A4-Team 17 https://github.com/google/timesketch/issues/1931
Student ID: 2021111945 2021112845 2021112810 2021110669
2021112808 2021113278 2021111928 2021111933

Timesketch_importer : support psort filters

1. Description

Theme: Support psort filters when ingesting into Timesketch
Some hosts produce very large plaso data sets. As an example, a domain controller produced nearly 18 million parsed events when processed with log2timeline. Many of them are from a timeframe that is irrelevent. Also, the majority of parsed events from the DC are Windows event logs. The way log2timeline parses Windows event logs results in duplicate events: one for Creation Time and one for Last Modification Time. Most of the time we only care about the Creation Time events. And the author of the issue have a psort filter he can use that will output just the Windows event log Creation events and also filter to a time range of interest. This works great for psort output to CSV.
However, he doesn't know of a way to do an equivalent filter when importing into Timesketch. So, Our purpose is to make use of the psort filter in the timesketch_importer.
Here is an example of a psort filter that will narrow down those 18 million events to under 2 million. This is what I'd like to replicate with timesketch_importer or a similar option.
psort.py --output-time-zone 'UTC' -o dynamic -w dc-triage.csv dc-triage.plaso "(((parser == 'winevtx') and (timestamp_desc == 'Creation Time')) or (parser != 'winevtx')) and (date > datetime('2021-02-01T00:00:00'))"
In this regard, we can call psort in the background worker, and we can pass in arguments to the command. We can add a filter argument and let the user (timesketch_importer for example) set that. This should replicate exactly what you do with normal CSV output.

2. Scope

We aim to modify the source code of the Timesketch importer to add support for psort filters, which mainly involves the parts of the code that import and filter functions. We want to add support for psort filters. And this may include parsing and applying psort filter expressions, filtering unnecessary events, and retaining only those that match the criteria.

3.Pull request

Ours repository : https://github.com/yaioqin/timesketch/tree/new_branch
Comparing changes :
https://github.com/google/timesketch/compare/master...yaioqin:timesketch:new_branch?expand=1
Implementation:

Fork into my own repository

Open git bash on the windows desktop

$ git clone https://github.com/yaioqin/timesketch.git
$ git checkout -b new_branch
$ cd timesketch

```
$ git checkout -b new_branch
$ git remote add origin https://github.com/yaioqin/timesketch
$ git status
On branch new_branch
Untracked files:
   (use "git add <file>..." to include in what will be committed)
         description.md
         project_guidelines.md
         scope.md
nothing added to commit but untracked files present (use "git add" to track)
$ git add scope.md description.md project_guidelines.md
$ git commit -S -m "Support psort filters when ingesting into Timesketch "
$ git push -u origin new_branch
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 4.45 KiB | 4.45 MiB/s, done.
Total 5 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'new_branch' on GitHub by visiting:
remote:      https://github.com/yaioqin/timesketch/pull/new/new_branch
remote:
To https://github.com/yaioqin/timesketch
  * [new branch]          new_branch -> new_branch
branch 'new_branch' set up to track 'origin/new_branch'.ss
```

4. Project guidelines followed

Getting started

The supported environment for Timesketch development is Docker.
Note: Exclamation mark ! denotes commands that should run in the docker container shell, dollar sign $ denotes commands to run in your local shell.

Locations and concepts

Timesketch provides a webinterface and a REST API
The configurations is located at /data sourcecode folder
The front end uses Vue.js framework and is stored at /timesketch/frontend
Code that is used in potentially multiple locations is stored in /timesketch/lib
Analyzers are located at /timesketch/lib/analyzers
The API methods are defined in /timesketch/api
API client code is in /api_client/python/timesketch_api_client
Data models are defined in /timesketch/models

Setting up your development environment

Start a shell, change to the timesketch/docker/dev directory
$ git clone timesketch
$ cd timesketch/docker/dev
$ docker compose up

Wait a few minutes for the installation script to complete.
$ docker compose logs timesketch
Attaching to timesketch-dev
timesketch-dev | Obtaining file:///usr/local/src/timesketch
timesketch-dev | Installing collected packages: timesketch
timesketch-dev | Running setup.py develop for timesketch
timesketch-dev | Successfully installed timesketch
timesketch-dev | User dev created/updated
timesketch-dev | Timesketch development server is ready!

Per default a user dev with password dev is created for you. If you want to add additional users to your Timesketch server, run the following command:
$ docker compose exec timesketch tsctl create-user <USER> --password <PW>
User <USER> created/updated

Web server

Now, start the gunicon server that will serve the Timsketch WSGI app.
To make this task easier, we recommend using the timesketch/contrib/tsdev.sh script.
In one shell:
./tsdev.sh web
OR
$ docker compose exec timesketch gunicorn --reload -b 0.0.0.0:5000 --log-file - --timeout 120 timesketch.wsgi:application
[2021-05-25 16:36:32 +0000] [94] [INFO] Starting gunicorn 19.10.0
[2021-05-25 16:36:32 +0000] [94] [INFO] Listening at: http://0.0.0.0:5000 (94)
[2021-05-25 16:36:32 +0000] [94] [INFO] Using worker: sync
/usr/lib/python3.8/os.py:1023: RuntimeWarning: line buffering (buffering=1) isn't supported in binary mode, the default buffer size will be used
return io.open(fd, *args, **kwargs)
[2021-05-25 16:36:32 +0000] [102] [INFO] Booting worker with pid: 102
[2021-05-25 16:36:33,343] timesketch.wsgi_server/INFO Metrics server enabled

By now, you should be able to point your browser to http://localhost:5000/ and log in with the username and password combination you specified earlier (or dev:dev by default). Any changes to Python files (e.g. in the timesketch/api/v1 directory tree) will be picked up automatically.

Celery workers

Although they are written in Python, changes on importers, analyzers and other asynchronous

elements of the codebase are not picked up by the Gunicorn servers but by Celery workers.
If you're planning to work on those (or even just import timelines into your Timesketch instance), you'll need to launch a Celery worker, and re-launch it every time you bring changes to its code.
You can use timesketch/contrib/tsdev.sh for this task as well.
In a new shell, run the following:
$ ./tsdev.sh celery
OR
$ docker compose exec timesketch celery -A timesketch.lib.tasks worker --loglevel info

Restarting

To restart the webserver and celery workers, stop the execution. Depending on your system ctrl+c will do it. Then start them both as outlined before with:
$ ./tsdev.sh web
$ ./tsdev.sh celery
OR
$ docker compose exec timesketch gunicorn --reload -b 0.0.0.0:5000 --log-file - --timeout 120 timesketch.wsgi:application
$ docker compose exec timesketch celery -A timesketch.lib.tasks worker --loglevel info

Frontend-ng UI development

For development on the new frontend-ng UI, you need to install some dependencies once and start the new frontend. More on frontend development is documented here.
We recommend using the timesketch/contrib/tsdev.sh script for this task as well.
Install frontend-ng dependencies:
./tsdev.sh vue-install-deps frontend-ng
Start the new frondend-ng:
./tsdev.sh vue-dev frontend-ng
Point your browser to http://localhost:5001/ to access the new frontend UI. All changes to the timesketch/frontend-ng/ path will be automatically build and loaded in the new frontend.

API development

Exposing new functionality via the API starts at /timesketch/api/v1/routes.py. In that file the different routes / endpoints are defined that can be used. Typically every route has a dedicated Resource file in /timesketch/api/v1/resources.
A resource can have GET as well as POSTor other HTTP methods each defined in the same resource file. A good example of a resource that has a mixture is /timesketch/api/v1/resources/archive.py.
To write tests for the resource, add a section in /timesketch/api/v1/resources_test.py

Error handling

It is recommended to expose the error with as much detail as possible to the user / tool that is trying to access the resource.
For example the following will give a human readable information as well as a HTTP status code that client code can react on
if not sketch:

abort(HTTP_STATUS_CODE_NOT_FOUND, 'No sketch found with this ID.')

On the opposite side the following is not recommended:

if not sketch:

abort(HTTP_STATUS_CODE_BAD_REQUEST, 'Error')

Writing documentation

Writing documentation is critical for others to use your features, so we encourage to write documentation along side with shipping new features.

The documentation is auto generated by a Github workflow https://github.com/google/timesketch/blob/master/.github/workflows/mkdocs.yml which will execute mkdocs gh-deploy --forceand deploy changes to timesketch.org.

To test mkdocs locally, run the following in your container:

! cd /usr/local/src/timesketch

! pip3 install mkdocs mkdocs-material mkdocs-redirects

! mkdocs serve

And visit the results / review remarks, warnings or errors from mkdocs.

Formatting

Before merging a pull request, we expect the code to be formatted in a certain manner. You can use VS Code extensions to make your life easier in formatting your files. For example:

Vetur for Vue files

Python and Black for Python files.

Formatting Python files

We use black to format Python files. black is the uncompromising Python code formatter. There are two ways to use it:

Manually from the command line:

Install black following the official black documentation.

Format your file by running this command: $ black path/to/python/file

Automatically from VS Code:

Download the VS Code extension Python.

Navigate to Code -> Preferences -> Settings and search for Python Formatting Provider. Then, select black from the dropdown menu.

Enable the Format on Save option to automatically format your files every time you save them.