dfTimewolf Documentation

log2timeline

Jun 29, 2018

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. We leverage `pipenv` for that.

```
$ pip install pipenv
$ git clone https://github.com/log2timeline/dftimewolf.git && cd dftimewolf
$ pipenv install -e .
```

Then use `pipenv shell` to activate your freshly created virtual environment. You can then invoke the `dftimewolf` command from any directory.

You can still use `python setup.py install` or `pip install -e .` if you'd rather install dftimewolf this way.

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
usage: dftimewolf [-h]
                  {grr_huntresults_plaso_timesketch,local_plaso,...}

Available recipes:
```

(continues on next page)

(continued from previous page)

```
 local_plaso                        Processes a list of file paths using plaso and␣
↪sends results to Timesketch.

positional arguments:
  {grr_huntresults_plaso_timesketch,local_plaso,...}

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

To get more help on a recipe's specific flags, specify a recipe name before the `-h` flag:

```
$ dftimewolf local_plaso -h
usage: dftimewolf local_plaso [-h] [--incident_id INCIDENT_ID]
                              [--sketch_id SKETCH_ID]
                              paths

Analyze local file paths with plaso and send 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)
```

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
usage: dftimewolf [-h]
                  {grr_huntresults_plaso_timesketch,local_plaso,timesketch_upload,grr_
↪artifact_hosts,grr_hunt_artifacts,grr_flow_download,grr_hunt_file}
                  ...

Available recipes:

 grr_artifact_hosts                 Fetches default artifacts from a list of GRR␣
↪hosts, processes them with plaso, and sends the results to Timesketch.
```

(continues on next page)

(continued from previous page)

```
 grr_flow_download                   Downloads the contents of a specific GRR flow to␣
↪the filesystem.
 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_plaso_timesketch   Fetches the findings of a GRR hunt, processes␣
↪them with plaso, and sends the results to Timesketch.
 local_plaso                         Processes a list of file paths using plaso and␣
↪sends results to Timesketch.
 timesketch_upload                   Uploads a .plaso file to Timesketch.

positional arguments:
  {grr_huntresults_plaso_timesketch,local_plaso,timesketch_upload,grr_artifact_hosts,
↪grr_hunt_artifacts,grr_flow_download,grr_hunt_file}

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
usage: dftimewolf grr_artifact_hosts [-h] [--artifacts ARTIFACTS]
                                 [--extra_artifacts EXTRA_ARTIFACTS]
                                 [--use_tsk USE_TSK]
                                 [--approvers APPROVERS]
                                 [--sketch_id SKETCH_ID]
                                 [--incident_id INCIDENT_ID]
                                 [--grr_server_url GRR_SERVER_URL]
                                 hosts reason

Collect artifacts from hosts using GRR.

- Collect a predefined list of artifacts from hosts using GRR
- Process them with a local install of plaso
- Export them to a Timesketch sketch

positional arguments:
hosts                 Comma-separated list of hosts to process
reason                Reason for collection

optional arguments:
-h, --help            show this help message and exit
--artifacts ARTIFACTS
                    Comma-separated list of artifacts to fetch (override
                    default artifacts) (default: None)
--extra_artifacts EXTRA_ARTIFACTS
                    Comma-separated list of artifacts to append to the
                    default artifact list (default: None)
--use_tsk USE_TSK     Use TSK to fetch artifacts (default: False)
--approvers APPROVERS
                    Emails for GRR approval request (default: None)
--sketch_id SKETCH_ID
                    Sketch to which the timeline should be added (default:
```

(continues on next page)

(continued from previous page)

```
                    None)
--incident_id INCIDENT_ID
                    Incident ID (used for Timesketch description)
                    (default: None)
--grr_server_url GRR_SERVER_URL
                    GRR endpoint (default: http://localhost:8000/)
```

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_artifact_hosts tomchop.greendale.xyz collection_reason
```

If you only want to collect browser activity:

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

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

```
$ dftimewolf grr_artifact_hosts tomchop.greendale.xyz collection_reason --artifact_
↪list=BrowserHistory --approvers=admin
$ dftimewolf grr_artifact_hosts 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",
  "timesketch_endpoint": "http://timesketch.greendale.xyz/"
}
```

This will set your `timesketch_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.

1.3 Recipe list

dfTimewolf uses recipes, which are a way to configure Collectors, Processors, and Exporters (called Modules).

1.3.1 grr_artifact_hosts

Use this recipe to collect a predefined set of artifacts from a specific list of hosts. If you want to collect the `BrowserHistory` and `LinuxLogFiles` from `tomchop.greendale.xyz` and `admin.greendale.xyz`, use this command:

```
$ dftimewolf grr_artifact_hosts tomchop.greendale.xyz,admin.greendale.xyz --artifact_
↪list=BrowserHistory,LinuxLogFiles
```

If `artifact_list` is not provided, the list defaults to:

- Linux
  - AllUsersShellHistory
  - BrowserHistory
  - LinuxLogFiles
  - AllLinuxScheduleFiles
  - LinuxScheduleFiles
  - ZeitgeistDatabase
  - AllShellConfigs
- Mac OS
  - MacOSRecentItems
  - MacOSBashHistory
  - MacOSLaunchAgentsPlistFiles
  - MacOSAuditLogFiles
  - MacOSSystemLogFiles
  - MacOSAppleSystemLogFiles
  - MacOSMiscLogs
  - MacOSSystemInstallationTime
  - MacOSQuarantineEvents
  - MacOSLaunchDaemonsPlistFiles
  - MacOSInstallationHistory
  - MacOSUserApplicationLogs
  - MacOSInstallationLogFile
- Windows
  - WindowsAppCompatCache
  - WindowsEventLogs
  - WindowsPrefetchFiles
  - WindowsScheduledTasks
  - WindowsSearchDatabase
  - WindowsSuperFetchFiles

- WindowsSystemRegistryFiles
- WindowsUserRegistryFiles
- WindowsXMLEventLogTerminalServices

1.3.2 grr_flow_download

Use this recipe to download the results of a given GRR flow.

If because of `test_reason` you want to fetch flow `F:920AFD8` from `tomchop.greendale.xyz` and dump results into `/tmp/tomflow/`, use the following command:

```
$ dftimewolf grr_flow_download tomchop.greendale.xyz F:920AFD8 test_reason /tmp/
↪tomflow
```

1.3.3 grr_hunt_artifacts

Launches a hunt for specific artifacts. The hunt is launched with a client limit set to 100 hosts.

If because of `test_reason` you want to launch a fleet-wide artifact hunt on `BrowserHistory` artifacts, use the following command:

```
$ dftimewolf grr_hunt_artifacts BrowserHistory test_reason
```

NOTE: Since hunts take time to complete, dfTimewolf will launch the hunt and return a Hunt ID that you can then feed to `grr_huntresults_plaso_timesketch`.

1.3.4 grr_hunt_file

Launches a hunt for specific files. The hunt is launched with a client limit set to 100 hosts. This is standard procedure for creating new hunts anyways.

If because of `test_reason` you want to launch a fleet-wide file hunt on `/tmp/billgates.pl` files, use the following command:

```
$ dftimewolf grr_hunt_file /tmp/billgates.pl test_reason
```

1.3.5 grr_huntresults_plaso_timesketch

Use this recipe to collect results from a GRR Hunt, process them with a local instance of plaso, and send them to our Timesketch server.

If you want to fetch results for `H:7481F262` because of `test_reason`, use the following command:

```
$ dftimewolf grr_huntresults_plaso_timesketch H:7481F262 test_reason
```

1.3.6 local_plaso

Use this recipe to process a local file using plaso and send the results to our Timesketch server.

If because of `test_reason` you want to process all files in `/mnt/winroot` with plaso and send results to Timesketch, use the following command:

```
$ dftimewolf local_plaso /mnt/winroot test_reason
```

1.3.7 timesketch_upload

Use this recipe to upload a `.plaso` or `.csv` file to Timesketch:

```
$ dftimewolf timesketch_upload ~/cases/sem12345/sdb1.plaso
```

1.4 Module list

This is a list of existing dfTimewolf modules. To see how well they play together, see the recipe list.

1.4.1 Collectors

- `FilesystemCollector` - a simple collector that just passes a local path on to the processors.

GRR hunts

Launch or fetch results from fleet-wide GRR hunts.

- `GRRHuntArtifactCollector` - Launches a fleet-wide GRR `ArtifactCollectorFlow`
- `GRRHuntFileCollector` - Launches a fleet-wide GRR `FileFinder`
- `GRRHuntDownloader` - Downloads results from a GRR hunt.

GRR flows

Launch and fetch flows on a specific list of hosts.

- `GRRArtifactCollector` - Launches a GRR `ArtifactCollectorFlow` on specific hosts.
- `GRRFileCollector` - Launches a `FileFinder` flow on specific hosts.
- `GRRFlowCollector` - Downloads the results of an arbitrary flow.

NOTE: As a general rule, `GRRHuntArtifactCollector` and `GRRHuntFileCollector` collectors are asynchronous. They will create a hunt and return the hunt ID that should be used with `GRRHuntDownloader` once the hunt is complete. `GRRArtifactCollector`, `GRRFileCollector` and `GRRFlowCollector` will wait for results before exiting.

1.4.2 Processors

- `LocalPlasoProcessor` - processes a list of file paths with a local plaso (`log2timeline.py`) instance.

1.4.3 Exporters

- `TimesketchExporter` - exports the result of a processor to a remote Timesketch instance.
- `LocalFileSystemExporter` - exports the results of a processor to the local filesystem.

1.5 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.

1.5.1 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 = [
    ('hosts', 'Comma-separated list of hosts to process', None),
    ('reason', 'Reason for collection', None),
    ('--artifacts', 'Comma-separated list of artifacts to fetch '
     '(override default artifacts)', None),
    ('--extra_artifacts', 'Comma-separated list of artifacts to append '
     'to the default artifact list', None),
    ('--use_tsk', 'Use TSK to fetch artifacts', False),
    ('--approvers', 'Emails for GRR approval request', None),
    ('--sketch_id', 'Sketch to which the timeline should be added', None),
    ('--incident_id', 'Incident ID (used for Timesketch description)', None),
    ('--grr_server_url', 'GRR endpoint', 'http://localhost:8000')

]
```

`hosts` and `reason` are positional arguments - they **must** be provided through the command line. `artifact_list`, `extra_artifacts`, `use_tsk`, `sketch_id`, and `grr_server_url` are all optional. If they are not specified through the command line, the default argument will be used.

1.5.2 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.

1.6 Architecture

1.6.1 Three main objects

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

- Modules
- Recipes
- The `state` attribute

Modules are individual Python objects that will (for the most part) take some kind of input and produce some kind of output. **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.

Modules

Modules all extend the `BaseModule` class, and implement the `setup`, `process` and `cleanup` methods.

`setup` is what is called with the recipe's modified arguments. Actions here should include things that have low overhead and can be accomplished sequentially with no big delay, like checking for permissions on a cloud project, creating an analysis VM, verifying that a file exists, etc.

`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 adding information to the state (e.g. processed plaso files) in the module's output as you go. You can access a previous module's output (i.e. your input) using `self.state.input` and manipulate the current module's output using `self.state.output`.

`cleanup` is mostly optional, in case you manipulated the state in a way that needs post-processing (e.g. adding a "# out of #" description to the module's output)

Recipes

Recipes are a Python dictionary that describe how Modules are chained, and which parameters can be ingested from the command-line. These dictionaries have a few specific keys:

- `name`: This is the name with which the recipe will be invoked (e.g. `local_plaso`)
- `short_description`: This is what will show up in the help message when invoking `dftimewolf -h`
- `modules`: A list of dicts describing modules and their corresponding arguments.
  - `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()` method. If `argument` starts with an `@`, it will be replaced with its corresponding value from the command-line or the `~/.dftimewolfrc` file.

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` method for later parsing.

State

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

- `add_error`: Used by modules to indicate that an error occurred during execution (e.g. missing file, unauthorized access).
- `check_errors`: Display any errors that have been added. If any critical errors were added, dftimewolf will stop the execution of the recipe and exit. Non-critical errors will just be displayed and execution will continue.
- `cleanup`: Resets the state: moves the output data to the input attribute and clears the output for the next Module. Moves remaining (and therefore non-critical) errors to global_errors for later processing.

1.6.2 What happens when you run a recipe

The dftimewolf cycle is as follows:

- The recipe is parsed, and the first Module is instantiated
- Command-line arguments are taken into account and passed to Module's `setup` method.
  - Errors are checked
- The module's `process` method is called
  - Errors are checked
- Cleanup occurs; the output becomes input and the process is repeated with the next module in the recipe.