an attribute from an object

```
obj.add_reference(attribute.uuid, "related-to",
                  "Synthetic teaching relation.")
# Add this before saving the event in the workshop.
```

- Keep the referenced UUID and relationship meaning explicit.
- A local reference is not a correlation or Galaxy attachment.

Tag at the intended scope

```
from examples.common import expect_success

expect_success(misp.tag(created.uuid, "tlp:clear"))
expect_success(misp.tag(attribute.uuid, "tlp:clear",
                        local=True))
```

Sightings and proposals have distinct meanings

- `MISPSighting` plus `add_sighting` records an observation.
- Target the intended attribute UUID, not a broad shared value.
- Proposal methods create suggested changes for review.
- A returned proposal does not confirm an original was edited.

Run the disposable write workshop deliberately

```
python -m examples.workshop --run
```

- Create a private event, attributes, object and reference.
- Edit, tag, add a sighting and refetch the result.
- Query bounded pages restricted to the new event UUID.
- Delete only that fixture and verify it is absent.

Handle failures and diagnostics without losing meaning

- Separate transport failure, API error and valid empty result.
- Check saved state before retrying an uncertain write.
- Keep keys, private payloads and verbose logs out of notebooks.
- Record versions, query policy and validation limits.

Use the notebook as an experiment

- Default: create, inspect and round-trip local models.
- Optional live reads: compare raw and typed attribute UUIDs.
- Separate opt-in: execute the disposable write workshop.
- Clear outputs before sharing; retain source provenance.

Make the integration reproducible

- Pin tested versions and preserve UUID-based identity.
- Separate local construction from remote persistence.
- Keep scope, selection policy and completion explicit.
- Use the README, executable examples and REST API course.