MythX Docs

The MythX team

Nov 19, 2019

Contents

MythX is a security analysis API for Ethereum smart contracts. It allows any developer or developer team to integrate advanced security analysis directly into development environments and build pipelines. It detects many common Solidity vulnerabilities and EVM bytecode vulnerabilities automatically.

MythX is integrated into popular developer tools you use today such as *Truffle*. Plus, you can integrate MythX into your own security tools, apps, and existing blockchain services.

MythX has multiple target audiences:

- **Developers and dev teams** who wish to verify smart contract security as part of their project workflow.
- **Tool creators and integrators** who wish to build a new MythX tool or integrate the MythX API into their product or service.

Note: While MythX is designed with Ethereum in mind, the service should be compatible with any chain that uses the EVM (such as VeChain and Tron). In most cases, you will just have to change a setting in your development environments to deploy to the different blockchain and then you can proceed to analyzing your contracts.

Please continue on to learn more about MythX.

CHAPTER 1

Contents

1.1 Getting started with MythX security tools

This page will show how to start using MythX in the shortest number of steps.

1.1.1 Quickstart

The quickest way to see MythX in action is by using a MythX tool such as the *MythX for Truffle* plugin.

No registration is required for this, though the results will be limited. See the section on *using MythX for Truffle* for more details.

1. In a terminal, navigate to the root of your local Truffle project.
2. Install the MythX plugin:

```
npm install truffle-security
```

3. Analyze all the contracts in the project with MythX:

```
truffle run verify
```

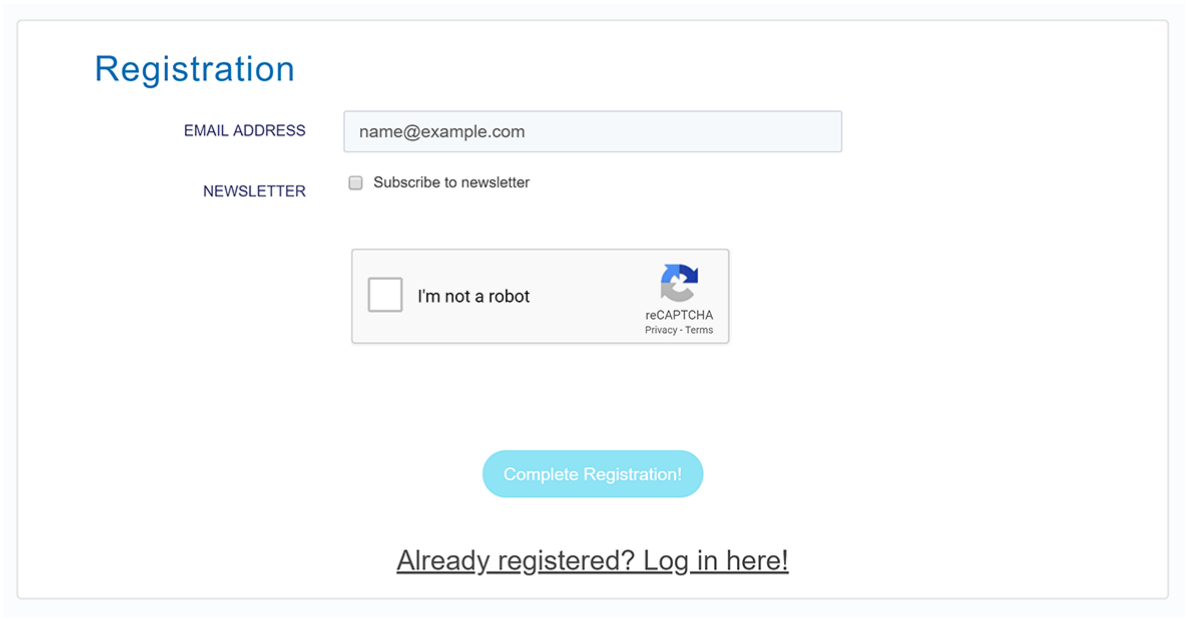
This will run a security analysis in Trial Mode, which limits the returned results to three vulnerabilities only. For a more complete report, you will need to create an account with MythX.

1.1.2 Creating an account

To get the full functionality of MythX, you need to create an account.

Note: MythX requires MetaMask, a browser-based wallet application, in order to create accounts. You must be logged in to MetaMask before continuing. The active address will be associated with your account and be used for any plans you may purchase.

1. Go to https://mythx.io and slick the *Sign Up* button on the top right.
2. Fill out the registration form. You will need to supply a valid email address.

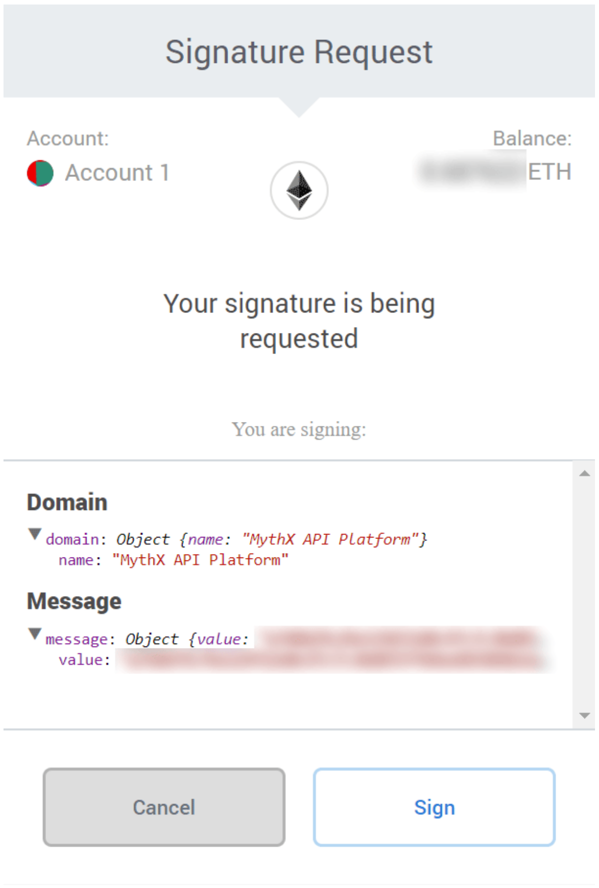


Registration

EMAIL ADDRESS name@example.com

NEWSLETTER ☐ Subscribe to newsletter

3. When finished, click *Complete Registration*.
4. A MetaMask popup will display, asking for confirmation. Click *Sign* to continue.



Signature Request

Account: Account 1

Balance: [redacted] ETH

Your signature is being requested

You are signing:

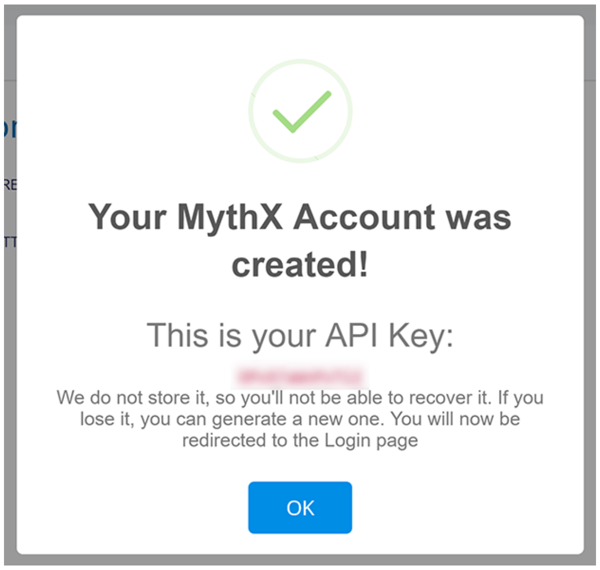
Domain

```
▼domain: Object {name: "MythX API Platform"}
   name: "MythX API Platform"
```

Message

```
▼message: Object {value: [redacted]
   value: [redacted]
```

5. An API key will be generated for you and displayed. Please copy this key down, as you will not be able to retrieve it later. (You can generate a new one in your account dashboard if necessary.)



Your MythX Account was created!

This is your API Key:

[redacted]

We do not store it, so you'll not be able to recover it. If you lose it, you can generate a new one. You will now be redirected to the Login page

6. You will be sent an email to verify your address. You will need to verify your email address before you can use the MythX service.

1.1.3 Linking your account with tools

Your account, once verified, is on the Free plan. This means that you can receive the complete report of vulnerabilities when running scans.

Note: MythX offers both free and paid plans. For information on plans and features, please see our Plans page.

If using one of the *MythX tools & integrations*, you will need to link your account to the tool to take advantage of your account's plan.

While the specifics of each tool differ, most tools will pick up your account information when stored in your system's environment variables.

Environment variable	Value
`MYTHX_ETH_ADDRESS`	Your MythX account (Ethereum address)
`MYTHX_PASSWORD`	API key supplied to you during registration

Please see *the specific page for your tool* to see more details about linking your account.

1.2 MythX tools & integrations

Welcome! Here you will find an overview of the most common tools that you can use to interact with the MythX API.

1.2.1 Current tools

Truffle integration with MythX

MythX for Truffle is a Truffle plugin that adds automated smart contract security analysis to the Truffle framework. With this plugin, you can run security analysis directly from your Truffle development environment.

The MythX plugin requires Truffle 5.0 or higher. Note that your Truffle project must compile successfully for the security analysis to work.

Installation

You can install the plugin on a per-project basis or globally.

Warning: Windows users need to install the following dependencies:

- Git
- Windows Build Tools npm package:

  ```
  npm install --global --production windows-build-tools
  ```

Both will need to be installed with Administrator privileges.

In addition, those using **Windows PowerShell** may need to set the Execution Policy to Unrestricted in order to install the Windows Build Tools package:

```
Set-ExecutionPolicy Unrestricted
```

This will also need to be run with Administrator privileges.

Individual project

Install the plugin on an individual Truffle project by running the following inside the root of your Truffle project:

```
npm install truffle-security
```

The plugin will install for that Truffle project only. In addition, the plugin will **edit the project's configuration file** (`truffle-config.js`) to add the necessary plugin configuration. You do not need to edit this file.

Note: If you have existing plugins activated for the project, they will not be affected.

Global installation

Install the plugin globally so that it is accessible to all projects:

```
npm install -g truffle-security
```

If you install MythX for Truffle in this manner, **you will in addition need to edit each project's configuration file** (`truffle-config.js`) to add the necessary plugin:

```
module.exports = {

  // ...

    plugins: [ "truffle-security" ],

  // ...

};
```

Usage

Analyzing an entire project

To run MythX for Truffle, run the following command in the root of your configured Truffle project:

```
truffle run verify
```

Note: The project must compile successfully in order for the plugin to run. If the project hasn't been compiled yet, MythX for Truffle will try to compile it first.

Analyzing whole contract files

By default, all contracts in all contract files in the project will be analyzed. To analyze only a single contract file, use the following syntax:

```
truffle run verify contract.sol
```

This will analyze all the contracts found in the file `contract.sol`.

Multiple contract files can be specified here as well:

```
truffle run verify contract1.sol contract2.sol
```

All contracts inside both `contract1.sol` and `contract2.sol` will be analyzed.

Analyzing specific contracts

You can also analyze a specific contract:

```
truffle run verify contract.sol:MyContract
```

This will analyze the contract named `MyContract` found in the file `contract.sol`.

Multiple contracts can be specified here too. For example:

```
truffle run verify contract1.sol:MyContract1 contract2.sol:MyContract2
```

This will analyze both `MyContract1` and `MyContract2`, which are found in the `contract1.sol` and `contract2.sol` files respectively.

Warning: The following syntax has been deprecated and should not be used:

```
truffle run verify MyContract
```

Options

To see the various command options available to you, run the following:

```
truffle run verify --help
```

You can pass options to the tool in two ways:

- Command line options (`--option`)
- Configuration file (`truffle-security.json`)

Command line options take precedence over any options specified in the configuration file.

Command line options

`--all`

Compile all contracts. Without this, only the contracts changed since last compile will be recompiled.

`--debug`

Provide additional debug output. Use `--debug=2` for more verbose output. Implies `--no-progress`.

`--initial-delay <N>`

Minimum amount of time (in seconds) to wait before attempting a first status poll. Default is 45 seconds. Read more about improving polling response.

`--json`

Output results in unprocessed JSON format. Differs from `--style=json` which provides an es-lint compatible output format. See also `--yaml`.

`--limit <N>`

Limit the number of parallel analysis requests to no more than `<N>`. As results come back, remaining contracts are submitted. The default and mamximum is 4, but this can be set lower.

`--min-severity <LEVEL>`

Ignore SWCs below the designated severity level. Options are `warning` or `error`.

Note: Currently, the only severity levels are `warning` and `error`, so choosing `warning` here has no effect (ignores nothing). Future versions may add support for an `info` severity level, which would be ignored.

`--mode <MODE>`

Perform `quick` or in-depth (`full`) analysis.

`--no-color`

Disable output coloring.

`--no-progress`

Disable progress bars during analysis.

`--style <STYLE>`

Output the report in the given es-lint style. Options include `stylish`, `json`, `table`, `tap`, `unix`, and `markdown`.

`--swc-blacklist <LIST>`

Ignore a specific SWC or list of SWCs. Use the number only (`107` instead of `SWC-107`). If using a list, use commas and no spaces to separate the SWCs (`103,111,115`).

`--timeout <N>`

Limit MythX analyses time to `<N>` seconds. The default is 300 seconds.

`--uuid <UUID>`

(Experimental) Display results from a prior analysis with the given UUID. Result is in YAML.

`--version`

Show package and MythX version information.

`--yaml`

Output results in unprocessed YAML format. Differs from `--style=yaml` which provides an es-lint compatible output format. See also `--json`.

Configuration file

In addition to command line options, you can specify a configuration file named `truffle-security.json`. Placed in the root of the project, this file can contain a list of options and values. Every option available on the command line is available here.

An example format of this file is as follows:

```
{
  "style": "table",
  "mode": "quick",
  "min-severity": "warning",
  "swc-blacklist": [103,111]
}
```

For arguments that don't take a value (such as `no-progress`) use the format:

```
{
  "no-format": true
}
```

For arguments that take a list (such as `swc-blacklist`), brackets for the values are optional.

Note: Command line options take precedence over any options specified in the configuration file.

Accounts and access

You do not need to sign up for a MythX account in order to use the MythX plugin for Truffle.

By default the plugin runs in Trial mode. **Trial mode returns a limited report**, with not all vulnerabilities listed. To get access to an unrestricted report, sign up for an account at https://mythx.io.

Note: Both free and paid plans are available. See *Getting started with MythX security tools* for more details.

Once you have signed up for an account, you will need to add your account and password as environment variables on your system.

Environment variable	Value
`MYTHX_ETH_ADDRESS`	Your MythX account (Ethereum address)
`MYTHX_PASSWORD`	Your MythX password

You can temporarily add these environment variables to your terminal with the following commands (which will need to be customized with your account information):

- **Linux / macOS**:

```
export MYTHX_ETH_ADDRESS=0x1234567891235678900000000000000000000000
export MYTHX_PASSWORD='Put your password in here!'
```

- **Windows**:

```
set MYTHX_ETH_ADDRESS=0x1234567891235678900000000000000000000000
set MYTHX_PASSWORD="Put your password in here!"
```

Once you have done this, the MythX plugin should recognize your credentials and elevate your privileges.

See also:

- MythX for Truffle (npm)
- MythX for Truffle (GitHub)

MythX CLI Guide: Installation & Usage

The MythX CLI Python package aims to provide a simple to use command line interface for the MythX 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.

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
```

(continues on next page)

(continued from previous page)

```
                                  dashboard
  --eth-address TEXT              Your MythX account's Ethereum address
  --password TEXT                 Your MythX account's password as set in the
                                  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.
```

Installation

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

To get started, simply run

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

Advanced Usage

For in-depth details on how to use the MythX CLI, alternative ways to install it, as well as helpful documentation about authentication, analyzing complex projects and more, head over to the MythX CLI documentation.

The MythX CLI project is MIT-licensed and contributions are always welcome! For more information on how to get involved, check out our Contribution guidelines.

Useful Links

Github repository	https://github.com/dmuhs/mythx-cli/
Documentation	https://mythx-cli.readthedocs.io/
Python Package Index	https://pypi.org/project/mythx-cli/
Current CI builds	https://travis-ci.org/dmuhs/mythx-cli/
Test coverage	https://coveralls.io/github/dmuhs/mythx-cli

Remix integration with MythX

MythX is available as a plugin for Remix, a popular web-based IDE for smart contract development and deployment, created and hosted by the Ethereum Foundation.

Setup

Note: These instructions will show the Remix interface that is current as of mid-2019. We recommend everyone use this interface.

Because Remix is a web-based interface, no local installation is required. However, MythX will need to be specifically activated from within the Remix Plugin Manager before use.

To activate MythX in Remix:

1. Click the *Plugins* icon (which resembles a plug).

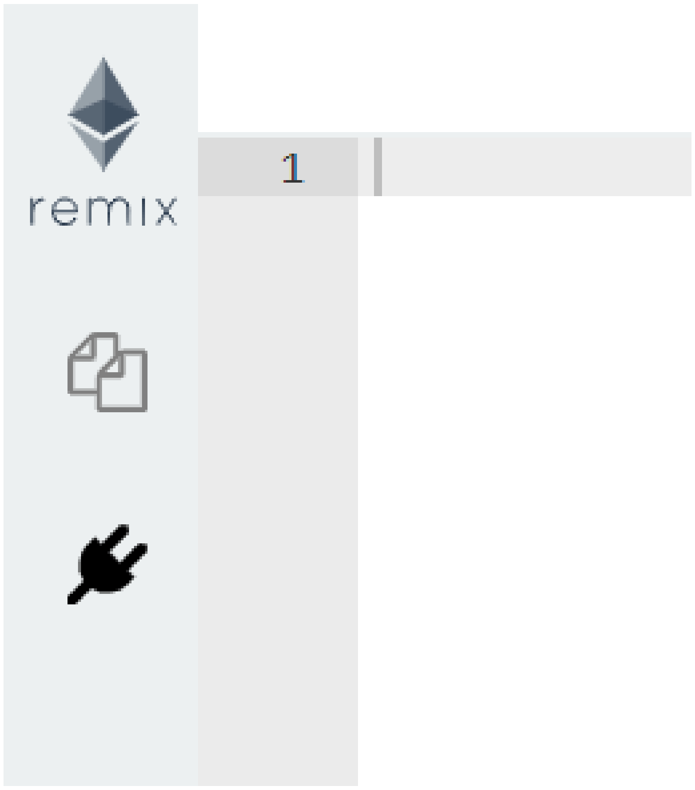
   Alternately, click the *See all Plugins* button under *Featured Plugins*.

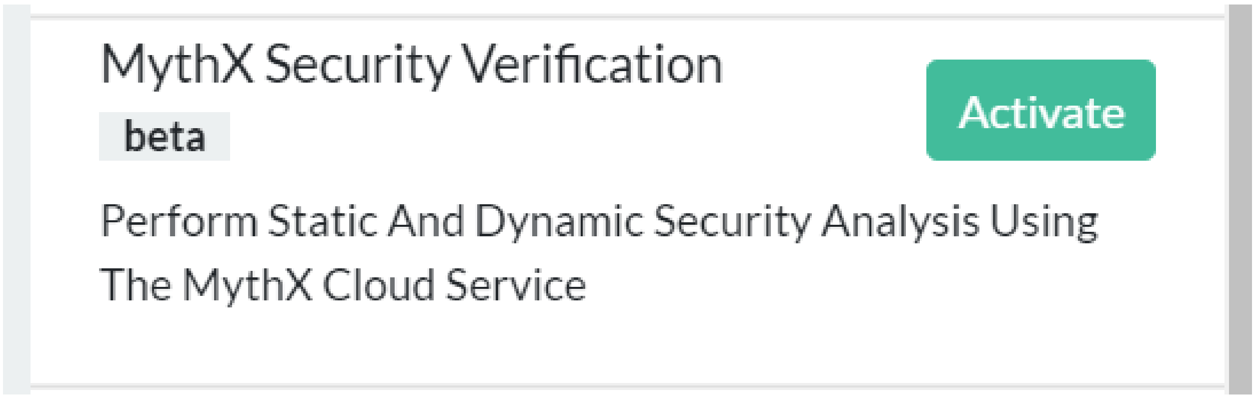
2. The full list of plugins for Remix will be displayed. Scroll down to the entry titled *MythX Security Verification* and click *Activate*.

   If done correctly, the plugin will be listed under *Active Modules* and the MythX icon will be shown in the sidebar.

3. *(Optional but recommended)* Click the MythX logo and enter your MythX credentials. This consists of your Ethereum address (also known as User ID) and the password supplied to you when you created your account at mythx.io. When done, click *Save*.

   Note: Without MythX credentials, you will be running in Trial Mode, which will only return a limited report of vulnerabilities. You can go to https://mythx.io to create a free account which will offer an unrestricted report.

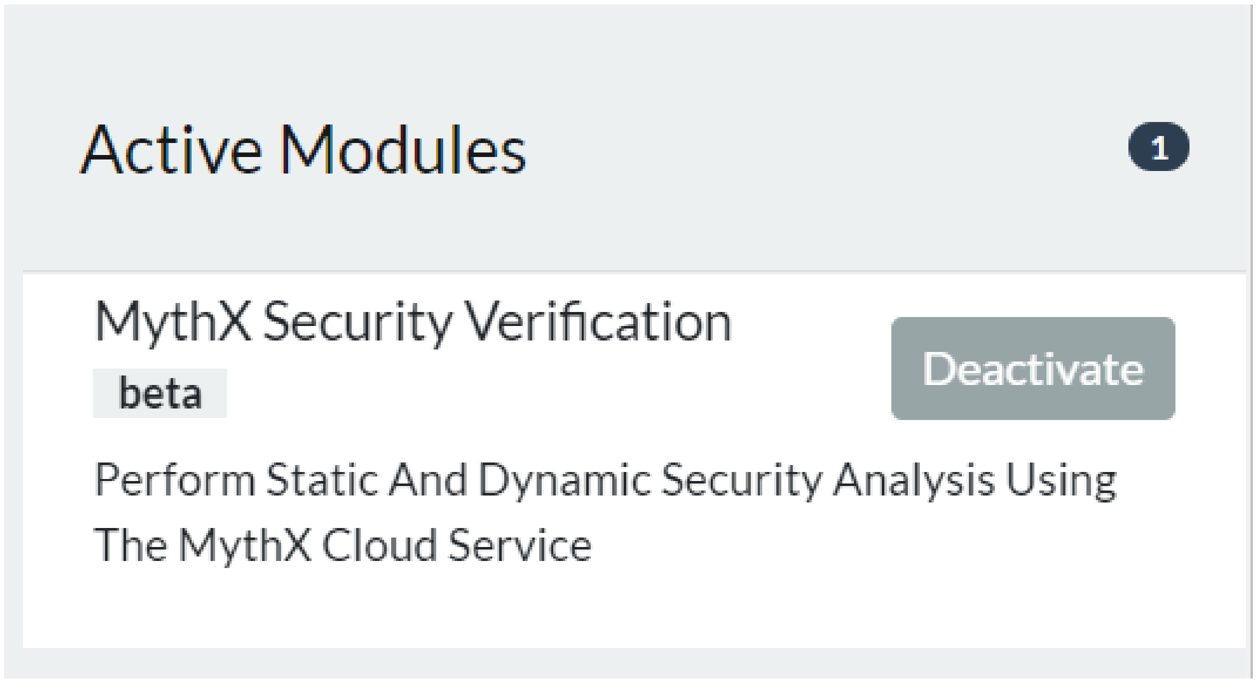



MythX Security Verification

beta

Activate

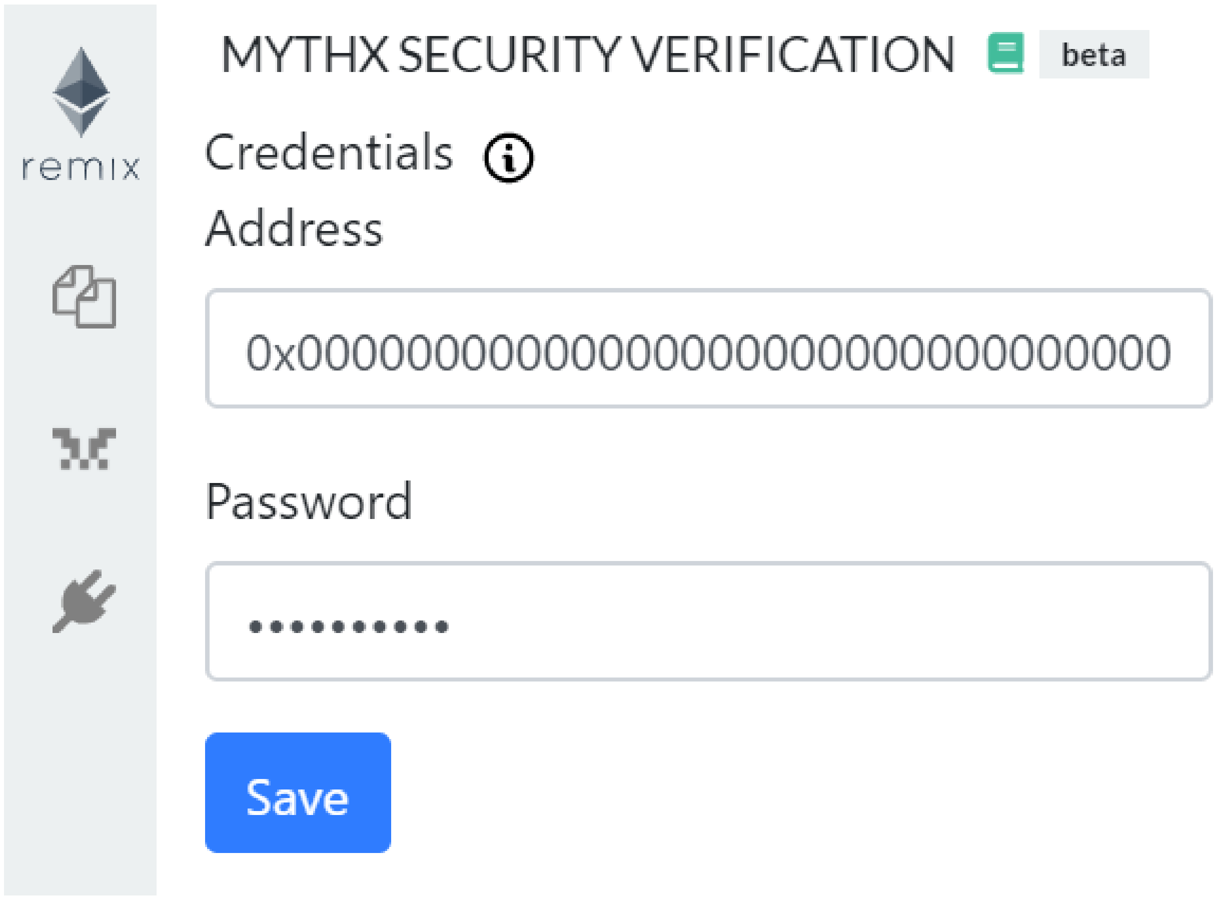
Perform Static And Dynamic Security Analysis Using The MythX Cloud Service



Active Modules 1

MythX Security Verification beta Deactivate

Perform Static And Dynamic Security Analysis Using The MythX Cloud Service



MYTHX SECURITY VERIFICATION beta

Credentials

Address

0x0000000000000000000000000000000000

Password

•••••••••••

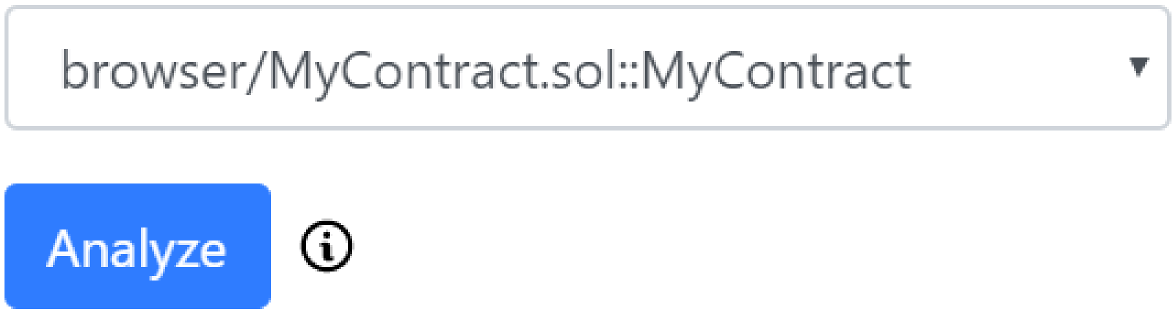
Save

Usage

You can perform a security analysis on any contract in any open file on Remix.

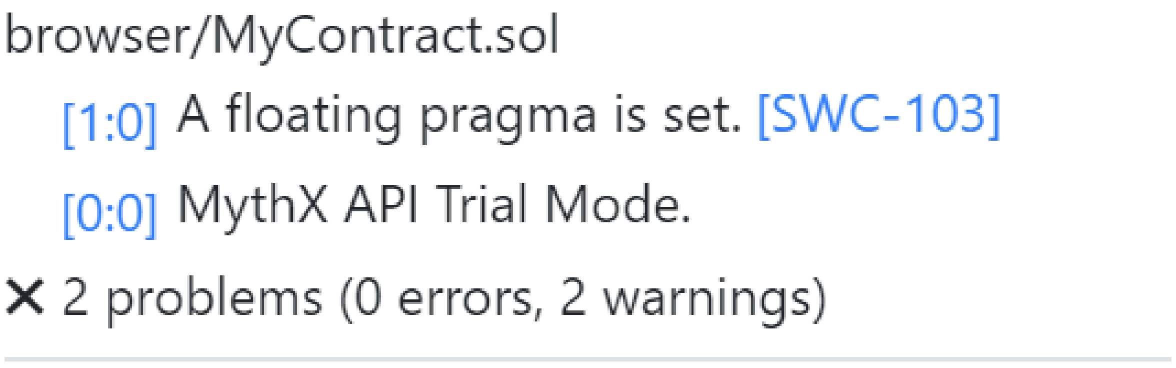
To perform an analysis:

1. Click the MythX logo on the sidebar to open the MythX control panel (if it isn't already open).
2. Below the credentials section, there will be a select box containing a list of all applicable contracts. Select the one you wish to analyze and click the *Analyze* button.



Note: The contract may need to be compiled first, depending on the current Remix settings. Make sure the *Solidity Compiler* plugin for Remix is activated in your project. You will have to click the Solidity icon and then click the *Compile* button for your contract.

3. The analysis may take a few minutes. When completed, a list of vulnerabilities will be displayed, along with a link to the SWC Registry for each vulnerability found.



```
browser/MyContract.sol
    [1:0] A floating pragma is set. [SWC-103]
    [0:0] MythX API Trial Mode.
✕ 2 problems (0 errors, 2 warnings)
```

Warning: If you are running in Trial Mode, you will see a response here saying so. This means that some vulnerabilities may not be shown in the output.

See also:

- Remix MythX plugin README (GitHub)

GuardRails Guide: Installation & Usage

GuardRails is a service that provides continuous security feedback for your GitHub repositories.

What is GuardRails?

GuardRails makes open-source security tools easily available in your GitHub pull requests. It has been tuned to keep the noise low and only report high-impact and relevant security issues.

Installing and configuring security tools, even for one repository, typically takes a lot of time and effort. GuardRails makes that process painless and rewarding for developers.

GuardRails can be installed across all your repositories in minutes. Once installed, GuardRails identifies security problems in your codebase and helps you fix them.

In addition, GuardRails leverages the power of MythX in order to scan for security vulnerabilities in your smart contracts.

Setup

To view to latest setup instructions for GuardRails, please see their Getting Started page.

Note: The free version of GuardRails only works with public repositories.

MythX comes built-in to GuardRails integration; **no special installation is required.**

Configuration

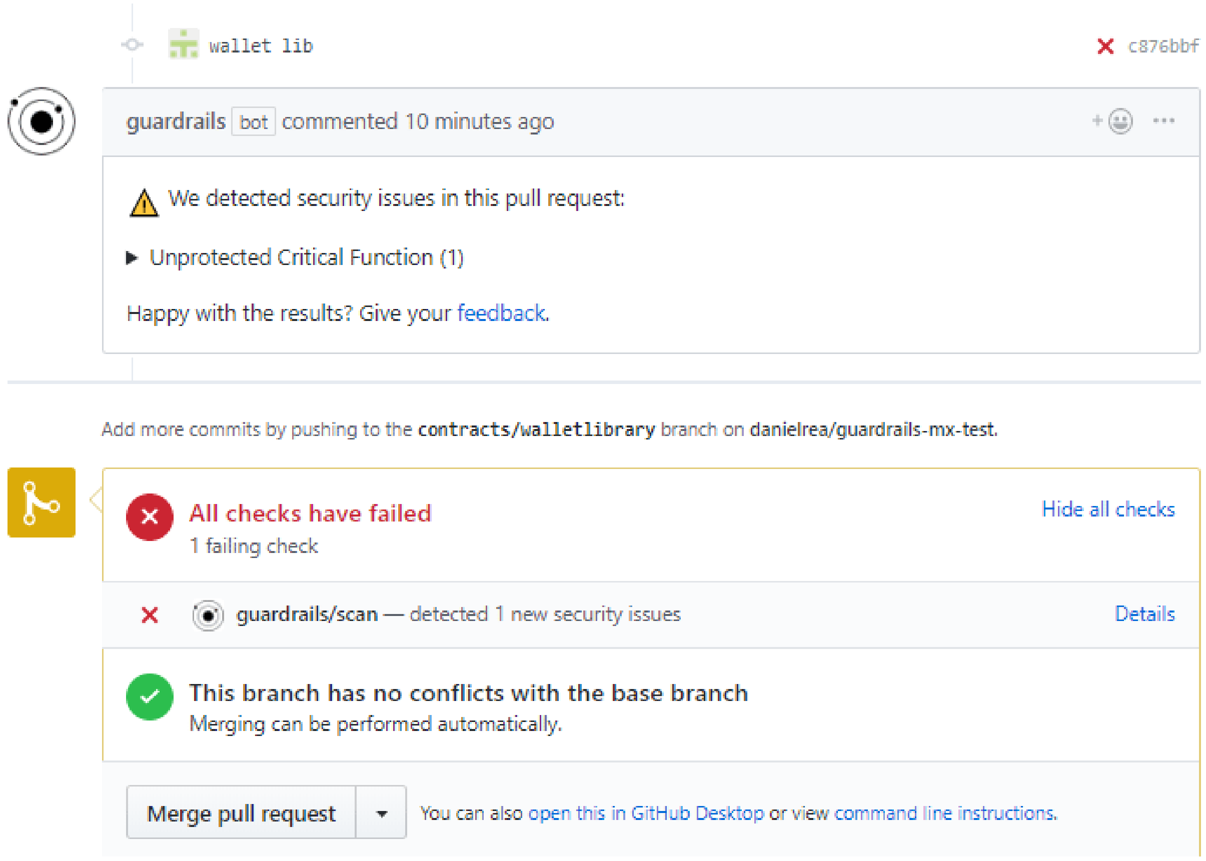
By default GuardRails will decide how to analyze based on the code tag on the repository. Solidity isn't always caught correctly, but you can set the *bundle* setting default to *solidity* in the Dashboard.

Note: View all GuardRails configuration options.

Scanning

Once GuardRails is properly configured, whenever a pull request is created on a branch, GuardRails will start a scan in the comments. This scan may take some time, especially if you have multiple languages or multiple smart contracts.

When completed, you will see the results of the scan in the pull request:



wallet lib c876bbf

guardrails bot commented 10 minutes ago

We detected security issues in this pull request:

▸ Unprotected Critical Function (1)

Happy with the results? Give your feedback.

Add more commits by pushing to the **contracts/walletlibrary** branch on danielrea/guardrails-mx-test.

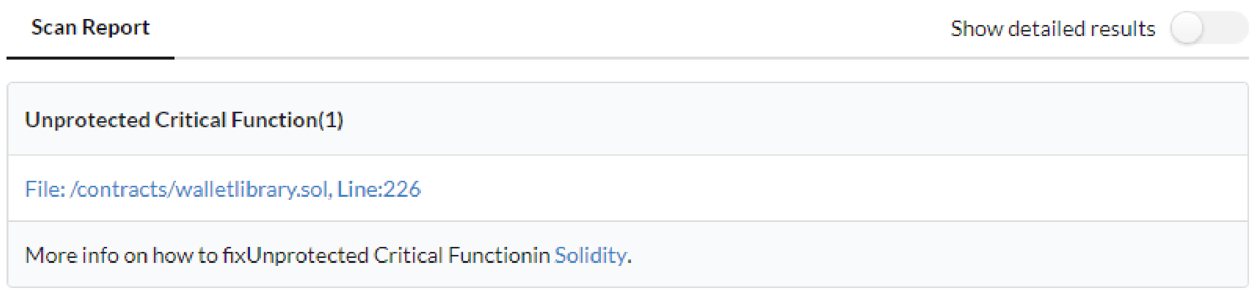
All checks have failed
1 failing check

guardrails/scan — detected 1 new security issues

This branch has no conflicts with the base branch
Merging can be performed automatically.

You can also open this in GitHub Desktop or view command line instructions.

You can see more detailed information about the issue and where it came from by clicking *Details* to see the scan report in the GuardRails dashboard:



Scan Report

Unprotected Critical Function(1)

File: /contracts/walletlibrary.sol, Line:226

More info on how to fixUnprotected Critical Functionin Solidity.

See also:

- GuardRails dashboard
- GuardRails configuration

Mythos Guide: Installation & Usage

Mythos helps you run scans right in your console, returning the output with source code mapping.

Install Mythos

You must have nodejs and npm installed on your system.

```
$ npm install @cleanunicorn/mythos
```

Usage

First of all you need to have an active account created on the MythX platform.

You need to specify your Ethereum address and the password that you generated on the platform.

I prefer to add them to the console as environment variables:

```
$ export MYTHX_ETH_ADDRESS='mythxEthAddress'
$ export MYTHX_PASSWORD='mythxPassword'
```

And then you can start the scan:

```
$ mythos analyze ./contract.sol Contract
```

But you can also specify them as flags `--mythxEthAddress=mythxEthAddress` and `--mythxPassword=mythxPassword` when running the tool.

Check the GitHub page and star the project, open an issue if you need more functionality or have some kind of problem.

It is a work in progress so any feedback is appreciated and your input will help shape the tool to fit your needs.

Brownie Guide: Installation & Usage

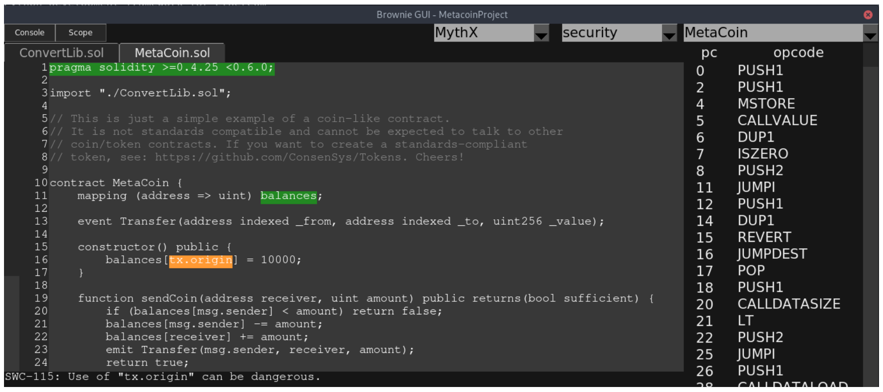
Brownie is a Python framework for Ethereum smart contract testing, interaction and deployment.

- **Testing**: Unit test your project with `pytest`, and evaluate test coverage through stack trace analysis. We make *no promises*.
- **Debugging**: Get detailed information when a transaction reverts, to help you locate and solve the issue quickly.
- **Interaction**: Write scripts or use the console to interact with your contracts on the main-net, or for quick testing in a local environment.
- **Deployment**: Automate the deployment of many contracts onto the blockchain, and any transactions needed to initialize or integrate the contracts.

Scanning for Security Vulnerabilities

To prevent vulnerabilities from being introduced to the code base, Brownie includes a plugin that integrates automated security scans using the MythX analysis API. Simply run `brownie analyze` on your compiled project directory. This will send the compiled build artifacts to MythX for analysis. By default no login is required and the analysis is going to be executed as a trial user. To access more vulnerability information, register for free on the MythX website and pass your login data via environment variables or command line arguments.

Once the analysis is done, the vulnerabilities are stored in the `reports/` directory. With `brownie analyze --gui` the GUI can be started automatically once the analysis has finished.



```
pragma solidity >=0.4.25 <0.6.0;

import "./ConvertLib.sol";

// This is just a simple example of a coin-like contract.
// It is not standards compatible and cannot be expected to talk to other
// coin/token contracts. If you want to create a standards-compliant
// token, see: https://github.com/ConsenSys/Tokens. Cheers!

contract MetaCoin {
    mapping (address => uint) balances;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);

    constructor() public {
        balances[tx.origin] = 10000;
    }

    function sendCoin(address receiver, uint amount) public returns(bool sufficient) {
        if (balances[msg.sender] < amount) return false;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Transfer(msg.sender, receiver, amount);
        return true;
SWC-115: Use of "tx.origin" can be dangerous.
```

```
 pc     opcode
0     PUSH1
2     PUSH1
4     MSTORE
5     CALLVALUE
6     DUP1
7     ISZERO
8     PUSH2
11    JUMPI
12    PUSH1
14    DUP1
15    REVERT
16    JUMPDEST
17    POP
18    PUSH1
20    CALLDATASIZE
21    LT
22    PUSH2
25    JUMPI
26    PUSH1
```

Note: Head to the Brownie documentation to get more information on how to use it in your smart contract development workflow!

1.2.2 New tools wanted

Are you working on an integration with an IDE, a web interface, or something else that's super useful? If you are building on MythX, we want to know about it!

Please join our community on Discord and reach out to the team for more information.

Note: We plan to offer a revenue sharing system for qualifying tools. More details will be announced soon.

1.3 Building smart contract security tools using MythX

This section of the guide is aimed at developers who want build security tools using the MythX API.

1.3.1 Contents

1.3.2 What Tools to Build?

Using MythX API, you can build security tools that find bugs in smart contracts for Ethereum or compatible blockchains such as Tron and Quorum. Currently, we support Solidity code and EVM bytecode, but we are working on support for additional bytecode formats (e.g. eWASM) and languages.

Some examples for potential MythX tools are:

- Plugins for dapp development environments, such as Truffle, Remix and Embark;
- Plugins for code editors (Sublime Text, Atom, vim,. . .);
- Apps and CI hooks for code repositories (GitHub, Gitlab);
- Command-line tools for security auditors;
- Integrations into dapp browsers and wallets;

- Standalone web interfaces.

Feel free however to experiment with the API in any way you want!

1.3.3 API Specification

Besides whatever you might find in this guide, the MythX OpenAPI Spec is the ultimate authority. Beyond that be dragons.

1.3.4 Language Bindings

In most cases you'll want to use an existing client library that abstracts the low-level details of interacting with MythX.

MythXJS Guide: Installation & Usage

MythXJS is a JavaScript library for interacting with the MythX API.

Installation

MythXJS can be installed via `npm`:

```
npm install mythxjs
```

Examples

Creating a new instance of the library using ES6 modules

```
import { Client } from 'mythxjs'

const mythx = new Client('0x0000000000000000000000000000000000000000', 'trial',
↪'testTool');
```

Performing a `login` request

```
// Logs in and returns an object containing access and refresh token
const tokens = await mythx.login()
```

Submitting an analysis using bytecode only

```
const bytecode = '0xfe'
await mythx.submitBytecode(bytecode)
```

Getting a list of detected issues

```
await mythx.getDetectedIssues('1111-2222-3333-4444')
```

Logging in with MetaMask

In order to keep MythXJS as lean as possible it does not handle MetaMask integration directly. Instead it provides two methods:

- `getChallenge()`
- `loginWithSignature()`

With these methods, you can handle the MetaMask integration as you prefer. You can also work with your preferred version of *web3*.

Here is an example using React and `web3@1.0.0-beta.37`:

```
const handleSignMessage = (account, data) => {
    try {
        return new Promise((resolve) => {
            const {value} = data.message
            if (!account) {
              console.error('no-account')
            }
              const params = [account, JSON.stringify(data)]
              web3.currentProvider.send(
                          { method: 'eth_signTypedData_v3', params, from: account },
                          (err, result) => {
                            if (err || result.error) {
                              console.error('Error with handling signature.', err)
                            }
                            resolve(value + '.' + result.result)
                          }
                    )
            }).catch((error) => {
              console.error(error)
            })
    } catch(err) {
        console.error(err)
    }
}

const loginWithMM = async () => {
    const accounts = await web3.eth.getAccounts();
    const account = accounts[0]

    const data = await mythx.getChallenge(account.toLowerCase())

    handleSignMessage(account, data).then(
        async (message) => {
            // Returns set of tokens
            const result = await mythx.loginWithSignature(message)
            console.log(result, 'ress')
        }
    ).catch(err => console.error(err))
}
```

See also:

- MythXJS documentation (GitHub)
- Source (GitHub)
- OpenAPI spec (MythX)

PythX Guide: Installation & Usage

PythX is a Python library for the MythX API.

Its goal is to facilitate development of use cases that go beyond bare API interaction by making it easy for developers to hook into high-level interfaces. When writing a complex tool around MythX you shouldn't have to worry about refreshing your JWT tokens regularly or basic consistency checking of your requests. PythX takes these tedious tasks away from you - unless you explicitly tell it not to.

Installation

PythX runs on Python 3.5+.

To get started, simply run

```
$ pip3 install pythx
```

Alternatively, clone the repository and run

```
$ pip3 install .
```

Or directly through Python's `setuptools`:

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

Example

PythX aims to provide an easy-to-use interface to the official MythX API. Its goal is to turbocharge tool development and make it easy to deal with even complex use cases.

```
from pythx import Client


# login as free trial user
c = Client(eth_address="0x0000000000000000000000000000000000000000", password="trial")

# submit bytecode, source files, their AST and more!
resp = c.analyze(bytecode="0xfe")

# wait for the analysis to finish
while not c.analysis_ready(resp.uuid):
    time.sleep(1)

# have all your security report data at your fingertips
for issue in c.report(resp.uuid):
    print(issue.swc_title or "Undefined", "-", issue.description_short)

```

(continues on next page)

(continued from previous page)

```
# Output:
# Assert Violation - A reachable exception has been detected.
# Undefined - MythX API trial mode.
```

The PythX CLI has now become the MythX CLI!

Originally, the PythX CLI was a proof of concept to display to interested developers what can be done using the library. The interest in the CLI grew so large that a lot of developers contacted me and asked for support and new features.

This is the PSA that **I will no longer maintain the PythX CLI**. But wait! There's more!

Because a PoC is not exactly what you would call future-proof and maintainable software, I have decided to do a complete revamp. It is called *mythx-cli* and incorporates all feature requests I have gotten so far. Check it out *here* and let me know what you think!

Enjoy! :)

Check out the repository, read through our documentation, and hit us up on Discord if you have any issues, feedback, or just want to say hello. :)

Armlet Guide: Installation & Usage

Armlet is a thin wrapper around the MythX API written in JavaScript which simplifies interaction with MythX. For example, the library wraps API analysis requests into a promise.

Warning: Please note that as of August 2019 the Armlet library has been **deprecated** in favour of the newer and more advanced `mythxjs` library whose documentation can be found *here*.

Installation

Just as with any nodejs package, install with:

```
$ npm install armlet
```

Example

Here is a small example of how you might use this client. For demonstration purposes, we'll set the credentials created on the MythX, you can use either the Ethereum address or email used during registration and the password you created:

```
$ export MYTHX_PASSWORD=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
$ export MYTHX_ETH_ADDRESS=0x.............
```

Then to get the MythX analysis results with the promise returned by the exposed function:

```
const armlet = require('armlet')
const client = new armlet.Client(
  {
      ethAddress: process.env.MYTHX_ETH_ADDRESS,
      password: process.env.MYTHX_PASSWORD,
```

(continues on next page)

(continued from previous page)

```
  })
const data = {
  contractName: 'TestMe',
  abi: [
    {
      constant: false,
      inputs: [
        {
          name: 'first_input',
          type: 'uint256',
        },
      ],
      name: 'lol',
      outputs: [
        {
          name: '',
          type: 'uint256',
        },
      ],
      payable: false,
      stateMutability: 'nonpayable',
      type: 'function',
    },
  ],
  bytecode: '0xf6...',
  deployedBytecode: '0xf6...',
  sourceMap: '25:78:1:-;;;;8:9:-1;5:2;;;30:1;27;20:12;5:2;25:78:1;;;;;;;',
  deployedSourceMap: '25:78:1:-;;;;8:9:-1;5:2;;;30:1;27;20:12;5:2;25:78:1;;;;;;;',
  sourceList: [
    'basecontract.sol',
    'maincontract.sol',
  ],
  sources: {
    'basecontract.sol': {
      source: '[... source code ...]',
    },
    'maincontract.sol': {
      source: '[... source code ...]',
    },
  },
  analysisMode: 'full',
};
client.analyze({data})
  .then(issues => {
    console.log(issues)
  }).catch(err => {
    console.log(err)
  })
```

You can also specify the timeout in milliseconds to wait for the analysis to be done (the default is 10 seconds). For instance, to wait up to 5 seconds:

```
client.analyze({data}, 5000)
  .then(issues => {
    console.log(issues)
  }).catch(err => {
```

(continues on next page)

(continued from previous page)

```
    console.log(err)
  })
```

See also:

- npm package
- The GitHub project
- MythX API spec
- MythX JS SDK

1.3.5 API Walkthrough

The MythX API curl scripts demonstrate interaction with MythX API at the most basic level. The scripts will show you the HTTP requests that get sent as well with the JSON output returned as the result of each request.

The process for analyzing a smart contract works as follows:

- Authenticate with Ethereum address and password to retrieve an access token;
- Submit a contract for analysis, creating a job run with a UUID;
- See the status of job using the UUID of a previously submitted analysis;
- Get the results of a previously finished analysis using the UUID.

Let's run through a basic example. Make sure that curl is installed and clone the GitHub repository to get started:

```
$ git clone https://github.com/rocky/mythx-api-curl
$ cd mythx-api-curl
```

To verify that you can connect to the API, run the api-version script:

```
$ ./api-version.sh
  Running: curl -v GET https://api.mythx.io/v1/version
  curl completed sucessfully. See /tmp/curljs.err53890 for verbose logs.
  Processed output from /tmp/curljs.out53890 follows...
  {
    "api": "v1.3.3",
    "maru": "0.3.4",
    "mythril": "0.20.0",
    "harvey": "0.0.7"
  }
```

Authentication

MythX uses JSON Web Token (JWT) authentication. In this authentication scheme, the user submits their username and password to the login endpoint. Upon successful login the server returns the following:

- A timed access token. This token is needs to be sent by the client with every request to access the API.
- A refresh token that can be used to request a new access token once the current one times out.

To execute the login process on the shell, set the *MYTHX_PASSWORD* to and *MYTHX_ETH_ADDRESS* environment variables to your Ethereum address and API password:

```
$ export MYTHX_PASSWORD=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
$ export MYTHX_ETH_ADDRESS=0x.............
```

Then run the login script as follows:

```
$ . ./login.sh
Successfully logged into MythX
```

If authentication succeeds, the login script will set the *MYTHX_ACCESS_TOKEN* and *MYTHX_REFRESH_TOKEN* environment variables.

```
$ echo $MYTHX_ACCESS_TOKEN
  eyJhb(...)m38eVH7P2TRjM
$ echo $MYTHX_REFRESH_TOKEN
  eyJhd(...)Ni00ODg5LTRjM
```

With these variables set you can submit security analysis jobs.

Submitting an Analysis Job

To launch a security analysis job the client needs to submit the Solidity source code and EVM bytecode via a POST request to the analyses endpoint. The input format is similar to the JSON output generated by solc and Truffle. Here is an example:

```
{
   "clientToolName": "mythx-api-curl",
   "data":
   {
      "contractName" : "Token"
      "deployedBytecode" : "6080605482019055600192(...)",
      "deployedSourceMap" : "25:503:0:-;;;;6202:0;(...)",
      "bytecode" : "6080604052348015610 01(...)",
      "sourceMap" : "25:503:0:-;;;;8:9:-1(...)",
      "analysisMode" : "quick",
      "sourceList" : [
         "token.sol"
      ],
      "sources" : {
         "Token.sol" : {
            "source" : "pragma solidity ^0.5.0;\n\ncontract Token {\n\n ␣
↪mapping(address => uint) balances;\n  uint public totalSupply;\n\n ␣
↪constructor(uint _initialSupply) public {\n    balances[msg.sender] = totalSupply =␣
↪_initialSupply;\n  }\n\n  function transfer(address _to, uint _value) public␣
↪returns (bool) {\n    require(balances[msg.sender] - _value >= 0);\n    ␣
↪balances[msg.sender] -= _value;\n    balances[_to] += _value;\n    return true;\n  }
↪\n\n  function balanceOf(address _owner) public view returns (uint balance) {\n    ␣
↪return balances[_owner];\n  }\n}\n"
         }
      },
      "mainSource: "Token.sol"
   }
}
```

The input JSON contains the following fields:

- clientToolName: A name that uniquely identifies your MythX tool.

- data: A dictionary containing the data of the contract to be analyzed
- contractName: The name of the main contract class to be analyzed.
- deployedBytecode: The runtime bytecode of the contract.
- deployedSourceMap: The source mapping generated by solc for the runtime bytecode.
- bytecode: The creation bytecode of the contract.
- sourceMap: The source mapping generated buy solc for the creation bytecode.
- analysisMode: The type of analysis ("quick" of "full").
- sourceList: A list of source files referred to by the source maps.
- sources: A dictionary containing the original source code of each code file.
- mainSource: The filename of the Solidity file containing the main contract class.

Note that both source code and bytecode must be submitted to receive complete results.

The *analysisMode* field is used to select the type of analysis. Currently two modes are supported:

- quick: Perform static analysis and shallow symbolic analysis and input fuzzing. Returns a result within 90 seconds.
- full: Perform static analysis and deep symbolic analysis and input fuzzing. May run for up to 2 hours.

About sourceList and sources

The *sourceList* must contain all files that were used for compiling the contract - i.e. the main Solidity file as well as imports (and imports of imports, etc.). Each filename listed in *sourceList* must have a matching entry in the *sources* dict. Note that the order of the files in the *sourceList* field is crucial for receiving correct issue locations and should match the order that the compiler used to build the source maps.

Assembling the *sourceList* correctly is currenly difficult for complex projects - we're working on an easier way that allows ASTs to be submitted instead. In the meantime, you can check out the code of Sabre which implements support for imports.

If successful, the API returns a UUID that can then be used to retrieve the status and results of the analysis.

```
$ ./analyses.sh sample-json/Token.json
Issuing HTTP POST http://api.mythx.io/v1/analyses
  (with MYTHX_API_KEY and EVM bytecode)
curl completed successfully. Output follows...
HTTP/1.1 200 OK
{
  "result": "Queued",
  "uuid": "bf9fe267-d322-4641-aae2-a89e62f40770"
}
```

Polling the API to Obtain Job Status

You can determined the status of your analysis by sending a GET request to */analyses/<UUID>*:

```
$ ./analyses-status.sh bf9fe267-d322-4641-aae2-a89e62f40770
Issuing HTTP GET https://api.mythx.io/v1/analyses/bf9fe267-d322-4641-aae2-a89e62f40770
  (with MYTHX_API_KEY)
curl completed successfully. Output follows...
HTTP/1.1 200 OK
{
```

(continues on next page)

(continued from previous page)

```
  "result": "Finished",
  "uuid": "bf9fe267-d322-4641-aae2-a89e62f40770"
}
```

- Queued: Your job is in the queue but has not been started yet. Note that you can queue up to 10 jobs at a time.
- In Progress: Your job is currently running. In quick mode, the job will remain in running state for approximately one minute. In full mode, the analysis may take up to two hours depending on the complexity of the code.
- Finished: The job has been completed successfully and the results can be retrieved.

Estimating Analysis Duration

How long an analysis takes to complete depends on the analysis mode ("quick" or "full") and overall API load.

Analysis mode

- In "quick" mode, the analysis finishes takes between 30 seconds 5 minute to complete after entering the "in progress" state.
- In "full" mode, the analysis may take up to 2-5 hours. Note that "full" mode is still highly experimental.

Overall API load

We aim to process all incoming requests immediately. However, in times of high load, our jobs might remain in the queue for some time before a worker becomes available.

Polling Recommendations

In order to help users keep below their rate limits (and not overload the API too much), we recommend implementing the following polling algorithm:

- Set an initial delay before sending the first poll after submitting an analysis. Even in quick mode, results rarely are ready in less than 45 seconds. In full mode, results will usually only be ready after 2 hours or more.
- After the initial delay has passed, poll in reasonable regular intervals (e.g. every 8 seconds in "quick" mode and once per minute in "full" mode). Alternatively, start with a low timeout that increases geometrically over time.
- Also set an overall timeout. If a job has been in "in progress" state for more than 12 hours, it is reasonable to assume that there's a problem on the API side and return an error message to the user. Include the job UUID in the error message. API bugs can be submitted via one of our support channels

Obtaining Analysis Results

Once the job is in "finished:" state the result can be obtained using as follows:

```
$ ./analyses-results.sh 8a15c859-3245-4d73-bdef-77bf53c5b9b2
Issuing HTTP GET https://api.mythx.io/v1/analyses/8a15c859-3245-4d73-bdef-
↪77bf53c5b9b2/issues
  (with MYTHX_ACCESS_TOKEN)

curl completed sucessfully. See /tmp/curljs.err54326 for verbose logs.
Processed output from /tmp/curljs.out54326 follows...
[
  {
```

(continues on next page)

(continued from previous page)

```
    "issues": [
      {
        "swcID": "SWC-101",
        "swcTitle": "Integer Overflow and Underflow",
        "description": {
          "head": "The binary subtraction can underflow.",
          "tail": "The operands of the subtraction operation are not sufficiently␣
↪constrained. The subtraction could therefore result in an integer underflow.␣
↪Prevent the underflow by checking inputs or ensure sure that the underflow is␣
↪caught by an assertion."
        },
        "severity": "High",
        "locations": [
          {
            "sourceMap": "296:29:0"
          }
        ],
        "extra": {}
      },
    (...)
    ],
    "sourceType": "solidity-file",
    "sourceFormat": "text",
    "sourceList": [
      "token.sol"
    ],
    "meta": {
      "coveredInstructions": 213,
      "coveredPaths": 10,
      "error": "",
      "selected_compiler": "0.5.0",
      "warning": []
    }
  }
]
```

The output contains a list of issues with title, short description and long description and source mappings, as well as additional information:

- The *swcID* field contains a reference to the Smart Contract Weakness Classification Registry.
- The *description* field describes the found issue in two entries.
  - *head* is the summary of the found issue.
  - *tail* contains the details on what causes the issue well as any possible remediation.
- The *locations* list contain one or more solc-style *sourceMap* entries that contain bytecode offsets into the provided source code files. This source mapping format is described in the solc documentation. The notation *s:l:f* where *s* is the byte-offset to the start of the range in the source file, *l* is the length of the source range in bytes and *f* is the index of the source code file in the *sourceList*.
- The *meta* field contains meta information about the analysis run.

We recommend that users submit both bytecode and source code to obtain a full analysis. If only the creation bytecode is given, and not the source code, MythX will return a result like the following:

```
[
  {
```

(continues on next page)

(continued from previous page)

```
    "issues": [
      (...)
    ],
    "sourceType": "raw-bytecode",
    "sourceFormat": "evm-byzantium-bytecode",
    "sourceList": [
      "0x98d243270da47b324377a628b45e42110ba7520280b8e13ac94aad07501fbb2e"
    ],
    "meta": {
      "coveredInstructions": 111,
      "coveredPaths": 5
    },
  }
]
```

In this instance, `sourceList: [0x98...]` refers to the Keccak256 hash of the runtime bytecode within which the issue(s) were found.

1.3.6 API Details

Token Expiration Times

Validity times for the JSON Web Tokens are set as follows:

- Access tokens are valid for 10 minutes;
- Refresh tokens are valid for 4 weeks.

Rate Limits

API rate limits need to be considered when designing MythX tools as sending excessive requests may cause API errors. Currently the following rate limits apply:

- The client can submit up to **2 requests per second.**
- The API can queue up to **10 analysis jobs** per client. However, a maximum of **four workers** will be allocated to a single client. It is therefore recommended to limit the number of parallel analysis jobs to four.
- The client can perform up to **10,000 API requests within 24 hours.**

1.3.7 Compiler Settings

It is recommended to activate optimization when compiling source code for analysis. This reduces the complexity of the bytecode, allowing for better performance in the fuzzing and symbolic analysis steps and increasing code coverage.

For example, when using *solcjs*, add the following to the compiler settings:

```
settings: {
    (...)
    optimizer: {
      enabled: true,
      runs: 200
    }
}
```

1.3.8 Example Code

Sabre is a minimal MythX CLI written in JavaScript. It shows how to compile a Solidity file using solc-js and submit the compiler output to MythX using the armlet JavaScript library.

See also:

- Sabre - Minimum Viable MythX Client

1.3.9 Revenue Sharing Program

Once paid subscription plans for MythX go live, we'll share back some of the revenue from subscription fees back to tool builders.

The amount revenue share you receive will depend on the number of daily active paying users of your tool. In your tool, set the clientToolName field to a unique name of your choice when submitting analysis requests. That way we can keep track of usage statistic.

More details about this program will be announced during the beta in 2019.

1.4 MythX support

There are multiple ways to get in touch with us, whether you want to get help using MythX, you want to develop an integration, or you want to give feedback.

1.4.1 Users

You can connect with the MythX team on Discord at #mythx-support.

To get support for third-party tools it is usually best to communicate directly with the tool developers. For example:

- If you run into an issue with the *MythX plugin for Truffle*, create an issue in its GitHub repository.
- For issues with the *MythX Remix plugin*, you can create an issue in the Remix plugin GitHub repository as well.

For direct support, please go to mythx.io and click the *Contact* link.

1.4.2 Tool developers

If you're building tools on our MythX API you can reach out to our team on the following channels:

- Discord: #mythx-tool-developers - General chat for MythX builders
- Discord: #api-bugs - For reporting bugs in the MythX service

For direct support, please go to mythx.io and click the *Contact* link.