MISP core development crash course

How I learned to stop worrying and love the PHP

Andras Iklody - *TLP:WHITE*

CIRCL
Computer Incident
Response Center
Luxembourg

CIRCL global HQ

Some things to know in advance...

- MISP is based on PHP 5.6+
- Using the MVC framework CakePHP 2.x
- What we'll look at now will be a quick glance at the structuring / layout of the code

MVC frameworks in general

- separation of business logic and views, interconnected by controllers
- main advantage is clear separation of the various components
- lean controllers, fat models (kinda...)
- domain based code reuse
- No interaction between Model and Views, ever

Structure of MISP Core app directories

- Config: general configuration files
- Console: command line tools
- Controller: Code dealing with requests/responses, generating data for views based on interactions with the models
- Lib: Generic reusable code / libraries
- Model: Business logic, data gathering and modification
- Plugin: Alternative location for plugin specific codes, ordered into controller, model, view files
- View: UI views, populated by the controller

Controllers - scope

- Each public function in a controller is exposed as an API action
- request routing (admin routing)
- multi-use functions (POST/GET)
- request/response objects
- contains the action code, telling the application what data fetching/modifying calls to make, preparing the resulting data for the resulting view
- grouped into controller files based on model actions
- Accessed via UI, API, AJAX calls directly by users
- For code reuse: behaviours
- Each controller bound to a model

Controllers - functionalities of controllers

- pagination functionality
- logging functionality
- Controllers actions can access functionality / variables of Models
- Controllers cannot access code of other controller actions (kind of...)
- Access to the authenticated user's data
- beforeFilter(), afterFilter() methods
- Inherited code in AppController

Controllers - components

- Components = reusable code for Controllers
  - Authentication components
  - RestResponse component
  - ACL component
  - Cidr component
  - IOCImport component (should be moved)

Controllers - additional functionalities

- code handling API requests
- auth/session management
- ACL management
- API management
- Security component
- important: quertString/PyMISP versions, MISP version handler
- future improvements to the export mechanisms

Models - scope

- Controls anything that has to do with:
  - finding subsets of data
  - altering existing data
  - inherited model: AppModel
  - reusable code for models: Behaviours
  - regex, trim

Models - hooking system

- Versatile hooking system
  - manipulate the data at certain stages of execution
  - code can be located in 3 places: Model hook, AppModel hook, behaviour

Model - hooking pipeline (add/edit)

- Hooks / model pipeline for data creation / edits
  - beforeValidate() (lowercase all hashes)
  - validate() (check hash format)
  - afterValidate() (we never use it
  - could be interesting if we ever validated without saving)
  - beforeSave() (purge existing correlations for an attribute)
  - afterSave() (create new correlations for an attribute / zmq)

Models - hooking pipeline (delete/read)

- Hooks for deletions
  - beforeDelete() (purge correlations for an attribute)
  - afterDelete() (zmq)
- Hooks for retrieving data
  - beforeFind() (modify the find parameters before execution, we don't use it)
  - afterFind() (json decode json fields)

Models - misc

- code to handle version upgrades contained in AppModel
- generic cleanup/data migration tools
- centralised redis/pubsub handlers
- (Show example of adding an attribute with trace)

Views - scope and structure

- templates for views
- layouts
- reusable template code: elements
  - attribute list, rows (if reused)
- reusable code: helpers
  - commandhelper (for discussion boards), highlighter for searches, tag colour helper
- views per controller

Views - Types of views and helpers

- ajax views vs normal views
- data views vs normal views vs serialisation in the controller
- sanitisation h()
- creating forms
  - sanitisation
  - CSRF

Distribution

- algorithm for checking if a user has access to an attribute
- creator vs owner organisation
- distribution levels and inheritance (events -¿ objects -¿ attributes)
- shorthand inherit level
- sharing groups (org list, instance list)
- correlation distribution
- algorithms for safe data fetching (fetchEvents(), fetchAttributes(),...)

Testing your code

- funtional testing
- impact scope
  - view code changes: only impacts request type based views
  - controller code changes: Should only affect given action
  - model code changes: can have impact on entire application
  - lib changes: can have affect on the entire application
- Don't forget: queryACL, change querystring