Open Source, Security Vulnerability Disclosure and Workflow

How To Improve Coding/Review Workflows in MeliCERTes II

Team MISP Project

http://www.misp-project.org/
Twitter: *@MISPProject*

Tooling WG - MeliCERTes II code review

Reporting security vulnerabilities in MISP/Cerebrate

- **If you find security vulnerabilities (even minor ones) in MISP project, send an encrypted email** (info@circl.lu) with the details and especially how to reproduce the issues. Avoid to share publicly the vulnerability before a fix is available in MISP. PGP key fingerprint: CA57 2205 C002 4E06 BA70 BE89 EAAD CFFC 22BD 4CD5.
- We usually fix reported and confirmed security vulnerabilities in less than 48 hours.
- **We will request a CVE number** if the reporters didn't ask for one (don't forget to mention how you want to be credited).

CVE allocation at CIRCL

- We request for NVD CVE via MITRE. The CVE request is sent only if the following has been done:
  - If the bug is fixed (committed publicly)
  - The report acknowledgement is present and clear (even it’s anonymous)
  - If the original reporter has been notified (and didn’t ask for a CVE directly or via CNA)

CVE assigned and its publication

- When the CVE is published (available in the NVD database):
  - Publish the vulnerability in the website of the project (example [1])
  - Make a software release (at least a tagged version) to track down which exact version is vulnerable
  - Send a reminder to existing users via different channels about the security vulnerability

[1] `https://www.misp-project.org/security/`

CVE allocation for MeliCERTes II

- We propose to use the same model (except if there is an objection or existing modules have their own vulnerability disclosure process)
- If an organisation or author of a module used in MeliCERTes II cannot assign a CVE, we propose to take the lead for the CVE allocation (3 rules as described before)
- To add in MeliCERTes/docs[2] repository a reference to each vulnerability disclosure process

[2] https://github.com/melicertes/docs

Some random practices from MISP

- A series of random open source practices and workflow used by MISP
- Maybe some could be reused or improved for MeliCERTes II

Code of Conduct

- The MISP project has a Contributor Covenant Code of Conduct[3].
- The goal of the code of conduct is to foster an **open, fun and welcoming environment**.
- Another important aspect of the MISP projects is to welcome different areas of expertise in information sharing and analysis. The **diversity of the MISP community** is important to make the project useful for everyone.

[3] https://github.com/MISP/MISP/code_of_conduct.md

Reporting a bug, an issue or suggesting features

- The most common way to contribute to the MISP project is to report a bug, issues or suggesting features.
- Each project (MISP core, misp-modules, misp-book, misp-taxonomies, misp-galaxy, misp-object or PyMISP) has their **own issue management**.
- Don't forget that you can **cross-reference issues** from other sub-projects.
- If you know an answer or could help on a specific issue, we welcome all contributions including **useful comments to reach a resolution**.

Automatic integration and testing

- The majority of the repositories within the MISP GitHub organisation includes automatic integration with TravisCI or GitHub Actions.
- If you contribute and make a pull-request, **verify if your changes affect the result of the tests**.
- Automatic integration is not perfect including Travis but it's a quick win to catch new bugs or major issues in contribution.
- When you do a pull-request, TravisCI is automatically called[4].
  - If this fails, no worries, **review the output at Travis** (it's not always you).
- We are working on additional automatic tests including unit testing for the MISP core software (contributors are welcome).

[4] https://travis-ci.org/MISP

JSON validation for MISP libraries

- All JSON format (**galaxy, taxonomies, objects or warning-lists**) are described in a JSON Schema[5].
- The TravisCI tests are including JSON validation (via *jq*) and validated with the associated JSON schema.
- How to contribute a JSON library (objects, taxonomies, galaxy or warning-list):
  - If you update a JSON library, don't forget to run *jq_all_the_things.sh*. It's fast and easy. If it fails, review your JSON.
  - Commit your code and make a pull-request.
- Documentations (in PDF and HTML format) for the librairies are automatically generated from the JSON via asciidoctor[6].

[5] schema_name.json

[6] example `https://github.com/MISP/misp-galaxy/blob/master/tools/adoc_galaxy.py`

Documentation

- In addition to the automatic generation of documentations from JSON files, we maintain **misp-book**[7] which is a generic documentation for MISP including usage, API documentation, best practices and specific configuration settings.
- The book is generated in HTML, PDF, epub and mobi using GitBook[8] which is a framework to write documentation in MarkDown format.
- TravisCI is included in misp-book and **the book generation is tested at each commit**.
- The MISP book is regularly published on misp-project.org and circl.lu website.
- Contributors are welcome especially for new topics[9] and also fixing our broken english.

[7] `https://github.com/MISP/misp-book`

[8] `https://github.com/GitbookIO`

[9] Topics of interest are analysts best-practices,

Internet-Draft - IETF for MISP formats

- If you want to contribute to our IETF Internet-Draft for the MISP standard, misp-rfc[10] is the repository where to contribute.
- **Update only the markdown file**, the XML and ASCII for the IETF I-D are automatically generated.
- If a major release or updates happen in the format, we will publish the I-D to the IETF[11].
- The process is always MISP implementation $\rightarrow$ IETF I-D updates.

[10] https://github.com/MISP/misp-rfc

[11] https://datatracker.ietf.org/doc/search/?name=misp&activedrafts=on&rfcs=on