MISP

INTELLIGENCE SHARING

Building dashboard widgets in MISP 2.5

Typed configuration, scoped data, reusable rendering

CIRCL / MISP Project

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

Reference snapshot: 29 October 2026

Build one widget, understand the whole path

- Discover and load a custom PHP class.
- Declare typed settings and a shared time filter.
- Query only the signed-in user’s own records.
- Return renderer data and choose a safe cache scope.
- Verify the contract through PHP, HTTP and the browser.

Reference environment

MISP 2.5.50, reviewed 29 October 2026. Patch level and theme matter.

Understand the model

A widget extends the integrated dashboard

The board owns

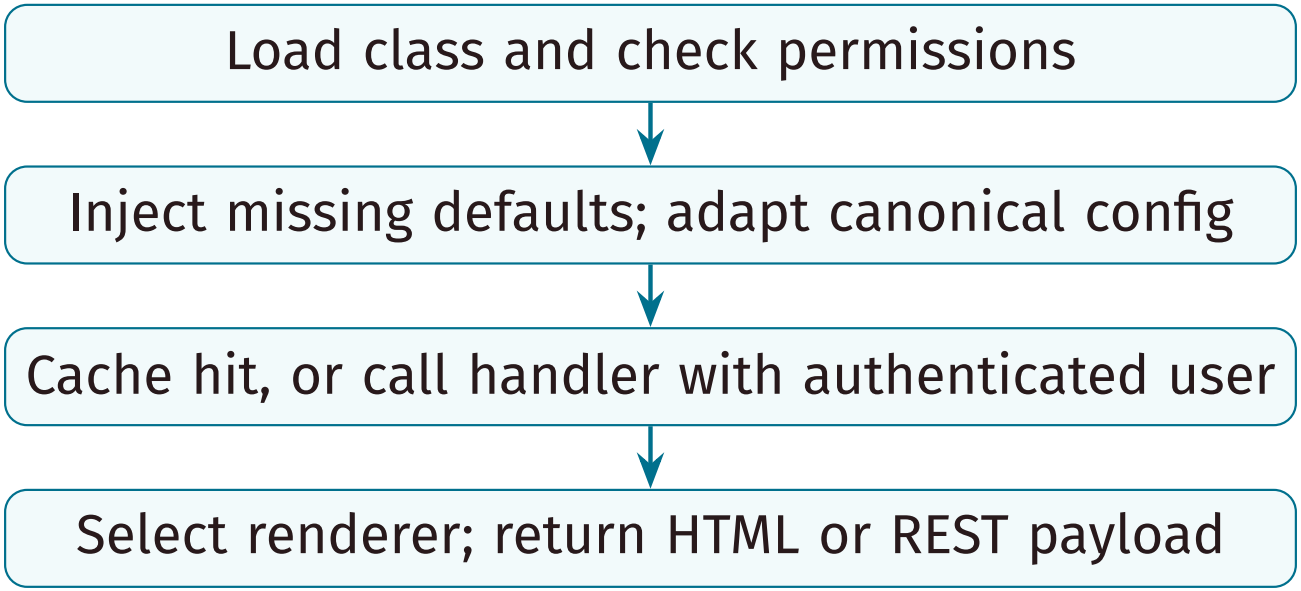
Layout, gallery, configuration, filters, scheduling and export.

Your widget owns

Metadata, input validation, data access and renderer-shaped output.

Reuse an existing renderer first. Add custom rendering only when the data needs a different visual form.

The render request has a defined pipeline



Source: DashboardsController::renderWidget

Discovery: filename, class and location

```
app/Lib/Dashboard/Custom/
  TrainingLoginSummaryWidget.php

class TrainingLoginSummaryWidget
{
    // Metadata and handler live here.
}
```

- Use a unique class ending in `Widget`.
- Match the filename exactly; no registration list.
- Built-in classes take precedence over Custom.

Metadata gives the gallery a useful entry

Property	Purpose
`title`, `description`	Explain the result and its data scope.
`category`, `thumbnail`	Help discovery; thumbnail is optional.
`render`	Name an existing renderer, here `SimpleList`.
`width`, `height`	Initial tile dimensions.
`schema`	Typed configuration and canonical filter declarations.
`params`, `placeholder`	Advanced option documentation and example JSON.

Two decisions: who may load it, what may they read?

Widget permission

`checkPermissions($user)` gates discovery and loading.

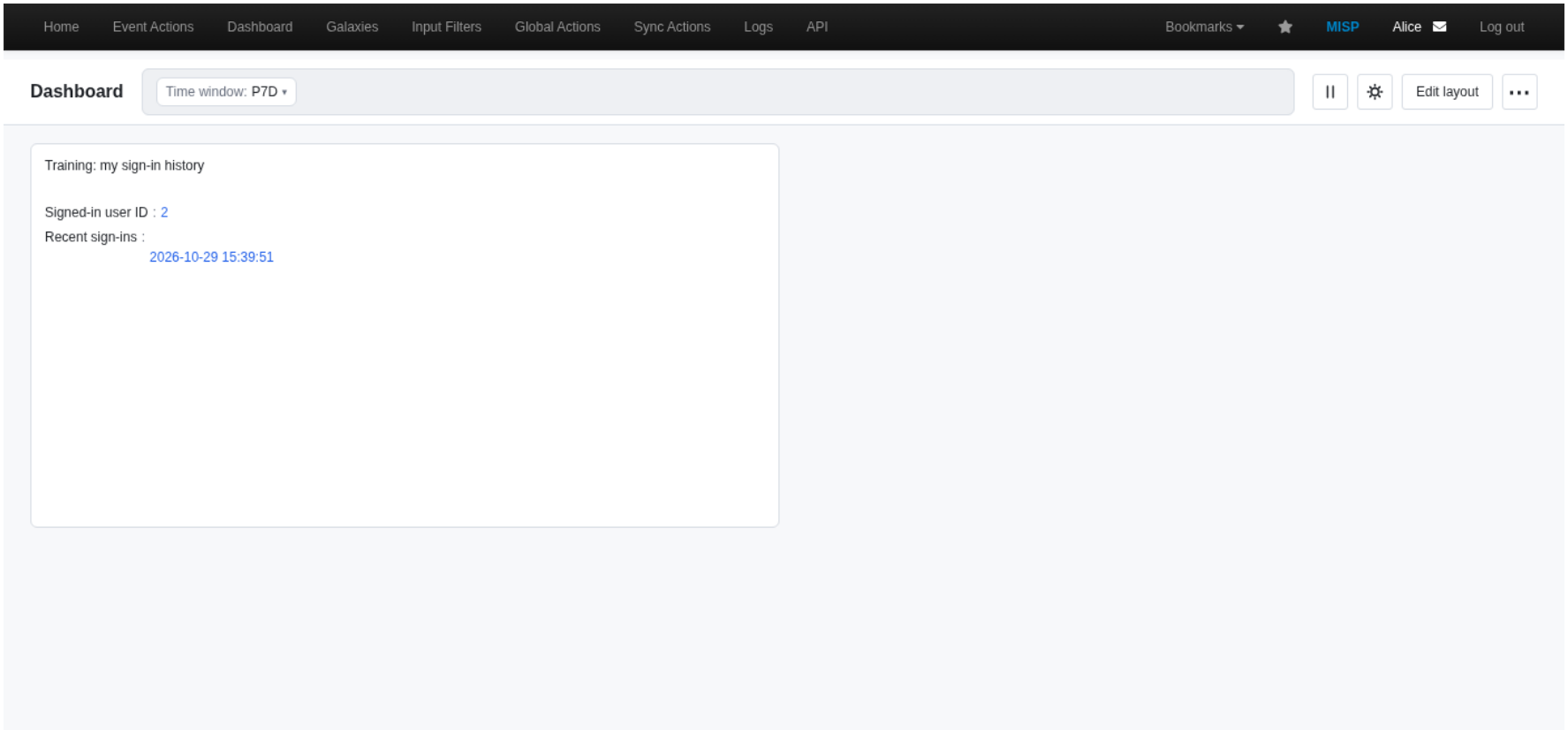
Data permission

The handler must apply the right user, organisation or event-access scope to its query.

A visible gallery card does not make every database row visible.

Build the example

Our exercise: my sign-in history



Declare the canonical time-window setting

```
public $schema = [
    'lookback' => [
        'type' => 'time_window',
        'default' => 'P7D',
        'help' => 'Your own sign-ins; at most 366 days.',
    ],
    // Scalar fields follow in the complete example.
];
```

The shared toolbar follows the declared **type**, not the name `lookback`.

Canonical inputs and scalar settings serve different jobs

Shared concepts

Time, tags, organisations, sharing groups, galaxy clusters, distribution, threat level and analysis state.

Widget-specific settings

`string`, `int`, `bool`, `enum`. Here: caption, result limit and organisation checkbox.

`attribute_type_filter` and `event_id_filter` are canonical, but are not toolbar controls in this snapshot.

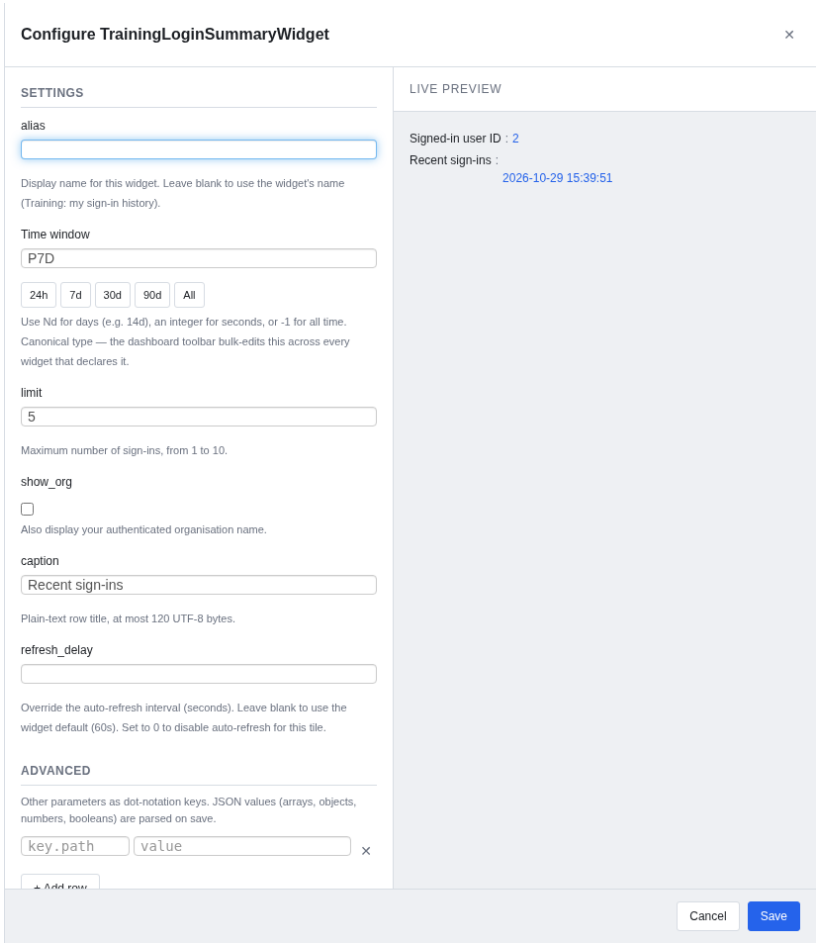
The adapter changes what the handler receives

Saved / wire value	Handler value
`P7D`	`7d`
`P2W`	`14d`
`PT6H`	`21600` seconds
`-1`	All-time sentinel, unchanged

Defaults do not mean validation

Missing keys get defaults. Explicit nulls, empty strings and unknown shapes may reach your handler.

Typed configuration, visible in the real panel



- Time-window control.
- Integer result limit.
- Boolean checkbox.
- Plain-text caption.

The panel saves configuration separately from layout editing.

Validate before querying

- `limit`: integer from 1 to 10.
- `show_org`: real boolean, not the string `"false"`.
- `caption`: nonempty UTF-8, at most 120 bytes, no controls.
- Reject unknown keys, including a supplied `user_id`.

Framework-owned settings

Accept `alias` and `refresh_delay`; do not let them change data.

Apply identity in the actual SQL conditions

```
$conditions = [
    'Log.user_id' => (int)$user['id'],
    'Log.action' => 'login',
];
if ($seconds !== -1) {
    $conditions['Log.created >='] =
        date('Y-m-d H:i:s', time() - $seconds);
}
```

Query identity comes from the authenticated user, never configuration.

Return the renderer's data, not a page

```
$rows = [
    ['title' => 'Signed-in user ID',
     'value' => (string)$user['id']],
    ['title' => $caption,
     'value' => $times ?: 'No matching sign-ins.'],
];
return $rows;
```

`SimpleList` accepts scalar or array values.
Empty history is a valid result. A database failure is an error.

Escaping belongs at the output sink

Use plain values

`<b>Plain text</b>` should appear as text in the widget.

- `SimpleList` escapes `title` and `value`.
- `html` and `html_title` deliberately bypass escaping.
- `drilldown` is checked by the dashboard URL validator.

Refresh interval and cache TTL are different

Example property	Meaning
`autoRefreshDelay = 60`	Request a refresh every 60 seconds
`cache_duration = 30`	Reuse a result for 30 seconds
`cache_scope = 'user'`	Separate cached results by user

Manual refresh can return a cache hit. Redis failure falls back to live work.

Choose cache scope from the output's audience

Scope	Appropriate when
Default	The same config yields identical data for every viewer.
`org`	An aggregate is identical within an organisation; site admins have a separate unrestricted bucket.
`user`	Personal identity, role or access changes the result.

This example needs user scope

Two users in the same organisation have different sign-in histories.

Verify and contribute

Test the wrapped REST contract

```
{
  "widget": "TrainingLoginSummaryWidget",
  "config": {
    "lookback": "P7D",
    "limit": 5,
    "show_org": false,
    "caption": "Recent sign-ins"
  }
}
```

`POST /dashboards/renderWidget`

- Response: widget, instance ID, translated config, renderer, data.
- `/exportjson:1`: bare handler data.
- `/exportcsv:1`: flattened CSV.

A layout identifies instances, not just classes

```
[
  {
    "instance_id": "training_signins",
    "widget": "TrainingLoginSummaryWidget",
    "config": {"lookback": "P7D", "limit": 5},
    "position": {"x": 0, "y": 0, "w": 6, "h": 4}
  }
]
```

Keep instance IDs stable. Importing a layout changes the personal board.

Saving layout does not save every kind of change

Layout staging

Add, move, resize and remove; then Save changes or Discard.

Configuration persistence

The configure panel and shared toolbar save settings separately.

A layout Discard is not a universal undo.

Three layers of evidence

Layer	Verify
PHP contracts	Real schema, adapter, cache keys and template; recorded database boundary.
Live HTTP	Actual SQL results, user A/B/A, invalid inputs, export and layout import/readback.
Browser	Discovery, typed settings, saved state, literal caption, resize and usable controls.

The supplied verifier runs **59 checks**; it is not an integration test.

Workshop: install, challenge, remove

1. Export your board; install the one custom PHP file.
2. Lint it and run `verify_widget.php` against MISP.
3. Add the widget; save a caption and checkbox setting.
4. Render as A, B, A within the cache TTL.
5. Try an invalid boolean, limit, null and supplied user ID.
6. Restore the board and remove the training PHP file.

Port the older lesson deliberately

Earlier pattern	Reviewed 2.5 model
Name collision / mismatched file	Unique matching class and filename
`params` only	Schema plus deliberate legacy support
`cacheLifetime`	Duration and appropriate cache scope
Renderer assumptions	Actual element and data contract
Size-only layout	Stable IDs and position coordinates
Ad hoc testing	Contracts, live requests and browser checks

If a new renderer is necessary

- Start with the PHP element’s precise data shape.
- Reuse theme tokens and accessible, responsive markup.
- Extend the existing dashboard chart modules where needed.
- Add a gallery preview for the renderer.
- Test empty data, failures, resizing, keyboard and themes.

Make the contribution reviewable

- State the question the widget answers and the audience.
- Document schema, data visibility, refresh and cache scope.
- Include focused checks and a reproducible lab procedure.
- Show real UI evidence with synthetic data.
- Explain saved-config compatibility and deployment needs.

Continue from the repository

Working example, request, layout, verifier and pinned source audit are in `a.a-widget-dev/README.md`.