MythX CLI Documentation

Release 0.1.7

Dominik Muhs

Sep 26, 2019

Contents:

CHAPTER 1

A PythX-driven CLI for MythX

This package aims to provide a simple to use command line interface for the MythX smart contract security analysis API. It's main purpose is to demonstrate how advanced features can be implemented using the PythX Python language bindings for MythX to simplify API interaction.

1.1 What is MythX?

MythX is a security analysis API that allows anyone to create purpose-built security tools for smart contract developers. Tools built on MythX integrate seamlessly into the development environments and continuous integration pipelines used throughout the Ethereum ecosystem.

1.2 Usage

```
$ mythx
Usage: mythx [OPTIONS] COMMAND [ARGS]...

  Your CLI for interacting with https://mythx.io/

Options:
  --debug / --no-debug            Provide additional debug output
  --access-token TEXT             Your access token generated from the MythX
                                  dashboard
  --eth-address TEXT              Your MythX account's Ethereum address
  --password TEXT                 Your MythX account's password as set in the
```

(continues on next page)

(continued from previous page)

```
                                dashboard
  --format [simple|json|json-pretty]
                                The format to display the results in
  --help                        Show this message and exit.

Commands:
  analyze  Analyze the given directory or arguments with MythX.
  list     Get a list of submitted analyses.
  report   Fetch the report for a single or multiple job UUIDs.
  status   Get the status of an already submitted analysis.
  version  Display API version information.
```

1.3 Installation

The MythX CLI runs on Python 3.6+, including 3.8-dev and pypy.

To get started, simply run

```
$ pip3 install mythx-cli
```

Alternatively, clone the repository and run

```
$ pip3 install .
```

Or directly through Python's `setuptools`:

```
$ python3 setup.py install
```

- Free software: MIT license
- Documentation: https://mythx-cli.readthedocs.io.

CHAPTER 2

Installation

2.1 Stable release

To install MythX CLI, run this command in your terminal:

```
$ pip install mythx-cli
```

This is the preferred method to install MythX CLI, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for MythX CLI can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dmuhs/mythx-cli
```

Or download the tarball:

```
$ curl  -OL https://github.com/dmuhs/mythx-cli/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

3.1 Format Options

A format option is passed to the `--format` option of the `mythx` root command. E.g.:

```
$ mythx --format json-pretty report ab9092f7-54d0-480f-9b63-1bb1508280e2
```

This will print the report for the given analysis job UUID in pretty-printed JSON format to stdout. Currently the following formatters are avialable:

- `simple` (default): Print the results in simple plain text (easy to grep). This does not include all result data but a subset of it that seems relevant for most use-cases.
- `json`: Print all of the result data as a single-line JSON string to stdout.
- `json-pretty`: The same as `json`, just pretty-printed, with an indentation of two spaces and alphabetically sorted object keys.

3.2 Authentication

By default the MythX CLI authenticates the user under the free trial account. This means that no account needs to be created on first use. Simply run an analysis, fetch the results and enjoy the free MythX service!

Of course, registering for a free MythX account and upgrading come with *additional perks <https://mythx.io/plans/>*. If you have set up an account, head over to the MythX *analysis dashboard <https://dashboard.mythx.io/>*. Head to your *Profile* settings and enter your password in the *MythX API Key* section. You will be able to copy a new API access token once it has been generated. Set the environment variable `MYTHX_ACCESS_TOKEN` with your JWT token and start using the MythX CLI as authenticated user. You will be able to see all your submitted analyses, their status, reports, and more on the dashboard.

Note that you can also pass the JWT token directly to the CLI via the `--access-token` option. For security reasons it is however recommended to always pass the token through a pre-defined environment variable or a shell script you `source` from.

Alternatively, username and password can be used for authentication. This functionality is considered deprecated due to security concerns and will be removed from the MythX API in the future. For compatibility reasons it has been included, however. The username corresponds to the Ethereum address the MythX account has been registered under, and the password is the one that the user can set in the MythX dashboard. Both can be passed with the `--eth-address` and `--password` option respectively, or by setting the `MYTHX_ETH_ADDRESS` and `MYTHX_PASSWORD` environment variables.

Note that if an access token is passed in directly as well, it will take precedence and no login with username and password is performed.

3.3 The Analysis Functionality

```
Usage: mythx analyze [OPTIONS] [TARGET]...

Options:
--async / --wait     Submit the job and print the UUID, or wait for
                     execution to finish
--mode [quick|full]
--help               Show this message and exit.
```

Submit a new analysis to MythX. This command works in different scenarios, simply by calling `mythx analyze`:

1. Either `truffle-config.js` or `truffle.js` are found in the directory. In this case, the MythX CLI checks the `<project_dir>/build/contracts` path for artifact JSON files generated by the `truffle compile` command. For each artifact found a new job is submitted to the MythX API.
2. If no Truffle project can be detected, the MythX CLI will automatically enumerate all Solidity files (having the `.sol` extension) in the current directory. A confirmation prompt will be displayed asking the user to confirm the submission of the number of smart contracts found. This is done to make sure a user does not accidentally submit a huge repository of Solidity files (unless they actually want it). For automation purposes the prompt can automatically be confirmed by piping `yes` into the command, i.e. `yes | mythx analyze`.
3. To analyze specific Solidity files or bytecode, data can also explicitly be passed to the `analyze` subcommand. The two supported argument types are creation bytecode strings (beginning with `0x`) and Solidity files (valid files ending with with `.sol`). The arguments can have arbitrary order and for each a new analysis request will be submitted.

If a Solidity file is analyzed in any of the given scenarios, the MythX CLI will attempt to automatically compile the file and obtain data such as the creation bytecode and the Solidity AST to enrich the request data submitted to the MythX API. This will increase the number of detected issues (as e.g. symbolic execution tools in the MythX backend can pick up on the bytecode), as well as reduce the number of false positive issues. The MythX CLI will try to estimate the `solc` version based on the pragma set in the source code.

3.3.1 Asynchronous Analysis

In any of the above scenarios the `analyze` subcommand will poll the MythX API for job completion and print the analysis report in the user-specified format. In some situations it might not be desired to wait for the results. The MythX CLI offers an option to only submit the analysis, print the job's UUID, and exit. In any scenario, simply pass the `--async` flag. E.g. in the scenario of a Continuous Integration (CI) server the submitted UUIDs can be stored in the first step:

```
$ mythx analyze --async > uuids.txt
```

This file can be stored as a CI job artifact. Later, when the (potentially very exhaustive and long) analysis run has finished, the reports can be retrieved. This is done by simply providing the stored job IDs as an argument list to the `mythx report` command:

```
$ cat uuids.txt | xargs mythx report
```

Optionally, the format can be changed here as well, e.g. to JSON, to allow for easier automated processing further on.

3.4 Listing Past Analyses

```
Usage: mythx list [OPTIONS]

Options:
--number INTEGER RANGE  The number of most recent analysis jobs to display
--help                  Show this message and exit.
```

This subcommand lists the past analyses associated to the current user. Note that this functionality is not available for the default trial account to ensure the confidentiality of analyses submitted by its users.

By default this subcommand will list the past five analyses associated to the authenticated user account. The number of returned analyses can be updated by passing the `--number` option. It is worth noting that in its current version (`v1.4.34.4`) the API returns only objects of 20 analyses per call. If a number greater than this is passed to `mythx list`, the MythX CLI will automatically query the next page until the desired number is reached.

To prevent too many network requests, the maximum number of analyses that can be fetched it capped at 100.:

```
$ mythx list
UUID: ab9d5681-0283-4ac5-bedb-1d241b5f2bf5
Submitted at: 2019-09-13 14:21:15.063000+00:00
Status: Finished

UUID: f5e4b742-5c90-4ee2-9079-4efaec9d4e2c
Submitted at: 2019-09-13 14:21:13.582000+00:00
Status: Finished

UUID: a5f9d7c7-7d33-440d-bea7-6ad8e1b2b734
Submitted at: 2019-09-13 14:21:11.367000+00:00
Status: Finished

UUID: f66d3c91-bc77-49a2-9e84-7e00c8689b0f
Submitted at: 2019-09-13 14:21:07.076000+00:00
Status: Finished

UUID: f1164a4c-91a6-4d81-a12f-6519090cb81e
Submitted at: 2019-09-13 14:21:05.386000+00:00
Status: Finished
```

3.5 Fetching Analysis Reports

```
Usage: mythx report [OPTIONS] [UUIDS]...

Options:
--help  Show this message and exit.
```

This subcommand prints the report of one or more finished analyses in the user-specified format. By default, it will print a simple text representation of the report to stdout. This will alos resolve the report's source map locations to the corresponding line and column numbers in the Solidity source file. This is only possible if the user has specified the source map in their request and is passing the Solidity source code as text.:

```
$ mythx report ab9092f7-54d0-480f-9b63-1bb1508280e2
UUID: ab9092f7-54d0-480f-9b63-1bb1508280e2
Title: Assert Violation (Low)
Description: It is possible to trigger an exception (opcode 0xfe). Exceptions can be␣
↪caused by type errors, division by zero, out-of-bounds array access, or assert␣
↪violations. Note that explicit `assert()` should only be used to check invariants.␣
↪Use `require()` for regular input checking.


/home/spoons/diligence/mythx-qa/land/contracts/estate/EstateStorage.sol:24
  mapping(uint256 => uint256[]) public estateLandIds;
```

3.6 Fetching Analysis Status

```
Usage: mythx status [OPTIONS] [UUIDS]...

Options:
--help  Show this message and exit.
```

This subcommand prints the status of an already submitted analysis.:

```
$ mythx --staging status 381eff48-04db-4f81-a417-8394b6614472
UUID: 381eff48-04db-4f81-a417-8394b6614472
Submitted at: 2019-09-05 20:34:27.606000+00:00
Status: Finished
```

By default a simple text representation is printed to stdout, more data on the MythX API's status response can be obtained by specifying an alternative output format such as `json-pretty`.

3.7 Fetching API Version Information

```
Usage: mythx version [OPTIONS]

Options:
--help  Show this message and exit.
```

This subcommand hits the MythX API's `/version` endpoint and obtains version information on the API. This can be especially useful for continuous scans as the backend tool capabilities of MythX are constantly being improved. This means that it's a good idea to rerun old scans with newer versions of MythX as potentially more vulnerabilities can be found, false positives are removed, and additional helpful data can be returned.

The MythX team has included a hash of all versions so changes are easily noticed simply by comparing the hash an analysis has run under with the one returned by the API.:

```
$ mythx version
API: v1.4.34.4
Harvey: 0.0.33
```

(continues on next page)

(continued from previous page)

```
Maru: 0.5.3
Mythril: 0.21.14
Hashed: 00c17c8b0ae13bebc9a7f678d8ee55db
```

This output can be adapted using the `--format` parameter as well to fetch e.g. JSON output for easier parsing.

CHAPTER 4

The MythX CLI

4.1 mythx_cli package

4.1.1 Subpackages

mythx_cli.formatter package

mythx_cli.formatter.base

This module contains the base formatter interface.

class mythx_cli.formatter.base.**BaseFormatter**
Bases: abc.ABC

The base formatter interface for printing various response types.

static format_analysis_list(*obj: mythx_models.response.analysis_list.AnalysisListResponse*)
Format an analysis list response.

static format_analysis_status(*resp: mythx_models.response.analysis_status.AnalysisStatusResponse*) → str
Format an analysis status response.

static format_detected_issues(*obj: mythx_models.response.detected_issues.DetectedIssuesResponse*, *inp: mythx_models.response.analysis_input.AnalysisInputResponse*)
Format an issue report response.

static format_version(*obj: mythx_models.response.version.VersionResponse*)
Format a version response.

mythx_cli.formatter.json

This module contains the compressed and pretty-printing JSON formatters.

class mythx_cli.formatter.json.**JSONFormatter**
Bases: *mythx_cli.formatter.base.BaseFormatter*

static format_analysis_list(*resp: mythx_models.response.analysis_list.AnalysisListResponse*) → str
Format an analysis list response as compressed JSON.

static format_analysis_status(*resp: mythx_models.response.analysis_status.AnalysisStatusResponse*) → str
Format an analysis status response as compressed JSON.

static format_detected_issues(*resp: mythx_models.response.detected_issues.DetectedIssuesResponse*, *inp: mythx_models.response.analysis_input.AnalysisInputResponse*) → str
Format an issue report response as compressed JSON.

static format_version(*resp: mythx_models.response.version.VersionResponse*) → str
Format a version response as compressed JSON.

class mythx_cli.formatter.json.**PrettyJSONFormatter**
Bases: *mythx_cli.formatter.base.BaseFormatter*

static format_analysis_list(*obj: mythx_models.response.analysis_list.AnalysisListResponse*) → str
Format an analysis list response as pretty-printed JSON.

static format_analysis_status(*obj: mythx_models.response.analysis_status.AnalysisStatusResponse*) → str
Format an analysis status response as pretty-printed JSON.

static format_detected_issues(*obj: mythx_models.response.detected_issues.DetectedIssuesResponse*, *inp: mythx_models.response.analysis_input.AnalysisInputResponse*)
Format an issue report response as pretty-printed JSON.

static format_version(*obj: mythx_models.response.version.VersionResponse*)
Format a version response as pretty-printed JSON.

mythx_cli.formatter.simple_stdout

This module contains a simple text formatter class printing a subset of the response data.

class mythx_cli.formatter.simple_stdout.**SimpleFormatter**
Bases: *mythx_cli.formatter.base.BaseFormatter*

static format_analysis_list(*resp: mythx_models.response.analysis_list.AnalysisListResponse*) → str
Format an analysis list response to a simple text representation.

static format_analysis_status(*resp: mythx_models.response.analysis_status.AnalysisStatusResponse*) → str
Format an analysis status response to a simple text representation.

static format_detected_issues(*resp: mythx_models.response.detected_issues.DetectedIssuesResponse*, *inp: mythx_models.response.analysis_input.AnalysisInputResponse*) → str
Format an issue report to a simple text representation.

static format_version(*resp: mythx_models.response.version.VersionResponse*) → str
Format a version response to a simple text representation.

mythx_cli.payload package

mythx_cli.payload.bytecode

This module contains functions to generate bytecode-only analysis request payloads.

mythx_cli.payload.bytecode.**generate_bytecode_payload**(*code*)
Generate a payload containing only the creation bytecode.

Parameters **code** – The creation bytecode as hex string starting with `0x`

Returns The payload dictionary to be sent to MythX

mythx_cli.payload.solidity

This module contains functions to generate Solidity-related payloads.

mythx_cli.payload.solidity.**generate_solidity_payload**(*file*)
Generate a MythX analysis request from a given Solidity file.

This function will open the file, try to detect the used solc version from the pragma definition, and automatically compile it. If the given solc version is not installed on the client's system, it will be automatically downloaded.

From the solc output, the following data is sent to the MythX API for analysis:

- `abi`
- `ast`
- `bin`
- `bin-runtime`
- `srcmap`
- `srcmap-runtime`

Parameters **file** – The path pointing towards the Solidity file

Returns The payload dictionary to be sent to MythX

mythx_cli.payload.truffle

This module contains functions to generate payloads for Truffle projects.

mythx_cli.payload.truffle.**generate_truffle_payload**(*file*)
Generate a MythX analysis request payload based on a truffle build artifact.

This will send the following artifact entries to MythX for analysis:

- `contractName`
- `bytecode`
- `deployedBytecode`
- `sourceMap`
- `deployedSourceMap`
- `sourcePath`
- `source`

- `ast`
- `legacyAST`
- the compiler version

Parameters **`file`** – The path to the Truffle build artifact

Returns The payload dictionary to be sent to MythX

`mythx_cli.payload.truffle.`**`zero_srcmap_indices`**(*src_map: str*) → str

Zero the source map file index entries.

Parameters **`src_map`** – The source map string to process

Returns The processed source map string

4.1.2 mythx_cli.cli

The main runtime of the MythX CLI.

`mythx_cli.cli.`**`find_solidity_files`**(*project_dir*)

Return all Solidity files in the given directory.

This will match all files with the *.sol* extension.

Parameters **`project_dir`** – The directory to search in

Returns Solidity files in *project_dir* or *None*

`mythx_cli.cli.`**`find_truffle_artifacts`**(*project_dir*)

Look for a Truffle build folder and return all relevant JSON artifacts.

This function will skip the Migrations.json file and return all other files under `<project-dir>/build/contracts/`. If no files were found, `None` is returned.

Parameters **`project_dir`** – The base directory of the Truffle project

Returns Files under `<project-dir>/build/contracts/` or `None`

4.1.3 mythx_cli.util

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/dmuhs/mythx-cli/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

MythX CLI could always use more documentation, whether as part of the official MythX CLI docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dmuhs/mythx-cli/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *mythx-cli* for local development.

1. Fork the *mythx-cli* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/mythx-cli.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv mythx-cli
$ cd mythx-cli/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 mythx_cli tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7, the 3.8-dev branch, and for PyPy. Check https://travis-ci.org/dmuhs/mythx-cli/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.<test_name>
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Dominik Muhs <dominik.muhs@consensys.net>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.0 (2019-08-31)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

S

Z