MISP core development hands-on exercise

Building a small nifty feature for the MISP core

Team CIRCL

MISP Training @ CIRCL
20190923

Some practical things first...

- If you'd like to take a peak at the main files already implemented:
  https://github.com/iglocska/misp-dev-training-cheat-sheet
- Full implementation:
  https://github.com/MISP/MISP/tree/dev_session/app

Let's try to develop a feature together

- Idea: Users should have the option to set alert filters for the publish alert e-mails
- By default receive all alerts as before
- If a filter is set, check if the alert is interesting for us or not

How to ensure that the feature is useful for the community at large?

- Always try to think in reusable systems instead of fixing a single issue
  - Much higher chance of getting a PR merged if it doesn't just cover your specific use-case
  - Try to stay two steps ahead, see how your feature can be reused for other tasks

User settings - a long overdue feature

- Allow users to set preferences for certain views
- For high level users, all the technical details are sometimes wasted
- Simply not being interested in certain types of data points
- Non-standard MISP deployments (island only MISP instances, etc)
- User pre-sets for certain settings

Objectives of the feature

- User should be able to do the following with filter rules:
  - set
  - get
  - remove
  - index
- Filter rules should be flexible - we do not want to anticipate all possible settings in advance
- Ensure that the system is easy to extend and reuse

Before we start with anything...

- Update our MISP instance (git pull origin 2.4)
- Fork github.com/MISP/MISP (via the github interface)
- Add a new remote to our fork:
  - via username/password auth: git remote add my_fork https://github.com/iglocska/MISP
  - via ssh: git remote add my_fork gitgithub.com:iglocska/MISP.git
- Generally a good idea to work on a new branch: git checkout -b dev_exercise
- Enable debug in MISP

Implementation

- Storage:
  - Single key/value table for all settings
  - Each user should be able to set a single instance of a key
  - Values could possible become complex, let's use JSON!
  - Add timestamping for traceability
  - Consider which fields we might want to look-up frequently for indexing

The database changes we need

- The table structure:
  - id int(11) auto increment //primary key
  - key varchar(100) //add index!
  - value text //json
  - user_id int(11) //add index!
  - timestamp int(11) //add index!
- Tie it to into the upgrade system (app/Model/AppModel.php)
- Test our upgrade process! Check the output in the audit logs

Checklist

- Outline of the changes needed:
  - New Controller (UserSettingsController.php)
  - New Model (UserSetting.php)
  - New Views (setSetting, index)
  - Add new controller actions to ACL
  - Update the e-mail alert system to use the functionality

Create the new Model skeleton

- location: /var/www/MISP/app/Model/UserSetting.php
- Create basic skeleton
- Add model relationships (hasMany/BelongsTo)
- Use the hooking functionality to deal with the JSON field (beforeSave(), beforeFind())
- Add a function that can be used to check if a user should get an alert based on filters (checkPublishFilter())
- Add a function to check if a user can access/modify a setting (checkAccess())

Create the Controller skeleton

- location: /var/www/MISP/app/Model/UserSetting.php
- Create basic skeleton
- Set pagination rules
- Define CRUD functions (exceptionally, we diverge here from the norm)
  - setSetting()
  - getSetting()
  - index()
  - delete()

Start with an API only approach at first

- setSetting():
  - Accepted methods: ADD / POST
  - Separate handling of API / UI
  - POST should create/update an entry
  - GET should describe the API

getSetting / index

- getSetting():
  - Accepted methods: GET
  - Retrieves a single setting based on either ID or setting key and user_id
  - Encode the data depending on API/UI
  - Accepted methods: GET
  - List all settings
  - Filter user scope on demand
  - Filter available scopes based on role

DELETE

- delete():
  - Accepted methods: POST / DELETE
  - Deletes a single entry based on ID or setting key
  - Encode the data depending on API/UI

Add the ACL functionalities

- Tie functions into checkAccess():
  - Check if user is allowed to execute actions and throw exceptions if not
  - Add it to: setSetting() / getSetting() / delete()
- Consider that:
  - Site admins have full reign
  - Org admins can manage their own users
  - Everyone else can self-manage

Test the functionalities

- Use the REST client
- Expectations
  - GET on /setSetting and /delete describing our endpoints
  - POST /setSetting with "key": "publish_filter", "value": "Event.tags":"%sofacy%" should return newly added or modified filter
  - GET on /index should list our entries, GET on /getSetting should show an individual entry
  - DELETE on /delete should delete the entry

Start adding the UI components

- We now have a rudimentary CRUD, let's add some simple UI views
  - setSetting as a simple form
  - index should use the parametrised generators (IndexTable)
  - Add both views to the menu systems (side-menu, global menu)
  - Don't forget about sanitisation and translations!

Add the checkPublishFilter() function to the e-mailing

- Trace the code path of the e-mail sending to understand the process
- Decide on the best place to inject our check
- Don't break the flow of the process!
- What do we have access to at this point? What format are they in?

Test if our code works correctly

- Do we see any notices / errors?
- Is our code easily accessible?
- Consider other roles! Can users/org admins do things we don't want them to do?
- Is our code-base breaking the default behaviour?
- Is our update script working as expected?

Push our code to our fork and create a pull request

- git status to check what changed / got added
- git add /path/to/file to add files we want to commit
- git commit (format: is "new/fix/chg: [topic] My description"
- git push my_fork
- Create pull request from the github interface
- Wait for Travis to run, update the code if needed