CIRCL
Computer Incident
Response Center
Luxembourg

MISP Foundations for Integration

MISP Integration Workshop

The data model: events, attributes, objects

Tags, taxonomies & galaxies

Feeds & servers

PyMISP & the REST API

Filtering exports

MISP modules

TLP:CLEAR

The data model

Events, attributes, objects — what every integration consumes.

- **Event** — a container for a report/incident; the unit of sharing and publishing.
- **Attribute** — a single data point (IP, domain, hash, URL) with a **type** and **category**.
- **Object** — a structured group of attributes following a template (e.g. `file`, `network-connection`).

> Integrations read attributes; events give them context and scope.

Tags, taxonomies & galaxies

Machine-readable labels that drive filtering and automation downstream.

- **Tags** — free-form or taxonomy-backed labels attached to events/attributes.
- **Taxonomies** — namespaced, agreed vocabularies — e.g. `tlp:clear`, `tlp:amber`, confidence, workflow state.
- **Galaxy clusters** — rich knowledge objects — e.g. `misp-galaxy:mitre-attack-pattern`.

> Downstream tools filter on these tags — consistent tagging = reliable automation.

Feeds & servers

Ingesting external intelligence.

- **Feeds** — pull indicators from remote sources (MISP feeds, CSV, free-text) on a schedule.
- **Servers** — synchronise events with other MISP instances (push / pull), respecting distribution and TLP.
- Both populate your instance so integrations have data to act on.

> Curate what you ingest — feeds are the top of the quality funnel.

PyMISP & the REST API

The backbone of every integration.

- **REST API** — everything the UI does is an authenticated HTTP call (`/events/restSearch`, `/attributes/restSearch`).
- **PyMISP** — the official Python client wrapping that API.
- **Auth keys** — per-user API keys scope access; treat them as secrets.

> If a tool integrates with MISP, it speaks REST — usually via PyMISP.

Filtering exports

Push only what detection tools should act on.

- `to_ids` **flag** — export only attributes marked for detection.
- **Tags** — include/exclude by taxonomy (e.g. only `tlp:clear` / `tlp:green`).
- **Attribute types** — restrict to what the sensor understands (IPs, domains, hashes).

> Filtering at export keeps noise and TLP-restricted data out of your sensors.

MISP modules

Enrich IOCs and interact with external services.

- **Expansion modules** — enrich an attribute (passive DNS, WHOIS, VirusTotal, Shodan) on demand.
- **Import / export modules** — parse external formats in, push formats out.
- Run as a separate `misp-modules` service the instance calls.

> Enrichment adds context without leaving the MISP workflow.

Hands-on

Hands-on: end-to-end

Take one IOC through the full integration path.

1. **Create** an event (set distribution + TLP).
2. **Tag** it (`tlp:clear`, a galaxy cluster).
3. **Add** attributes and flag IOCs `to_ids=true`.
4. **Enrich** an attribute with a MISP module.
5. **Publish** the event.
6. **Pull** it via PyMISP (`restSearch`) and inspect the result.

> This is exactly what a downstream integration does — you're doing it by hand first.

Questions?

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

MISP ↔ Wazuh Integration

MISP Integration Workshop

What is Wazuh?

Main features

MISP → Wazuh (detection)

The lookup pattern

Tuning rules & decoders

Rate-limiting & caching

Hands-on

TLP:CLEAR

What is Wazuh?

An open-source security platform — SIEM and XDR in one stack.

- **Wazuh Agent** — runs on each monitored host, shipping log, file-integrity and system data to the manager.
- **Wazuh Manager** — **decodes** that data into fields and runs the detection **rules** that produce **alerts**.
- **Wazuh Indexer & Dashboard** — store, search and visualise the alerts.

> Free, open source and self-hosted — the telemetry layer we plug MISP into.

Main features

Detection, compliance and response across the endpoints you already run.

- **Log analysis** — decoders and rules turn raw logs into scored alerts.
- **File Integrity Monitoring** — what changed on disk, when, and by whom.
- **Vulnerability detection** and **security configuration assessment**.
- **Active response** — run a command on the agent when a rule fires.
- **Integrations & API** — call out to external services for enrichment.

> Rules are the extension point — and integrations are where MISP comes in.

MISP → Wazuh (detection)

Enriching alerts with MISP

A custom integration script that queries MISP for IOCs found in Wazuh events.

- Wazuh calls an **integration script** on matching events.
- The script queries the **MISP REST API** for the extracted indicator.
- A hit turns a routine log line into an **intelligence-backed alert**.

> Wazuh sees the event; MISP says whether it matters.

The lookup pattern

Decode → extract → query → alert.

- **Decode / extract** — Wazuh pulls an IOC from the log (hash, IP, domain).
- **Query** — the custom integration script hits the MISP REST API (`/attributes/restSearch`).
- **Alert** — a match generates a **high-severity** alert with MISP context.

> The same pattern works for hashes, IPs, and domains — only the extraction changes.

Tuning rules & decoders

Make MISP-matched events surface clearly.

- Add a **dedicated rule** for the integration's output so matches stand out.
- Set an appropriate **alert level** so MISP hits rise above the noise.
- Enrich the alert with MISP fields (event id, tags, category) for the analyst.

> If a MISP match looks like every other alert, the integration adds no value.

Rate-limiting & caching

Don't hammer MISP on noisy logs.

- Noisy sources can fire **thousands** of lookups per minute.
- **Cache** recent lookups (hit **and** miss) with a short TTL.
- **Rate-limit** or pre-filter which events trigger a lookup.
- Consider a **pull model** (local IOC list) for high-volume sources.

> Protect the MISP instance — a lookup storm degrades it for everyone.

Hands-on

Hands-on: see it fire

Trigger a match and watch the pipeline end-to-end.

1. **Trigger** a test event — a simulated malicious hash or DNS lookup.
2. **Watch** the MISP lookup fire from the integration script.
3. **See** the enriched, high-severity Wazuh alert with MISP context.

> One synthetic IOC proves the whole detection path works.

Questions?

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

Log4Shell

MISP ⇄ Wazuh interoperability demo

Intel published & synced

Detection — network and file

Triage & case management

Exposure assessment & fleet hunt

Response

TLP:CLEAR

The scenario

One incident, end to end — from community advisory to response.

- We run **Wazuh** across a fleet and subscribe to a **MISP** sharing community.
- A member publishes a **Log4Shell** campaign — `CVE-2021-44228`, RCE in Apache Log4j.
- Exploited by logging a crafted `${jndi:...}` string that makes the app fetch and run attacker code.
- Watch shared intel become **detection**, **triage**, **risk assessment** and **mitigation**.

> Everything from the tutorial, tied together by a single incident.

Stage 1 — Intel published upstream

What an analyst builds when writing up a fresh campaign.

- **Event metadata** — `info`, an event report, a **TLP tag**, and the `Exploit Public-Facing Application (T1190)` attack-pattern galaxy.
- **Attributes** — callback `domain` and `ip-dst` flagged `to_ids`, plus the JNDI payload as `text` for context.
- **Objects** — a `file` object with `log4j-core-2.14.1.jar` and its **md5**.

> A human-readable report **plus** machine-readable IOCs — the shape of a real advisory.

Stage 2 — Sync into the local MISP

Turning "someone published intel" into "our stack knows about it".

- Pull from the upstream via a **feed** or a **server-to-server** connection.
- The event arrives with attributes, objects and tags intact.
- Once pulled, the `to_ids` IOCs are immediately matchable — no import step.

> Everything downstream depends on the event being visible to the API user the Wazuh integration authenticates as.

Detection

Stage 3.1 — Wazuh catches the network IOC

Simulate the exploited host phoning home.

```
 # agent — force the query onto the monitored interface
dig @8.8.8.8 <CALLBACK_DOMAIN>
curl -m5 -k https://<CALLBACK_IP>/ ; true

# manager — watch the enriched alert
sudo tail -f /var/ossec/logs/alerts/alerts.json | grep -i misp
```

- Suricata logs the DNS/connection event; Wazuh queries MISP for the indicator.
- **Expected:** a MISP-match alert (rule `100620` / `100622`).

> Wazuh had no signature — MISP supplied the knowledge.

Stage 3.2 — …and the file hash, via FIM

The same integration matches hashes with no code changes.

- `custom-misp.py` reads `md5_after` / `sha1_after` / `sha256_after` from syscheck events and looks them up the same way.
- Enable FIM hashing on a watched directory, then drop the vulnerable JAR:

```
cd /tmp && wget https://repo1.maven.org/maven2/org/apache/logging/\
log4j/log4j-core/2.14.1/log4j-core-2.14.1.jar
```

- Produces a hash-match IoC alert (rule `100623`).

> One MISP event, two detection surfaces — **network** and **file**.

Stage 4 — Triage & open a case

Capture the investigation somewhere durable.

- Confirm the MISP match in the alert, pivot to the event, read the write-up.
- Open a **Flowintel** case (`http://192.168.56.50`) holding:
  - the **Wazuh alert** — rule id, agent, timestamp, matched IOC;
  - the **MISP event UUID** — linking case ⇄ intel;
  - the **initial scope** and room for later findings.

> Consumed intel can become produced intel — the loop closes here.

Stage 5 — Am I actually exploitable?

An IOC hit says a host talked to something. Not that it is vulnerable.

- Wazuh → **Threat Intelligence** → **Vulnerability Detection**, query `CVE-2021-44228` across agents.
- Search the software inventory for log4j-core `2.0-beta9` → `2.14.1`.
- **Blind spot:** syscollector enumerates OS **packages**, not loose files. A bundled `.jar` in a `lib/` folder never shows up.

> An IOC hit is a signal, not a verdict. This is what turns an alert into a risk assessment.

Stage 6 — Fleet hunt with osquery

Ask "where does this file exist?" across many hosts.

```
 SELECT path, filename, size FROM file
WHERE path LIKE '/opt/%%/log4j-core-%.jar'
   OR path LIKE '/usr/%%/log4j-core-%.jar'
   OR path LIKE '/usr/%%/bin/log4j-core-%.jar'
   OR path LIKE '/home/%%/log4j-core-%.jar';
```

- osquery exposes the OS as **SQL tables**; Wazuh integrates it **config-driven** (scheduled queries / packs), not ad-hoc dispatch.
- Demoed live with `osqueryi`; in production, an osquery server fronts the fleet.

> Finds the vulnerable component inventory can't see.

Stage 6 — Exploitation attempts with YARA

Spot the `${jndi:...}` payload — including obfuscated variants.

```
$ yara -s log4shell.yar /var/log/apache2/access.log
EXPL_Log4j_CVE_2021_44228_Dec21_OBFUSC  /var/log/apache2/access.log
0x4c:$x1:  $%7Bjndi:
0x1c5:$x4: ${jndi:${lower:
0x348:$x7: ${::-l}${::-d}${::-a}${::-p}
```

- Rules from **Rulezet.org** — search `log4j`.
- The long list of `$x*` patterns is why a hand-written grep misses real attacks.

> Alert = someone probed us. Inventory = who is exposed. Hunt = where the component actually lives.

Stage 7 — Response

Detection without a decision is not enough.

- **Contain** — isolate the exposed host (Wazuh **Active Response**, or a documented manual step).
- **Mitigate** — patch log4j-core to `2.17.1`, or remove the `JndiLookup` class where an immediate patch isn't possible.
- **Record** — update the Flowintel case: status, actions, owner, and any new IOCs (which can be fed back into MISP).

> Ending on an action is what makes this incident response rather than tool usage.

Know the vulnerability

vulnerability.circl.lu/vuln/CVE-2021-44228

- **Affected:** `2.0-beta9` → `2.15.0`. `2.15.0` only **disabled** JNDI lookup — removed in `2.16.0`; back-ports `2.12.2`, `2.12.3`, `2.3.1`.
- **Conditions:** needs JNDI message-lookup enabled **and** attacker-influenced logging — which is why the callback and log hunt are the real evidence.
- **Severity:** CVSS **10.0**, in **CISA KEV**, EPSS ≈ **0.99999**.

> MISP can pull this in automatically — expand the `vulnerability` attribute so CVSS/KEV/EPSS travel **with** the intel.

Questions?

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

Broadening the Ecosystem

MISP Integration Workshop

What is Suricata?

Suricata — network events → MISP lookup

What is Zeek?

Zeek — Threat Intel module

TLP:CLEAR

What is Suricata?

An open-source network IDS/IPS and traffic-logging engine.

- Inspects packets from a **span port, tap or inline bridge** — traffic no host agent ever sees.
- Matches them against **signature rulesets** (e.g. ET Open) to raise alerts.
- Also logs protocol metadata — DNS, HTTP, TLS, flows — to `eve.json`.

> Signature-first: "did this traffic match a rule I already know about?"

Suricata

Ship Suricata's network events to Wazuh and let Wazuh query MISP.

- Suricata logs DNS, HTTP, TLS and flow events to `eve.json`.
- The Wazuh agent reads that log and forwards the observed IPs and domains.
- The manager's `custom-misp` integration looks up each one and alerts on a hit.

> Network-layer coverage — catch IOCs on the wire, not just on the host.

What is Zeek?

An open-source network analysis framework (formerly Bro).

- Doesn't alert on signatures — it **describes** traffic, connection by connection.
- Writes structured per-protocol logs: `conn.log`, `dns.log`, `http.log`, `ssl.log`.
- Scriptable in its own event-driven language, with an **Intel framework** for matching indicator lists.

> Context-first: "what happened on the wire, and who talked to whom?"

Zeek

Export MISP IOCs for the Zeek Threat Intel module.

- Export attributes into the **Zeek Intel framework** format.
- Zeek's **Threat Intel module** watches traffic for those indicators.
- Matches land in `intel.log` with the surrounding connection context.

> Zeek adds rich context — who talked to the IOC, when, and how.

Hands-on

Hands-on: Suricata → MISP

Push model — a MISP lookup per network event.

- Send Suricata's network events to Wazuh.
- Let the manager query MISP for each observed IP and domain.
- Trigger traffic to a known-bad indicator and watch the alert fire.

> Full steps in the tutorial.

Hands-on: Zeek → MISP

Pull model — Zeek fetches IOCs once and matches traffic itself.

- Point Zeek at MISP so it loads the IOC set into memory.
- Ship Zeek's matches to Wazuh for alerting.
- Trigger a connection to a known-bad IP and watch the match.

> Full steps in the tutorial.

Questions?

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

Scaling the Integration

MISP Integration Workshop

Why query-per-event doesn't scale

Tune the trigger

Cache MISP responses

Pull model with a CDB list

Insulate the MISP instance

TLP:CLEAR

The starting point

query-per-event integration.

- Every matching alert is handed to `custom-misp.py`.
- The script makes a **synchronous MISP REST call** per IOC.
- Fine for a few lab agents — it does **not** scale.

> Lookup volume tracks **alert** volume, not the number of real matches.

```
<ossec_config>
    <integration>
    <name>custom-misp.py</name>
    <group>syscheck</group>
    <hook_url>https://YOUR_MISP_IP:PORT/</hook_url>
    <api_key>YOUR_API_KEY</api_key>
    <alert_format>json</alert_format>
    </integration>
</ossec_config>
```

Why it breaks down

A single MISP instance on the hot path of every event.

- Every FIM and Suricata record becomes one+ MISP queries — mostly **no match**.
- A few hundred agents can produce **thousands of lookups/second**.
- Under load: rising API latency, a backed-up `wazuh-integratord` queue, dropped alerts, and MISP going unresponsive for analysts.

> Fixes go cheapest → most robust; in production you combine several.

Reduce the lookups

The cheapest win is not making the query at all.

- **Tighten the trigger** — narrow the `<group>` filter; only high-value alerts.
- **Reduce noise at the source** — scope FIM directories, tune Suricata logging.
- **Prefer specific IOC types** — hashes/domains over raw IPs; drop what you don't act on.

```
<syscheck>
  <!-- report added files -->
  <alert_new_files>yes</alert_new_files>
  <!-- scope dirs + restrict to executables -->
  <directories check_all="yes" realtime="yes"
    restrict="\.exe$|\.dll$|\.sh$|\.bin$">/usr/local/bin
  </directories>
</syscheck>
```

> Lowers load, but keeps the per-event, single-instance dependency.

Cache MISP responses

The same IOCs recur constantly across agents.

- Add a local cache (Redis or on-disk KV) in front of the MISP call.
- **Cache negative results too** — they dominate — with a short TTL.
- Or put a **caching reverse proxy** in front of `/attributes/restSearch`.

> A flood of lookups for one benign value hits MISP once, not thousands of times.

Switch to a pull model (recommended)

Export IOCs from MISP; match locally on the manager with a CDB list.

- Removes MISP from the hot path — matching is an **in-memory lookup**.
- **Zero API calls per event**, no matter how many agents you run.
- Reuse the PyMISP export (`get_iocs_csv.py`) from cron every 15–30 min.

> Trade-off: detections are only as fresh as your last sync.

Insulate the MISP instance

Treat MISP as a shared production service.

- Give the integration a **dedicated API user** — rate-limit, audit, revoke.
- Serve API traffic via a **caching proxy**; use a **read-only replica** for lookups.
- Publish IOCs as a **feed/export** that consumers pull, not live API hits.

> Keep the primary instance responsive for analysts on the web UI.

Questions?

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

Operationalizing & Automation

MISP Integration Workshop · CIRCL

Automation patterns

Data quality

Pitfalls to avoid

Q&A + pointers

TLP:CLEAR

Automation patterns

Event-driven pipelines — don't poll, react.

- **Workflows** — trigger actions on MISP events (on-publish, on-attribute-add): enrich, tag, block-list push, notify.
- **Webhooks** — push events out to external systems (SIEM, SOAR, chat) the moment they happen.
- Combine with the **pull model** (CDB list) we built for Wazuh — automation keeps the list fresh without manual exports.

> Goal: an IOC published in MISP reaches your sensors with zero human steps.

Data quality

Garbage in → false positives out. Discipline matters more than volume.

- **Warninglists** — auto-flag known-good indicators (RFC1918, public DNS, CDNs, Alexa/Tranco top sites) so they never reach detection.
- **`to_ids` discipline** — only attributes meant for detection get `to_ids=1`; context-only IOCs stay out of exports.
- **Taxonomy consistency** — agree on TLP, confidence, and workflow tags across the org so filtering is predictable.

> A small, curated feed beats a huge noisy one.

Pitfalls to avoid

The failure modes that bite teams in production.

- **False-positive IOCs** — warninglists + review before `to_ids=1`.
- **TLP leakage** — check TLP before any cross-org sync or integration push; `TLP:RED`/amber must not flow to broad block-lists.
- **IOC expiration** — set decay / first-seen–last-seen so stale indicators stop firing.
- **API auth-key hygiene** — scope keys per integration, rotate them, never commit them to git.

Q&A + pointers

Where to go next.

- **Docs** — https://www.misp-project.org/documentation/
- **MISP on GitHub** — https://github.com/MISP/MISP
- **PyMISP** — https://github.com/MISP/PyMISP
- **Training materials** — https://github.com/MISP/misp-training
- **This workshop's repo** — walkthrough, install guide, and lab OVAs.

Questions?

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

AI & MCP

MISP Integration Workshop

Why MCP for MISP

Configure misp-mcp

Available queries

Hands-on

TLP:CLEAR

AI: MCP servers

Bring MISP data to LLM assistants over the Model Context Protocol.

- **misp-mcp** — a server exposing MISP as read-only tools an AI can call.
- The assistant turns a natural-language question into the right MISP query.
- **Read-only** — the assistant can search and read, never modify.

> AI accelerates the analyst — it does not replace `to_ids` discipline.

Configure misp-mcp

Point the server at your MISP instance.

- Install: `pip install misp-mcp`
- Set `MISP_URL` and `MISP_API_KEY`.
- Use `MISP_VERIFYCERT=false` for the lab's self-signed cert.
- Register it with your MCP client — Claude Desktop / Claude Code.

> Access is read-only — a scoped API key is enough.

Client config

Register the server in your MCP client (e.g. `mcp.json`).

```
{
  "mcpServers": {
    "misp": {
      "command": "misp-mcp",
      "env": {
        "MISP_URL": "https://192.168.56.30",
        "MISP_API_KEY": "your-api-key",
        "MISP_VERIFYCERT": "false"
      }
    }
  }
}
```

What you can ask it

The server exposes read-only MISP queries as tools.

- **Events & attributes** — `search_events`, `search_attributes`, `get_event`
- **Tags & taxonomies** — `search_tags`, `list_taxonomies`, `get_taxonomy`
- **Galaxies** — `search_galaxies` (ATT&CK, threat actors, malware)
- **Feeds** — `search_feeds`

> The assistant picks the right tool — you just ask the question.

Hands-on

Hands-on: example queries

Ask MISP in natural language.

- "Any events mentioning `185.194.93.14`?"
- "Show attributes tagged `tlp:clear` added this week."
- "What ATT&CK techniques relate to this threat actor?"
- "Summarise event 1234 and list its IOCs."
- "Which feeds are enabled?"

Questions?

TLP:CLEAR

CIRCL
Computer Incident
Response Center
Luxembourg

MISP Workbench

MISP Integration Workshop

What it is & why it exists

Architecture

Ingest, correlate, search, export

Enrichment, hunts & alerting

AI / MCP, reactors & notebooks

Hands-on

TLP:CLEAR

What is MISP Workbench?

A modern, MISP-compatible threat intelligence platform.

- Runs **standalone** — no full MISP instance required.
- Ingest feeds, **correlate** indicators, search, and export.
- Stays **compatible** with the wider MISP ecosystem.

> Built for the analyst's investigation workflow, not just data entry.

MISP
WORKBENCH

Architecture

A few focused services working together.

- **FastAPI** backend exposes the API and UI.
- **OpenSearch** indexes attributes for fast search & correlation.
- **Celery** workers process feeds, hunts, and events asynchronously.
- **S3-compatible** or local storage for attachments.

> Index-first design — everything is searchable the moment it lands.

Feed management

Bring intelligence in from anywhere.

- Ingest **MISP, CSV, JSON, and freetext** feeds.
- Run them **on a schedule** or trigger manually.
- New data is indexed and ready to correlate immediately.

> One place to pull together every source you track.

Correlation & analysis

Find the overlaps that matter.

- **Batch** and **incremental** correlation scans over indexed attributes.
- Surface relationships and shared indicators across the dataset.
- Correlations update as new feed data arrives.

> Correlation is where isolated indicators become a picture.

Search & exploration

Ask precise questions, get fast answers.

- **Lucene** query syntax against the OpenSearch index.
- Fast indicator lookups across the whole dataset.
- Pivot from a hit into related attributes and events.

> The dataset is only as useful as your ability to query it.

Export

Get intelligence back out in the right shape.

- Export in **JSON, CSV, MISP, or STIX 2.1**.
- Feed downstream tools, partners, and detection pipelines.
- Same data, multiple standards-compliant formats.

> Interoperability keeps Workbench a good citizen of the ecosystem.

Enrichment

Add context automatically.

- IOC enrichment powered by **misp-modules**.
- Augment indicators with WHOIS, geolocation, reputation, and more.
- Turn a bare indicator into an actionable one.

> Enrichment answers the "so what?" behind an indicator.

Hunts & alerting

Saved searches that keep working for you.

- **Hunts** are saved searches that run **periodically**.
- A match **triggers an alert** — no manual re-running.
- Notifications are processed by **Celery** workers.

> Let the platform watch the feeds while you do other work.

AI / MCP integration

Query threat intel from your AI assistant.

- Built-in **MCP Server** (Model Context Protocol).
- Ask **Claude, Cursor, and others** to query the dataset directly.
- Natural-language access to search, correlation, and context.

> Bring the intelligence to the tools analysts already use.

Reactors & notebooks

Automate and explore with Python.

- **Reactor scripts** — Python that responds to platform events in a sandbox.
- **Interactive notebooks** for ad-hoc analyst exploration.
- Analytical tools pre-imported and ready to go.

> Scripted automation for the routine, notebooks for the novel.

Hands-on

Hands-on: ingest & explore

Load a feed and query it end-to-end.

1. **Add a feed** and run it — watch attributes get indexed.
2. **Search** with a Lucene query and pivot on a hit.
3. **Correlate** to surface overlaps with existing data.

> Full steps in the tutorial.

Hands-on: hunt & export

Turn a search into a standing alert, then share it.

1. **Save a search as a hunt** and let it run on a schedule.
2. **Trigger** a match and watch the alert fire.
3. **Export** the results as STIX 2.1 for a downstream tool.

> Full steps in the tutorial.

Questions?

TLP:CLEAR