Cerebrate

A quick intro into Cerebrate

Andras Iklody

Internal

Plan for this session

- What is Cerebrate? (tl;dr edition)
- Why do we need integration modules?
- What does an integration module look like?
- Integrating tools with Cerebrate
- Interconnection requests
- Future plans / discussion

What is Cerebrate?

- Central tool for the Melicertes project...
- ...but also a stand-alone open-source community management and orchestration tool

What issues is it trying to tackle?

- Repository of organisations and individuals along with associated public keys
- Management of trust circle objects
- Exchange of contact and sharing group information
- Instrumentation of local tool interconnections
- Local tool fleet management

What are integration modules?

- Tool agnostic integration layer
- Integrate other tools with MISP for a set of tasks
  - Manage tools
  - Feed the tool with / feed off the tool's contact information
  - Orchestrate the interconnection between local tools
  - Open a dialogue with partners to interconnect tools

Design goals

- Keep it simplistic
- Monolithic packages
- Using a common toolkit for ease of integration
- Have the ability to go beyond the default required functionalities
- Reuse visualisation systems of Cerebrate
- Still heavily WiP

What is available today?

- Some sample modules
  - Skeleton module
  - MISP connector
- Auto detection, custom / default module separation
- Local tool management
- Diagnostics
- Remote tool interconnection negotiation

What is missing?

- Mass / fleet management
- Local tool interconnections
- Opening up the local tool functions to the API
- More implementations to test our design against!

Demo time

- How does the MISP connector work?
- How to issue interconnection requests?

Let us dig into the module design

- Each module is a php file in cerebrate/src/Lib/custom (ending in *Connector.php)
- A standard toolkit is always extended (CommonConnectorTools.php)
- Each module consists of:
  - Meta information
  - Custom exposed function mapping
  - Default functions for diagnostics
  - Default functions for interconnections
  - Custom exposed functions
  - Private helper functions

Meta information

- name: The name of the module as shown in selectors, inbox messages, etc
- connectorName: The name of the module - this is the canonical name used across the application
- version: The version of the connector (not the connected tool)
- description: A description of the connector in freetext

Exposed function mapping

- List of custom functions with their configurations
- Two main types so far:
  - Index: list of items with their associated actions
  - form: GET/POST pair for interacting with the tool
- Two main scopes so far:
  - child: Automatically added accordion below the connection's metadata
  - childAction: Function that can be placed anywhere (index action, chained/embedded actions)

Default function for diagnostics

- health(): returns a standardised message about the current status of the connection
- returns a list with 2 data points
  - status: UNKNOWN/OK/ISSUES/ERROR
  - message: message shown along with the status
- For more in-depth diagnostics / error reporting, use custom actions!

Default functions for interconnections

- 3-way handhsake
- functions need to be implemented to interact with the local tool
  - initiateConnection(): returns connection payload
  - acceptConnection(): returns connection payload
  - finaliseConnection(): returns boolean
- Automatically tied into the messaging system
- Built in repeating, discarding, error handling in cerebrate
- Looped through common tools, state handling done automatically

Exploring the modules

- Skeleton module
- MISP module

Local tool interconnections

- Create plugins that purely serve to connect different local tools
- Ownership of the modules was a difficulty to handle if we kept it in the main modules
- Unlike the cross cerebrate interconnections for like tools, the connection can be done in one shot

Future plans - fleet management

- Push settings / data to multiple tools of the same type in one shot
- Visualise your connected tools
- More fine grained diagnostics

API

- Currently the modules are built for UI only
- This will change and we'll add tools to easily create API responses

Discussion

- What would you like to see to be able to integrate your tool?
- Do you have tools in mind that you would like to integrate?
- What issues do you foresee with the implementation?
  - Difficulty?
  - Lack of functionalities?
  - Lack of apetite before it's more mature?