Contributing MISP modules with an AI agent

From a precise brief to a tested pull request

CIRCL / MISP Project

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

Reference snapshot: 29 October 2026



What a successful contribution contains

- A module that fits existing discovery and execution.
- Correct MISP inputs, outputs, configuration and provenance.
- Tests that demonstrate behavior, not just Python syntax.
- Useful metadata, documentation and a focused pull request.

The agent assists development

It is not a new module family or a reason to add a model service to the runtime implementation.

What a successful contribution contains
A module that fits existing discovery and execution.
Correct MISP inputs, outputs, configuration and provenance.
Tests that demonstrate behavior, not just Python syntax.
Useful metadata, documentation and a focused pull request.
The agent assists development
It is not a new module family or a reason to add a model service to the runtime implementation.

This companion assumes the MISP modules introduction. It is provider-neutral and works with the coding assistant the contributor already uses. The reference exercise requires no model credentials or agent SDK.

Keep ownership and evidence explicit

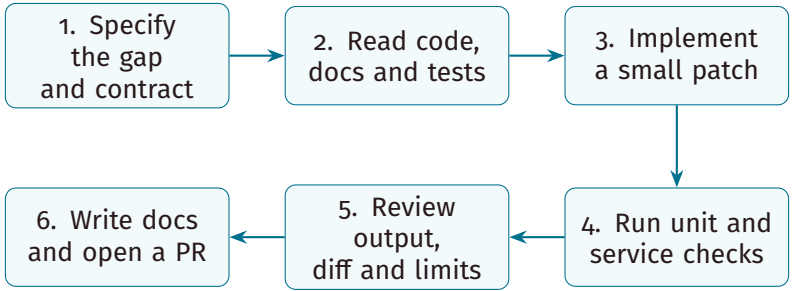
Contributor	Coding agent	Maintainer
Defines the need and boundaries.	Inspects code and prepares a plan and patch.	Assesses fit and maintenance cost.
Checks contracts and reviews results.	Runs permitted checks and reports actual outcomes.	Reviews the patch and evidence.
Owns the submitted contribution.	Identifies assumptions and limits.	Decides whether to merge.

A confident explanation from an agent is not execution evidence.

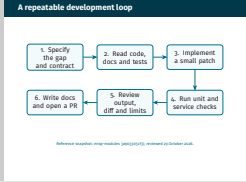
Keep ownership and evidence explicit
Contributor | Coding agent | Maintainer
Defines the need and boundaries. | Inspects code and prepares a plan and patch. | Assesses fit and maintenance cost.
Checks contracts and reviews results. | Runs permitted checks and reports actual outcomes. | Reviews the patch and evidence.
Owns the submitted contribution. | Identifies assumptions and limits. | Decides whether to merge.
A confident explanation from an agent is not execution evidence.

Agree on allowed network access and writes in the task brief. Do not claim a provider was tested when only mocked responses ran. Accountability remains with the contributor and maintainers.

A repeatable development loop



Reference snapshot: misp-modules 3a903305cf37, reviewed 29 October 2026.



Repeat the relevant loop when a check reveals a defect. Read the current target checkout: a training snapshot or agent memory is not the authority for a later API. The README pins the reviewed evidence.

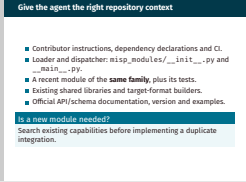
Read the existing contract

Give the agent the right repository context

- Contributor instructions, dependency declarations and CI.
- Loader and dispatcher: `misp_modules/__init__.py` and `__main__.py`.
- A recent module of the **same family**, plus its tests.
- Existing shared libraries and target-format builders.
- Official API/schema documentation, version and examples.

Is a new module needed?

Search existing capabilities before implementing a duplicate integration.



The agent should report the checkout revision and reference files. A marketing description is not a wire contract. If official details are missing, record the ambiguity and continue independent work rather than inventing behavior.

Select a family before writing code

Directory	Metadata type	Typical contract
`expansion`	expansion / hover	Observable or attribute to enrichment results.
`import_mod`	import	File, paste or report to parsed MISP data.
`export_mod`	export	Supplied MISP data to an encoded artifact.
`action_mod`	action	Workflow data and parameters to an operation/result.

Match the specific module's schema; this is not one universal request format.

Select a family before writing code

Directory | Metadata type | Typical contract
expansion | expansion / hover | Observable or attribute to enrichment results.
import_mod | import | File, paste or report to parsed MISP data.
export_mod | export | Supplied MISP data to an encoded artifact.
action_mod | action | Workflow data and parameters to an operation/result.
Match the specific module's schema; this is not one universal request format.

The filename supplies the API identifier; the display name is separate. Hover is an expansion capability, not a fifth directory. Discovery scans valid Python files without a manual all-list registration. Do not introduce a replacement loader.

Preserve the module interface

- `introspection()`: supported input/output or family-specific setup.
- `version()`: accurate capabilities, configuration and metadata.
- `dict_handler(request=...)`: dictionary execution path.
- `handler(q=...)`: supported legacy JSON execution path.
- Return fresh result/error objects; keep request state local.

Read the fallback behavior

The reviewed dispatcher catches `AttributeError` for legacy fallback. Do not use that exception as input validation.

Preserve the module interface
- introspection(): supported input/output or family-specific setup.
- version(): accurate capabilities, configuration and metadata.
- dict_handler(request=...): dictionary execution path.
- handler(q=...): supported legacy JSON execution path.
- Return fresh result/error objects; keep request state local.

Read the fallback behavior
The reviewed dispatcher catches AttributeError for legacy fallback. Do not use that exception as input validation.

The reference implements both paths and compares semantic results. Avoid mutable shared error dictionaries or metadata modified by a request. Importing a module should not contact a provider or change a target system.

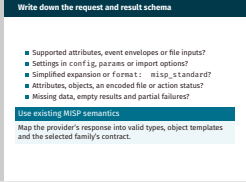
Write down the request and result schema

- Supported attributes, event envelopes or file inputs?
- Settings in `config`, `params` or import options?
- Simplified expansion or `format: misp_standard`?
- Attributes, objects, an encoded file or action status?
- Missing data, empty results and partial failures?

Use existing MISP semantics

Map the provider's response into valid types, object templates and the selected family's contract.

Request and response mappings belong in the plan. Export modules often return base64 data; actions commonly use params and data, but these summaries do not replace reading actual contracts. JSON-serializable output can still have the wrong MISP structure.

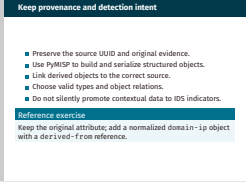


Keep provenance and detection intent

- Preserve the source UUID and original evidence.
- Use PyMISP to build and serialize structured objects.
- Link derived objects to the correct source.
- Choose valid types and object relations.
- Do not silently promote contextual data to IDS indicators.

Reference exercise

Keep the original attribute; add a normalized `domain-ip` object with a `derived-from` reference.



Tests inspect real PyMISP output: source UUID, value/comment, references and detection flags. The derived attribute is non-IDS in this exercise. Validate flags before PyMISP conversion: a nonempty string such as false can otherwise become true. The exercise accepts only boolean flags when supplied. It is the full normalized name, not a guessed registrable domain.

Dependencies and configuration are not incidental

- Use the service's actual Python environment and dependency workflow.
- Add necessary dependencies without unrelated lockfile upgrades.
- Metadata describing a dependency does not install it.
- Distinguish request config, service defaults and module-specific environment variables.
- Keep credentials and private incidents out of prompts and fixtures.

Dependencies and configuration are not incidental
- Use the service's actual Python environment and dependency workflow.
- Add necessary dependencies without unrelated lockfile upgrades.
- Metadata describing a dependency does not install it.
- Distinguish request config, service defaults and module-specific environment variables.
- Keep credentials and private incidents out of prompts and fixtures.

Do not bypass artifact or TLS verification to make setup pass. Missing optional imports can skip a module; syntax errors can stop startup. Neither is the same as MISP enablement or provider authorization. Read family-specific configuration and action semantics before implementing.

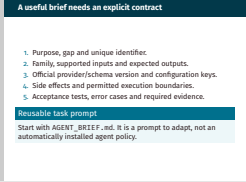
Direct the agent and test behavior

A useful brief needs an explicit contract

1. Purpose, gap and unique identifier.
2. Family, supported inputs and expected outputs.
3. Official provider/schema version and configuration keys.
4. Side effects and permitted execution boundaries.
5. Acceptance tests, error cases and required evidence.

Reusable task prompt

Start with `AGENT_BRIEF.md`. It is a prompt to adapt, not an automatically installed agent policy.



Ask for a short contract and implementation plan first. Continue useful reversible work without repeatedly asking preferences. Clarify genuine blockers and keep live submissions or writes within the user's authorization.

Worked exercise: normalize a DNS name locally

- Input: `domain/hostname`, value and source UUID.
- Normalize case, one root dot and standard-library IDNA.
- Reject malformed labels, unsupported types, IP literals and bad UUIDs.
- Preserve the input; add a referenced MISP-standard object.
- No DNS lookup, provider account, model call or runtime agent framework.

Training scope

This is a contract exercise, not a proposal to add another normalizer upstream. A contribution must address a real project gap.

Worked exercise: normalize a DNS name locally
Input: domain/hostname, value and source UUID.
Normalize case, one root dot and standard-library IDNA.
Reject malformed labels, unsupported types, IP literals and bad UUIDs.
Preserve the input; add a referenced MISP-standard object.
No DNS lookup, provider account, model call or runtime agent framework.
Training scope
This is a contract exercise, not a proposal to add another normalizer upstream. A contribution must address a real project gap.

The README defines the exact grammar and acceptance cases. The exercise uses multi-label names, at most 63 bytes per ASCII label and 253 bytes overall after IDNA. It does not check registration, reachability, public suffixes or maliciousness.

An acceptance request the agent can test

```
{
  "module": "training_domain_normalize",
  "attribute": {
    "type": "domain",
    "value": "B\u00dcCHER.Example.",
    "uuid": "13d8d18b-784a-42a0-9a58-a9cddbfda561",
    "to_ids": true,
    "comment": "Synthetic training source"
  }
}
```

Expected: `xn--bcher-kva.example` in a `domain-ip` object linked to the original source UUID.

An acceptance request the agent can test
{
"module": "training_domain_normalize",
"attribute": {
"type": "domain",
"value": "B\u00dcCHER.Example.",
"uuid": "13d8d18b-784a-42a0-9a58-a9cddbfda561",
"to_ids": true,
"comment": "Synthetic training source"
}
}
Expected: xn--bcher-kva.example in a domain-ip object linked to the original source UUID.

The JSON escape represents uppercase U with umlaut. Original input value and detection flag stay unchanged in the source Attribute. The new object's domain attribute is non-IDS. Check the serialized result, not just an agent's printed summary.

Ask for small, reviewable implementation steps

1. Metadata and introspection.
2. Pure parsing/normalization and validation.
3. MISP result construction with references.
4. A thin provider boundary, if the task needs one.
5. Tests and documentation for the actual contract.

Inspect the diff

Reject unrelated framework changes, dependency churn, import-time networking and invented endpoint behavior.

Ask for small, reviewable implementation steps
1. Metadata and introspection.
2. Pure parsing/normalization and validation.
3. MISP result construction with references.
4. A thin provider boundary, if the task needs one.
5. Tests and documentation for the actual contract.
Inspect the diff
Reject unrelated framework changes, dependency churn, import-time networking and invented endpoint behavior.

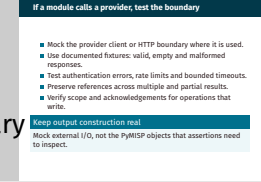
The reference is an answer key for the exercise. Compare a real integration with a recent maintained sibling. Reuse existing helpers/builders instead of duplicating target-format logic or changing the dispatcher to accommodate one module.

If a module calls a provider, test the boundary

- Mock the provider client or HTTP boundary where it is used.
- Use documented fixtures: valid, empty and malformed responses.
- Test authentication errors, rate limits and bounded timeouts.
- Preserve references across multiple and partial results.
- Verify scope and acknowledgements for operations that write.

Keep output construction real

Mock external I/O, not the PyMISP objects that assertions need to inspect.



The reference has no external I/O to mock. The source audit links real upstream DNS tests as an example. Tests should not need real credentials by default. A transport success does not establish complete publication or activation at the target.

Make tests challenge the implementation

Dimension	What can go wrong
Input/schema	Missing fields, unsupported types, Unicode and length boundaries.
MISP semantics	Wrong relations, lost source UUIDs, changed detection flags.
State isolation	Caller or request mutates another result or shared metadata.
Errors	Exceptions hidden as empty success; errors ignored at HTTP 200.
Deployment	Service imports another package or skips the new file.

Make tests challenge the implementation

Dimension	What can go wrong
Input/schema	Missing fields, unsupported types, Unicode and length boundaries.
MISP semantics	Wrong relations, lost source UUIDs, changed detection flags.
State isolation	Caller or request mutates another result or shared metadata.
Errors	Exceptions hidden as empty success; errors ignored at HTTP 200.
Deployment	Service imports another package or skips the new file.

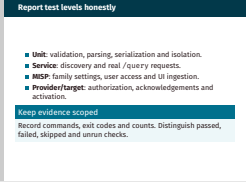
Ask which failure each assertion would catch. Do not weaken assertions or delete fixtures just to turn a run green. The invalid inputs are subtests; positive cases inspect actual MISP objects. Syntax-only tests cannot demonstrate these semantics.

Report test levels honestly

- **Unit**: validation, parsing, serialization and isolation.
- **Service**: discovery and real `/query` requests.
- **MISP**: family settings, user access and UI ingestion.
- **Provider/target**: authorization, acknowledgements and activation.

Keep evidence scoped

Record commands, exit codes and counts. Distinguish passed, failed, skipped and unrun checks.



The verifier runs eight unit tests by default. Adding –url runs three HTTP tests too. It says explicitly when HTTP checks were not requested. Unit passes do not imply a tested service, MISP UI or provider integration.

Verify and contribute

Check what the service actually imports

- Use an editable package install or explicit custom root.
- Install dependencies in the service environment.
- Restart the correct process.

```
# From this companion deck's directory:
misp-modules -l 127.0.0.1 -p 6666 \
  --custom "$PWD/examples/custom"
```

The root contains `expansion/training_domain_normalize.py`.

Check what the service actually imports
Use an editable package install or explicit custom root.
Install dependencies in the service environment.
Restart the correct process.
From this companion deck's directory:
misp-modules -l 127.0.0.1 -p 6666 \
--custom "$PWD/examples/custom"
The root contains
expansion/training_domain_normalize.py.

Pass the root, not its expansion directory. Upstream modules use the normal package family directory. A duplicate custom identifier replaces an installed module, so choose names deliberately and use a separate workshop service.

Verify discovery and dispatch

```
curl --fail --silent --show-error \
  http://127.0.0.1:6666/modules |
  jq '.[] | select(.name == "training_domain_normalize")'

python3 examples/verify_module.py \
  --url http://127.0.0.1:6666 -v
```

- Check family, capabilities, input format and config schema.
- Verify valid results and invalid-input errors through HTTP.
- Inspect `/openapi.json`, not just the handler.



Expected: eight unit plus three service tests, all executed. Discovery failures can reflect dependencies, paths, stale installation or startup errors. Health alone does not prove the new module loaded.

HTTP 200 is not the acceptance criterion

- Inspect `error`/`errors` and the actual output envelope.
- Check references and source fields after serialization.
- Decode exports and inspect the target format.
- Check action outcomes appropriate to the operation.
- Verify MISP enablement separately from service discovery.

HTTP 200 is not the acceptance criterion
- Inspect `error`/`errors` and the actual output envelope.
- Check references and source fields after serialization.
- Decode exports and inspect the target format.
- Check action outcomes appropriate to the operation.
- Verify MISP enablement separately from service discovery.

The wildcard example returns a module error over HTTP 200. A MISP UI check also needs the relevant family configured, module enabled, supported input and organisation access. Do not turn a localhost API pass into a claim of production integration.

Use the project's quality and documentation workflow

- Read current CI, Poetry groups, extras and native prerequisites.
- Run relevant tests and formatting/lint under project configuration.
- Broaden checks when changes or failures warrant it.
- Supply complete metadata and user documentation.
- Keep dependency/lock edits tied to the implementation.

The brief supplies command templates; substitute actual paths and family.

Use the project's quality and documentation workflow

Use the project's quality and documentation workflow
- Read current CI, Poetry groups, extras and native prerequisites.
- Run relevant tests and formatting/lint under project configuration.
- Broaden checks when changes or failures warrant it.
- Supply complete metadata and user documentation.
- Keep dependency/lock edits tied to the implementation.
The brief supplies command templates; substitute actual paths and family.

The reviewed CI installs the test group and all extras, starts a service and runs pytest across its matrix. This is broader than the base environment used for the reference. Documentation generation also has prerequisites; a formatter pass or zero-test run is not upstream CI success.

What the pull request should explain

1. Integration gap and resulting behavior.
2. Input/output examples and configuration, without secrets.
3. Provider/schema version and side effects.
4. Actual validation, commands, counts and limitations.
5. Dependencies and documentation needed by users.

Keep it focused

Include the implementation, tests and documentation a maintainer needs. Remove incidental edits and generated noise.

What the pull request should explain
1. Integration gap and resulting behavior.
2. Input/output examples and configuration, without secrets.
3. Provider/schema version and side effects.
4. Actual validation, commands, counts and limitations.
5. Dependencies and documentation needed by users.
Keep it focused
Include the implementation, tests and documentation a maintainer needs. Remove incidental edits and generated noise.

Describe the final implementation, not an agent conversation or abandoned plan. Separate mocked, local-service and live-integration checks. An AI-generated assertion that code is ready is not execution evidence.

Human review is part of the workflow

- Compare mappings with official API/schema documentation.
- Read validation, errors, references and config behavior.
- Inspect tests for missing cases, not only their exit status.
- Check for unexpected target writes or runtime model calls.
- Reconsider a duplicate or unnecessary upstream integration.

Human review is part of the workflow
- Compare mappings with official API/schema documentation.
- Read validation, errors, references and config behavior.
- Inspect tests for missing cases, not only their exit status.
- Check for unexpected target writes or runtime model calls.
- Reconsider a duplicate or unnecessary upstream integration.

The contributor owns the proposal and maintainers decide on merge. Prepare or publish a PR within the user's authorization. This exercise does not submit anything to misp-modules upstream; do not propose the toy normalizer without a genuine project need.

Workshop: brief, implement, verify

1. Adapt `AGENT_BRIEF.md` to the exercise.
2. Ask for a contract and acceptance plan.
3. Implement in a separate local directory; use the reference as an answer key.
4. Run unit and service checks and review source/object relationships.
5. Draft a PR description stating what was and was not tested.

Workshop: brief, implement, verify
1. Adapt AGENT_BRIEF.md to the exercise.
2. Ask for a contract and acceptance plan.
3. Implement in a separate local directory; use the reference as an answer key.
4. Run unit and service checks and review source/object relationships.
5. Draft a PR description stating what was and was not tested.

Suggested timing: ten minutes for source reading and brief, twenty to thirty for implementation/tests, ten for service checks and review. Do not create an upstream PR for the toy exercise. A real contribution replaces the capstone with an identified missing capability.

The logic to preserve

1. Existing loader, family and dispatch contracts.
2. Types, object templates, source references and detection intent.
3. Explicit configuration, bounded I/O and useful failures.
4. Repeatable tests and installation/discovery evidence.
5. A contribution that a maintainer can review.

The logic to preserve
1. Existing loader, family and dispatch contracts.
2. Types, object templates, source references and detection intent.
3. Explicit configuration, bounded I/O and useful failures.
4. Repeatable tests and installation/discovery evidence.
5. A contribution that a maintainer can review.

Use these as a final rubric. An assistant can help at each stage without inventing evidence, protocols or an architecture that bypasses the project's existing behavior.

Resources for the next contribution

- `README.md`: setup, exercise specification and source audit.
- `AGENT_BRIEF.md`: reusable task prompt.
- `examples/`: reference, request and verifier.
- MISP/misp-modules and its current contribution guide.
- Module and installation documentation.
- The companion *MISP modules* introduction and local API lab.

Resources for the next contribution
- README.md: setup, exercise specification and source audit.
- AGENT_BRIEF.md: reusable task prompt.
- examples/: reference, request and verifier.
- MISP/misp-modules and its current contribution guide.
- Module and installation documentation.
- The companion MISP modules introduction and local API lab.

Reread the target checkout before the next integration. Pinned evidence documents a reviewed snapshot, not a permanent guarantee about future APIs, dependencies or CI commands.