Mythril Documentation

Release v0.22.21

Bernhard Mueller

Jun 10, 2021

Table of Contents:

CHAPTER 1

What is Mythril?

Mythril is a security analysis tool for Ethereum smart contracts. It was introduced at HITBSecConf 2018.

Mythril detects a range of security issues, including integer underflows, owner-overwrite-to-Ether-withdrawal, and others. Note that Mythril is targeted at finding common vulnerabilities, and is not able to discover issues in the business logic of an application. Furthermore, Mythril and symbolic executors are generally unsound, as they are often unable to explore all possible states of a program.

CHAPTER 2

Installation and Setup

Mythril can be setup using different methods.

2.1 PyPI on Mac OS

```
brew update
brew upgrade
brew tap ethereum/ethereum
brew install leveldb
brew install solidity
pip3 install mythril
```

2.2 PyPI on Ubuntu

```
# Update
sudo apt update

# Install solc
sudo apt install software-properties-common
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt install solc

# Install libssl-dev, python3-dev, and python3-pip
sudo apt install libssl-dev python3-dev python3-pip

# Install mythril
pip3 install mythril
myth --version
```

2.3 Docker

All Mythril releases, starting from v0.18.3, are published to DockerHub as Docker images under the `mythril/myth` name.

After installing Docker CE:

```
# Pull the latest release of mythril/myth
$ docker pull mythril/myth
```

Use `docker run mythril/myth` the same way you would use the `myth` command

```
docker run mythril/myth --help
docker run mythril/myth disassemble -c "0x6060"
```

To pass a file from your host machine to the dockerized Mythril, you must mount its containing folder to the container properly. For `contract.sol` in the current working directory, do:

```
docker run -v $(pwd):/tmp mythril/myth analyze /tmp/contract.sol
```

CHAPTER 3

Security Analysis

Run `myth analyze` with one of the input options described below will run the analysis modules in the /analysis/modules directory.

3.1 Analyzing Solidity Code

In order to work with Solidity source code files, the solc command line compiler needs to be installed and in PATH. You can then provide the source file(s) as positional arguments.

```
$ myth analyze ether_send.sol
==== Unprotected Ether Withdrawal ====
SWC ID: 105
Severity: High
Contract: Crowdfunding
Function name: withdrawfunds()
PC address: 730
Estimated Gas Usage: 1132 - 1743
Anyone can withdraw ETH from the contract account.
Arbitrary senders other than the contract creator can withdraw ETH from the contract␣
↪account without previously having sent an equivalent amount of ETH to it. This is␣
↪likely to be a vulnerability.
--------------------
In file: tests/testdata/input_contracts/ether_send.sol:21

msg.sender.transfer(address(this).balance)

--------------------
```

If an input file contains multiple contract definitions, Mythril analyzes the *last* bytecode output produced by solc. You can override this by specifying the contract name explicitly:

```
myth analyze OmiseGo.sol:OMGToken
```

3.1.1 Specifying Solc Versions

You can specify a version of the solidity compiler to be used with `--solv <version number>`. Please be aware that this uses py-solc and will only work on Linux and macOS. It will check the version of solc in your path, and if this is not what is specified, it will download binaries on Linux or try to compile from source on macOS.

3.1.2 Output Formats

By default, analysis results are printed to the terminal in text format. You can change the output format with the `-o` argument:

```
myth analyze underflow.sol -o jsonv2
```

Available formats are `text`, `markdown`, `json`, and `jsonv2`. For integration with other tools, `jsonv2` is generally preferred over `json` because it is consistent with other MythX tools.

3.2 Analyzing On-Chain Contracts

When analyzing contracts on the blockchain, Mythril will by default attempt to query INFURA. You can use the built-in INFURA support or manually configure the RPC settings with the `--rpc` argument.

`--rpc ganache`	Connect to local Ganache
`--rpc infura-[netname] --infura-id <ID>`	Connect to mainnet, rinkeby, kovan, or ropsten.
`--rpc host:port`	Connect to custom rpc
`--rpctls <True/False>`	RPC connection over TLS (default: False)

To specify a contract address, use `-a <address>`

Analyze mainnet contract via INFURA:

```
myth analyze -a 0x5c436ff914c458983414019195e0f4ecbef9e6dd --infura-id <ID>
```

You can also use the environment variable *INFURA_ID* instead of the cmd line argument or set it in ~/.mythril/config.ini. Adding the `-l` flag will cause mythril to automatically retrieve dependencies, such as dynamically linked library contracts:

```
myth -v4 analyze -l -a 0xEbFD99838cb0c132016B9E117563CB41f2B02264 --infura-id <ID>
```

3.3 Speed vs. Coverage

The execution timeout can be specified with the `--execution-timeout <seconds>` argument. When the timeout is reached, mythril will stop analysis and print out all currently found issues.

The maximum recursion depth for the symbolic execution engine can be controlled with the `--max-depth` argument. The default value is 22. Lowering this value will decrease the number of explored states and analysis time, while increasing this number will increase the number of explored states and increase analysis time. For some contracts, it helps to fine tune this number to get the best analysis results. -

CHAPTER 4

Analysis Modules

Mythril's detection capabilities are written in modules in the /analysis/modules directory.

4.1 Modules

4.1.1 Delegate Call To Untrusted Contract

The delegatecall module detects SWC-112 (DELEGATECALL to Untrusted Callee).

4.1.2 Dependence on Predictable Variables

The predictable variables module detects SWC-120 (Weak Randomness) and SWC-116 (Timestamp Dependence).

4.1.3 Deprecated Opcodes

The deprecated opcodes module detects SWC-111 (Use of Deprecated Functions).

4.1.4 Ether Thief

The Ether Thief module detects SWC-105 (Unprotected Ether Withdrawal).

4.1.5 Exceptions

The exceptions module detects SWC-110 (Assert Violation).

4.1.6 External Calls

The external calls module warns about SWC-117 (Reentrancy) by detecting calls to external contracts.

4.1.7 Integer

The integer module detects SWC-101 (Integer Overflow and Underflow).

4.1.8 Multiple Sends

The multiple sends module detects SWC-113 (Denial of Service with Failed Call) by checking for multiple calls or sends in a single transaction.

4.1.9 Suicide

The suicide module detects SWC-106 (Unprotected SELFDESTRUCT).

4.1.10 State Change External Calls

The state change external calls module detects SWC-107 (Reentrancy) by detecting state change after calls to an external contract.

4.1.11 Unchecked Retval

The unchecked retval module detects SWC-104 (Unchecked Call Return Value).

4.1.12 User Supplied assertion

The user supplied assertion module detects SWC-110 (Assert Violation) for user-supplied assertions. User supplied assertions should be log messages of the form: `emit AssertionFailed(string)`.

4.1.13 Arbitrary Storage Write

The arbitrary storage write module detects SWC-124 (Write to Arbitrary Storage Location).

4.1.14 Arbitrary Jump

The arbitrary jump module detects SWC-127 (Arbitrary Jump with Function Type Variable).

4.2 Creating a Module

Create a module in the `analysis/modules` directory, and create an instance of a class that inherits `DetectionModule` named `detector`. Take a look at the suicide module as an example.

CHAPTER 5

MythX Analysis

Run `myth pro` with one of the input options described below will run a MythX analysis on the desired input. This includes a run of Mythril, the fuzzer Harvey, and the static analysis engine Maru and has some false-positive filtering only possible by combining the tool capabilities.

5.1 Authentication

In order to authenticate with the MythX API, set the environment variables `MYTHX_PASSWORD` and `MYTHX_ETH_ADDRESS`.

```
$ export MYTHX_ETH_ADDRESS='0x0000000000000000000000000000000000000000'
$ export MYTHX_PASSWORD='password'
```

5.2 Analyzing Solidity Code

The input format is the same as a regular Mythril analysis.

```
$ myth pro ether_send.sol
==== Unprotected Ether Withdrawal ====
SWC ID: 105
Severity: High
Contract: Crowdfunding
Function name: withdrawfunds()
PC address: 730
Anyone can withdraw ETH from the contract account.
Arbitrary senders other than the contract creator can withdraw ETH from the contract␣
↪account without previously having sent an equivalent amount of ETH to it. This is␣
↪likely to be a vulnerability.
--------------------
In file: tests/testdata/input_contracts/ether_send.sol:21
```

(continues on next page)

(continued from previous page)

```

msg.sender.transfer(address(this).balance)

--------------------
```

If an input file contains multiple contract definitions, Mythril analyzes the *last* bytecode output produced by solc. You can override this by specifying the contract name explicitly:

```
myth pro OmiseGo.sol:OMGToken
```

To specify a contract address, use `-a <address>`

5.3 Analyzing On-Chain Contracts

Analyzing a mainnet contract via INFURA:

```
myth pro -a 0x5c436ff914c458983414019195e0f4ecbef9e6dd
```

Adding the `-l` flag will cause mythril to automatically retrieve dependencies, such as dynamically linked library contracts:

```
myth -v4 pro -l -a 0xEbFD99838cb0c132016B9E117563CB41f2B02264
```

CHAPTER 6

Mythril Package

6.1 Subpackages

6.1.1 mythril.analysis package

Subpackages

mythril.analysis.modules package

Submodules

mythril.analysis.modules.base module

mythril.analysis.modules.delegatecall module

mythril.analysis.modules.dependence_on_predictable_vars module

mythril.analysis.modules.deprecated_ops module

mythril.analysis.modules.ether_thief module

mythril.analysis.modules.exceptions module

mythril.analysis.modules.external_calls module

mythril.analysis.modules.integer module

mythril.analysis.modules.multiple_sends module

mythril.analysis.modules.suicide module

mythril.analysis.modules.transaction_order_dependence module

mythril.analysis.modules.unchecked_retval module

Module contents

Submodules

mythril.analysis.callgraph module

This module contains the configuration and functions to create call graphs.

`mythril.analysis.callgraph.`**`extract_edges`**(*statespace*)

> **Parameters** **`statespace`** –
>
> **Returns**

`mythril.analysis.callgraph.`**`extract_nodes`**(*statespace*)

> **Parameters**
>
> - **`statespace`** –
> - **`color_map`** –
>
> **Returns**

`mythril.analysis.callgraph.`**`generate_graph`**(*statespace, title='Mythril / Ethereum LASER Symbolic VM', physics=False, phrackify=False*)

> **Parameters**
>
> - **`statespace`** –
> - **`title`** –
> - **`physics`** –
> - **`phrackify`** –
>
> **Returns**

mythril.analysis.ops module

This module contains various helper methods for dealing with EVM operations.

`class` `mythril.analysis.ops.`**`Call`**(*node, state, state_index, _type, to, gas, value=<mythril.analysis.ops.Variable object>, data=None*)

> Bases: *`mythril.analysis.ops.Op`*
>
> The representation of a CALL operation.

class `mythril.analysis.ops.`**`Op`**(*node*, *state*, *state_index*)

Bases: `object`

The base type for operations referencing current node and state.

class `mythril.analysis.ops.`**`VarType`**

Bases: `enum.Enum`

An enum denoting whether a value is symbolic or concrete.

`CONCRETE = 2`

`SYMBOLIC = 1`

class `mythril.analysis.ops.`**`Variable`**(*val*, *_type*)

Bases: `object`

The representation of a variable with value and type.

`mythril.analysis.ops.`**`get_variable`**(*i*)

Parameters **`i`** –

Returns

mythril.analysis.report module

This module provides classes that make up an issue report.

class `mythril.analysis.report.`**`Issue`**(*contract*, *function_name*, *address*, *swc_id*, *title*, *bytecode*, *gas_used=(None, None)*, *severity=None*, *description_head=”*, *description_tail=”*, *transaction_sequence=None*)

Bases: `object`

Representation of an issue and its location.

static add_block_data(*transaction_sequence: Dict[KT, VT]*)

Adds sane block data to a transaction_sequence

`add_code_info`(*contract*)

Parameters **`contract`** –

`as_dict`

Returns

`resolve_function_names`()

Resolves function names for each step

`transaction_sequence_jsonv2`

Returns the transaction sequence as a json string with pre-generated block data

`transaction_sequence_users`

Returns the transaction sequence without pre-generated block data

class `mythril.analysis.report.`**`Report`**(*contracts=None*, *exceptions=None*, *execution_info: Optional[List[mythril.laser.execution_info.ExecutionInfo]] = None*)

Bases: `object`

A report containing the content of multiple issues.

append_issue(*issue*)

Parameters **issue** –

as_json()

Returns

as_markdown()

Returns

as_swc_standard_format()

Format defined for integration and correlation.

Returns

as_text()

Returns

environment = <jinja2.environment.Environment object>

sorted_issues()

Returns

mythril.analysis.security module

This module contains functionality for hooking in detection modules and executing them.

mythril.analysis.security.**fire_lasers**(*statespace*, *white_list: Optional[List[str]] = None*) → List[mythril.analysis.report.Issue]

Fire lasers at analysed statespace object

Parameters

- **statespace** – Symbolic statespace to analyze
- **white_list** – Optionally whitelist modules to use for the analysis

Returns Discovered issues

mythril.analysis.security.**retrieve_callback_issues**(*white_list: Optional[List[str]] = None*) → List[mythril.analysis.report.Issue]

Get the issues discovered by callback type detection modules

mythril.analysis.solver module

This module contains analysis module helpers to solve path constraints.

mythril.analysis.solver.**get_transaction_sequence**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*, *constraints: mythril.laser.ethereum.state.constraints.Constraints*) → Dict[KT, VT]

Generate concrete transaction sequence.

Parameters

- **global_state** – GlobalState to generate transaction sequence for
- **constraints** – list of constraints used to generate transaction sequence

`mythril.analysis.solver.`**`pretty_print_model`**(*model*)

Pretty prints a z3 model

Parameters **`model`** –

Returns

mythril.analysis.swc_data module

This module maps SWC IDs to their registry equivalents.

mythril.analysis.symbolic module

This module contains a wrapper around LASER for extended analysis purposes.

`class` `mythril.analysis.symbolic.`**`SymExecWrapper`**(*contract, address: Union[int, str, mythril.laser.smt.bitvec.BitVec], strategy: str, dynloader=None, max_depth: int = 22, execution_timeout: Optional[int] = None, loop_bound: int = 3, create_timeout: Optional[int] = None, transaction_count: int = 2, modules: Optional[List[str]] = None, compulsory_statespace: bool = True, disable_dependency_pruning: bool = False, run_analysis_modules: bool = True, custom_modules_directory: str = ''*)

Bases: `object`

Wrapper class for the LASER Symbolic virtual machine.

Symbolically executes the code and does a bit of pre-analysis for convenience.

`execution_info`

mythril.analysis.traceexplore module

This module provides a function to convert a state space into a set of state nodes and transition edges.

`mythril.analysis.traceexplore.`**`get_serializable_statespace`**(*statespace*)

Parameters **`statespace`** –

Returns

Module contents

6.1.2 mythril.disassembler package

Submodules

mythril.disassembler.asm module

This module contains various helper classes and functions to deal with EVM code disassembly.

class `mythril.disassembler.asm.`**`EvmInstruction`**(*address*, *op_code*, *argument=None*)

Bases: `object`

Model to hold the information of the disassembly.

`to_dict`() → dict

Returns

`mythril.disassembler.asm.`**`disassemble`**(*bytecode: bytes*) → list

Disassembles evm bytecode and returns a list of instructions.

Parameters **`bytecode`** –

Returns

`mythril.disassembler.asm.`**`find_op_code_sequence`**(*pattern: list*, *instruction_list: list*) → collections.abc.Generator

Returns all indices in instruction_list that point to instruction sequences following a pattern.

Parameters

- **`pattern`** – The pattern to look for, e.g. [[“PUSH1”, “PUSH2”], [“EQ”]] where [“PUSH1”, “EQ”] satisfies pattern
- **`instruction_list`** – List of instructions to look in

Returns Indices to the instruction sequences

`mythril.disassembler.asm.`**`get_opcode_from_name`**(*operation_name: str*) → int

Get an op code based on its name.

Parameters **`operation_name`** –

Returns

`mythril.disassembler.asm.`**`instruction_list_to_easm`**(*instruction_list: list*) → str

Convert a list of instructions into an easm op code string.

Parameters **`instruction_list`** –

Returns

`mythril.disassembler.asm.`**`is_sequence_match`**(*pattern: list*, *instruction_list: list*, *index: int*) → bool

Checks if the instructions starting at index follow a pattern.

Parameters

- **`pattern`** – List of lists describing a pattern, e.g. [[“PUSH1”, “PUSH2”], [“EQ”]] where [“PUSH1”, “EQ”] satisfies pattern
- **`instruction_list`** – List of instructions
- **`index`** – Index to check for

Returns Pattern matched

mythril.disassembler.disassembly module

This module contains the class used to represent disassembly code.

class `mythril.disassembler.disassembly.`**`Disassembly`**(*code: str*, *enable_online_lookup: bool = False*)

Bases: `object`

Disassembly class.

Stores bytecode, and its disassembly. Additionally it will gather the following information on the existing functions in the disassembled code: - function hashes - function name to entry point mapping - function entry point to function name mapping

assign_bytecode(*bytecode*)

get_easm()

Returns

mythril.disassembler.disassembly.**get_function_info**(*index: int*, *instruction_list: list*, *signature_database: mythril.support.signatures.SignatureDB*) → Tuple[str, int, str]

Finds the function information for a call table entry Solidity uses the first 4 bytes of the calldata to indicate which function the message call should execute The generated code that directs execution to the correct function looks like this:

- PUSH function_hash
- EQ
- PUSH entry_point
- JUMPI

This function takes an index that points to the first instruction, and from that finds out the function hash, function entry and the function name.

Parameters

- **index** – Start of the entry pattern
- **instruction_list** – Instruction list for the contract that is being analyzed
- **signature_database** – Database used to map function hashes to their respective function names

Returns function hash, function entry point, function name

Module contents

6.1.3 mythril.ethereum package

Subpackages

mythril.ethereum.interface package

Subpackages

mythril.ethereum.interface.leveldb package

Submodules

mythril.ethereum.interface.leveldb.accountindexing module

This module contains account indexing functionality.

This includes a sedes class for lists, account storage receipts for LevelDB and a class for updating account addresses.

class `mythril.ethereum.interface.leveldb.accountindexing.`**`AccountIndexer`**(*ethDB*)
Bases: `object`

Updates address index.

`get_contract_by_hash`(*contract_hash*)
get mapped contract_address by its hash, if not found try indexing.

`updateIfNeeded`()
update address index.

class `mythril.ethereum.interface.leveldb.accountindexing.`**`CountableList`**(*element_sedes*)
Bases: `object`

A sedes for lists of arbitrary length.

Parameters **`element_sedes`** – when (de-)serializing a list, this sedes will be applied to all of its elements

`deserialize`(*serial*)

Parameters **`serial`** –

Returns

`serialize`(*obj*)

Parameters **`obj`** –

Returns

class `mythril.ethereum.interface.leveldb.accountindexing.`**`ReceiptForStorage`**(**args*, ***kwargs*)
Bases: `rlp.sedes.serializable.Serializable`

Receipt format stored in levelDB.

`bloom`

`contractAddress`

`cumulative_gas_used`

`gas_used`

`logs`

`state_root`

`tx_hash`

mythril.ethereum.interface.leveldb.client module

This module contains a LevelDB client.

class `mythril.ethereum.interface.leveldb.client.`**`EthLevelDB`**(*path*)
Bases: `object`

Go-Ethereum LevelDB client class.

`contract_hash_to_address`(*contract_hash*)
Try to find corresponding account address.

Parameters **`contract_hash`** –

Returns

eth_getBalance(*address*)

Get account balance.

Parameters **address** –

Returns

eth_getBlockByNumber(*number*)

Get block body by block number.

Parameters **number** –

Returns

eth_getBlockHeaderByNumber(*number*)

Get block header by block number.

Parameters **number** –

Returns

eth_getCode(*address*)

Get account code.

Parameters **address** –

Returns

eth_getStorageAt(*address*, *position*)

Get account storage data at position.

Parameters

- **address** –
- **position** –

Returns

get_contracts()

Iterate through all contracts.

search(*expression*, *callback_func*)

Search through all contract accounts.

Parameters

- **expression** –
- **callback_func** –

class mythril.ethereum.interface.leveldb.client.**LevelDBReader**(*db*)

Bases: `object`

LevelDB reading interface, can be used with snapshot.

class mythril.ethereum.interface.leveldb.client.**LevelDBWriter**(*db*)

Bases: `object`

level db writing interface.

mythril.ethereum.interface.leveldb.eth_db module

This module contains the ETH_DB class, which the base database used by pyethereum.

class `mythril.ethereum.interface.leveldb.eth_db.`**`ETH_DB`**(*path*)

Bases: `ethereum.db.BaseDB`

Adopts pythereum BaseDB using plyvel.

`get`(*key*)

gets value for key.

`put`(*key*, *value*)

puts value for key.

`write_batch`()

start writing a batch.

mythril.ethereum.interface.leveldb.state module

This module implements wrappers around the pyethereum state for LevelDB.

class `mythril.ethereum.interface.leveldb.state.`**`Account`**(*nonce*, *balance*, *storage*, *code_hash*, *db*, *addr*)

Bases: `rlp.sedes.serializable.Serializable`

adjusted account from ethereum.state.

`balance`

classmethod **`blank_account`**(*db*, *addr*, *initial_nonce=0*)

creates a blank account.

Parameters

- **`db`** –
- **`addr`** –
- **`initial_nonce`** –

Returns

`code`

code rlp data.

`code_hash`

`get_storage_data`(*key*)

get storage data.

Parameters **`key`** –

Returns

`is_blank`()

checks if is a blank account.

Returns

`nonce`

`storage`

class `mythril.ethereum.interface.leveldb.state.`**`State`**(*db*, *root*)

Bases: `object`

adjusted state from ethereum.state.

get_all_accounts()
iterates through trie to and yields non-blank leafs as accounts.

get_and_cache_account(*addr*)
Gets and caches an account for an addres, creates blank if not found.

Parameters `addr` –

Returns

Module contents

mythril.ethereum.interface.rpc package

Submodules

mythril.ethereum.interface.rpc.base_client module

This module provides a basic RPC interface client.

This code is adapted from: https://github.com/ConsenSys/ethjsonrpc

class `mythril.ethereum.interface.rpc.base_client.`**BaseClient**
Bases: `object`

The base RPC client class.

eth_blockNumber()
TODO: documentation

https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_blocknumber

TESTED

eth_coinbase()
TODO: documentation

https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_coinbase

TESTED

eth_getBalance(*address=None*, *block='latest'*)
TODO: documentation

https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_getbalance

TESTED

eth_getBlockByNumber(*block='latest'*, *tx_objects=True*)
TODO: documentation

https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_getblockbynumber

TESTED

eth_getCode(*address*, *default_block='latest'*)
TODO: documentation

https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_getcode

NEEDS TESTING

eth_getStorageAt(*address=None, position=0, block='latest'*)
TODO: documentation

https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_getstorageat

TESTED

eth_getTransactionReceipt(*tx_hash*)
TODO: documentation

https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_gettransactionreceipt

TESTED

mythril.ethereum.interface.rpc.client module

This module contains a basic Ethereum RPC client.

This code is adapted from: https://github.com/ConsenSys/ethjsonrpc

class mythril.ethereum.interface.rpc.client.**EthJsonRpc**(*host='localhost', port=8545, tls=False*)
Bases: *mythril.ethereum.interface.rpc.base_client.BaseClient*

Ethereum JSON-RPC client class.

close()
Close the RPC client's session.

mythril.ethereum.interface.rpc.constants module

This file contains constants used used by the Ethereum JSON RPC interface.

mythril.ethereum.interface.rpc.exceptions module

This module contains exceptions regarding JSON-RPC communication.

exception mythril.ethereum.interface.rpc.exceptions.**BadJsonError**
Bases: *mythril.ethereum.interface.rpc.exceptions.EthJsonRpcError*

An RPC exception denoting that the RPC instance returned a bad JSON object.

exception mythril.ethereum.interface.rpc.exceptions.**BadResponseError**
Bases: *mythril.ethereum.interface.rpc.exceptions.EthJsonRpcError*

An RPC exception denoting that the RPC instance returned a bad response.

exception mythril.ethereum.interface.rpc.exceptions.**BadStatusCodeError**
Bases: *mythril.ethereum.interface.rpc.exceptions.EthJsonRpcError*

An RPC exception denoting a bad status code returned by the RPC instance.

exception mythril.ethereum.interface.rpc.exceptions.**ConnectionError**
Bases: *mythril.ethereum.interface.rpc.exceptions.EthJsonRpcError*

An RPC exception denoting there was an error in connecting to the RPC instance.

exception mythril.ethereum.interface.rpc.exceptions.**EthJsonRpcError**
Bases: Exception

The JSON-RPC base exception type.

mythril.ethereum.interface.rpc.utils module

This module contains various utility functions regarding the RPC data format and validation.

`mythril.ethereum.interface.rpc.utils.`**`clean_hex`**`(`*d*`)`

Convert decimal to hex and remove the "L" suffix that is appended to large numbers.

Parameters **`d`** –

Returns

`mythril.ethereum.interface.rpc.utils.`**`ether_to_wei`**`(`*ether*`)`

Convert ether to wei.

Parameters **`ether`** –

Returns

`mythril.ethereum.interface.rpc.utils.`**`hex_to_dec`**`(`*x*`)`

Convert hex to decimal.

Parameters **`x`** –

Returns

`mythril.ethereum.interface.rpc.utils.`**`validate_block`**`(`*block*`)`

Parameters **`block`** –

Returns

`mythril.ethereum.interface.rpc.utils.`**`wei_to_ether`**`(`*wei*`)`

Convert wei to ether.

Parameters **`wei`** –

Returns

Module contents

Module contents

Submodules

mythril.ethereum.evmcontract module

This module contains the class representing EVM contracts, aka Smart Contracts.

`class` `mythril.ethereum.evmcontract.`**`EVMContract`**`(`*code='', creation_code='', name='Unknown', enable_online_lookup=False*`)`

Bases: `persistent.Persistent`

This class represents an address with associated code (Smart Contract).

`as_dict``()`

Returns

`bytecode_hash`

Returns runtime bytecode hash

creation_bytecode_hash

Returns Creation bytecode hash

get_creation_easm()

Returns

get_easm()

Returns

matches_expression(*expression*)

Parameters **expression** –

Returns

mythril.ethereum.util module

This module contains various utility functions regarding unit conversion and solc integration.

mythril.ethereum.util.**encode_calldata**(*func_name*, *arg_types*, *args*)

Parameters

- **func_name** –
- **arg_types** –
- **args** –

Returns

mythril.ethereum.util.**get_indexed_address**(*index*)

Parameters **index** –

Returns

mythril.ethereum.util.**get_random_address**()

Returns

mythril.ethereum.util.**get_solc_json**(*file*, *solc_binary='solc'*, *solc_settings_json=None*)

Parameters

- **file** –
- **solc_binary** –
- **solc_settings_json** –

Returns

mythril.ethereum.util.**safe_decode**(*hex_encoded_string*)

Parameters **hex_encoded_string** –

Returns

mythril.ethereum.util.**solc_exists**(*version*)

Parameters **version** –

Returns

Module contents

6.1.4 mythril.interfaces package

Submodules

mythril.interfaces.cli module

mythril.py: Bug hunting on the Ethereum blockchain

http://www.github.com/ConsenSys/mythril

`mythril.interfaces.cli.`**`contract_hash_to_address`**(*args: argparse.Namespace*)
prints the hash from function signature :param args: :return:

`mythril.interfaces.cli.`**`create_analyzer_parser`**(*analyzer_parser: argparse.ArgumentParser*)
Modify parser to handle analyze command :param analyzer_parser: :return:

`mythril.interfaces.cli.`**`create_disassemble_parser`**(*parser: argparse.ArgumentParser*)
Modify parser to handle disassembly :param parser: :return:

`mythril.interfaces.cli.`**`create_func_to_hash_parser`**(*parser: argparse.ArgumentParser*)
Modify parser to handle func_to_hash command :param parser: :return:

`mythril.interfaces.cli.`**`create_hash_to_addr_parser`**(*hash_parser: argparse.ArgumentParser*)
Modify parser to handle hash_to_addr command :param hash_parser: :return:

`mythril.interfaces.cli.`**`create_leveldb_parser`**(*parser: argparse.ArgumentParser*)
Modify parser to handle leveldb-search :param parser: :return:

`mythril.interfaces.cli.`**`create_pro_parser`**(*parser: argparse.ArgumentParser*)
Modify parser to handle mythx analysis :param parser: :return:

`mythril.interfaces.cli.`**`create_read_storage_parser`**(*read_storage_parser: argparse.ArgumentParser*)
Modify parser to handle storage slots :param read_storage_parser: :return:

`mythril.interfaces.cli.`**`execute_command`**(*disassembler: mythril.mythril.mythril_disassembler.MythrilDisassembler, address: str, parser: argparse.ArgumentParser, args: argparse.Namespace*)
Execute command :param disassembler: :param address: :param parser: :param args: :return:

`mythril.interfaces.cli.`**`exit_with_error`**(*format_, message*)
Exits with error :param **format_**: The format of the message :param message: message

`mythril.interfaces.cli.`**`get_creation_input_parser`**() → argparse.ArgumentParser
Returns Parser which handles input :return: Parser which handles input

`mythril.interfaces.cli.`**`get_output_parser`**() → argparse.ArgumentParser
Get parser which handles output :return: Parser which handles output

`mythril.interfaces.cli.`**`get_rpc_parser`**() → argparse.ArgumentParser
Get parser which handles RPC flags :return: Parser which handles rpc inputs

`mythril.interfaces.cli.`**`get_runtime_input_parser`**() → argparse.ArgumentParser
Returns Parser which handles input :return: Parser which handles input

`mythril.interfaces.cli.`**`get_utilities_parser`**() → argparse.ArgumentParser
Get parser which handles utilities flags :return: Parser which handles utility flags

`mythril.interfaces.cli.`**`leveldb_search`**(*config: mythril.mythril.mythril_config.MythrilConfig, args: argparse.Namespace*)

Handle leveldb search :param config: :param args: :return:

`mythril.interfaces.cli.`**`load_code`**(*disassembler: mythril.mythril.mythril_disassembler.MythrilDisassembler, args: argparse.Namespace*)

Loads code into disassembly and returns address :param disassembler: :param args: :return: Address

`mythril.interfaces.cli.`**`main`**() → None

The main CLI interface entry point.

`mythril.interfaces.cli.`**`parse_args_and_execute`**(*parser: argparse.ArgumentParser, args: argparse.Namespace*) → None

Parses the arguments :param parser: The parser :param args: The args

`mythril.interfaces.cli.`**`set_config`**(*args: argparse.Namespace*)

Set config based on args :param args: :return: modified config

`mythril.interfaces.cli.`**`validate_args`**(*args: argparse.Namespace*)

Validate cli args :param args: :return:

mythril.interfaces.epic module

Don't ask.

`class` `mythril.interfaces.epic.`**`LolCat`**(*mode=256, output=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

Bases: `object`

Cats lel.

`ansi`(*rgb*)

Parameters **`rgb`** –

Returns

`cat`(*fd, options*)

Parameters

- **`fd`** –
- **`options`** –

`println`(*s, options*)

Parameters

- **`s`** –
- **`options`** –

`println_ani`(*s, options*)

Parameters

- **`s`** –
- **`options`** –

Returns

`println_plain`(*s, options*)

Parameters

- **s** –
- **options** –

rainbow(*freq*, *i*)

Parameters

- **freq** –
- **i** –

Returns

wrap(**codes*)

Parameters **codes** –

Returns

mythril.interfaces.epic.**detect_mode**(*term_hint='xterm-256color'*)

Poor-mans color mode detection.

mythril.interfaces.epic.**reset**()

mythril.interfaces.epic.**run**()

Main entry point.

Module contents

6.1.5 mythril.laser package

Subpackages

mythril.laser.ethereum package

Subpackages

mythril.laser.ethereum.state package

Submodules

mythril.laser.ethereum.state.account module

This module contains account-related functionality.

This includes classes representing accounts and their storage.

class mythril.laser.ethereum.state.account.**Account**(*address: Union[mythril.laser.smt.bitvec.BitVec, str], code=None, contract_name=None, balances: mythril.laser.smt.array.Array = None, concrete_storage=False, dynamic_loader=None*)

Bases: object

Account class representing ethereum accounts.

add_balance(*balance: Union[int, mythril.laser.smt.bitvec.BitVec]*) → None

Parameters **balance** –

as_dict

Returns

set_balance(*balance: Union[int, mythril.laser.smt.bitvec.BitVec]*) → None

Parameters **balance** –

class mythril.laser.ethereum.state.account.**Storage**(*concrete=False*, *address=None*, *dynamic_loader=None*)

Bases: object

Storage class represents the storage of an Account.

mythril.laser.ethereum.state.annotation module

This module includes classes used for annotating trace information.

This includes the base StateAnnotation class, as well as an adaption, which will not be copied on every new state.

class mythril.laser.ethereum.state.annotation.**MergeableStateAnnotation**

Bases: *mythril.laser.ethereum.state.annotation.StateAnnotation*

This class allows a base annotation class for annotations that can be merged.

check_merge_annotation(*annotation*) → bool

merge_annotation(*annotation*)

class mythril.laser.ethereum.state.annotation.**NoCopyAnnotation**

Bases: *mythril.laser.ethereum.state.annotation.StateAnnotation*

This class provides a base annotation class for annotations that shouldn't be copied on every new state.

Rather the same object should be propagated. This is very useful if you are looking to analyze a property over multiple substates

class mythril.laser.ethereum.state.annotation.**StateAnnotation**

Bases: object

The StateAnnotation class is used to persist information over traces.

This allows modules to reason about traces without the need to traverse the state space themselves.

persist_over_calls

If this function returns true then laser will propagate the annotation between calls

The default is set to False

persist_to_world_state

If this function returns true then laser will also annotate the world state.

If you want annotations to persist through different user initiated message call transactions then this should be enabled.

The default is set to False

mythril.laser.ethereum.state.calldata module

This module declares classes to represent call data.

class mythril.laser.ethereum.state.calldata.**BaseCalldata**(*tx_id: str*)

Bases: `object`

Base calldata class This represents the calldata provided when sending a transaction to a contract.

calldatasize

Returns Calldata size for this calldata object

concrete(*model: z3.z3.Model*) → list

Returns a concrete version of the calldata using the provided model.

Parameters model –

get_word_at(*offset: int*) → mythril.laser.smt.expression.Expression

Gets word at offset.

Parameters offset –

Returns

size

Returns the exact size of this calldata, this is not normalized.

Returns unnormalized call data size

class mythril.laser.ethereum.state.calldata.**BasicConcreteCalldata**(*tx_id: str, calldata: list*)

Bases: *mythril.laser.ethereum.state.calldata.BaseCalldata*

A base class to represent concrete call data.

concrete(*model: z3.z3.Model*) → list

Parameters model –

Returns

size

Returns

class mythril.laser.ethereum.state.calldata.**BasicSymbolicCalldata**(*tx_id: str*)

Bases: *mythril.laser.ethereum.state.calldata.BaseCalldata*

A basic class representing symbolic call data.

concrete(*model: z3.z3.Model*) → list

Parameters model –

Returns

size

Returns

class mythril.laser.ethereum.state.calldata.**ConcreteCalldata**(*tx_id: str, calldata: list*)

Bases: *mythril.laser.ethereum.state.calldata.BaseCalldata*

A concrete call data representation.

concrete (*model: z3.z3.Model*) → list

Parameters **model** –

Returns

size

Returns

class mythril.laser.ethereum.state.calldata.**SymbolicCalldata** (*tx_id: str*)

Bases: *mythril.laser.ethereum.state.calldata.BaseCalldata*

A class for representing symbolic call data.

concrete (*model: z3.z3.Model*) → list

Parameters **model** –

Returns

size

Returns

mythril.laser.ethereum.state.constraints module

This module contains the class used to represent state-change constraints in the call graph.

class mythril.laser.ethereum.state.constraints.**Constraints** (*constraint_list: Optional[List[mythril.laser.smt.bool.Bool]] = None*)

Bases: list

This class should maintain a solver and it's constraints, This class tries to make the Constraints() object as a simple list of constraints with some background processing.

append (*constraint: Union[bool, mythril.laser.smt.bool.Bool]*) → None

Parameters **constraint** – The constraint to be appended

as_list

Returns returns the list of constraints

copy () → mythril.laser.ethereum.state.constraints.Constraints

Return a shallow copy of the list.

is_possible

Returns True/False based on the existence of solution of constraints

pop (*index: int = -1*) → None

Parameters **index** – Index to be popped from the list

mythril.laser.ethereum.state.environment module

This module contains the representation for an execution state's environment.

class mythril.laser.ethereum.state.environment.**Environment**(*active_account: mythril.laser.ethereum.state.account.Account, sender: z3.z3.ExprRef, calldata: mythril.laser.ethereum.state.calldata.BaseCalld gasprice: z3.z3.ExprRef, callvalue: z3.z3.ExprRef, origin: z3.z3.ExprRef, code=None, static=False*)

Bases: `object`

The environment class represents the current execution environment for the symbolic executor.

as_dict

Returns

mythril.laser.ethereum.state.global_state module

This module contains a representation of the global execution state.

class mythril.laser.ethereum.state.global_state.**GlobalState**(*world_state: WorldState, environment: mythril.laser.ethereum.state.environment.Envi node: mythril.laser.ethereum.cfg.Node, machine_state=None, transaction_stack=None, last_return_data=None, annotations=None*)

Bases: `object`

GlobalState represents the current globalstate.

accounts

Returns

add_annotations(*annotations: List[mythril.laser.ethereum.state.annotation.StateAnnotation]*)
Function used to add annotations to global state :param annotations: :return:

annotate(*annotation: mythril.laser.ethereum.state.annotation.StateAnnotation*) → None

Parameters **annotation** –

annotations

Returns

current_transaction

Returns

get_annotations(*annotation_type: type*) → Iterable[mythril.laser.ethereum.state.annotation.StateAnnotation]
Filters annotations for the queried annotation type. Designed particularly for modules with annotations: globalstate.get_annotations(MySpecificModuleAnnotation)

Parameters **`annotation_type`** – The type to filter annotations for

Returns filter of matching annotations

`get_current_instruction``()` → Dict[KT, VT]
Gets the current instruction for this GlobalState.

Returns

`instruction`

Returns

`new_bitvec``(`*name: str*, *size=256*, *annotations=None*`)` → z3.z3.BitVec

Parameters

- **`name`** –
- **`size`** –

Returns

mythril.laser.ethereum.state.machine_state module

This module contains a representation of the EVM's machine state and its stack.

`class` `mythril.laser.ethereum.state.machine_state.`**`MachineStack`**`(`*default_list=None*`)`
Bases: `list`

Defines EVM stack, overrides the default list to handle overflows.

`STACK_LIMIT = 1024`

`append``(`*element: Union[int, mythril.laser.smt.expression.Expression]*`)` → None

This function ensures the following properties when appending to a list:

- Element appended to this list should be a BitVec
- Ensures stack overflow bound

Parameters **`element`** – element to be appended to the list

Function appends the element to list if the size is less than STACK_LIMIT, else throws an error

`pop``(`*index=-1*`)` → Union[int, mythril.laser.smt.expression.Expression]
This function ensures stack underflow bound :param index:index to be popped, same as the list() class. :returns popped value :function: same as list() class but throws StackUnderflowException for popping from an empty list

class mythril.laser.ethereum.state.machine_state.**MachineState**(*gas_limit: int*, *pc=0*, *stack=None*, *subroutine_stack=None*, *memory: Optional[mythril.laser.ethereum.state.memory = None*, *constraints=None*, *depth=0*, *max_gas_used=0*, *min_gas_used=0*, *prev_pc=-1*)

Bases: `object`

MachineState represents current machine state also referenced to as mu.

as_dict

Returns

calculate_extension_size(*start: int*, *size: int*) → int

Parameters

- **start** –
- **size** –

Returns

calculate_memory_gas(*start: int*, *size: int*)

Parameters

- **start** –
- **size** –

Returns

check_gas()

Check whether the machine is out of gas.

mem_extend(*start: Union[int, mythril.laser.smt.bitvec.BitVec], size: Union[int, mythril.laser.smt.bitvec.BitVec]*) → None

Extends the memory of this machine state.

Parameters

- **start** – Start of memory extension
- **size** – Size of memory extension

memory_size

Returns

memory_write(*offset: int, data: List[Union[mythril.laser.smt.bitvec.BitVec, int]]*) → None

Writes data to memory starting at offset.

Parameters

- **offset** –
- **data** –

pc

Returns

pop (*amount=1*) → Union[mythril.laser.smt.bitvec.BitVec, List[mythril.laser.smt.bitvec.BitVec]]
Pops amount elements from the stack.

Parameters **amount** –

Returns

mythril.laser.ethereum.state.memory module

This module contains a representation of a smart contract's memory.

class mythril.laser.ethereum.state.memory.**Memory**
Bases: object

A class representing contract memory with random access.

extend (*size: int*)

Parameters **size** –

get_word_at (*index: int*) → Union[int, mythril.laser.smt.bitvec.BitVec]
Access a word from a specified memory index.

Parameters **index** – integer representing the index to access

Returns 32 byte word at the specified index

write_word_at (*index: int, value: Union[int, mythril.laser.smt.bitvec.BitVec, bool, mythril.laser.smt.bool.Bool]*) → None
Writes a 32 byte word to memory at the specified index'

Parameters

- **index** – index to write to
- **value** – the value to write to memory

mythril.laser.ethereum.state.memory.**convert_bv** (*val: Union[int, mythril.laser.smt.bitvec.BitVec]*) → mythril.laser.smt.bitvec.BitVec

mythril.laser.ethereum.state.world_state module

This module contains a representation of the EVM's world state.

class mythril.laser.ethereum.state.world_state.**WorldState** (*transaction_sequence=None, annotations: List[mythril.laser.ethereum.state.annotation.Stat = None, constraints: mythril.laser.ethereum.state.constraints.Constrai = None*)
Bases: object

The WorldState class represents the world state as described in the yellow paper.

accounts

accounts_exist_or_load(*addr*, *dynamic_loader: mythril.support.loader.DynLoader*) → mythril.laser.ethereum.state.account.Account
returns account if it exists, else it loads from the dynamic loader :param addr: address :param dynamic_loader: Dynamic Loader :return: The code

annotate(*annotation: mythril.laser.ethereum.state.annotation.StateAnnotation*) → None

Parameters **annotation** –

annotations

Returns

create_account(*balance=0*, *address=None*, *concrete_storage=False*, *dynamic_loader=None*, *creator=None*, *code=None*) → mythril.laser.ethereum.state.account.Account
Create non-contract account.

Parameters

- **address** – The account's address
- **balance** – Initial balance for the account
- **concrete_storage** – Interpret account storage as concrete
- **dynamic_loader** – used for dynamically loading storage from the block chain
- **creator** – The address of the creator of the contract if it's a contract
- **code** – The code of the contract, if it's a contract

Returns The new account

create_initialized_contract_account(*contract_code*, *storage*) → None
Creates a new contract account, based on the contract code and storage provided The contract code only includes the runtime contract bytecode.

Parameters

- **contract_code** – Runtime bytecode for the contract
- **storage** – Initial storage for the contract

Returns The new account

get_annotations(*annotation_type: type*) → Iterator[mythril.laser.ethereum.state.annotation.StateAnnotation]
Filters annotations for the queried annotation type. Designed particularly for modules with annotations: worldstate.get_annotations(MySpecificModuleAnnotation)

Parameters **annotation_type** – The type to filter annotations for

Returns filter of matching annotations

put_account(*account: mythril.laser.ethereum.state.account.Account*) → None

Parameters **account** –

Module contents

mythril.laser.ethereum.strategy package

Submodules

mythril.laser.ethereum.strategy.basic module

This module implements basic symbolic execution search strategies.

class `mythril.laser.ethereum.strategy.basic.`**`BreadthFirstSearchStrategy`**(*work_list*, *max_depth*)

Bases: *mythril.laser.ethereum.strategy.BasicSearchStrategy*

Implements a breadth first search strategy I.E.

Execute all states of a "level" before continuing

`get_strategic_global_state`() → mythril.laser.ethereum.state.global_state.GlobalState

Returns

class `mythril.laser.ethereum.strategy.basic.`**`DepthFirstSearchStrategy`**(*work_list*, *max_depth*)

Bases: *mythril.laser.ethereum.strategy.BasicSearchStrategy*

Implements a depth first search strategy I.E.

Follow one path to a leaf, and then continue to the next one

`get_strategic_global_state`() → mythril.laser.ethereum.state.global_state.GlobalState

Returns

class `mythril.laser.ethereum.strategy.basic.`**`ReturnRandomNaivelyStrategy`**(*work_list*, *max_depth*)

Bases: *mythril.laser.ethereum.strategy.BasicSearchStrategy*

chooses a random state from the worklist with equal likelihood.

`get_strategic_global_state`() → mythril.laser.ethereum.state.global_state.GlobalState

Returns

class `mythril.laser.ethereum.strategy.basic.`**`ReturnWeightedRandomStrategy`**(*work_list*, *max_depth*)

Bases: *mythril.laser.ethereum.strategy.BasicSearchStrategy*

chooses a random state from the worklist with likelihood based on inverse proportion to depth.

`get_strategic_global_state`() → mythril.laser.ethereum.state.global_state.GlobalState

Returns

Module contents

class `mythril.laser.ethereum.strategy.`**`BasicSearchStrategy`**(*work_list*, *max_depth*)

Bases: `abc.ABC`

`get_strategic_global_state`()

`max_depth`

`work_list`

mythril.laser.ethereum.transaction package

Submodules

mythril.laser.ethereum.transaction.concolic module

This module contains functions to set up and execute concolic message calls.

`mythril.laser.ethereum.transaction.concolic.`**`execute_message_call`**(*laser_evm*, *callee_address*, *caller_address*, *origin_address*, *code*, *data*, *gas_limit*, *gas_price*, *value*, *track_gas=False*) → Union[None, List[mythril.laser.ethereum.state.globa

Execute a message call transaction from all open states.

Parameters

- **`laser_evm`** –
- **`callee_address`** –
- **`caller_address`** –
- **`origin_address`** –
- **`code`** –
- **`data`** –
- **`gas_limit`** –
- **`gas_price`** –
- **`value`** –
- **`track_gas`** –

Returns

mythril.laser.ethereum.transaction.symbolic module

This module contains functions setting up and executing transactions with symbolic values.

`class` `mythril.laser.ethereum.transaction.symbolic.`**`Actors`**(*creator=100475310549029526324481294656594*, *attacker=127127061300004165581744834813227*, *someguy=974334424887268612135789888477*

Bases: `object`

`attacker`

`creator`

mythril.laser.ethereum.transaction.symbolic.**execute_contract_creation**(*laser_evm*, *contract_initialization_code*, *contract_name=None*, *world_state=None*) → mythril.laser.ethereum.state.ac

Executes a contract creation transaction from all open states.

Parameters

- **laser_evm** –
- **contract_initialization_code** –
- **contract_name** –

Returns

mythril.laser.ethereum.transaction.symbolic.**execute_message_call**(*laser_evm*, *callee_address: mythril.laser.smt.bitvec.BitVec*) → None

Executes a message call transaction from all open states.

Parameters

- **laser_evm** –
- **callee_address** –

mythril.laser.ethereum.transaction.transaction_models module

This module contians the transaction models used throughout LASER's symbolic execution.

class mythril.laser.ethereum.transaction.transaction_models.**BaseTransaction**(*world_state: mythril.laser.ethereum callee_account: mythril.laser.ethereum = None, caller: z3.z3.ExprRef = None, call_data=None, identifier: Optional[str] = None, gas_price=None, gas_limit=None, origin=None, code=None, call_value=None, init_call_data=True, static=False*)

Bases: object

Basic transaction class holding common data.

initial_global_state() → mythril.laser.ethereum.state.global_state.GlobalState

initial_global_state_from_environment(*environment*, *active_function*)

Parameters

- **environment** –
- **active_function** –

Returns

class mythril.laser.ethereum.transaction.transaction_models.**ContractCreationTransaction**(*wor myt call z3.z = Nor call iden ti- fier Op- tion = Nor gas gas ori- gin cod call con trac con trac*

Bases: *mythril.laser.ethereum.transaction.transaction_models.BaseTransaction*

Transaction object models an transaction.

end(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*, *return_data=None*, *revert=False*)

Parameters

- **global_state** –
- **return_data** –
- **revert** –

initial_global_state() → mythril.laser.ethereum.state.global_state.GlobalState
Initialize the execution environment.

class mythril.laser.ethereum.transaction.transaction_models.**MessageCallTransaction**(**args*, ***kwargs*)

Bases: *mythril.laser.ethereum.transaction.transaction_models.BaseTransaction*

Transaction object models an transaction.

end(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*, *return_data=None*, *revert=False*) → None

Parameters

- **global_state** –
- **return_data** –
- **revert** –

initial_global_state() → mythril.laser.ethereum.state.global_state.GlobalState
Initialize the execution environment.

exception mythril.laser.ethereum.transaction.transaction_models.**TransactionEndSignal**(*global_ mythril. re- vert=Fa*

Bases: Exception

Exception raised when a transaction is finalized.

exception mythril.laser.ethereum.transaction.transaction_models.**TransactionStartSignal**(*trans Unio Con- tract ation Tran ac- tion] op_c str, globa myth*

Bases: Exception

Exception raised when a new transaction is started.

mythril.laser.ethereum.transaction.transaction_models.**get_next_transaction_id**() → str

Returns

Module contents

Submodules

mythril.laser.ethereum.call module

This module contains the business logic used by Instruction in instructions.py to get the necessary elements from the stack and determine the parameters for the new global state.

mythril.laser.ethereum.call.**get_call_data**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState, memory_start: Union[int, mythril.laser.smt.bitvec.BitVec], memory_size: Union[int, mythril.laser.smt.bitvec.BitVec]*)

Gets call_data from the global_state.

Parameters

- **global_state** – state to look in
- **memory_start** – Start index
- **memory_size** – Size

Returns Tuple containing: call_data array from memory or empty array if symbolic, type found

mythril.laser.ethereum.call.**get_call_parameters**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState, dynamic_loader: mythril.support.loader.DynLoader, with_value=False*)

Gets call parameters from global state Pops the values from the stack and determines output parameters.

Parameters

- **global_state** – state to look in
- **dynamic_loader** – dynamic loader to use
- **with_value** – whether to pop the value argument from the stack

Returns callee_account, call_data, value, call_data_type, gas

mythril.laser.ethereum.call.**get_callee_account**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState, callee_address: Union[str, mythril.laser.smt.bitvec.BitVec], dynamic_loader: mythril.support.loader.DynLoader*)

Gets the callees account from the global_state.

Parameters

- **global_state** – state to look in
- **callee_address** – address of the callee
- **dynamic_loader** – dynamic loader to use

Returns Account belonging to callee

mythril.laser.ethereum.call.**get_callee_address**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState, dynamic_loader: mythril.support.loader.DynLoader, symbolic_to_address: mythril.laser.smt.expression.Expression*)

Gets the address of the callee.

Parameters

- **global_state** – state to look in
- **dynamic_loader** – dynamic loader to use
- **symbolic_to_address** – The (symbolic) callee address

Returns Address of the callee

mythril.laser.ethereum.call.**insert_ret_val**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*)

mythril.laser.ethereum.call.**native_call**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState, callee_address: Union[str, mythril.laser.smt.bitvec.BitVec], call_data: mythril.laser.ethereum.state.calldata.BaseCalldata, memory_out_offset: Union[int, mythril.laser.smt.expression.Expression], memory_out_size: Union[int, mythril.laser.smt.expression.Expression]*) → Optional[List[mythril.laser.ethereum.state.global_state.GlobalState]]

mythril.laser.ethereum.cfg module

This module.

class mythril.laser.ethereum.cfg.**Edge**(*node_from: int*, *node_to: int*, *edge_type=<JumpType.UNCONDITIONAL: 2>*, *condition=None*)

Bases: `object`

The respresentation of a call graph edge.

as_dict

Returns

class mythril.laser.ethereum.cfg.**JumpType**

Bases: `enum.Enum`

An enum to represent the types of possible JUMP scenarios.

CALL = 3

CONDITIONAL = 1

RETURN = 4

Transaction = 5

UNCONDITIONAL = 2

class mythril.laser.ethereum.cfg.**Node**(*contract_name: str*, *start_addr=0*, *constraints=None*, *function_name='unknown'*)

Bases: `object`

The representation of a call graph node.

get_cfg_dict() → Dict[KT, VT]

Returns

class mythril.laser.ethereum.cfg.**NodeFlags**

Bases: `flags.Flags`

A collection of flags to denote the type a call graph node can have.

mythril.laser.ethereum.evm_exceptions module

This module contains EVM exception types used by LASER.

exception mythril.laser.ethereum.evm_exceptions.**InvalidInstruction**

Bases: *mythril.laser.ethereum.evm_exceptions.VmException*

A VM exception denoting an invalid op code has been encountered.

exception mythril.laser.ethereum.evm_exceptions.**InvalidJumpDestination**

Bases: *mythril.laser.ethereum.evm_exceptions.VmException*

A VM exception regarding JUMPs to invalid destinations.

exception mythril.laser.ethereum.evm_exceptions.**OutOfGasException**

Bases: *mythril.laser.ethereum.evm_exceptions.VmException*

A VM exception denoting the current execution has run out of gas.

exception mythril.laser.ethereum.evm_exceptions.**StackOverflowException**

Bases: *mythril.laser.ethereum.evm_exceptions.VmException*

A VM exception regarding stack overflows.

exception mythril.laser.ethereum.evm_exceptions.**StackUnderflowException**

Bases: IndexError, *mythril.laser.ethereum.evm_exceptions.VmException*

A VM exception regarding stack underflows.

exception mythril.laser.ethereum.evm_exceptions.**VmException**

Bases: Exception

The base VM exception type.

exception mythril.laser.ethereum.evm_exceptions.**WriteProtection**

Bases: *mythril.laser.ethereum.evm_exceptions.VmException*

A VM exception denoting that a write operation is executed on a write protected environment

mythril.laser.ethereum.gas module

mythril.laser.ethereum.instructions module

This module contains a representation class for EVM instructions and transitions between them.

class mythril.laser.ethereum.instructions.**Instruction**(*op_code: str, dynamic_loader: mythril.support.loader.DynLoader, pre_hooks: List[Callable] = None, post_hooks: List[Callable] = None*)

Bases: object

Instruction class is used to mutate a state according to the current instruction.

add_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

addmod_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

address_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

and_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

assert_fail_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

balance_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

beginsub_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

blockhash_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

byte_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

call_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`call_post`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`callcode_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`callcode_post`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`calldatacopy_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`calldataload_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`calldatasize_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

caller_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

callvalue_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

chainid_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

codecopy_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

codesize_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

coinbase_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

create2_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

create2_post(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

create_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

create_post(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

delegatecall_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

delegatecall_post(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

difficulty_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

div_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

dup_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

eq_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

evaluate(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*, *post=False*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Performs the mutation for this instruction.

Parameters

- **global_state** –
- **post** –

Returns

exp_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

extcodecopy_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

extcodehash_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

extcodesize_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

gas_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

gaslimit_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

gasprice_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

gt_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

invalid_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

iszero_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

jump_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

jumpdest_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

jumpi_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

jumpsub_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

log_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

lt_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`mload_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`mod_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`msize_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`mstore8_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`mstore_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

`mul_`(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **`func_obj`** –
- **`global_state`** –

Returns

mulmod_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

not_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

number_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

or_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

origin_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

pc_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

pop_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

post_handler(*global_state*, *function_name: str*)

push_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

return_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

returndatacopy_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

returndatasize_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

returnsub_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

revert_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

sar_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

sdiv_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

selfbalance_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

sgt_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

sha3_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

shl_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

shr_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

signextend_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

sload_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

slt_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

smod_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

sstore_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

staticcall_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

staticcall_post(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

stop_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

sub_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

suicide_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

swap_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

timestamp_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

xor_(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters

- **func_obj** –
- **global_state** –

Returns

class mythril.laser.ethereum.instructions.**StateTransition**(*increment_pc=True*, *enable_gas=True*, *is_state_mutation_instruction=False*)

Bases: `object`

Decorator that handles global state copy and original return.

This decorator calls the decorated instruction mutator function on a copy of the state that is passed to it. After the call, the resulting new states' program counter is automatically incremented if *increment_pc=True*.

accumulate_gas(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*)

Parameters **global_state** –

Returns

static call_on_state_copy(*func: Callable*, *func_obj: mythril.laser.ethereum.instructions.Instruction*, *state: mythril.laser.ethereum.state.global_state.GlobalState*)

Parameters

- **func** –
- **func_obj** –
- **state** –

Returns

static check_gas_usage_limit(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*)

Parameters **global_state** –

Returns

increment_states_pc(*states: List[mythril.laser.ethereum.state.global_state.GlobalState]*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

Parameters **states** –

Returns

mythril.laser.ethereum.instructions.**transfer_ether**(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*, *sender: mythril.laser.smt.bitvec.BitVec*, *receiver: mythril.laser.smt.bitvec.BitVec*, *value: Union[int, mythril.laser.smt.bitvec.BitVec]*)

Perform an Ether transfer between two accounts

Parameters

- **global_state** – The global state in which the Ether transfer occurs
- **sender** – The sender of the Ether
- **receiver** – The recipient of the Ether

- **value** – The amount of Ether to send

Returns

mythril.laser.ethereum.keccak module

mythril.laser.ethereum.natives module

This nodule defines helper functions to deal with native calls.

exception mythril.laser.ethereum.natives.**NativeContractException**
Bases: Exception

An exception denoting an error during a native call.

mythril.laser.ethereum.natives.**blake2b_fcompress**(*data: List[int]*) → List[int]
blake2b hashing :param data: :return:

mythril.laser.ethereum.natives.**ec_add**(*data: List[int]*) → List[int]

mythril.laser.ethereum.natives.**ec_mul**(*data: List[int]*) → List[int]

mythril.laser.ethereum.natives.**ec_pair**(*data: List[int]*) → List[int]

mythril.laser.ethereum.natives.**ecrecover**(*data: List[int]*) → List[int]

Parameters **data** –

Returns

mythril.laser.ethereum.natives.**identity**(*data: List[int]*) → List[int]

Parameters **data** –

Returns

mythril.laser.ethereum.natives.**mod_exp**(*data: List[int]*) → List[int]
TODO: Some symbolic parts can be handled here Modular Exponentiation :param data: Data with <length_of_BASE> <length_of_EXPONENT> <length_of_MODULUS> <BASE> <EXPONENT> <MODULUS> :return: modular exponentiation

mythril.laser.ethereum.natives.**native_contracts**(*address: int, data: mythril.laser.ethereum.state.calldata.BaseCalldata*) → List[int]
Takes integer address 1, 2, 3, 4.

Parameters

- **address** –
- **data** –

Returns

mythril.laser.ethereum.natives.**ripemd160**(*data: List[int]*) → List[int]

Parameters **data** –

Returns

mythril.laser.ethereum.natives.**sha256**(*data: List[int]*) → List[int]

Parameters **data** –

Returns

mythril.laser.ethereum.svm module

This module implements the main symbolic execution engine.

class mythril.laser.ethereum.svm.**LaserEVM**(*dynamic_loader=None*, *max_depth=inf*, *execution_timeout=60*, *create_timeout=10*, *strategy=<class 'mythril.laser.ethereum.strategy.basic.DepthFirstSearchStrategy'>*, *transaction_count=2*, *requires_statespace=True*, *iprof=None*)

Bases: object

The LASER EVM.

Just as Mithril had to be mined at great efforts to provide the Dwarves with their exceptional armour, LASER stands at the heart of Mythril, digging deep in the depths of call graphs, unearthing the most precious symbolic call data, that is then hand-forged into beautiful and strong security issues by the experienced smiths we call detection modules. It is truly a magnificent symbiosis.

exec(*create=False*, *track_gas=False*) → Optional[List[mythril.laser.ethereum.state.global_state.GlobalState]]

Parameters

- **create** –
- **track_gas** –

Returns

execute_state(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*) → Tuple[List[mythril.laser.ethereum.state.global_state.GlobalState], Optional[str]]

Execute a single instruction in global_state.

Parameters **global_state** –

Returns A list of successor states.

extend_strategy(*extension: abc.ABCMeta*, **args*) → None

handle_vm_exception(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*, *op_code: str*, *error_msg: str*) → List[mythril.laser.ethereum.state.global_state.GlobalState]

instr_hook(*hook_type*, *opcode*) → Callable

Registers the annoted function with register_instr_hooks

Parameters

- **hook_type** – Type of hook pre/post
- **opcode** – The opcode related to the function

laser_hook(*hook_type: str*) → Callable

Registers the annotated function with register_laser_hooks

Parameters **hook_type** –

Returns hook decorator

manage_cfg(*opcode: str, new_states: List[mythril.laser.ethereum.state.global_state.GlobalState]*) → None

Parameters

- **opcode** –

- **new_states** –

post_hook (*op_code: str*) → Callable

Parameters **op_code** –

Returns

pre_hook (*op_code: str*) → Callable

Parameters **op_code** –

Returns

register_hooks (*hook_type: str, hook_dict: Dict[str, List[Callable]]*)

Parameters

- **hook_type** –
- **hook_dict** –

register_instr_hooks (*hook_type: str*, *opcode: str*, *hook: Callable*)
Registers instructions hooks from plugins

register_laser_hooks (*hook_type: str*, *hook: Callable*)
registers the hook with this Laser VM

sym_exec (*world_state: mythril.laser.ethereum.state.world_state.WorldState = None*, *target_address: int = None*, *creation_code: str = None*, *contract_name: str = None*) → None
Starts symbolic execution There are two modes of execution. Either we analyze a preconfigured configuration, in which case the world_state and target_address variables must be supplied. Or we execute the creation code of a contract, in which case the creation code and desired name of that contract should be provided.

:param world_state The world state configuration from which to perform analysis :param target_address The address of the contract account in the world state which analysis should target :param creation_code The creation code to create the target contract in the symbolic environment :param contract_name The name that the created account should be associated with

exception mythril.laser.ethereum.svm.**SVMError**
Bases: Exception

An exception denoting an unexpected state in symbolic execution.

mythril.laser.ethereum.taint_analysis module

mythril.laser.ethereum.util module

This module contains various utility conversion functions and constants for LASER.

mythril.laser.ethereum.util.**bytearray_to_int** (*arr*)

Parameters **arr** –

Returns

mythril.laser.ethereum.util.**concrete_int_from_bytes** (*concrete_bytes: Union[List[Union[mythril.laser.smt.bitvec.BitVec, int]], bytes], start_index: int*) → int

Parameters

- **concrete_bytes** –
- **start_index** –

Returns

mythril.laser.ethereum.util.**concrete_int_to_bytes**(*val*)

Parameters **val** –

Returns

mythril.laser.ethereum.util.**extract32**(*data: bytearray*, *i: int*) → int

Parameters

- **data** –
- **i** –

Returns

mythril.laser.ethereum.util.**extract_copy**(*data: bytearray*, *mem: bytearray*, *memstart: int*, *datastart: int*, *size: int*)

mythril.laser.ethereum.util.**get_concrete_int**(*item: Union[int, mythril.laser.smt.expression.Expression]*) → int

Parameters **item** –

Returns

mythril.laser.ethereum.util.**get_instruction_index**(*instruction_list: List[Dict[KT, VT]]*, *address: int*) → Optional[int]

Parameters

- **instruction_list** –
- **address** –

Returns

mythril.laser.ethereum.util.**get_trace_line**(*instr: Dict[KT, VT]*, *state: MachineState*) → str

Parameters

- **instr** –
- **state** –

Returns

mythril.laser.ethereum.util.**pop_bitvec**(*state: MachineState*) → mythril.laser.smt.bitvec.BitVec

Parameters **state** –

Returns

mythril.laser.ethereum.util.**safe_decode**(*hex_encoded_string: str*) → bytes

Parameters **hex_encoded_string** –

Returns

mythril.laser.ethereum.util.**to_signed**(*i: int*) → int

Parameters **i** –

Returns

Module contents

mythril.laser.smt package

Submodules

mythril.laser.smt.bitvec module

This module provides classes for an SMT abstraction of bit vectors.

class mythril.laser.smt.bitvec.**BitVec**(*raw: z3.z3.BitVecRef, annotations: Optional[Set[Any]] = None*)

Bases: *mythril.laser.smt.expression.Expression*

A bit vector symbol.

size() → int

TODO: documentation

Returns

symbolic

Returns whether this symbol doesn't have a concrete value.

Returns

value

Returns the value of this symbol if concrete, otherwise None.

Returns

mythril.laser.smt.bool module

This module provides classes for an SMT abstraction of boolean expressions.

mythril.laser.smt.bool.**And**(**args*) → mythril.laser.smt.bool.Bool

Create an And expression.

class mythril.laser.smt.bool.**Bool**(*raw: T, annotations: Optional[Set[Any]] = None*)

Bases: *mythril.laser.smt.expression.Expression*

This is a Bool expression.

is_false

Specifies whether this variable can be simplified to false.

Returns

is_true

Specifies whether this variable can be simplified to true.

Returns

value

Returns the concrete value of this bool if concrete, otherwise None.

Returns Concrete value or None

`mythril.laser.smt.bool.`**`Not`**(*a: mythril.laser.smt.bool.Bool*) → mythril.laser.smt.bool.Bool

Create a Not expression.

Parameters **`a`** –

Returns

`mythril.laser.smt.bool.`**`Or`**(**args*) → mythril.laser.smt.bool.Bool

Create an or expression.

Parameters

- **`a`** –
- **`b`** –

Returns

`mythril.laser.smt.bool.`**`Xor`**(*a: mythril.laser.smt.bool.Bool*, *b: mythril.laser.smt.bool.Bool*) → mythril.laser.smt.bool.Bool

Create an And expression.

`mythril.laser.smt.bool.`**`is_false`**(*a: mythril.laser.smt.bool.Bool*) → bool

Returns whether the provided bool can be simplified to false.

Parameters **`a`** –

Returns

`mythril.laser.smt.bool.`**`is_true`**(*a: mythril.laser.smt.bool.Bool*) → bool

Returns whether the provided bool can be simplified to true.

Parameters **`a`** –

Returns

mythril.laser.smt.expression module

This module contains the SMT abstraction for a basic symbol expression.

`class` `mythril.laser.smt.expression.`**`Expression`**(*raw: T*, *annotations: Optional[Set[Any]] = None*)

Bases: `typing.Generic`

This is the base symbol class and maintains functionality for simplification and annotations.

`annotate`(*annotation: Any*) → None

Annotates this expression with the given annotation.

Parameters **`annotation`** –

`annotations`

Gets the annotations for this expression.

Returns

`get_annotations`(*annotation: Any*)

`simplify`() → None

Simplify this expression.

`size`()

`mythril.laser.smt.expression.`**`simplify`**(*expression: G*) → G

Simplify the expression .

Parameters **`expression`** –

Returns

Module contents

class `mythril.laser.smt.`**`SymbolFactory`**

Bases: `typing.Generic`

A symbol factory provides a default interface for all the components of mythril to create symbols

static **`BitVecSym`**(*name: str*, *size: int*, *annotations: Optional[Set[Any]] = None*) → U

Creates a new bit vector with a symbolic value.

Parameters

- **`name`** – The name of the symbolic bit vector
- **`size`** – The size of the bit vector
- **`annotations`** – The annotations to initialize the bit vector with

Returns The freshly created bit vector

static **`BitVecVal`**(*value: int*, *size: int*, *annotations: Optional[Set[Any]] = None*) → U

Creates a new bit vector with a concrete value.

Parameters

- **`value`** – The concrete value to set the bit vector to
- **`size`** – The size of the bit vector
- **`annotations`** – The annotations to initialize the bit vector with

Returns The freshly created bit vector

static **`Bool`**(*value: __builtins__.bool*, *annotations: Optional[Set[Any]] = None*) → T

Creates a Bool with concrete value :param value: The boolean value :param annotations: The annotations to initialize the bool with :return: The freshly created Bool()

static **`BoolSym`**(*name: str*, *annotations: Optional[Set[Any]] = None*) → T

Creates a boolean symbol :param name: The name of the Bool variable :param annotations: The annotations to initialize the bool with :return: The freshly created Bool()

Module contents

6.1.6 mythril.solidity package

Submodules

mythril.solidity.soliditycontract module

This module contains representation classes for Solidity files, contracts and source mappings.

class `mythril.solidity.soliditycontract.`**`SolidityContract`**(*input_file*, *name=None*, *solc_settings_json=None*, *solc_binary='solc'*)

Bases: *`mythril.ethereum.evmcontract.EVMContract`*

Representation of a Solidity contract.

static get_full_contract_src_maps(*ast: Dict[KT, VT]*) → Set[str]
Takes a solc AST and gets the src mappings for all the contracts defined in the top level of the ast :param ast: AST of the contract :return: The source maps

static get_solc_indices(*data: Dict[KT, VT]*) → Dict[KT, VT]
Returns solc file indices

get_source_info(*address*, *constructor=False*)

Parameters

- **address** –
- **constructor** –

Returns

class mythril.solidity.soliditycontract.**SolidityFile**(*filename: str, data: str, full_contract_src_maps: Set[str]*)
Bases: object

Representation of a file containing Solidity code.

class mythril.solidity.soliditycontract.**SourceCodeInfo**(*filename*, *lineno*, *code*, *mapping*)
Bases: object

class mythril.solidity.soliditycontract.**SourceMapping**(*solidity_file_idx*, *offset*, *length*, *lineno*, *mapping*)
Bases: object

mythril.solidity.soliditycontract.**get_contracts_from_file**(*input_file*, *solc_settings_json=None*, *solc_binary='solc'*)

Parameters

- **input_file** –
- **solc_settings_json** –
- **solc_binary** –

Module contents

6.1.7 mythril.support package

Submodules

mythril.support.loader module

This module contains the dynamic loader logic to get on-chain storage data and dependencies.

class mythril.support.loader.**DynLoader**(*eth: Optional[mythril.ethereum.interface.rpc.client.EthJsonRpc]*, *active=True*)
Bases: object

The dynamic loader class.

dynld

Parameters **dependency_address** –

Returns

read_balance

Parameters **address** –

Returns

read_storage

Parameters

- **contract_address** –
- **index** –

Returns

mythril.support.signatures module

The Mythril function signature database.

class mythril.support.signatures.**SQLiteDB**(*path*)

Bases: object

Simple context manager for sqlite3 databases.

Commits everything at exit.

class mythril.support.signatures.**SignatureDB**(*enable_online_lookup: bool = False*, *path: str = None*)

Bases: object

add(*byte_sig: str*, *text_sig: str*) → None

Adds a new byte - text signature pair to the database. :param byte_sig: 4-byte signature string :param text_sig: resolved text signature :return:

get(*byte_sig: str*, *online_timeout: int = 2*) → List[str]

Get a function text signature for a byte signature 1) try local cache 2) try online lookup (if enabled; if not flagged as unavailable)

Parameters

- **byte_sig** – function signature hash as hexstr
- **online_timeout** – online lookup timeout

Returns list of matching function text signatures

import_solidity_file(*file_path: str*, *solc_binary: str = 'solc'*, *solc_settings_json: str = None*)

Import Function Signatures from solidity source files.

Parameters

- **solc_binary** –
- **solc_settings_json** –
- **file_path** – solidity source code file path

Returns

static lookup_online(*byte_sig: str*, *timeout: int*, *proxies=None*) → List[str]

Lookup function signatures from 4byte.directory.

Parameters

- **byte_sig** – function signature hash as hexstr
- **timeout** – optional timeout for online lookup
- **proxies** – optional proxy servers for online lookup

Returns a list of matching function signatures for this hash

class mythril.support.signatures.**Singleton**

Bases: type

A metaclass type implementing the singleton pattern.

mythril.support.signatures.**synchronized**(*sync_lock*)

A decorator synchronizing multi-process access to a resource.

mythril.support.support_utils module

This module contains utility functions for the Mythril support package.

class mythril.support.support_utils.**Singleton**

Bases: type

A metaclass type implementing the singleton pattern.

mythril.support.support_utils.**get_code_hash**(*code: str*) → str

Parameters **code** – bytecode

Returns Returns hash of the given bytecode

mythril.support.truffle module

Module contents

6.2 Submodules

6.3 mythril.exceptions module

This module contains general exceptions used by Mythril.

exception mythril.exceptions.**AddressNotFoundError**

Bases: *mythril.exceptions.MythrilBaseException*

A Mythril exception denoting the given smart contract address was not found.

exception mythril.exceptions.**CompilerError**

Bases: *mythril.exceptions.MythrilBaseException*

A Mythril exception denoting an error during code compilation.

exception mythril.exceptions.**CriticalError**

Bases: *mythril.exceptions.MythrilBaseException*

A Mythril exception denoting an unknown critical error has been encountered.

exception mythril.exceptions.**DetectorNotFoundError**

Bases: *mythril.exceptions.MythrilBaseException*

A Mythril exception denoting attempted usage of a non-existant detection module.

exception `mythril.exceptions.`**`MythrilBaseException`**

Bases: `Exception`

The Mythril exception base type.

exception `mythril.exceptions.`**`NoContractFoundError`**

Bases: *`mythril.exceptions.MythrilBaseException`*

A Mythril exception denoting that a given contract file was not found.

exception `mythril.exceptions.`**`UnsatError`**

Bases: *`mythril.exceptions.MythrilBaseException`*

A Mythril exception denoting the unsatisfiability of a series of constraints.

6.4 mythril.mythril module

6.5 mythril.version module

6.6 Module contents

CHAPTER 7

Indices and Tables

- genindex
- modindex
- search

Python Module Index

Index