MISP

INTELLIGENCE SHARING

Integrating MISP

APIs, modules, workflows and operational feedback

CIRCL / Team MISP Project

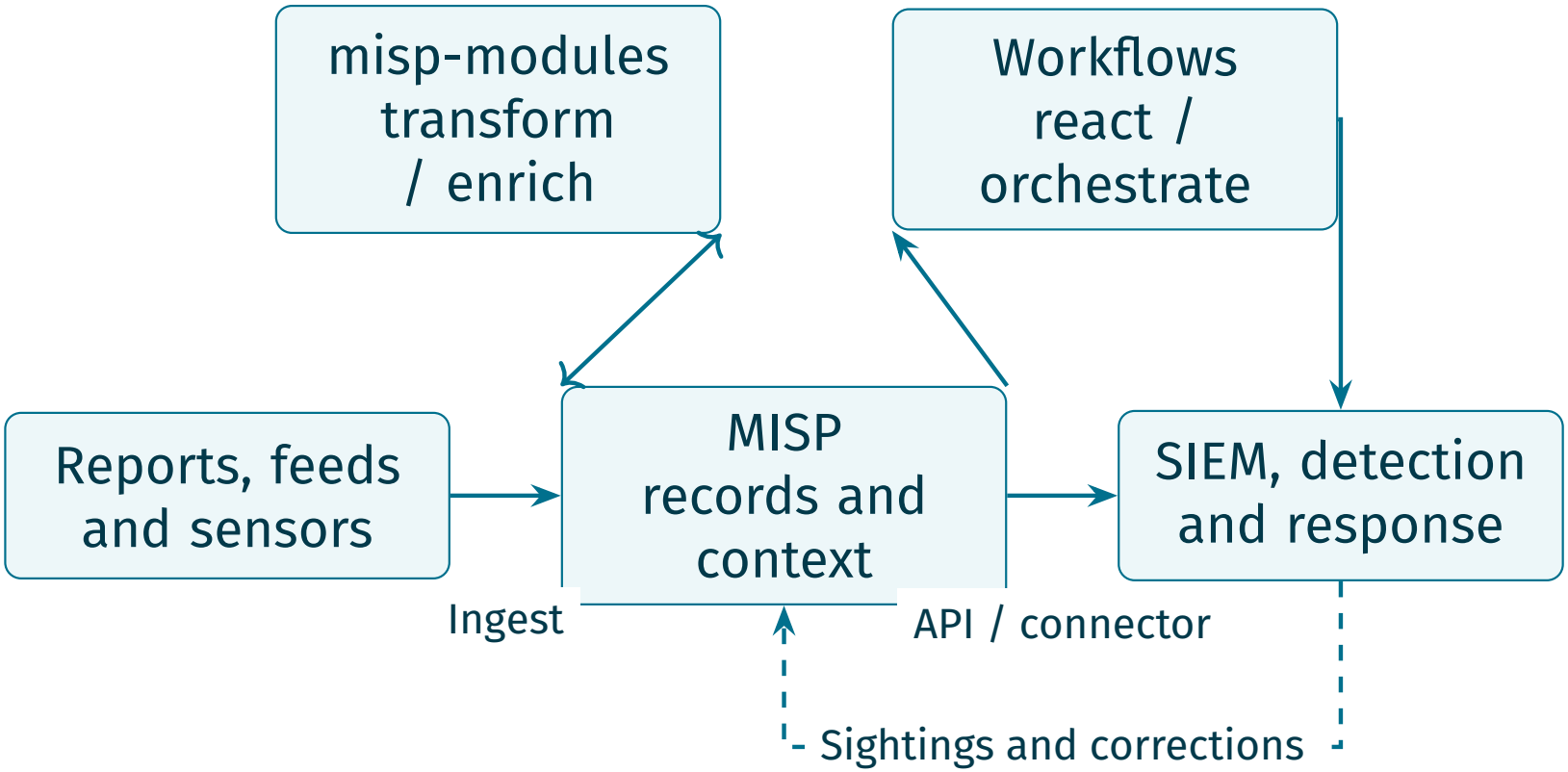
https://www.misp-project.org/

Reference: 4 November 2026

Follow a finding all the way to its consumer

- Bring a report, sensor finding or feed into MISP.
- Add structure and context through modules and analysts.
- Select an audience and an automation path.
- Deliver useful data, then return feedback and corrections.

An integration is several connected data flows



Give every arrow an owner, a payload and a failure policy.

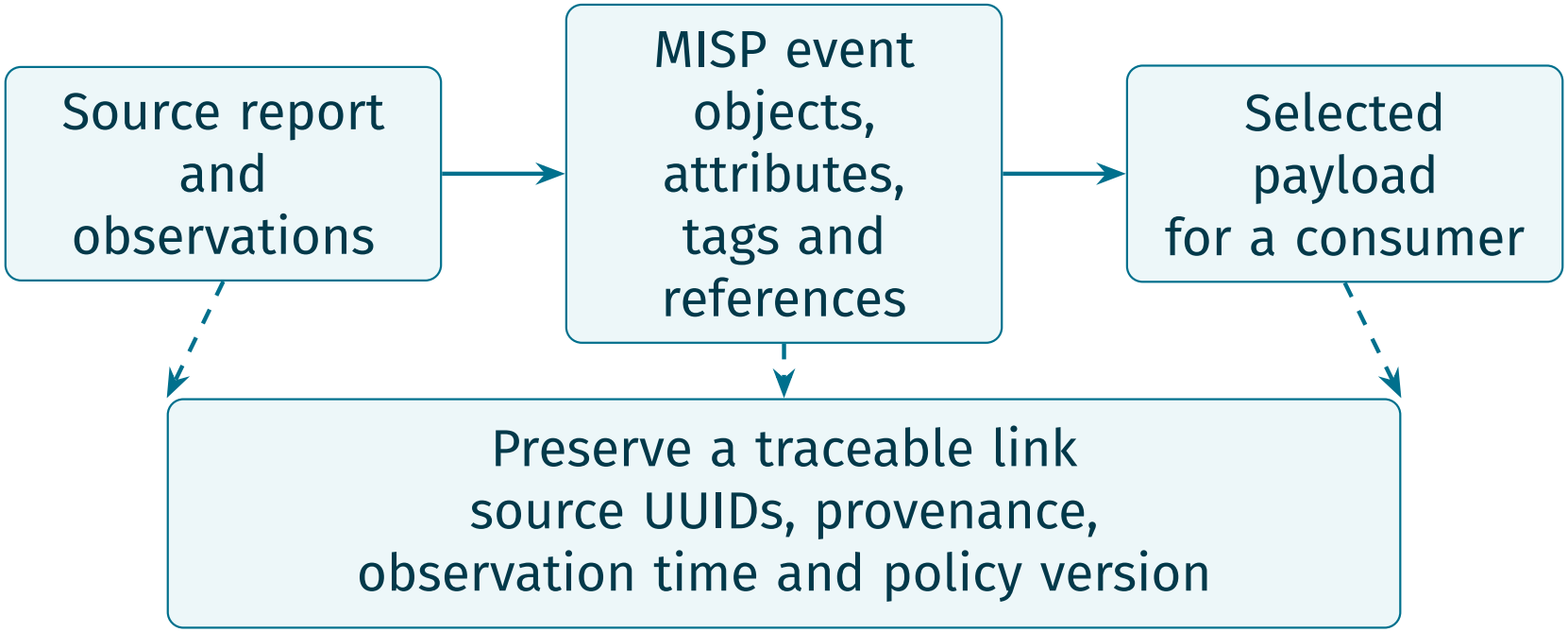
Choose the interface for the job

Need	Start with
Peer exchange	Synchronisation with reviewed organisations and rules
Dataset intake	A configured feed; evaluate provenance and overlap
Search or edit	REST API, directly or through PyMISP
Enrich or convert	A compatible misp-modules capability
React to an action	A workflow trigger, conditions and actions
Update a tool	Its connector or API, with a defined delivery process

Write the integration contract first

- **Meaning:** which question or action does this data support?
- **Identity:** which UUIDs and source references survive conversion?
- **Audience:** which organisations and external services may receive it?
- **Lifecycle:** how do updates, expiration and removals arrive?
- **Operations:** who handles retries, limits and failures?

Ingest deliberately and preserve context



A flat list can serve a tool; keep the explanation traceable.

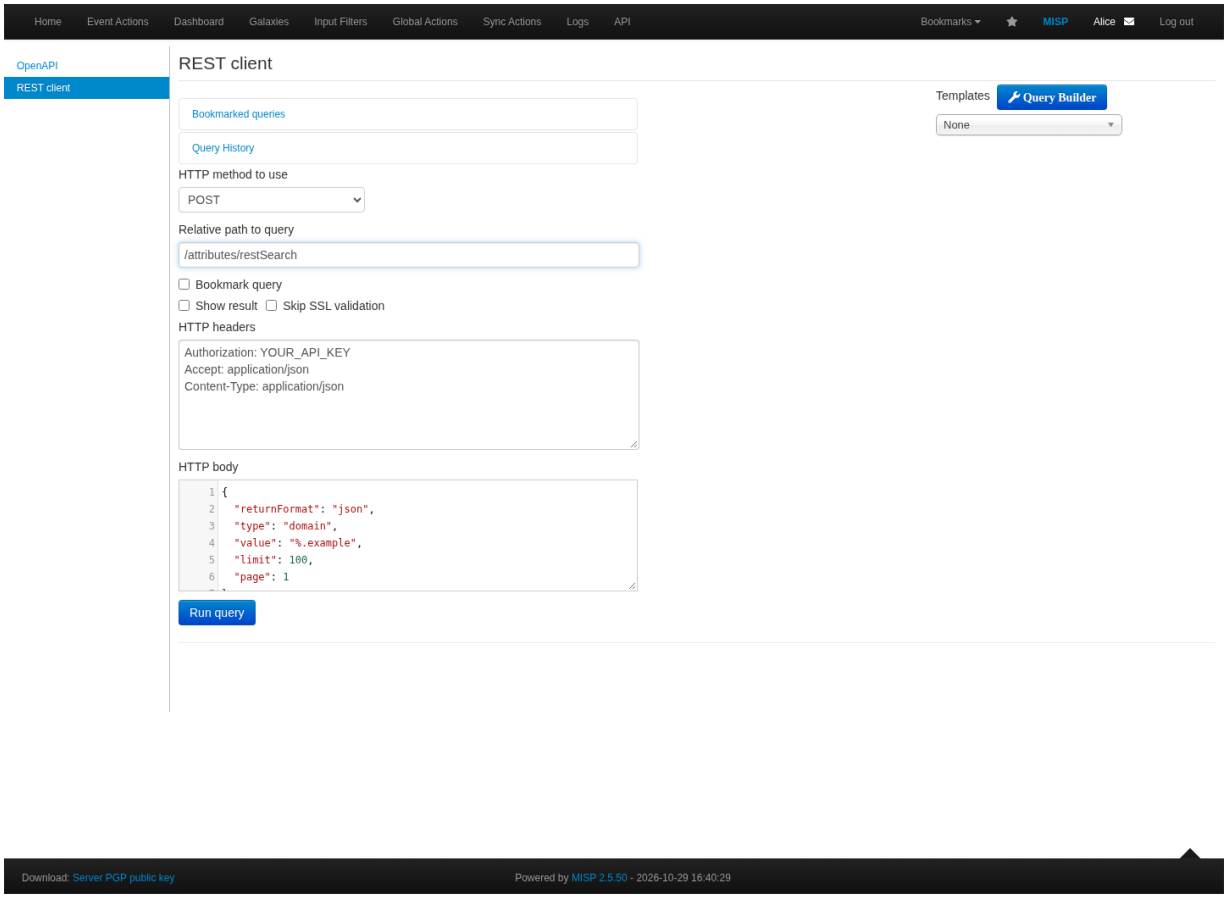
Feeds and sync solve different ingestion problems

- Compare feed coverage, freshness and overlap before importing.
- Review sync identities, distribution and push/pull filters.
- Keep upstream UUIDs when maintaining the same native records.
- Give synthetic or unreviewed imports a controlled staging path.

API authentication is also a data-access decision

- Use a dedicated integration account and an appropriately scoped key.
- Send the raw key in the `Authorization` header.
- Keep TLS verification, timeouts and secrets outside source files.
- A read-only key can POST a search; it cannot perform writes.

Prepare a request in MISP's REST client



Choose a method, path and JSON body.

Inspect the deployed `/api/openapi` and `/api/rest`.

This capture contains a placeholder key.

A detection-oriented attribute query: one page

```
{
  "returnFormat": "json",
  "type": "domain",
  "value": "%.example",
  "published": true,
  "to_ids": true,
  "enforceWarninglist": true,
  "includeEventUuid": true,
  "includeEventTags": true,
  "limit": 100,
  "page": 1
}
```

POST this body to `/attributes/restSearch`.

The reserved `.example` filter keeps the exercise synthetic.

Check the response contract, not only HTTP status

- Attribute JSON search returns `response.Attribute`.
- Event JSON search has a different, event-oriented envelope.
- Validate the content type, errors and required UUID fields.
- Distinguish empty results, denied access and incomplete retrieval.

Pagination and checkpoints need a completion rule

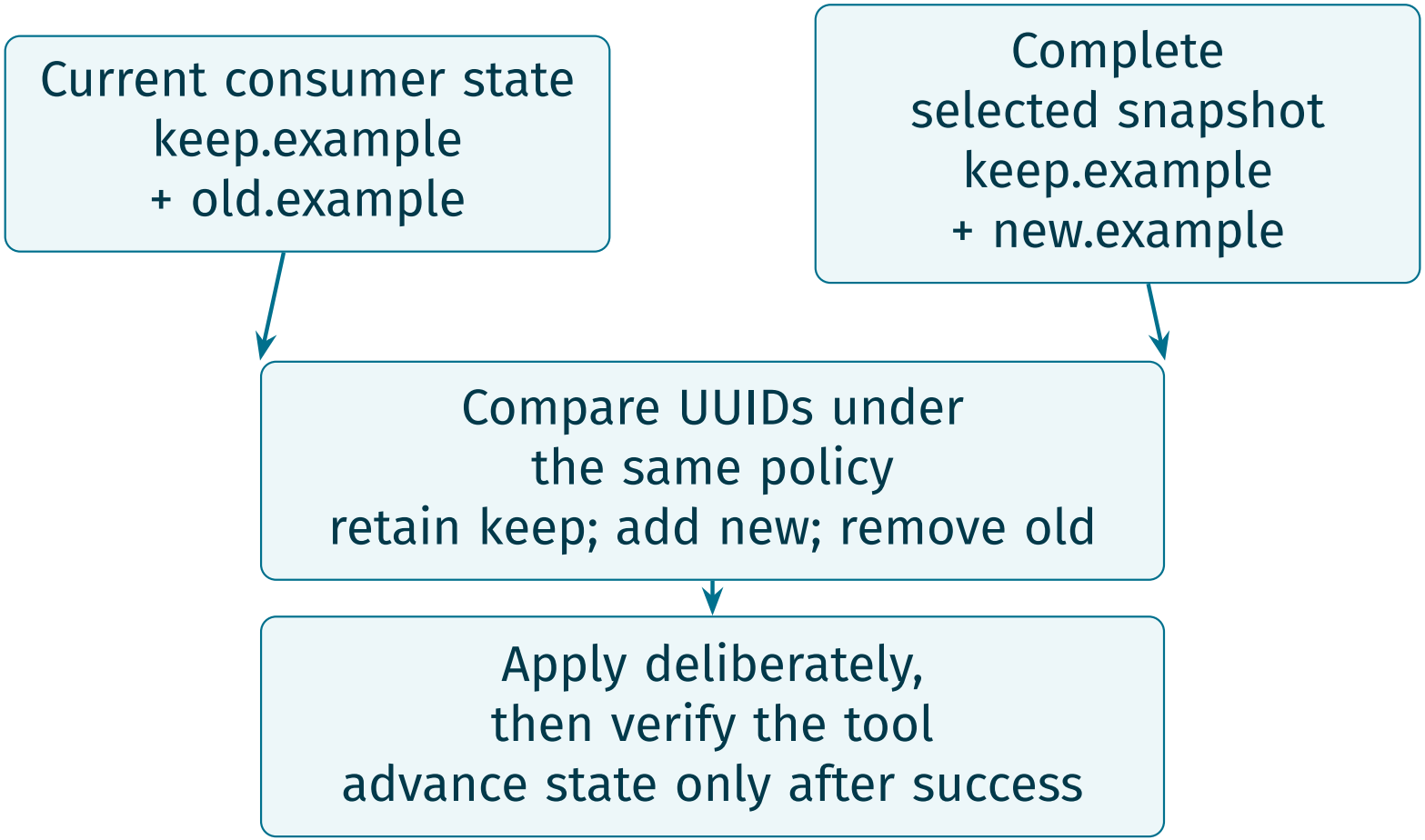
- Bound page size, runtime and the number of requests.
- Use stable ordering and deduplicate by UUID.
- Role caps and post-query filters can produce sparse pages.
- Advance a checkpoint only after the whole run succeeds.

PyMISP makes API objects easier to work with

```
rows = misp.search(
    controller="attributes", return_format="json",
    type_attribute="domain", value="%.example",
    published=True, to_ids=True,
    include_event_uuid=True,
    limit=100, page=1, pythonify=True,
)
```

- Reuse the authenticated client setup from the PyMISP/API course.
- Check for error dictionaries; this is still only one page.

Plan additions, changes and removals



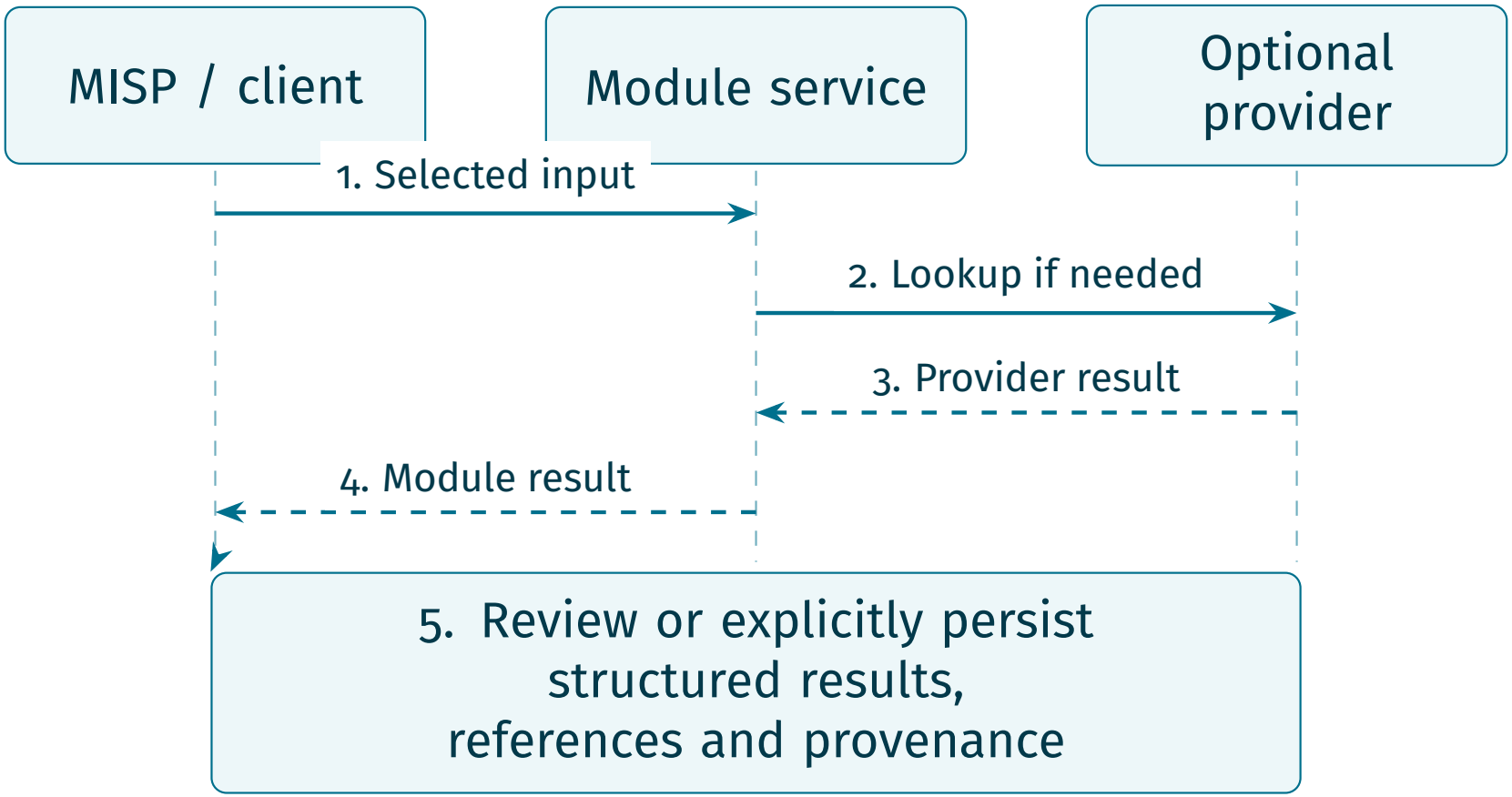
An incomplete run must not become an empty desired state.

misp-modules adds capabilities around MISP

Family	Typical role
Enrichment	Hover or expansion: look up or derive context
Import	Convert an external payload to MISP data
Export	Produce a consumer-oriented artefact
Action	Perform a configured workflow operation

Discover what the running service actually loaded.

A module query is not a database write



Review or explicitly persist the returned data in MISP.

Connect the service from the right network context

- Check `/healthcheck`, `/version` and `/modules`.
- Inspect `/openapi.json` on the deployed module service.
- Configure the MISP module family, URL, enablement and timeouts.
- Test reachability from the MISP worker as well as the web process.

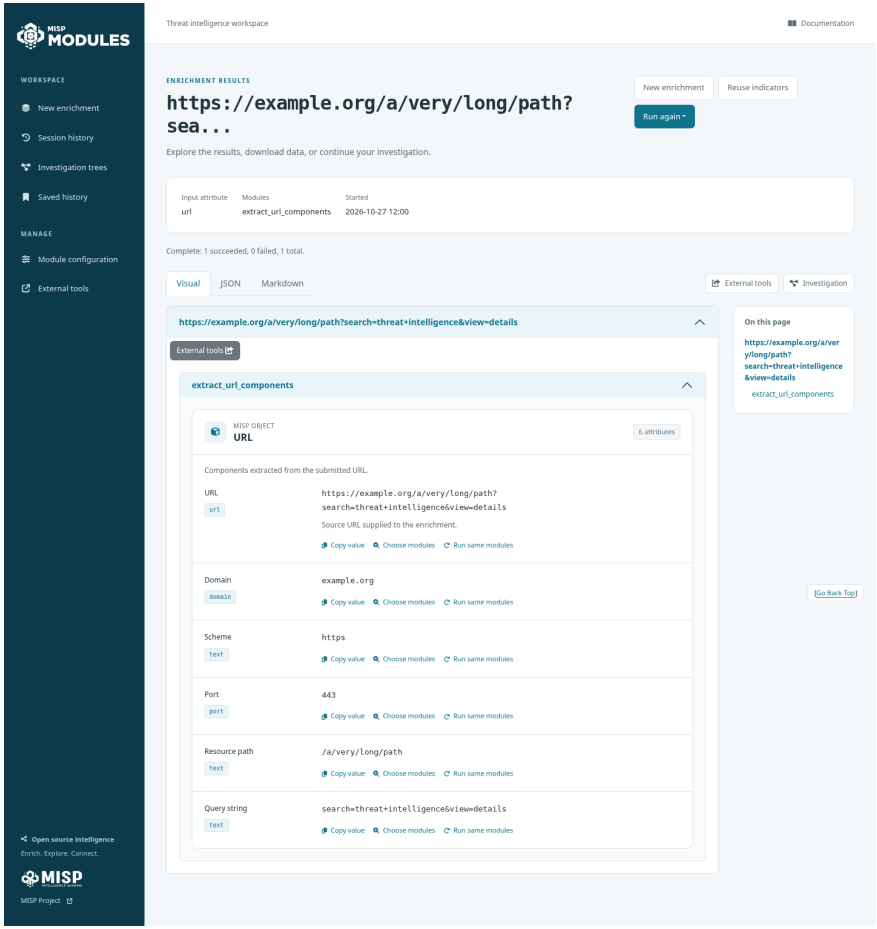
A local parsing example without a provider lookup

```
{
  "module": "extract_url_components",
  "attribute": {
    "type": "url",
    "value": "https://portal.example.org/a?item=1",
    "uuid": "13d8d18b-784a-42a0-9a58-a9cddbfda561"
  }
}
```

POST to the lab module service's `/query`.

Inspect `results.Object` and references to the input UUID.

The modules website is a separate client



- Inspect structured objects and source values.
- Reuse results in a selected analysis path.
- Distinguish the website, module API and MISP UI.

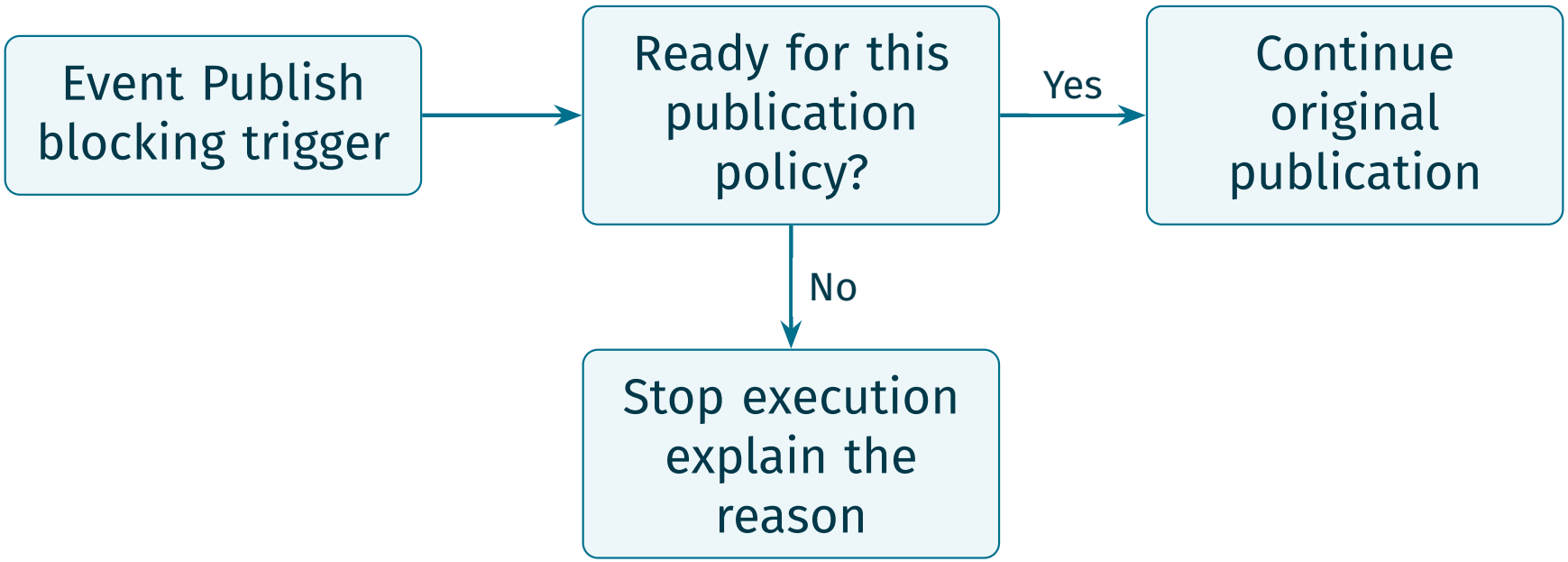
Choose how enrichment becomes part of the record

- An analyst can review proposed enrichment before saving.
- **Attach enrichment** adds results to workflow roaming data.
- **Enrich Event** invokes event enrichment and can persist changes.
- Track source, observation time, uncertainty and resulting writes.

Treat an external lookup as disclosure

- Review the provider, submitted fields and permitted purpose.
- Avoid exposing victim information through values or samples.
- Apply handling policy before the request leaves the environment.
- Keep provider keys and debug logs out of shared examples.

Workflows connect triggers, conditions and actions



Illustrative publication gate; implement and test the policy.

Blocking and background work serve different purposes

- Use a blocking path for checks that must precede the action.
- Move slow callbacks to an appropriate non-blocking path.
- A non-blocking failure cannot undo the originating operation.
- Review workers, timeouts and every action’s error behaviour.

Workflow execution has its own trust boundary

- UI and API actions can both trigger workflows.
- Workflow processing uses a privileged system context.
- Select and minimise data before sending a webhook or module request.
- Prevent loops when workflow writes trigger more workflow activity.

Use the right trigger for the decision

- `event-publish`: check a publication prerequisite.
- `enrichment-before-query`: guard MISP's enrichment path.
- After-save triggers: process the relevant saved data.
- Inspect the payload available to each installed trigger.

A webhook needs a deliberate payload contract

- Agree the destination, authentication and expected body.
- An empty webhook payload can send the full roaming data.
- Review `Security.workflow_enable_arbitrary_urls`.
- Send a minimal reference when the receiver can fetch authorised data.

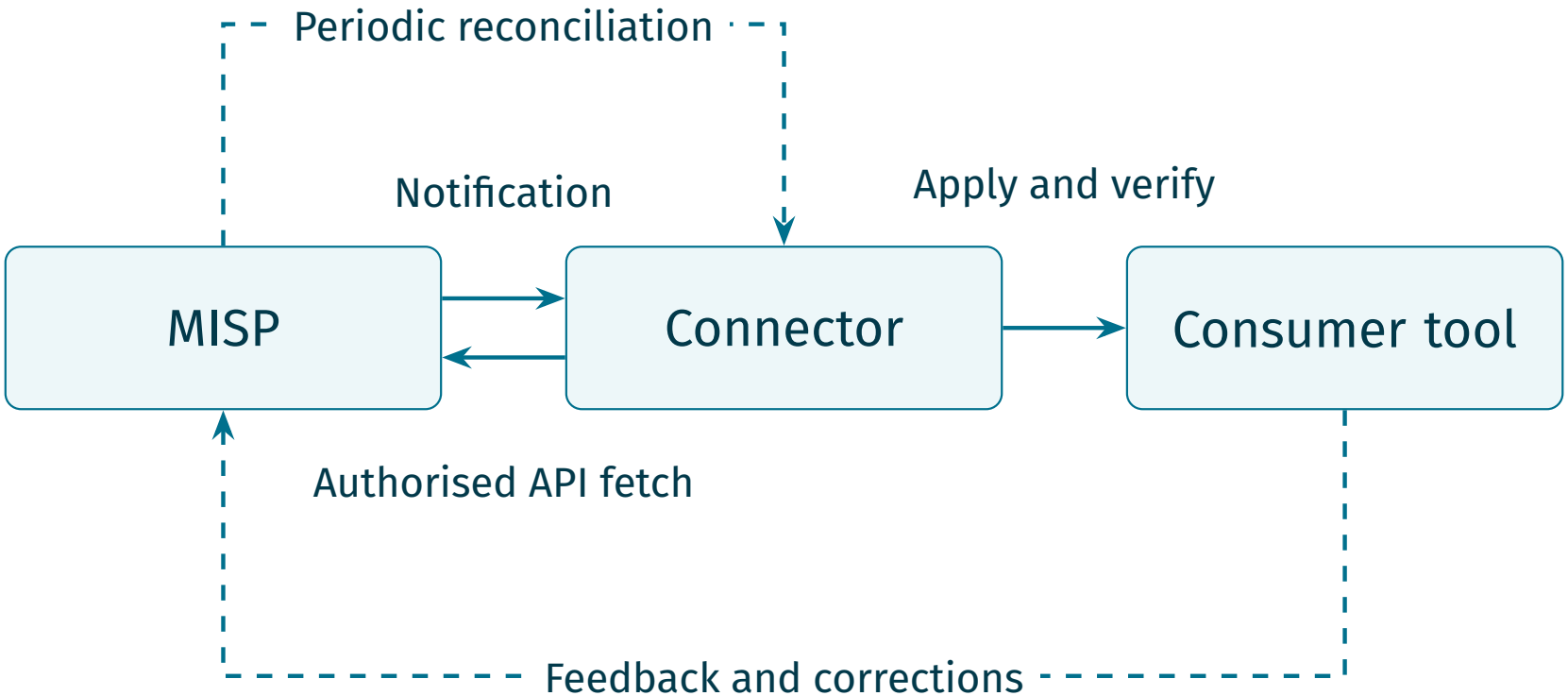
Module actions can produce an artefact, not activation

- Example: `osquery_pack_publish` can plan a query pack.
- Start with `dry_run=true`; inspect the planned content.
- Real publication writes on the module server.
- Distribution, configuration reload and endpoint checks are separate.

Three complementary SIEM integration patterns

Pattern	Useful for	Check
Scheduled API pull	A selected lookup dataset	Freshness, completion and removals
Event notification	Prompt processing after a change	Missed messages and authorised retrieval
MISP-to-SIEM lookup	An analyst asking whether a value was seen	Query scope, permissions and response meaning

Use notifications for speed and reconciliation for repair



Prove what the receiver loaded, not only what MISP sent.

Select detection data for the consuming tool

- Use published state, types, `to_ids`, context and freshness.
- Review warninglist matches and the false-positive impact.
- Inspect Suricata rules, Zeek intel or osquery packs before delivery.
- Apply a stricter policy for blocking than for investigation.

Test format conversion with a representative record

- Include objects, references, tags, times and source UUIDs.
- Inspect native MISP JSON and the consumer's chosen format.
- Check STIX/TAXII mapping, version and extensions explicitly.
- Record what is preserved, transformed or omitted.

Return feedback against the right attribute

```
{
  "type": 0,
  "source": "isolated-integration-lab",
  "date_sighting": "1793793600"
}
```

POST to `/sightings/add/{attribute-uuid}` in the lab.

- Types: 0 observation, 1 false positive, 2 expiration.
- Record the source and actual observation time.

Diagnose the failing boundary

Symptom	Next check
API query has no useful rows	Key, role, distribution, query and completion
Module absent or unavailable	Runtime discovery, dependencies, family settings and network
Workflow did not run	Enabled trigger, workflow, workers and diagnostics
Callback succeeded, tool unchanged	Receiver contract, load/reload and activation evidence
Old indicator is still active	Reconciliation, removal policy and last successful refresh

Offline lab: inspect a consumer update plan

1. Run the supplied snapshot planner on the two fictional manifests.
2. Explain the one addition, one removal and one retained item.
3. Change a retained UUID’s value and observe an update.
4. Set `complete=false`; confirm the planner refuses the input.

Optional lab: connect the three integration surfaces

1. API: preview the supplied first-page query in an isolated MISP.
2. Modules: parse the fictional URL and inspect structured output.
3. Workflow: build a lab publication gate and test both branches.
4. Consumer: confirm a correction or removal reaches the lab output.

Roll out with evidence at every boundary

- Start with previews and a small, representative dataset.
- Test errors, retries, access exclusions and schema changes.
- Verify consumer activation, corrections and removals.
- Monitor freshness and failures; assign an operational owner.

Continue with the specialist courses

- MISP Book: automation and API
- misp-modules: capabilities and service documentation
- MISP Book: workflows
- Training sources: `a.7-rest-API`, `a.2-pymisp`, `3.1-misp-modules`, `a.12-misp-workflows`, `a.13-misp-stix`

https://www.misp-project.org/ — https://github.com/MISP