dfTimewolf Documentation

log2timeline

Apr 21, 2021

Contents

A framework for orchestrating forensic collection, processing and data export.

dfTimewolf consists of collectors, processors and exporters (modules) that pass data on to one another. How modules are orchestrated is defined in predefined "recipes".

CHAPTER 1

Table of contents

1.1 Getting started

1.1.1 Installation

Ideally you'll want to install dftimewolf in its own virtual environment.

```
git clone https://github.com/log2timeline/dftimewolf.git && cd dftimewolf
pip install -r requirements.txt
pip install -e .
```

You can then invoke the `dftimewolf` command from any directory.

You can also install dfTimewolf the SetupTools way: `python setup.py install`

1.1.2 Quick how-to

dfTimewolf is typically run by specifying a recipe name and any arguments the recipe defines. For example:

```
dftimewolf local_plaso /tmp/path1,/tmp/path2 --incident_id 12345
```

This will launch the local_plaso recipe against `path1` and `path2` in `/tmp`. In this recipe `--incident_id` is used by Timesketch as a sketch description.

Details on a recipe can be obtained using the standard python help flags:

```
$ dftimewolf -h
[2020-10-06 14:29:42,111] [dftimewolf          ] INFO     Logging to stdout and /tmp/
↪dftimewolf.log
[2020-10-06 14:29:42,111] [dftimewolf          ] DEBUG    Recipe data path: /Users/
↪tomchop/code/dftimewolf/data
[2020-10-06 14:29:42,112] [dftimewolf          ] DEBUG    Configuration loaded from: /
↪Users/tomchop/code/dftimewolf/data/config.json
```

(continues on next page)

(continued from previous page)

```
usage: dftimewolf [-h]
                              {aws_forensics,gce_disk_export,gcp_forensics,gcp_logging_
↪cloudaudit_ts,gcp_logging_cloudsql_ts,...}

Available recipes:

 aws_forensics                      Copies a volume from an AWS account to an␣
↪analysis VM.
 gce_disk_export                    Export disk image from a GCP project to Google␣
↪Cloud Storage.
 gcp_forensics                      Copies disk from a GCP project to an analysis VM.
 gcp_logging_cloudaudit_ts          Collects GCP logs from a project and exports them␣
↪to Timesketch.
 [...]

positional arguments:
  {aws_forensics,gce_disk_export,gcp_forensics,gcp_logging_cloudaudit_ts,...}

optional arguments:
  -h, --help            show this help message and exit
```

To get details on an individual recipe, call the recipe with the `-h` flag.

```
$ dftimewolf gcp_forensics -h
[...]
usage: dftimewolf gcp_forensics [-h] [--instance INSTANCE]
                                          [--disks DISKS] [--all_disks]
                                          [--analysis_project_name ANALYSIS_PROJECT_
↪NAME]
                                          [--boot_disk_size BOOT_DISK_SIZE]
                                          [--boot_disk_type BOOT_DISK_TYPE]
                                          [--zone ZONE]
                                          remote_project_name incident_id

Copies a disk from a project to another, creates an analysis VM, and attaches the␣
↪copied disk to it.

positional arguments:
  remote_project_name   Name of the project containing the instance / disks to
                        copy
  incident_id           Incident ID to label the VM with.

optional arguments:
  -h, --help            show this help message and exit
  --instance INSTANCE   Name of the instance to analyze. (default: None)
  --disks DISKS         Comma-separated list of disks to copy. (default: None)
  --all_disks           Copy all disks in the designated instance. Overrides
                        disk_names if specified (default: False)
  --analysis_project_name ANALYSIS_PROJECT_NAME
                        Name of the project where the analysis VM will be
                        created (default: None)
  --boot_disk_size BOOT_DISK_SIZE
                        The size of the analysis VM boot disk (in GB)
                        (default: 50.0)
  --boot_disk_type BOOT_DISK_TYPE
                        Disk type to use [pd-standard, pd-ssd] (default: pd-
                        standard)
```

(continues on next page)

(continued from previous page)

```
  --zone ZONE           The GCP zone where the Analysis VM and copied disks
                        will be created (default: us-central1-f)
```

1.2 User manual

dfTimewolf ships with *recipes*, which are essentially instructions on how to launch and chain modules.

1.2.1 Listing all recipes

Since you won't know all the recipe names off the top of your head, start with:

```
$ dftimewolf -h
[2020-10-06 14:29:42,111] [dftimewolf          ] INFO     Logging to stdout and /tmp/
↪dftimewolf.log
[2020-10-06 14:29:42,111] [dftimewolf          ] DEBUG    Recipe data path: /Users/
↪tomchop/code/dftimewolf/data
[2020-10-06 14:29:42,112] [dftimewolf          ] DEBUG    Configuration loaded from: /
↪Users/tomchop/code/dftimewolf/data/config.json
usage: dftimewolf_recipes.py [-h]
                             {aws_forensics,gce_disk_export,gcp_forensics,gcp_logging_
↪cloudaudit_ts,gcp_logging_cloudsql_ts,gcp_logging_collect,gcp_logging_gce_instance_
↪ts,gcp_logging_gce_ts,gcp_turbinia_disk_copy_ts,gcp_turbinia_ts,grr_artifact_grep,
↪grr_artifact_ts,grr_files_collect,grr_flow_collect,grr_hunt_artifacts,grr_hunt_file,
↪grr_huntresults_ts,plaso_ts,upload_ts}
                             ...

Available recipes:

 aws_forensics                    Copies a volume from an AWS account to an␣
↪analysis VM.
 gce_disk_export                  Export disk image from a GCP project to Google␣
↪Cloud Storage.
 gcp_forensics                    Copies disk from a GCP project to an analysis VM.
 gcp_logging_cloudaudit_ts        Collects GCP logs from a project and exports them␣
↪to Timesketch.
 gcp_logging_cloudsql_ts          Collects GCP logs from Cloud SQL instances for a␣
↪project and exports them to Timesketch.
 gcp_logging_collect              Collects logs from a GCP project and dumps on the␣
↪filesystem.
 gcp_logging_gce_instance_ts      GCP Instance Cloud Audit to Timesketch
 gcp_logging_gce_ts               Loads GCP Cloud Audit Logs for GCE into Timesketch
 gcp_turbinia_disk_copy_ts        Imports a remote GCP persistent disk, processes␣
↪it with Turbinia and sends results to Timesketch.
 gcp_turbinia_ts                  Processes an existing GCP persistent disk in the␣
↪Turbinia project and sends results to Timesketch.
 grr_artifact_grep                Fetches ForensicArtifacts from GRR hosts and runs␣
↪grep with a list of keywords on them.
 grr_artifact_ts                  Fetches default artifacts from a list of GRR␣
↪hosts, processes them with plaso, and sends the results to Timesketch.
 grr_files_collect                Fetches specific files from one or more GRR hosts.
 grr_flow_collect                 Download GRR flows.

Download a GRR flow's results to the local filesystem.
```

(continues on next page)

(continued from previous page)

```
 grr_hunt_artifacts                 Starts a GRR hunt for the default set of␣
↪artifacts.
 grr_hunt_file                      Starts a GRR hunt for a list of files.
 grr_huntresults_ts                 Fetches the findings of a GRR hunt, processes␣
↪them with plaso, and sends the results to Timesketch.
 plaso_ts                           Processes a list of file paths using plaso and␣
↪sends results to Timesketch.
 upload_ts                          Uploads a CSV or Plaso file to Timesketch.

positional arguments:
  {aws_forensics,gce_disk_export,gcp_forensics,gcp_logging_cloudaudit_ts,gcp_logging_
↪cloudsql_ts,gcp_logging_collect,gcp_logging_gce_instance_ts,gcp_logging_gce_ts,gcp_
↪turbinia_disk_copy_ts,gcp_turbinia_ts,grr_artifact_grep,grr_artifact_ts,grr_files_
↪collect,grr_flow_collect,grr_hunt_artifacts,grr_hunt_file,grr_huntresults_ts,plaso_
↪ts,upload_ts}

optional arguments:
  -h, --help            show this help message and exit
```

1.2.2 Get detailed help for a specific recipe

To get more details on a specific recipe:

```
$ dftimewolf grr_artifact_hosts -h
[2020-10-06 14:31:40,553] [dftimewolf          ] INFO     Logging to stdout and /tmp/
↪dftimewolf.log
[2020-10-06 14:31:40,553] [dftimewolf          ] DEBUG    Recipe data path: /Users/
↪tomchop/code/dftimewolf/data
[2020-10-06 14:31:40,553] [dftimewolf          ] DEBUG    Configuration loaded from: /
↪Users/tomchop/code/dftimewolf/data/config.json
usage: dftimewolf_recipes.py plaso_ts [-h] [--incident_id INCIDENT_ID]
                                      [--sketch_id SKETCH_ID]
                                      [--token_password TOKEN_PASSWORD]
                                      paths

Processes a list of file paths using plaso and sends results to Timesketch.

- Collectors collect from a path in the FS
- Processes them with a local install of plaso
- Exports them to a new Timesketch sketch

positional arguments:
  paths                 Paths to process

optional arguments:
  -h, --help            show this help message and exit
  --incident_id INCIDENT_ID
                        Incident ID (used for Timesketch description)
                        (default: None)
  --sketch_id SKETCH_ID
                        Sketch to which the timeline should be added (default:
                        None)
  --token_password TOKEN_PASSWORD
                        Optional custom password to decrypt Timesketch
                        credential file with (default: )
```

1.2.3 Running a recipe

One typically invokes dftimewolf with a recipe name and a few arguments. For example:

```
$ dftimewolf <RECIPE_NAME> arg1 arg2 --optarg1 optvalue1
```

Given the help output above, you can then use the recipe like this:

```
$ dftimewolf grr_artifacts_ts tomchop.greendale.xyz collection_reason
```

If you only want to collect browser activity:

```
$ dftimewolf grr_artifacts_ts tomchop.greendale.xyz collection_reason --artifact_
↪list=BrowserHistory
```

In the same way, if you want to specify one (or more) approver(s):

```
$ dftimewolf grr_artifacts_ts tomchop.greendale.xyz collection_reason --artifact_
↪list=BrowserHistory --approvers=admin
$ dftimewolf grr_artifacts_ts tomchop.greendale.xyz collection_reason --artifact_
↪list=BrowserHistory --approvers=admin,tomchop
```

~/.dftimewolfrc

If you want to set recipe arguments to specific values without typing them in the command-line (e.g. your development Timesketch server, or your favorite set of GRR approvers), you can use a `.dftimewolfrc` file. Just create a `~/.dftimewolfrc` file containing a JSON dump of parameters to replace:

```
$ cat ~/.dftimewolfrc
{
  "approvers": "approver@greendale.xyz",
  "ts_endpoint": "http://timesketch.greendale.xyz/"
}
```

This will set your `ts_endpoint` and `approvers` parameters for all subsequent dftimewolf runs. You can still override these settings for one-shot usages by manually specifying the argument in the command-line.

1.2.4 Remove colorization

dfTimewolf output will not be colorized if the environment variable `DFTIMEWOLF_NO_RAINBOW` is set.

1.3 Developer's guide

This page gives a few hints on how to develop new recipes and modules for dftimewolf. Start with the *architecture* page if you haven't read it already.

1.3.1 Codereview

As for other Log2Timeline projects, all contributions to dfTimewolf undergo code review. The process is documented here.

1.3.2 Code style

dfTimewolf follows the Log2Timeline style guide.

1.3.3 Creating a recipe

If you're not satisfied with the way modules are chained, or default arguments that are passed to some of the recipes, then you can create your own. See existing recipes for simple examples like local_plaso. Details on recipe keys are given here.

Recipe arguments

Recipes launch Modules with a given set of arguments. Arguments can be specified in different ways:

- Hardcoded values in the recipe's Python code
- `@` parameters that are dynamically changed, either:
  - Through a `~/.dftimewolfrc` file
  - Through the command line

Parameters are declared for each Module in a recipe's `recipe` variable in the form of `@parameter` placeholders. How these are populated is then specified in the `args` variable right after, as a list of `(argument, help_text, default_value)` tuples that will be passed to `argparse`. For example, the public version of the grr_artifact_hosts.py recipe specifies arguments in the following way:

```
"args": [
  ["remote_project_name", "Name of the project containing the instance / disks to␣
↪copy ", null],
  ["incident_id", "Incident ID to label the VM with.", null],
  ["--instance", "Name of the instance to analyze.", null],
  ["--disks", "Comma-separated list of disks to copy.", null],
  ["--all_disks", "Copy all disks in the designated instance. Overrides disk_names if␣
↪specified", false],
  ["--analysis_project_name", "Name of the project where the analysis VM will be␣
↪created", null],
  ["--boot_disk_size", "The size of the analysis VM boot disk (in GB)", 50.0],
  ["--boot_disk_type", "Disk type to use [pd-standard, pd-ssd]", "pd-standard"],
  ["--zone", "The GCP zone where the Analysis VM and copied disks will be created",
↪"us-central1-f"]
]
```

`remote_project_name` and `incident_id` are positional arguments - they **must** be provided through the command line. `instance`, `disks`, `all_disks`, and all other arguments starting with `--` are optional. If they are not specified through the command line, the default argument will be used. `null` will be translated to a Python `None`, and `false` will be the python `False` boolean.

1.3.4 Modules

If dftimewolf lacks the actual processing logic, you need to create a new module. If you can achieve your goal in Python, then you can include it in dfTimewolf. "There is no learning curve™".

Check out the Module architecture and read up on simple existing modules such as the LocalPlasoProcessor module for an example of simple Module.

1.4 Architecture

The main concepts you need to be aware of when digging into dfTimewolf's codebase are:

- Modules
- Recipes
- The `state` object

Modules are individual Python objects that will interact with specific platforms depending on attributes passed through the command line or `AttributeContainer` objects created by a previous module's execution. **Recipes** are instructions that define how modules are chained, essentially defining which Module's output becomes another Module's input. Input and output are all stored in a **State** object that is attached to each module.

1.4.1 Modules

Modules all extend the `BaseModule` class, and implement the `SetUp`, and `Process` functions.

`SetUp` is what is called with the recipe's modified arguments. Actions here should include things that have low overhead and can be accomplished with no big delay, like checking for API permissions, verifying that a file exists, etc. The idea here is to detect working conditions and "fail early" if the module can't run correctly.

`Process` is where all the magic happens - here is where you'll want to parallelize things as much as possible (copying a disk, running plaso, etc.). You'll be reading from containers pushed by previous modules (e.g. processed plaso files) and adding your own for future modules to process. Accessing containers is done through the `GetContainers` and `StoreContainer` functions of the `state` object.

Logging

Modules can log messages to make the execution flow clearer for the user. This is done through the module's `logger` attribute: `self.logger.info('message')`. This uses the standard python `logging` module so can use functions like `info`, `warning`, `debug`.

Error reporting

Modules can also report errors using their `ModuleError` function. Errors added this way will be reported at the end of the run. Semantically, they mean that the recipe flow didn't go as expected and should be examined.

`ModuleError` also takes a `critical` parameter, that will raise an exception and interrupt the flow of the recipe. This should be used for errors that dftimewolf can't recover from (e.g. if a binary run by one of the modules can't be found on disk).

1.4.2 Recipes

Recipes are JSON files that describe how Modules are chained, and which parameters can be ingested from the command-line. A recipe JSON object follows a specific format:

- `name`: This is the name with which the recipe will be invoked (e.g. `local_plaso`).
- `description`: This is a longer description of what the recipe does. It will show up in the help message when invoking `dftimewolf recipe_hame -h`.
- `short_description`: This is what will show up in the help message when invoking `dftimewolf -h`.
- `modules`: An array of JSON objects describing modules and their corresponding arguments.

  - `wants`: What other modules this module should wait for before calling its `Process` function.
  - `name`: The name of the module class that will be instantiated.
  - `args`: A list of (argument_name, argument) tuples that will be passed on to the module's `SetUp()` function. If `argument` starts with an `@`, it will be replaced with its corresponding value from the command-line or the `~/.dftimewolfrc` file.
- `args`: Recipes need to describe the way arguments are handled in a global `args` variable. This variable is a list of `(switch, help_message, default_value)` tuples that will be passed to the `argparse.add_argument` function for later parsing.

1.4.3 State and AttributeContainers

The State object is an instance of the DFTimewolfState class. It has a couple of useful functions and attributes:

- `StoreContainer`: Store your containers to make them available to future modules.
- `GetContainers`: Retrieve the containers stored using `StoreContainer`. It takes a `container_class` param where you can select which containers you're interested in.
- `StreamContainer`: This will push a container on the streaming queue, and any registered streaming callbacks will be called on the container. Containers stored this way are not persistent (e.g. can't be accessed with `GetContainers` later on).
- `RegisterStreamingCallback`: Use this to register a function that will be called on the container as it is streamed in real-time.

1.4.4 Life of a dfTimewolf run

The dfTimewolf cycle is as follows:

- The recipe JSON is parsed, all requested modules are instantiated, as well as the semaphores that will schedule the execution of the Module's `Process` functions.
- Command-line arguments are taken into account and passed to Module's `SetUp` function. This occurs in parallel for all modules, regardless of the semaphores they declared in the recipe.
- The modules with no blocking semaphores start running their `Process` function. At the end of their run, they free their semaphore, signalling other modules that they can proceed with their own `Process` function.
- This cycle repeats until all modules have called their `Process` function.