MISP restSearch API

An easy way to query, add and update your threat

CIRCL / Team MISP Project

CIRCL - Uni.lu

- The MISP API has grown gradually with a UI first design in many cases
- Endpoints all solved specific issues with their own rulesets
- Growth was organic - whenever the need to add a new functionality / filter popped up we've added it
- Lead to frankenmonsters such as this:

  http://localhost:5000/events/csv/download/false/false/tag1&&tag2&&tag3/Network%20activity/domain

- Open up every functionality in MISP available via the UI to the API
- Including ones related to **instance management**
- APIs that expect input objects for data creation should be **self-describing**
- **URL parameters should be discouraged,** but still usable by legacy tools (deprecation)
- APIs should be heavily **tested** (Raphael Vinot's exhaustive test suite in PyMISP)
- Largest focus on Export APIs

- Scrapped all existing type specific APIs (**deprecated**, documentation moved to legacy, still available)
- **Single entry point** - all export APIs baked into restSearch
- Queries consist of a combination of:
  - ▶ **Scope** (Event, Attribute, Sighting, more coming in the future)
  - ▶ **Filter parameters** - passed via JSON objects, url parameters (key value or ordered list)
  - ▶ A **return format**
- Everything that we could do before the rework we should be able to accomplish after the rework
- Under the hood now also used by the UI search and exports

- One of our largest issues solved: **pagination**
  - ▶ **Scope specific** pagination (number of events, attributes, etc)
  - ▶ Simply control it via the framework friendly **page / limit** parameters
  - ▶ Alternatively, use the improved **time based controls** (timestamp, publish_timestamp windows)

- Single execution with subqueries
- Internal pagination **aligned with memory limits**
  - ▶ Probing of available memory for the current process
  - ▶ **Chunking of the query results** to fit in object specific memory envelopes
  - ▶ Constructing export set on disk in chunks has slashed memory usage considerably

Designing tools that use the APIs can be complex, but there's help

- The result of our own frustration
- Built in **ReST client** with templating
- Extensive query builder UI by Sami Mokaddem
- Build queries in a simple interface, automatically set URLs, headers, etc
- Uses the self documentation of APIs
- Export your queries as **cURL or Python scripts**
- Built in testing tools (performance measurements, result parsers)
- Store queries for reuse and download the results directly

- The **maturity** of the communities and threat intel sharing at large has improved
- We are sharing more
- Most importantly: we are sharing **more context** along with technical indicators
- This allows us to **manage our data more accurately** before feeding them to our protective tools
- Different contexts (APT targeting me? Persisting techniques?) - lifecycle management
- Use several queries / boolean operators to select the slice of data most relevant for the task

CLI TOOLS FOR THE CLI GOD, AUTOMATION FOR THE AUTOMATION THRONE

- Open up commonly used system management tasks to the CLI
  - ▶ sync servers/feeds
  - ▶ caching feeds
  - ▶ Password resets
  - ▶ Server settings
  - ▶ Bruteforce protection resets
  - ▶ Enrichment
  - ▶ Worker management
- Goal was also to move away from the often malfunctioning scheduler and have cron friendly CLI scripts

Demo time!

Plans for the future

- Add export modules to the restSearch API
- Improve the query language to support some missing features (such as AND boolean operators)
- Support for extended events via the restSearch API
  - We're missing a framing structure in the export module system (how are a list of conversions encapsulated and delimited?)
  - Proof of concept of the system implemented by Christian Studer already works using the STIX / STIX2 export subsystems
  - Would open us up to simple customiseable search APIs
- Open up search APIs to other scopes (objects, users, organisations, proposals, feeds, galaxies, taxonomies)