Mythril Documentation

Release v0.23.25

ConsenSys Dilligence

Sep 10, 2023

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 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

Tutorial

3.1 Introduction

Mythril is a popular security analysis tool for smart contracts. It is an open-source tool that can analyze Ethereum smart contracts and report potential security vulnerabilities in them. By analyzing the bytecode of a smart contract, Mythril can identify and report on possible security vulnerabilities, such as reentrancy attacks, integer overflows, and other common smart contract vulnerabilities. This tutorial explains how to use Mythril to analyze simple Solidity contracts for security vulnerabilities. A simple contract is one that does not have any imports.

3.2 Executing Mythril on Simple Contracts

To start, we consider this simple contract, `Exceptions`, which has a number of functions, including `assert1()`, `assert2()`, and `assert3()`, that contain Solidity `assert()` statements. We will use Mythril to analyze this contract and report any potential vulnerabilities.

```
contract Exceptions {

    uint256[8] myarray;
    uint counter = 0;
    function assert1() public pure {
        uint256 i = 1;
        assert(i == 0);
    }
    function counter_increase() public {
        counter+=1;
    }
    function assert5(uint input_x) public view{
        require(counter>2);
        assert(input_x > 10);
    }
    function assert2() public pure {
```

(continues on next page)

(continued from previous page)

```
        uint256 i = 1;
        assert(i > 0);
    }

    function assert3(uint256 input) public pure {
        assert(input != 23);
    }

    function require_is_fine(uint256 input) public pure {
        require(input != 23);
    }

    function this_is_fine(uint256 input) public pure {
        if (input > 0) {
            uint256 i = 1/input;
        }
    }

    function this_is_find_2(uint256 index) public view {
        if (index < 8) {
            uint256 i = myarray[index];
        }
    }

}
```

The sample contract has several functions, some of which contain vulnerabilities. For instance, the `assert1()` function contains an assertion violation. To analyze the contract using Mythril, the following command can be used:

```
$ myth analyze <file_path>
```

The output will show the vulnerabilities in the contract. In the case of the "Exceptions" contract, Mythril detected two instances of assertion violations.

```
==== Exception State ====
SWC ID: 110
Severity: Medium
Contract: Exceptions
Function name: assert1()
PC address: 708
Estimated Gas Usage: 207 - 492
An assertion violation was triggered.
It is possible to trigger an assertion violation. Note that Solidity␣
↪assert() statements should only be used to check invariants. Review the␣
↪transaction trace generated for this issue and either make sure your␣
↪program logic is correct, or use require() instead of assert() if your␣
↪goal is to constrain user inputs or enforce preconditions. Remember to␣
↪validate inputs from both callers (for instance, via passed arguments) and␣
↪callees (for instance, via return values).
--------------------
In file: solidity_examples/exceptions.sol:7

assert(i == 0)

--------------------
Initial State:
```

(continues on next page)

(continued from previous page)

```
Account: [CREATOR], balance: 0x2, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [ATTACKER], function: assert1(), txdata: 0xb34c3610, value: 0x0

==== Exception State ====
SWC ID: 110
Severity: Medium
Contract: Exceptions
Function name: assert3(uint256)
PC address: 708
Estimated Gas Usage: 482 - 767
An assertion violation was triggered.
It is possible to trigger an assertion violation. Note that Solidity␣
↪assert() statements should only be used to check invariants. Review the␣
↪transaction trace generated for this issue and either make sure your␣
↪program logic is correct, or use require() instead of assert() if your␣
↪goal is to constrain user inputs or enforce preconditions. Remember to␣
↪validate inputs from both callers (for instance, via passed arguments) and␣
↪callees (for instance, via return values).
--------------------
In file: solidity_examples/exceptions.sol:20

assert(input != 23)

--------------------
Initial State:

Account: [CREATOR], balance: 0x40207f9b0, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [SOMEGUY], function: assert3(uint256), txdata:␣
↪0x546455b500000000000000000000000000000000000000000000000000000000000000017,
↪ value: 0x0
```

One of the functions, `assert5(uint256)`, should also have an assertion failure, but it is not detected because Mythril's default configuration is to run three transactions. To detect this vulnerability, the transaction count can be increased to four using the `-t` option, as shown below:

```
$ myth analyze <file_path> -t 4
```

This gives the following execution output:

```
==== Exception State ====
SWC ID: 110
Severity: Medium
Contract: Exceptions
Function name: assert1()
PC address: 731
```

(continues on next page)

(continued from previous page)

```
Estimated Gas Usage: 207 - 492
An assertion violation was triggered.
It is possible to trigger an assertion violation. Note that Solidity␣
↪assert() statements should only be used to check invariants. Review the␣
↪transaction trace generated for this issue and either make sure your␣
↪program logic is correct, or use require() instead of assert() if your␣
↪goal is to constrain user inputs or enforce preconditions. Remember to␣
↪validate inputs from both callers (for instance, via passed arguments) and␣
↪callees (for instance, via return values).
--------------------
In file: solidity_examples/exceptions.sol:7

assert(i == 0)

--------------------
Initial State:

Account: [CREATOR], balance: 0x2, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [ATTACKER], function: assert1(), txdata: 0xb34c3610, value: 0x0

==== Exception State ====
SWC ID: 110
Severity: Medium
Contract: Exceptions
Function name: assert3(uint256)
PC address: 731
Estimated Gas Usage: 504 - 789
An assertion violation was triggered.
It is possible to trigger an assertion violation. Note that Solidity␣
↪assert() statements should only be used to check invariants. Review the␣
↪transaction trace generated for this issue and either make sure your␣
↪program logic is correct, or use require() instead of assert() if your␣
↪goal is to constrain user inputs or enforce preconditions. Remember to␣
↪validate inputs from both callers (for instance, via passed arguments) and␣
↪callees (for instance, via return values).
--------------------
In file: solidity_examples/exceptions.sol:22

assert(input != 23)

--------------------
Initial State:

Account: [CREATOR], balance: 0x3, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [ATTACKER], function: assert3(uint256), txdata:␣
↪0x546455b50000000000000000000000000000000000000000000000000000000000000017,
↪ value: 0x0
```

(continues on next page)

(continued from previous page)

```
==== Exception State ====
SWC ID: 110
Severity: Medium
Contract: Exceptions
Function name: assert5(uint256)
PC address: 731
Estimated Gas Usage: 1302 - 1587
An assertion violation was triggered.
It is possible to trigger an assertion violation. Note that Solidity␣
↪assert() statements should only be used to check invariants. Review the␣
↪transaction trace generated for this issue and either make sure your␣
↪program logic is correct, or use require() instead of assert() if your␣
↪goal is to constrain user inputs or enforce preconditions. Remember to␣
↪validate inputs from both callers (for instance, via passed arguments) and␣
↪callees (for instance, via return values).
--------------------
In file: solidity_examples/exceptions.sol:14

assert(input_x > 10)

--------------------
Initial State:

Account: [CREATOR], balance: 0x20000000, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x1000000, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], calldata: , value: 0x0
Caller: [ATTACKER], function: counter_increase(), txdata: 0xe47b0253, value:␣
↪0x0
Caller: [CREATOR], function: counter_increase(), txdata: 0xe47b0253, value:␣
↪0x0
Caller: [CREATOR], function: counter_increase(), txdata: 0xe47b0253, value:␣
↪0x0
Caller: [ATTACKER], function: assert5(uint256), txdata:␣
↪0x1d5d53dd0000000000000000000000000000000000000000000000000000000000000003,
↪ value: 0x0
```

For the violation in the 4th transaction, the input value should be less than 10. The transaction data generated by Mythril for the 4th transaction is `0x1d5d53dd0000000000000000000000000000000000000000000000000000000000000003`, the first 4 bytes `1d5d53dd` correspond to the function signature hence the input generated by Mythril is `0000000000000000000000000000000000000000000000000000000000000003` in hex, which is 3. For automated resolution of the input try using a different output format such as JSON.

```
$ myth analyze <file_path> -o json
```

This leads to the following output:

```
{
    "error": null,
    "issues": [{
        "address": 731,
        "code": "assert(i == 0)",
```

(continues on next page)

(continued from previous page)


```
        "contract": "Exceptions",
        "description": "An assertion violation was triggered.\nIt is␣
↪possible to trigger an assertion violation. Note that Solidity assert()␣
↪statements should only be used to check invariants. Review the transaction␣
↪trace generated for this issue and either make sure your program logic is␣
↪correct, or use require() instead of assert() if your goal is to constrain␣
↪user inputs or enforce preconditions. Remember to validate inputs from␣
↪both callers (for instance, via passed arguments) and callees (for␣
↪instance, via return values).",
        "filename": "solidity_examples/exceptions.sol",
        "function": "assert1()",
        "lineno": 7,
        "max_gas_used": 492,
        "min_gas_used": 207,
        "severity": "Medium",
        "sourceMap": ":::i",
        "swc-id": "110",
        "title": "Exception State",
        "tx_sequence": {
            "initialState": {
                "accounts": {
                    "0xaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe": {
                        "balance": "0x2",
                        "code": "",
                        "nonce": 0,
                        "storage": "{}"
                    },
                    "0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef": {
                        "balance": "0x0",
                        "code": "",
                        "nonce": 0,
                        "storage": "{}"
                    }
                }
            },
            "steps": [{
                "address": "",
                "calldata": "",
                "input":
↪"0x6080604052600060085534801561001557600080fd5b506103f7806100256000396000f3fe6080604052348015
↪",
                "name": "unknown",
                "origin": "0xaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe",
                "value": "0x0"
            }, {
                "address": "0x901d12ebe1b195e5aa8748e62bd7734ae19b51f",
                "calldata": "0xb34c3610",
                "input": "0xb34c3610",
                "name": "assert1()",
                "origin": "0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef",
                "resolved_input": null,
                "value": "0x0"
            }]
        }
    }, {
        "address": 731,
        "code": "assert(input != 23)",
```


(continues on next page)

(continued from previous page)


```
        "contract": "Exceptions",
        "description": "An assertion violation was triggered.\nIt is␣
↪possible to trigger an assertion violation. Note that Solidity assert()␣
↪statements should only be used to check invariants. Review the transaction␣
↪trace generated for this issue and either make sure your program logic is␣
↪correct, or use require() instead of assert() if your goal is to constrain␣
↪user inputs or enforce preconditions. Remember to validate inputs from␣
↪both callers (for instance, via passed arguments) and callees (for␣
↪instance, via return values).",
        "filename": "solidity_examples/exceptions.sol",
        "function": "assert3(uint256)",
        "lineno": 22,
        "max_gas_used": 789,
        "min_gas_used": 504,
        "severity": "Medium",
        "sourceMap": ":::i",
        "swc-id": "110",
        "title": "Exception State",
        "tx_sequence": {
            "initialState": {
                "accounts": {
                    "0xaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe": {
                        "balance": "0x3",
                        "code": "",
                        "nonce": 0,
                        "storage": "{}"
                    },
                    "0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef": {
                        "balance": "0x0",
                        "code": "",
                        "nonce": 0,
                        "storage": "{}"
                    }
                }
            },
            "steps": [{
                "address": "",
                "calldata": "",
                "input":
↪"0x6080604052600060085534801561001557600080fd5b506103f780610025600039600000f3fe60806040523480156
↪",
                "name": "unknown",
                "origin": "0xaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe",
                "value": "0x0"
            }, {
                "address": "0x901d12ebe1b195e5aa8748e62bd7734ae19b51f",
                "calldata":
↪"0x546455b5000000000000000000000000000000000000000000000000000000000000000017
↪",
                "input":
↪"0x546455b5000000000000000000000000000000000000000000000000000000000000000017
↪",
                "name": "assert3(uint256)",
                "origin": "0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef",
                "resolved_input": [23],
                "value": "0x0"
            }]
```


(continues on next page)

(continued from previous page)


```
        }
    }, {
        "address": 731,
        "code": "assert(input_x > 10)",
        "contract": "Exceptions",
        "description": "An assertion violation was triggered.\nIt is␣
↪possible to trigger an assertion violation. Note that Solidity assert()␣
↪statements should only be used to check invariants. Review the transaction␣
↪trace generated for this issue and either make sure your program logic is␣
↪correct, or use require() instead of assert() if your goal is to constrain␣
↪user inputs or enforce preconditions. Remember to validate inputs from␣
↪both callers (for instance, via passed arguments) and callees (for␣
↪instance, via return values).",
        "filename": "solidity_examples/exceptions.sol",
        "function": "assert5(uint256)",
        "lineno": 14,
        "max_gas_used": 1587,
        "min_gas_used": 1302,
        "severity": "Medium",
        "sourceMap": ":::i",
        "swc-id": "110",
        "title": "Exception State",
        "tx_sequence": {
            "initialState": {
                "accounts": {
                    "0xaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe": {
                        "balance": "0x0",
                        "code": "",
                        "nonce": 0,
                        "storage": "{}"
                    },
                    "0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef": {
                        "balance": "0x0",
                        "code": "",
                        "nonce": 0,
                        "storage": "{}"
                    }
                }
            },
            "steps": [{
                "address": "",
                "calldata": "",
                "input":
↪"0x6080604052600060085534801561001557600080fd5b506103f7806100256000396000f3fe60806040523480156
↪",
                "name": "unknown",
                "origin": "0xaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe",
                "value": "0x0"
            }, {
                "address": "0x901d12ebe1b195e5aa8748e62bd7734ae19b51f",
                "calldata": "0xe47b0253",
                "input": "0xe47b0253",
                "name": "counter_increase()",
                "origin": "0xaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe",
                "resolved_input": null,
                "value": "0x0"
            }, {
```


(continues on next page)

(continued from previous page)


```
                "address": "0x901d12ebe1b195e5aa8748e62bd7734ae19b51f",
                "calldata": "0xe47b0253",
                "input": "0xe47b0253",
                "name": "counter_increase()",
                "origin": "0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef",
                "resolved_input": null,
                "value": "0x0"
            }, {
                "address": "0x901d12ebe1b195e5aa8748e62bd7734ae19b51f",
                "calldata": "0xe47b0253",
                "input": "0xe47b0253",
                "name": "counter_increase()",
                "origin": "0xaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa",
                "resolved_input": null,
                "value": "0x0"
            }, {
                "address": "0x901d12ebe1b195e5aa8748e62bd7734ae19b51f",
                "calldata":
↪"0x1d5d53dd0000000000000000000000000000000000000000000000000000000000000003
↪",
                "input":
↪"0x1d5d53dd0000000000000000000000000000000000000000000000000000000000000003
↪",
                "name": "assert5(uint256)",
                "origin": "0xdeadbeefdeadbeefdeadbeefdeadbeefdeadbeef",
                "resolved_input": [3],
                "value": "0x0"
            }]
        }
    }],
    "success": true
}
```


We can observe that the "resolved_input" field for the final transaction resolves to `[3]`. Although this resolution fails in some circumstances where output generated by Mythril is although executable on the bytecode, it cannot be decoded due to not being a valid ABI.

There are interesting options such as `--execution-timeout <seconds>` and `--solver-timeout <milliseconds>` which can be increased for better results. The default execution-timeout and solver-timeout are 86400 seconds and 25000 milliseconds respectively.

3.3 Executing Mythril on Contracts with Imports

When using Mythril to analyze a Solidity contract, you may encounter issues related to import statements. Solidity does not have access to the import locations, which can result in errors when compiling the program. In order to provide import information to Solidity, you can use the remappings option in Mythril.

Consider the following Solidity contract, which imports the PRC20 contract from the `@openzeppelin/contracts/token/PRC20/PRC20.sol` file:

```
import "@openzeppelin/contracts/token/PRC20/PRC20.sol";

contract Nothing is PRC20{
    string x_0 = "";
```

(continues on next page)

(continued from previous page)

```
    bytes3 x_1 = "A";

    bytes5 x_2 = "E";

    bytes5 x_3 = "";

    bytes3 x_4 = "I";

    bytes3 x_5 = "U";

    bytes3 x_6 = "O";

    bytes3 x_7 = "0";

    bytes3 x_8 = "U";

    bytes3 x_9 = "U";
    function stringCompare(string memory a, string memory b) internal pure␣
↪returns (bool) {
        if(bytes(a).length != bytes(b).length) {
            return false;
        } else {
            return keccak256(bytes(a)) == keccak256(bytes(b));
        }
    }

    function nothing(string memory g_0, bytes3 g_5, bytes3 g_6, bytes3 g_7,␣
↪bytes3 g_8, bytes3 g_9, bytes3 g_10, bytes3 g_11) public view returns␣
↪(bool){
        if (!stringCompare(g_0, x_0)) return false;


        if (g_5 != x_5) return false;

        if (g_6 != x_6) return false;

        if (g_7 != x_7) return false;

        if (g_8 != x_8) return false;

        if (g_9 != x_9) return false;

        if (g_10 != x_9) return false;

        if (g_11 != x_9) return false;

        return true;

    }
}
```

When this contract is directly executed by using the following command:

```
$ myth analyze <file_path>
```

We encounter the following error:

```
mythril.interfaces.cli [ERROR]: Solc experienced a fatal error.

ParserError: Source "@openzeppelin/contracts/token/PRC20/PRC20.sol" not␣
↪found: File not found. Searched the following locations: "".
--> <file_path>:1:1:
|
1 | import "@openzeppelin/contracts/token/PRC20/PRC20.sol";
| ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

This error occurs because Solidity cannot locate the `PRC20.sol` file. To solve this issue, you need to provide remapping information to Mythril, which will relay it to the Solidity compiler. Remapping involves mapping an import statement to the path that contains the corresponding file.

In this example, we can map the import statement `@openzeppelin/contracts/token/PRC20/` to the path that contains `PRC20.sol`. Let's assume that the file is located at `node_modules/PRC20/PRC20.sol`. We can provide the remapping information to Mythril in a JSON file like this:

```
{
"remappings": [ "@openzeppelin/contracts/token/PRC20/=node_modules/PRC20/"]
}
```

This JSON file maps the prefix `@openzeppelin/contracts/token/PRC20/` to the path `node_modules/PRC20/` in the file system. When you run Mythril, you can use the `--solc-json` option to provide the remapping file:

```
$ myth analyze {file_path} --solc-json {json_file_path}
```

With this command, Mythril will be able to locate the `PRC20.sol` file, and the analysis should proceed without errors.

For more information on remappings, you can refer to the Solidity documentation.

3.4 Executing Mythril by Restricting Transaction Sequences

Mythril is a security analysis tool that can be used to search certain transaction sequences. The *–transaction-sequences* argument can be used to direct the search. You should provide a list of transactions that are sequenced in the same order that they will be executed in the contract. For example, suppose you want to find vulnerabilities in a contract that executes three transactions, where the first transaction is constrained with `func_hash1` and `func_hash2`, the second transaction is constrained with `func_hash2` and `func_hash3`, and the final transaction is unconstrained on any function. You would provide `--transaction-sequences [[func_hash1, func_hash2], [func_hash2,func_hash3],[]]` as an argument to Mythril.

You can use `-1` as a proxy for the hash of the *fallback()* function and `-2` as a proxy for the hash of the `receive()` function.

Here is an example contract that demonstrates how to use Mythril with `--transaction-sequences`.

Consider the following contract:

```
pragma solidity ^0.5.0;


contract Rubixi {
    //Declare variables for storage critical to contract
    uint private balance = 0;
    uint private collectedFees = 0;
```

(continues on next page)

(continued from previous page)

```
    uint private feePercent = 10;
    uint private pyramidMultiplier = 300;
    uint private payoutOrder = 0;

    address payable private creator;

    modifier onlyowner {
        if (msg.sender == creator) _;
    }

    struct Participant {
        address payable etherAddress;
        uint payout;
    }

    //Fallback function
    function() external payable {
        init();
    }

    //Sets creator
    function dynamicPyramid() public {
        creator = msg.sender;
    }

    Participant[] private participants;

    //Fee functions for creator
    function collectAllFees() public onlyowner {
        require(collectedFees > 0);
        creator.transfer(collectedFees);
        collectedFees = 0;
    }

    function collectFeesInEther(uint _amt) public onlyowner {
        _amt *= 1 ether;
        if (_amt > collectedFees) collectAllFees();

        require(collectedFees > 0);

        creator.transfer(_amt);
        collectedFees -= _amt;
    }

    function collectPercentOfFees(uint _pcent) public onlyowner {
        require(collectedFees > 0 && _pcent <= 100);

        uint feesToCollect = collectedFees / 100 * _pcent;
        creator.transfer(feesToCollect);
        collectedFees -= feesToCollect;
    }

    //Functions for changing variables related to the contract
    function changeOwner(address payable _owner) public onlyowner {
        creator = _owner;
    }
```

(continues on next page)

(continued from previous page)

```
    function changeMultiplier(uint _mult) public onlyowner {
        require(_mult <= 300 &&  _mult >= 120);
        pyramidMultiplier = _mult;
    }

    function changeFeePercentage(uint _fee) public onlyowner {
        require(_fee <= 10);
        feePercent = _fee;
    }

    //Functions to provide information to end-user using JSON interface or␣
↪other interfaces
    function currentMultiplier() public view returns (uint multiplier,␣
↪string memory info) {
        multiplier = pyramidMultiplier;
        info = "This multiplier applies to you as soon as transaction is␣
↪received, may be lowered to hasten payouts or increased if payouts are␣
↪fast enough. Due to no float or decimals, multiplier is x100 for a␣
↪fractional multiplier e.g. 250 is actually a 2.5x multiplier. Capped at 3x␣
↪max and 1.2x min.";
    }

    function currentFeePercentage() public view returns (uint fee, string␣
↪memory info) {
        fee = feePercent;
        info = "Shown in % form. Fee is halved(50%) for amounts equal or␣
↪greater than 50 ethers. (Fee may change, but is capped to a maximum of 10%)
↪";
    }

    function currentPyramidBalanceApproximately() public view returns (uint␣
↪pyramidBalance, string memory info) {
        pyramidBalance = balance / 1 ether;
        info = "All balance values are measured in Ethers, note that due to␣
↪no decimal placing, these values show up as integers only, within the␣
↪contract itself you will get the exact decimal value you are supposed to";
    }

    function nextPayoutWhenPyramidBalanceTotalsApproximately() public view␣
↪returns (uint balancePayout) {
        balancePayout = participants[payoutOrder].payout / 1 ether;
    }

    function feesSeperateFromBalanceApproximately() public view returns␣
↪(uint fees) {
        fees = collectedFees / 1 ether;
    }

    function totalParticipants() public view returns (uint count) {
        count = participants.length;
    }

    function numberOfParticipantsWaitingForPayout() public view returns␣
↪(uint count) {
        count = participants.length - payoutOrder;
    }
```

(continues on next page)

(continued from previous page)

```
    function participantDetails(uint orderInPyramid) public view returns␣
↪(address addr, uint payout) {
        if (orderInPyramid <= participants.length) {
            addr = participants[orderInPyramid].etherAddress;
            payout = participants[orderInPyramid].payout / 1 ether;
        }
    }

    //init function run on fallback
    function init() private {
        //Ensures only tx with value of 1 ether or greater are processed and␣
↪added to pyramid
        if (msg.value < 1 ether) {
            collectedFees += msg.value;
            return;
        }

        uint _fee = feePercent;
        // 50% fee rebate on any ether value of 50 or greater
        if (msg.value >= 50 ether) _fee /= 2;

        addPayout(_fee);
    }

    //Function called for valid tx to the contract
    function addPayout(uint _fee) private {
        //Adds new address to participant array
        participants.push(Participant(msg.sender, (msg.value *␣
↪pyramidMultiplier) / 100));

        // These statements ensure a quicker payout system to
        // later pyramid entrants, so the pyramid has a longer lifespan
        if (participants.length == 10) pyramidMultiplier = 200;
        else if (participants.length == 25) pyramidMultiplier = 150;

        // collect fees and update contract balance
        balance += (msg.value * (100 - _fee)) / 100;
        collectedFees += (msg.value * _fee) / 100;

        //Pays earlier participiants if balance sufficient
        while (balance > participants[payoutOrder].payout) {
            uint payoutToSend = participants[payoutOrder].payout;
            participants[payoutOrder].etherAddress.transfer(payoutToSend);

            balance -= participants[payoutOrder].payout;
            payoutOrder += 1;
        }
    }
}
```

Since this contract has `16` functions, it is infeasible to execute uninteresting functions such as `feesSeperateFromBalanceApproximately()`. To successfully explore useful transaction sequences we can use Mythril's `--transaction-sequences` argument.

```
$ myth analyze rubixi.sol -t 3 --transaction-sequences [["0x89b8ae9b"],[-1],[
↪"0x686f2c90","0xb4022950","0x4229616d"]]
```

The first transaction is constrained to the function `dynamicPyramid()`, the second one to the `fallback()` function, and finally, the third transaction is constrained to‘‘collectAllFees()‘‘, `collectFeesInEther(uint256)` and `collectPercentOfFees(uint256)`. Make sure to use `-t 3` argument, since the length of the transaction sequence should match with the transaction count argument.

CHAPTER 4

Security Analysis

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

4.1 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
```

4.1.1 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.

4.1.2 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.

4.2 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.

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

4.3 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 5

Analysis Modules

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

5.1 Modules

5.1.1 Delegate Call To Untrusted Contract

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

5.1.2 Dependence on Predictable Variables

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

5.1.3 Ether Thief

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

5.1.4 Exceptions

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

5.1.5 External Calls

The external calls module warns about SWC-107 (Reentrancy) by detecting calls to external contracts.

5.1.6 Integer

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

5.1.7 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.

5.1.8 Suicide

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

5.1.9 State Change External Calls

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

5.1.10 Unchecked Retval

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

5.1.11 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)`.

5.1.12 Arbitrary Storage Write

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

5.1.13 Arbitrary Jump

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

5.2 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 6

mythril package

6.1 Subpackages

6.1.1 mythril.analysis package

Subpackages

mythril.analysis.module package

Subpackages

mythril.analysis.module.modules package

Submodules

mythril.analysis.module.modules.arbitrary_jump module

This module contains the detection code for Arbitrary jumps.

class mythril.analysis.module.modules.arbitrary_jump.**ArbitraryJump**
Bases: *mythril.analysis.module.base.DetectionModule*

This module searches for JUMPs to a user-specified location.

description = '\n\nSearch for jumps to arbitrary locations in the bytecode\n'

entry_point = 2

name = 'Caller can redirect execution to arbitrary bytecode locations'

pre_hooks = ['JUMP', 'JUMPI']

reset_module()
Resets the module by clearing everything :return:

swc_id = '127'

mythril.analysis.module.modules.arbitrary_jump.**is_unique_jumpdest**(*jump_dest: mythril.laser.smt.bitvec.BitVec*, *state: mythril.laser.ethereum.state.global_s* → bool

Handles cases where jump_dest evaluates to a single concrete value

mythril.analysis.module.modules.arbitrary_write module

This module contains the detection code for arbitrary storage write.

class mythril.analysis.module.modules.arbitrary_write.**ArbitraryStorage**
Bases: *mythril.analysis.module.base.DetectionModule*

This module searches for a feasible write to an arbitrary storage slot.

description = '\n\nSearch for any writes to an arbitrary storage slot\n'

entry_point = 2

name = 'Caller can write to arbitrary storage locations'

pre_hooks = ['SSTORE']

reset_module()
Resets the module by clearing everything :return:

swc_id = '124'

mythril.analysis.module.modules.delegatecall module

This module contains the detection code for insecure delegate call usage.

class mythril.analysis.module.modules.delegatecall.**ArbitraryDelegateCall**
Bases: *mythril.analysis.module.base.DetectionModule*

This module detects delegatecall to a user-supplied address.

description = 'Check for invocations of delegatecall to a user-supplied address.'

entry_point = 2

name = 'Delegatecall to a user-specified address'

pre_hooks = ['DELEGATECALL']

swc_id = '112'

mythril.analysis.module.modules.dependence_on_origin module

This module contains the detection code for predictable variable dependence.

class mythril.analysis.module.modules.dependence_on_origin.**TxOrigin**
Bases: *mythril.analysis.module.base.DetectionModule*

This module detects whether control flow decisions are made based on the transaction origin.

description = 'Check whether control flow decisions are influenced by tx.origin'

entry_point = 2

name = 'Control flow depends on tx.origin'

post_hooks = ['ORIGIN']

pre_hooks = ['JUMPI']

swc_id = '115'

class mythril.analysis.module.modules.dependence_on_origin.**TxOriginAnnotation**
Bases: object

Symbol annotation added to a variable that is initialized with a call to the ORIGIN instruction.

mythril.analysis.module.modules.dependence_on_predictable_vars module

This module contains the detection code for predictable variable dependence.

class mythril.analysis.module.modules.dependence_on_predictable_vars.**OldBlockNumberUsedAnno**
Bases: *mythril.laser.ethereum.state.annotation.StateAnnotation*

Symbol annotation used if a variable is initialized from a predictable environment variable.

class mythril.analysis.module.modules.dependence_on_predictable_vars.**PredictableValueAnnota**

Bases: object

Symbol annotation used if a variable is initialized from a predictable environment variable.

class mythril.analysis.module.modules.dependence_on_predictable_vars.**PredictableVariables**
Bases: *mythril.analysis.module.base.DetectionModule*

This module detects whether control flow decisions are made using predictable parameters.

description = 'Check whether control flow decisions are influenced by block.coinbase,b

entry_point = 2

name = 'Control flow depends on a predictable environment variable'

post_hooks = ['BLOCKHASH', 'COINBASE', 'GASLIMIT', 'TIMESTAMP', 'NUMBER']

pre_hooks = ['JUMPI', 'BLOCKHASH']

swc_id = '116 120'

mythril.analysis.module.modules.ether_thief module

This module contains the detection code for unauthorized ether withdrawal.

class mythril.analysis.module.modules.ether_thief.**EtherThief**
Bases: *mythril.analysis.module.base.DetectionModule*

This module search for cases where Ether can be withdrawn to a user- specified address.

description = '\nSearch for cases where Ether can be withdrawn to a user-specified addr

entry_point = 2

name = 'Any sender can withdraw ETH from the contract account'

post_hooks = ['CALL', 'STATICCALL']

reset_module()
Resets the module by clearing everything :return:

swc_id = '105'

mythril.analysis.module.modules.exceptions module

This module contains the detection code for reachable exceptions.

class mythril.analysis.module.modules.exceptions.**Exceptions**
Bases: *mythril.analysis.module.base.DetectionModule*

description = 'Checks whether any exception states are reachable.'

entry_point = 2

name = 'Assertion violation'

pre_hooks = ['INVALID', 'JUMP', 'REVERT']

swc_id = '110'

class mythril.analysis.module.modules.exceptions.**LastJumpAnnotation**(*last_jump: Optional[int] = None*)
Bases: *mythril.laser.ethereum.state.annotation.StateAnnotation*

State Annotation used if an overflow is both possible and used in the annotated path

mythril.analysis.module.modules.exceptions.**is_assertion_failure**(*global_state*)

mythril.analysis.module.modules.external_calls module

This module contains the detection code for potentially insecure low-level calls.

class mythril.analysis.module.modules.external_calls.**ExternalCalls**
Bases: *mythril.analysis.module.base.DetectionModule*

This module searches for low level calls (e.g. call.value()) that forward all gas to the callee.

description = '\n\nSearch for external calls with unrestricted gas to a user-specified

entry_point = 2

name = 'External call to another contract'

pre_hooks = ['CALL']

swc_id = '107'

mythril.analysis.module.modules.integer module

This module contains the detection code for integer overflows and underflows.

class mythril.analysis.module.modules.integer.**IntegerArithmetics**
Bases: *mythril.analysis.module.base.DetectionModule*

This module searches for integer over- and underflows.

description = "For every SUB instruction, check if there's a possible state where op1 :

entry_point = 2

name = 'Integer overflow or underflow'

pre_hooks = ['ADD', 'MUL', 'EXP', 'SUB', 'SSTORE', 'JUMPI', 'STOP', 'RETURN', 'CALL']

reset_module()
Resets the module :return:

swc_id = '101'

class mythril.analysis.module.modules.integer.**OverUnderflowAnnotation**(*overflowing_state: mythril.laser.ethereum.state.gl*, *operator: str*, *constraint: mythril.laser.smt.bool.Bool*)

Bases: object

Symbol Annotation used if a BitVector can overflow

class mythril.analysis.module.modules.integer.**OverUnderflowStateAnnotation**

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

State Annotation used if an overflow is both possible and used in the annotated path

mythril.analysis.module.modules.multiple_sends module

This module contains the detection code to find multiple sends occurring in a single transaction.

class mythril.analysis.module.modules.multiple_sends.**MultipleSends**

Bases: *mythril.analysis.module.base.DetectionModule*

This module checks for multiple sends in a single transaction.

description = 'Check for multiple sends in a single transaction'

entry_point = 2

name = 'Multiple external calls in the same transaction'

pre_hooks = ['CALL', 'DELEGATECALL', 'STATICCALL', 'CALLCODE', 'RETURN', 'STOP']

swc_id = '113'

class mythril.analysis.module.modules.multiple_sends.**MultipleSendsAnnotation**

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

mythril.analysis.module.modules.state_change_external_calls module

class mythril.analysis.module.modules.state_change_external_calls.**StateChangeAfterCall**

Bases: *mythril.analysis.module.base.DetectionModule*

This module searches for state change after low level calls (e.g. call.value()) that forward gas to the callee.

description = '\n\nCheck whether the account state is accesses after the execution of

entry_point = 2

name = 'State change after an external call'

pre_hooks = ['CALL', 'DELEGATECALL', 'CALLCODE', 'SSTORE', 'SLOAD', 'CREATE', 'CREATE2

swc_id = '107'

class mythril.analysis.module.modules.state_change_external_calls.**StateChangeCallsAnnotatio**

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

get_issue(*global_state: mythril.laser.ethereum.state.global_state.GlobalState*, *detector: mythril.analysis.module.base.DetectionModule*) → Optional[mythril.analysis.potential_issues.PotentialIssue]

mythril.analysis.module.modules.suicide module

class mythril.analysis.module.modules.suicide.**AccidentallyKillable**

Bases: *mythril.analysis.module.base.DetectionModule*

This module checks if the contact can be 'accidentally' killed by anyone.

description = "\nCheck if the contact can be 'accidentally' killed by anyone.\nFor kil

entry_point = 2

name = 'Contract can be accidentally killed by anyone'

pre_hooks = ['SELFDESTRUCT']

reset_module()

Resets the module :return:

swc_id = '106'

mythril.analysis.module.modules.unchecked_retval module

This module contains detection code to find occurrences of calls whose return value remains unchecked.

class mythril.analysis.module.modules.unchecked_retval.**RetVal**

Bases: dict

class mythril.analysis.module.modules.unchecked_retval.**UncheckedRetval**

Bases: *mythril.analysis.module.base.DetectionModule*

A detection module to test whether CALL return value is checked.

description = 'Test whether CALL return value is checked. For direct calls, the Solidi

entry_point = 2

name = 'Return value of an external call is not checked'

post_hooks = ['CALL', 'DELEGATECALL', 'STATICCALL', 'CALLCODE']

pre_hooks = ['STOP', 'RETURN']

swc_id = '104'

class mythril.analysis.module.modules.unchecked_retval.**UncheckedRetvalAnnotation**

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

mythril.analysis.module.modules.user_assertions module

This module contains the detection code for potentially insecure low-level calls.

class mythril.analysis.module.modules.user_assertions.**UserAssertions**
Bases: *mythril.analysis.module.base.DetectionModule*

This module searches for user supplied exceptions: emit AssertionFailed("Error").

description = "\n\nSearch for reachable user-supplied exceptions.\nReport a warning if

entry_point = 2

name = 'A user-defined assertion has been triggered'

pre_hooks = ['LOG1', 'MSTORE']

swc_id = '110'

Module contents

Submodules

mythril.analysis.module.base module

Mythril Detection Modules

This module includes an definition of the DetectionModule interface. DetectionModules implement different analysis rules to find weaknesses and vulnerabilities.

class mythril.analysis.module.base.**DetectionModule**
Bases: abc.ABC

The base detection module.

All custom-built detection modules must inherit from this class.

There are several class properties that expose information about the detection modules

Parameters

- **name** – The name of the detection module
- **swc_id** – The SWC ID associated with the weakness that the module detects
- **description** – A description of the detection module, and what it detects
- **entry_point** – Mythril can run callback style detection modules, or modules that search the statespace. [IMPORTANT] POST entry points severely slow down the analysis, try to always use callback style modules
- **pre_hooks** – A list of instructions to hook the laser vm for (pre execution of the instruction)
- **post_hooks** – A list of instructions to hook the laser vm for (post execution of the instruction)

description = 'Detection module description'

entry_point = 2

execute(*target: mythril.laser.ethereum.state.global_state.GlobalState*) → Optional[List[mythril.analysis.report.Issue]]
The entry point for execution, which is being called by Mythril.

Parameters **target** – The target of the analysis, either a global state (callback) or the entire statespace (post)

Returns List of encountered issues

name = 'Detection Module Name / Title'

post_hooks = []

pre_hooks = []

reset_module()
Resets the storage of this module

swc_id = 'SWC-000'

update_cache(*issues=None*)
Updates cache with param issues, updates against self.issues, if the param is None :param issues: The issues used to update the cache

class mythril.analysis.module.base.**EntryPoint**
Bases: enum.Enum

EntryPoint Enum

This enum is used to signify the entry_point of detection modules. See also the class documentation of DetectionModule

CALLBACK = 2

POST = 1

mythril.analysis.module.loader module

class mythril.analysis.module.loader.**ModuleLoader**
Bases: object

The module loader class implements a singleton loader for detection modules.

By default it will load the detection modules in the mythril package. Additional detection modules can be loaded using the register_module function call implemented by the ModuleLoader

get_detection_modules(*entry_point: Optional[mythril.analysis.module.base.EntryPoint] = None*, *white_list: Optional[List[str]] = None*) → List[mythril.analysis.module.base.DetectionModule]
Gets registered detection modules

Parameters

- **entry_point** – If specified: only return detection modules with this entry point
- **white_list** – If specified: only return whitelisted detection modules

Returns The selected detection modules

register_module(*detection_module: mythril.analysis.module.base.DetectionModule*)
Registers a detection module with the module loader

mythril.analysis.module.module_helpers module

mythril.analysis.module.module_helpers.**is_prehook**() → bool
Check if we are in prehook. One of Bernhard's trademark hacks! Let's leave it to this for now, unless we need to check prehook for a lot more modules.

mythril.analysis.module.util module

mythril.analysis.module.util.**get_detection_module_hooks**(*modules: List[mythril.analysis.module.base.DetectionModule*, *hook_type='pre'*) → Dict[str, List[Callable]]
Gets a dictionary with the hooks for the passed detection modules

Parameters

- **modules** – The modules for which to retrieve hooks
- **hook_type** – The type of hooks to retrieve (default: "pre")

Returns Dictionary with discovered hooks

mythril.analysis.module.util.**reset_callback_modules**(*module_names: Optional[List[str]] = None*)
Clean the issue records of every callback-based module.

Module contents

Submodules

mythril.analysis.analysis_args module

mythril.analysis.call_helpers module

This module provides helper functions for the analysis modules to deal with call functionality.

mythril.analysis.call_helpers.**get_call_from_state**(*state: mythril.laser.ethereum.state.global_state.GlobalState*) → Optional[mythril.analysis.ops.Call]

Parameters **state** –

Returns

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.issue_annotation module

class mythril.analysis.issue_annotation.**IssueAnnotation**(*conditions: List[mythril.laser.smt.bool.Bool]*, *issue: mythril.analysis.report.Issue*, *detector*)

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

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() → bool

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.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.potential_issues module

class mythril.analysis.potential_issues.**PotentialIssue**(*contract*, *function_name*, *address*, *swc_id*, *title*, *bytecode*, *detector*, *severity=None*, *description_head=''*, *description_tail=''*, *constraints=None*)
Bases: object

Representation of a potential issue

class mythril.analysis.potential_issues.**PotentialIssuesAnnotation**
Bases: *mythril.laser.ethereum.state.annotation.StateAnnotation*

search_importance
Used in estimating the priority of a state annotated with the corresponding annotation. Default is 1

mythril.analysis.potential_issues.**check_potential_issues**(*state: mythril.laser.ethereum.state.global_state.GlobalSt*
→ None
Called at the end of a transaction, checks potential issues, and adds valid issues to the detector.

Parameters **state** – The final global state of a transaction

Returns

mythril.analysis.potential_issues.**get_potential_issues_annotation**(*state: mythril.laser.ethereum.state.global_s*
→ mythril.analysis.potential_issues.Pot
Returns the potential issues annotation of the given global state, and creates one if one does not already exist.

Parameters **state** – The global state

Returns

mythril.analysis.report module

This module provides classes that make up an issue report.

class mythril.analysis.report.**Issue**(*contract: str*, *function_name: str*, *address: int*, *swc_id: str*, *title: str*, *bytecode: str*, *gas_used=(None, None)*, *severity=None*, *description_head=''*, *description_tail=''*, *transaction_sequence=None*, *source_location=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

static decode_bytes(*val*)

resolve_function_names()
Resolves function names for each step

static resolve_input(*data*, *function_name*)
Adds decoded calldate to the tx sequence.

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[str, Any]

Generate concrete transaction sequence. Note: This function only considers the constraints in constraint argument, which in some cases is expected to differ from global_state's constraints

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.concolic package

Submodules

mythril.concolic.concolic_execution module

mythril.concolic.concolic_execution.**concolic_execution**(*concrete_data: mythril.concolic.concrete_data.ConcreteData, jump_addresses: List[T], solver_timeout=100000*) → List[Dict[str, Dict[str, Any]]]

Executes codes and prints input required to cover the branch flips :param input_file: Input file :param jump_addresses: Jump addresses to flip :param solver_timeout: Solver timeout

mythril.concolic.concolic_execution.**flip_branches**(*init_state: mythril.laser.ethereum.state.world_state.WorldState, concrete_data: mythril.concolic.concrete_data.ConcreteData, jump_addresses: List[str], trace: List[T]*) → List[Dict[str, Dict[str, Any]]]

Flips branches and prints the input required for branch flip

Parameters

- **concrete_data** – Concrete data
- **jump_addresses** – Jump addresses to flip
- **trace** – trace to follow

mythril.concolic.concrete_data module

class mythril.concolic.concrete_data.**AccountData**
Bases: dict

class mythril.concolic.concrete_data.**ConcreteData**
Bases: dict

class mythril.concolic.concrete_data.**InitialState**
Bases: dict

class mythril.concolic.concrete_data.**TransactionData**
Bases: dict

mythril.concolic.find_trace module

mythril.concolic.find_trace.**concrete_execution**(*concrete_data: mythril.concolic.concrete_data.ConcreteData*) → Tuple[mythril.laser.ethereum.state.world_state.WorldState, List[T]]
Executes code concretely to find the path to be followed by concolic executor :param concrete_data: Concrete data :return: path trace

mythril.concolic.find_trace.**setup_concrete_initial_state**(*concrete_data: mythril.concolic.concrete_data.ConcreteData*) → mythril.laser.ethereum.state.world_state.WorldSta
Sets up concrete initial state :param concrete_data: Concrete data :return: initialised world state

Module contents

6.1.3 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*) → 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

6.1.4 mythril.ethereum package

Subpackages

mythril.ethereum.interface package

Subpackages

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.**extract_binary**(*file: str*) → str

mythril.ethereum.util.**extract_version**(*file: Optional[str]*)

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.**parse_pragma**(*solidity_code*)

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

Parameters **hex_encoded_string** –

Returns

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

Parameters **version** –

Returns

Module contents

6.1.5 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.**add_analysis_args**(*options*)
Adds arguments for analysis

Parameters **options** – Analysis Options

mythril.interfaces.cli.**add_graph_commands**(*parser: argparse.ArgumentParser*)

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_concolic_parser**(*parser: argparse.ArgumentParser*) → argparse.ArgumentParser
Get parser which handles arguments for concolic branch flipping

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

mythril.interfaces.cli.**create_foundry_parser**(*foundry_parser: argparse.ArgumentParser*)

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_read_storage_parser**(*read_storage_parser: argparse.ArgumentParser*)
Modify parser to handle storage slots :param read_storage_parser: :return:

mythril.interfaces.cli.**create_safe_functions_parser**(*parser: argparse.ArgumentParser*)
The duplication exists between safe-functions and analyze as some of them have different default values. :param parser: Parser

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_safe_functions_parser**() → argparse.ArgumentParser
Returns Parser which handles checking for safe functions :return: Parser which handles checking for safe functions

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

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.**print_function_report**(*myth_disassembler: mythril.mythril.mythril_disassembler.MythrilDisassembler*, *report: mythril.analysis.report.Report*)
Prints the function report :param report: Mythril's report :return:

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

6.1.6 mythril.laser package

Subpackages

mythril.laser.ethereum package

Subpackages

mythril.laser.ethereum.function_managers package

Submodules

mythril.laser.ethereum.function_managers.exponent_function_manager module

class mythril.laser.ethereum.function_managers.exponent_function_manager.**ExponentFunctionMa**
Bases: object

Uses an uninterpreted function for exponentiation with the following properties: 1) power(a, b) > 0 2) if a = 256 => forall i if b = i then power(a, b) = (256 ^ i) % (2^256)

Only these two properties are added as to handle indexing of boolean arrays. Caution should be exercised when increasing the conditions since it severely affects the solving time.

create_condition(*base: mythril.laser.smt.bitvec.BitVec*, *exponent: mythril.laser.smt.bitvec.BitVec*)
→ Tuple[mythril.laser.smt.bitvec.BitVec, mythril.laser.smt.bool.Bool]
Creates a condition for exponentiation :param base: The base of exponentiation :param exponent: The exponent of the exponentiation :return: Tuple of condition and the exponentiation result

mythril.laser.ethereum.function_managers.keccak_function_manager module

class mythril.laser.ethereum.function_managers.keccak_function_manager.**KeccakFunctionManage**
Bases: object

A bunch of uninterpreted functions are considered like keccak256_160 ,... where keccak256_160 means the input of keccak256() is 160 bit number. the range of these functions are constrained to some mutually disjoint intervals All the hashes modulo 64 are 0 as we need a spread among hashes for array type data structures All the functions are kind of one to one due to constraint of the existence of inverse for each encountered input. For more info https://files.sri.inf.ethz.ch/website/papers/sp20-verx.pdf

create_conditions() → mythril.laser.smt.bool.Bool

create_keccak(*data: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bitvec.BitVec
Creates Keccak of the data :param data: input :return: Tuple of keccak and the condition it should satisfy

static find_concrete_keccak(*data: mythril.laser.smt.bitvec.BitVec*) →
mythril.laser.smt.bitvec.BitVec
Calculates concrete keccak :param data: input bitvecval :return: concrete keccak output

get_concrete_hash_data(*model*) → Dict[int, List[Optional[int]]]
returns concrete values of hashes in the self.hash_result_store :param model: The z3 model to query for concrete values :return: A dictionary with concrete hashes { <hash_input_size> : [<concrete_hash>, <concrete_hash>]}

static get_empty_keccak_hash() → mythril.laser.smt.bitvec.BitVec
returns sha3("") :return:

get_function(*length: int*) → Tuple[mythril.laser.smt.function.Function, mythril.laser.smt.function.Function]
Returns the keccak functions for the corresponding length :param length: input size :return: tuple of keccak and it's inverse

hash_matcher = 'fffffff'

reset()

Module contents

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, nonce=0*)

Bases: object

Account class representing ethereum accounts.

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

Parameters **balance** –

as_dict

Returns

serialised_code()

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

Parameters **balance** –

set_storage(*storage: Dict[KT, VT]*)
Sets concrete storage

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

search_importance
Used in estimating the priority of a state annotated with the corresponding annotation. Default is 1

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.

get_all_constraints()

get_model(*solver_timeout=None*) → Optional[mythril.laser.smt.model.Model]

Parameters **solver_timeout** – The default timeout uses analysis timeout from args.solver_timeout

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

is_possible(*solver_timeout=None*) → bool

Parameters **solver_timeout** – The default timeout uses analysis timeout from args.solver_timeout

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

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*, *basefee: 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*)

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[int, mythril.laser.smt.bitvec.BitVec]]*) → None

Writes data to memory starting at offset.

Parameters

- **offset** –
- **data** –

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.machine_state.**ceil32**(*value: int*, ***, *ceiling: int = 32*) → int

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.return_data module

This module declares classes to represent call data.

class mythril.laser.ethereum.state.return_data.**ReturnData**(*return_data: List[mythril.laser.smt.bitvec.BitVec], return_data_size: mythril.laser.smt.bitvec.BitVec*)
Bases: object

Base returndata class.

size

Returns Calldata size for this calldata object

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*, *nonce=0*) → 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
- **nonce** – Nonce of the account

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

Subpackages

mythril.laser.ethereum.strategy.extensions package

Submodules

mythril.laser.ethereum.strategy.extensions.bounded_loops module

class mythril.laser.ethereum.strategy.extensions.bounded_loops.**BoundedLoopsStrategy**(*super_str*
mythril.la
***kwargs*

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

Adds loop pruning to the search strategy. Ignores JUMPI instruction if the destination was targeted >JUMPDEST_LIMIT times.

static calculate_hash(*i: int, j: int, trace: List[int]*) → int
calculate hash(trace[i: j]) :param i: :param j: :param trace: :return: hash(trace[i: j])

static count_key(*trace: List[int], key: int, start: int, size: int*) → int
Count continuous loops in the trace. :param trace: :param key: :param size: :return:

static get_loop_count(*trace: List[int]*) → int
Gets the loop count :param trace: annotation trace :return:

get_strategic_global_state() → mythril.laser.ethereum.state.global_state.GlobalState
Returns the next state

Returns Global state

class mythril.laser.ethereum.strategy.extensions.bounded_loops.**JumpdestCountAnnotation**
Bases: *mythril.laser.ethereum.state.annotation.StateAnnotation*

State annotation that counts the number of jumps per destination.

Module contents

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*, ***kwargs*)

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

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

Returns

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

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

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

Returns

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

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

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

Returns

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

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

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

Returns

mythril.laser.ethereum.strategy.beam module

class mythril.laser.ethereum.strategy.beam.**BeamSearch**(*work_list*, *max_depth*, *beam_width*, ***kwargs*)

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

chooses a random state from the worklist with equal likelihood.

static beam_priority(*state*)

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

Returns

sort_and_eliminate_states()

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

Returns

mythril.laser.ethereum.strategy.concolic module

class mythril.laser.ethereum.strategy.concolic.**ConcolicStrategy**(*work_list: List[mythril.laser.ethereum.state.global*, *max_depth: int*, *trace: List[List[Tuple[int, str]]]*, *flip_branch_addresses: List[str]*)

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

Executes program concolically using the input trace till a specific branch

check_completion_criterion()

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

This function does the following:- 1) Choose the states by following the concolic trace. 2) In case we have an executed JUMPI that is in flip_branch_addresses, flip that branch. :return:

class mythril.laser.ethereum.strategy.concolic.**TraceAnnotation**(*trace=None*)

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

This is the annotation used by the ConcolicStrategy to store concolic traces.

persist_over_calls

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

The default is set to False

Module contents

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

Bases: abc.ABC

A basic search strategy which halts based on depth

get_strategic_global_state()

run_check()

class mythril.laser.ethereum.strategy.**CriterionSearchStrategy**(*work_list*, *max_depth*, ***kwargs*)

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

If a criterion is satisfied, the search halts

get_strategic_global_state()

set_criterion_satisfied()

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_contract_creation**(*laser_evm*, *callee_address*, *caller_address*, *origin_address*, *data*, *gas_limit*, *gas_price*, *value*, *code=None*, *track_gas=False*, *contract_name=None*)

Executes a contract creation transaction concretely.

Parameters

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

Returns

mythril.laser.ethereum.transaction.concolic.**execute_message_call**(*laser_evm*, *callee_address*, *caller_address*, *origin_address*, *data*, *gas_limit*, *gas_price*, *value*, *code=None*, *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.concolic.**execute_transaction**(**args*, ***kwargs*) → Union[None, List[mythril.laser.ethereum.state.global

Chooses the transaction type based on callee address and executes the transaction

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=1004753105490295263244812946565944*, *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*, *origin=100475310549029526324*, *caller=100475310549029526* → 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*, *func_hashes: List[List[int]] = None*) → None

Executes a message call transaction from all open states.

Parameters

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

mythril.laser.ethereum.transaction.symbolic.**execute_transaction**(**args*, ***kwargs*)

Chooses the transaction type based on callee address and executes the transaction

mythril.laser.ethereum.transaction.symbolic.**generate_function_constraints**(*calldata: mythril.laser.ethereum.st*, *func_hashes: List[List[int]]*) → List[mythril.laser.smt.bo

This will generate constraints for fixing the function call part of calldata :param calldata: Calldata :param func_hashes: The list of function hashes allowed for this transaction :return: Constraints List

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, base_fee=None*)

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
ider
ti-
fier
Op-
tion
=
Nor
gas
gas
ori-
gin
cod
call
con
trac
con
trac
bas

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_s 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 Trans ac- tion] op_c str, globa myth*

Bases: Exception

Exception raised when a new transaction is started.

class mythril.laser.ethereum.transaction.transaction_models.**TxIdManager**

Bases: object

get_next_tx_id()

restart_counter()

set_counter(*tx_id*)

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.**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.instruction_data module

mythril.laser.ethereum.instruction_data.**calculate_native_gas**(*size: int*, *contract: str*)

Parameters

- **size** –
- **contract** –

Returns

mythril.laser.ethereum.instruction_data.**calculate_sha3_gas**(*length: int*)

Parameters **length** –

Returns

mythril.laser.ethereum.instruction_data.**ceil32**(*value: int*, ***, *ceiling: int = 32*) → int

mythril.laser.ethereum.instruction_data.**get_opcode_gas**(*opcode: str*) → Tuple[int, int]

mythril.laser.ethereum.instruction_data.**get_required_stack_elements**(*opcode: str*) → int

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

basefee_(*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

selfdestruct_(*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

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.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.**ecrecover_to_pub**(*rawhash*, *v*, *r*, *s*)

mythril.laser.ethereum.natives.**encode_int32**(*v*)

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

Parameters **data** –

Returns

mythril.laser.ethereum.natives.**int_to_32bytearray**(*i*)

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.**safe_ord**(*value*)

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*, *use_reachability_check=True*, *beam_width=None*, *tx_strategy=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.

execute_transactions(*address*) → None

This function helps runs plugins that can order transactions. Such plugins should set self.executed_transactions as True after its execution

Parameters **address** – Address of the contract

Returns None

extend_strategy(*extension: abc.ABCMeta*, ***kwargs*) → 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.time_handler module

class mythril.laser.ethereum.time_handler.**TimeHandler**
Bases: object

start_execution(*execution_time*)

time_remaining()

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.**insert_ret_val**(*global_state: GlobalState*)

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.plugin package

Subpackages

mythril.laser.plugin.plugins package

Subpackages

mythril.laser.plugin.plugins.coverage package

Submodules

mythril.laser.plugin.plugins.coverage.coverage_plugin module

class mythril.laser.plugin.plugins.coverage.coverage_plugin.**CoveragePluginBuilder**
Bases: *mythril.laser.plugin.builder.PluginBuilder*

name = 'coverage'

class mythril.laser.plugin.plugins.coverage.coverage_plugin.**InstructionCoveragePlugin**
Bases: *mythril.laser.plugin.interface.LaserPlugin*

This plugin measures the instruction coverage of mythril. The instruction coverage is the ratio between the instructions that have been executed and the total amount of instructions.

Note that with lazy constraint solving enabled that this metric will be "unsound" as reachability will not be considered for the calculation of instruction coverage.

initialize(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM*)
Initializes the instruction coverage plugin

Introduces hooks for each instruction :param symbolic_vm: :return:

is_instruction_covered(*bytecode*, *index*)

mythril.laser.plugin.plugins.coverage.coverage_strategy module

class mythril.laser.plugin.plugins.coverage.coverage_strategy.**CoverageStrategy**(*super_strategy: mythril.laser.ethe in- struc- tion_coverage_pl mythril.laser.plug*

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

Implements a instruction coverage based search strategy

This strategy is quite simple and effective, it prioritizes the execution of instructions that have previously been uncovered. Once there is no such global state left in the work list, it will resort to using the super_strategy.

This strategy is intended to be used "on top of" another one

get_strategic_global_state() → mythril.laser.ethereum.state.global_state.GlobalState
Returns the first uncovered global state in the work list if it exists, otherwise super_strategy.get_strategic_global_state() is returned.

Module contents

mythril.laser.plugin.plugins.summary_backup package

Module contents

Submodules

mythril.laser.plugin.plugins.benchmark module

class mythril.laser.plugin.plugins.benchmark.**BenchmarkPlugin**(*name=None*)
Bases: *mythril.laser.plugin.interface.LaserPlugin*

Benchmark Plugin

This plugin aggregates the following information: - duration - code coverage over time - final code coverage - total number of executed instructions

initialize(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM*)
Initializes the BenchmarkPlugin

Introduces hooks in symbolic_vm to track the desired values :param symbolic_vm: Symbolic virtual machine to analyze

class mythril.laser.plugin.plugins.benchmark.**BenchmarkPluginBuilder**
Bases: *mythril.laser.plugin.builder.PluginBuilder*

name = 'benchmark'

mythril.laser.plugin.plugins.call_depth_limiter module

class mythril.laser.plugin.plugins.call_depth_limiter.**CallDepthLimit**(*call_depth_limit: int*)
Bases: *mythril.laser.plugin.interface.LaserPlugin*

initialize(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM*)
Initializes the mutation pruner

Introduces hooks for SSTORE operations :param symbolic_vm: :return:

class mythril.laser.plugin.plugins.call_depth_limiter.**CallDepthLimitBuilder**
Bases: *mythril.laser.plugin.builder.PluginBuilder*

name = 'call-depth-limit'

mythril.laser.plugin.plugins.dependency_pruner module

class mythril.laser.plugin.plugins.dependency_pruner.**DependencyPruner**
Bases: *mythril.laser.plugin.interface.LaserPlugin*

Dependency Pruner Plugin

For every basic block, this plugin keeps a list of storage locations that are accessed (read) in the execution path containing that block. This map is built up over the whole symbolic execution run.

After the initial build up of the map in the first transaction, blocks are executed only if any of the storage locations written to in the previous transaction can have an effect on that block or any of its successors.

initialize(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM*) → None
Initializes the DependencyPruner

:param symbolic_vm

update_calls(*path: List[int]*) → None
Update the dependency map for the block offsets on the given path.

:param path :param target_location

update_sloads(*path: List[int], target_location: object*) → None
Update the dependency map for the block offsets on the given path.

:param path :param target_location

update_sstores(*path: List[int], target_location: object*) → None
Update the dependency map for the block offsets on the given path.

:param path :param target_location

wanna_execute(*address: int*, *annotation: mythril.laser.plugin.plugins.plugin_annotations.DependencyAnnotation*) → bool
Decide whether the basic block starting at 'address' should be executed.

:param address :param storage_write_cache

class mythril.laser.plugin.plugins.dependency_pruner.**DependencyPrunerBuilder**
Bases: *mythril.laser.plugin.builder.PluginBuilder*

name = 'dependency-pruner'

mythril.laser.plugin.plugins.dependency_pruner.**get_dependency_annotation**(*state: mythril.laser.ethereum.sta* → mythril.laser.plugin.plugi

Returns a dependency annotation

Parameters **state** – A global state object

mythril.laser.plugin.plugins.dependency_pruner.**get_ws_dependency_annotation**(*state: mythril.laser.ethereu* → mythril.laser.plugin.p

Returns the world state annotation

Parameters **state** – A global state object

mythril.laser.plugin.plugins.instruction_profiler module

class mythril.laser.plugin.plugins.instruction_profiler.**InstructionProfiler**
Bases: *mythril.laser.plugin.interface.LaserPlugin*

Performance profile for the execution of each instruction.

initialize(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM*) → None
Initializes this plugin on the symbolic virtual machine

Parameters **symbolic_vm** – symbolic virtual machine to initialize the laser plugin on

class mythril.laser.plugin.plugins.instruction_profiler.**InstructionProfilerBuilder**
Bases: *mythril.laser.plugin.builder.PluginBuilder*

name = 'instruction-profiler'

mythril.laser.plugin.plugins.mutation_pruner module

class mythril.laser.plugin.plugins.mutation_pruner.**MutationPruner**
Bases: *mythril.laser.plugin.interface.LaserPlugin*

Mutation pruner plugin

Let S be a world state from which T is a symbolic transaction, and S' is the resulting world state. In a situation where T does not execute any mutating instructions we can safely abandon further analysis on top of state S'. This is for the reason that we already performed analysis on S, which is equivalent.

This optimization inhibits path explosion caused by "clean" behaviour

The basic operation of this plugin is as follows:

- Hook all mutating operations and introduce a MutationAnnotation to the global state on execution of the hook
- Hook the svm EndTransaction on execution filter the states that do not have a mutation annotation

initialize(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM*)
Initializes the mutation pruner

Introduces hooks for SSTORE operations :param symbolic_vm: :return:

class mythril.laser.plugin.plugins.mutation_pruner.**MutationPrunerBuilder**
Bases: *mythril.laser.plugin.builder.PluginBuilder*

name = 'mutation-pruner'

mythril.laser.plugin.plugins.plugin_annotations module

class mythril.laser.plugin.plugins.plugin_annotations.**DependencyAnnotation**
Bases: *mythril.laser.ethereum.state.annotation.MergeableStateAnnotation*

Dependency Annotation

This annotation tracks read and write access to the state during each transaction.

check_merge_annotation(*other: mythril.laser.plugin.plugins.plugin_annotations.DependencyAnnotation*)

extend_storage_write_cache(*iteration: int*, *value: object*)

get_storage_write_cache(*iteration: int*)

merge_annotation(*other: mythril.laser.plugin.plugins.plugin_annotations.DependencyAnnotation*)

class mythril.laser.plugin.plugins.plugin_annotations.**MutationAnnotation**
Bases: *mythril.laser.ethereum.state.annotation.StateAnnotation*

Mutation Annotation

This is the annotation used by the MutationPruner plugin to record mutations

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

The default is set to False

class mythril.laser.plugin.plugins.plugin_annotations.**WSDependencyAnnotation**
Bases: *mythril.laser.ethereum.state.annotation.MergeableStateAnnotation*

Dependency Annotation for World state

This world state annotation maintains a stack of state annotations. It is used to transfer individual state annotations from one transaction to the next.

check_merge_annotation(*annotation: mythril.laser.plugin.plugins.plugin_annotations.WSDependencyAnnotation*) → bool

merge_annotation(*annotation: mythril.laser.plugin.plugins.plugin_annotations.WSDependencyAnnotation*) → mythril.laser.plugin.plugins.plugin_annotations.WSDependencyAnnotation

Module contents

Plugin implementations

This module contains the implementation of some features

- benchmarking
- pruning

Submodules

mythril.laser.plugin.builder module

class mythril.laser.plugin.builder.**PluginBuilder**
Bases: abc.ABC

The plugin builder interface enables construction of Laser plugins

name = 'Default Plugin Name'

mythril.laser.plugin.interface module

class mythril.laser.plugin.interface.**LaserPlugin**
Bases: object

Base class for laser plugins

Functionality in laser that the symbolic execution process does not need to depend on can be implemented in the form of a laser plugin.

Laser plugins implement the function initialize(symbolic_vm) which is called with the laser virtual machine when they are loaded. Regularly a plugin will introduce several hooks into laser in this function

Plugins can direct actions by raising Signals defined in mythril.laser.ethereum.plugins.signals For example, a pruning plugin might raise the PluginSkipWorldState signal.

initialize(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM*) → None
Initializes this plugin on the symbolic virtual machine

Parameters **symbolic_vm** – symbolic virtual machine to initialize the laser plugin on

mythril.laser.plugin.loader module

class mythril.laser.plugin.loader.**LaserPluginLoader**
Bases: object

The LaserPluginLoader is used to abstract the logic relating to plugins. Components outside of laser thus don't have to be aware of the interface that plugins provide

add_args(*plugin_name*, ***kwargs*)

enable(*plugin_name: str*)

instrument_virtual_machine(*symbolic_vm: mythril.laser.ethereum.svm.LaserEVM, with_plugins: Optional[List[str]]*)
Load enabled plugins into the passed symbolic virtual machine :param symbolic_vm: The virtual machine to instrument the plugins with :param with_plugins: Override the globally enabled/disabled builders and load all plugins in the list

is_enabled(*plugin_name: str*) → bool
Returns whether the plugin is loaded in the symbolic_vm

Parameters **plugin_name** – Name of the plugin to check

load(*plugin_builder: mythril.laser.plugin.builder.PluginBuilder*) → None
Enables a Laser Plugin

Parameters **plugin_builder** – Builder that constructs the plugin

mythril.laser.plugin.signals module

exception mythril.laser.plugin.signals.**PluginSignal**
Bases: Exception

Base plugin signal

These signals are used by the laser plugins to create intent for certain actions in the symbolic virtual machine

exception mythril.laser.plugin.signals.**PluginSkipState**
Bases: *mythril.laser.plugin.signals.PluginSignal*

Plugin to skip world state

Plugins that raise this signal while the add_world_state hook is being executed will force laser to abandon that world state.

exception mythril.laser.plugin.signals.**PluginSkipWorldState**
Bases: *mythril.laser.plugin.signals.PluginSignal*

Plugin to skip world state

Plugins that raise this signal while the add_world_state hook is being executed will force laser to abandon that world state.

Module contents

Laser plugins

This module contains everything to do with laser plugins

Laser plugins are a way of extending laser's functionality without complicating the core business logic. Different features that have been implemented in the form of plugins are: - benchmarking - path pruning

Plugins also provide a way to implement optimisations outside of the mythril code base and to inject them. The api that laser currently provides is still unstable and will probably change to suit our needs as more plugins get developed.

For the implementation of plugins the following modules are of interest: - laser.plugins.plugin - laser.plugins.signals - laser.svm

Which show the basic interfaces with which plugins are able to interact

mythril.laser.smt package

Subpackages

mythril.laser.smt.solver package

Submodules

mythril.laser.smt.solver.independence_solver module

class mythril.laser.smt.solver.independence_solver.**DependenceBucket**(*variables=None*, *conditions=None*)

Bases: object

Bucket object to contain a set of conditions that are dependent on each other

class mythril.laser.smt.solver.independence_solver.**DependenceMap**

Bases: object

DependenceMap object that maintains a set of dependence buckets, used to separate independent smt queries

add_condition(*condition: z3.z3.BoolRef*) → None

Add condition to the dependence map :param condition: The condition that is to be added to the dependence map

class mythril.laser.smt.solver.independence_solver.**IndependenceSolver**

Bases: object

An SMT solver object that uses independence optimization

add(**constraints*) → None

Adds the constraints to this solver.

Parameters **constraints** – constraints to add

append(**constraints*) → None

Adds the constraints to this solver.

Parameters **constraints** – constraints to add

check(***kwargs*)

model() → mythril.laser.smt.model.Model

Returns z3 model for a solution.

pop(*num*) → None

Pop num constraints from this solver.

reset() → None

Reset this solver.

set_timeout(*timeout: int*) → None
Sets the timeout that will be used by this solver, timeout is in milliseconds.

Parameters **timeout** –

mythril.laser.smt.solver.solver module

This module contains an abstract SMT representation of an SMT solver.

class mythril.laser.smt.solver.solver.**BaseSolver**(*raw: T*)
Bases: typing.Generic

add(**constraints*) → None
Adds the constraints to this solver.

Parameters **constraints** –

Returns

append(**constraints*) → None
Adds the constraints to this solver.

Parameters **constraints** –

Returns

check(***kwargs*)

model() → mythril.laser.smt.model.Model
Returns z3 model for a solution.

Returns

set_timeout(*timeout: int*) → None
Sets the timeout that will be used by this solver, timeout is in milliseconds.

Parameters **timeout** –

sexpr()

class mythril.laser.smt.solver.solver.**Optimize**
Bases: *mythril.laser.smt.solver.solver.BaseSolver*

An optimizing smt solver.

maximize(*element: mythril.laser.smt.expression.Expression[z3.z3.ExprRef][z3.z3.ExprRef]*) → None
In solving this solver will try to maximize the passed expression.

Parameters **element** –

minimize(*element: mythril.laser.smt.expression.Expression[z3.z3.ExprRef][z3.z3.ExprRef]*) → None
In solving this solver will try to minimize the passed expression.

Parameters **element** –

class mythril.laser.smt.solver.solver.**Solver**
Bases: *mythril.laser.smt.solver.solver.BaseSolver*

An SMT solver object.

pop(*num: int*) → None
Pop num constraints from this solver.

Parameters **num** –

reset() → None
Reset this solver.

mythril.laser.smt.solver.solver_statistics module

class mythril.laser.smt.solver.solver_statistics.**SolverStatistics**
Bases: object

Solver Statistics Class

Keeps track of the important statistics around smt queries

mythril.laser.smt.solver.solver_statistics.**stat_smt_query**(*func: Callable*)
Measures statistics for annotated smt query check function

Module contents

Submodules

mythril.laser.smt.array module

This module contains an SMT abstraction of arrays.

This includes an Array class to implement basic store and set operations, as well as as a K-array, which can be initialized with default values over a certain range.

class mythril.laser.smt.array.**Array**(*name: str*, *domain: int*, *value_range: int*)
Bases: *mythril.laser.smt.array.BaseArray*

A basic symbolic array.

class mythril.laser.smt.array.**BaseArray**(*raw*)
Bases: object

Base array type, which implements basic store and set operations.

substitute(*original_expression*, *new_expression*)

Parameters

- **original_expression** –
- **new_expression** –

class mythril.laser.smt.array.**K**(*domain: int*, *value_range: int*, *value: int*)
Bases: *mythril.laser.smt.array.BaseArray*

A basic symbolic array, which can be initialized with a default value.

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.bitvec_helper module

mythril.laser.smt.bitvec_helper.**BVAddNoOverflow**(*a: Union[mythril.laser.smt.bitvec.BitVec, int], b: Union[mythril.laser.smt.bitvec.BitVec, int], signed: bool*) → mythril.laser.smt.bool.Bool
Creates predicate that verifies that the addition doesn't overflow.

Parameters

- **a** –
- **b** –
- **signed** –

Returns

mythril.laser.smt.bitvec_helper.**BVMulNoOverflow**(*a: Union[mythril.laser.smt.bitvec.BitVec, int], b: Union[mythril.laser.smt.bitvec.BitVec, int], signed: bool*) → mythril.laser.smt.bool.Bool
Creates predicate that verifies that the multiplication doesn't overflow.

Parameters

- **a** –
- **b** –
- **signed** –

Returns

mythril.laser.smt.bitvec_helper.**BVSubNoUnderflow**(*a: Union[mythril.laser.smt.bitvec.BitVec, int], b: Union[mythril.laser.smt.bitvec.BitVec, int], signed: bool*) → mythril.laser.smt.bool.Bool
Creates predicate that verifies that the subtraction doesn't overflow.

Parameters

- **a** –
- **b** –

- **signed** –

Returns

mythril.laser.smt.bitvec_helper.**Concat**(**args*) → mythril.laser.smt.bitvec.BitVec

Create a concatenation expression.

Parameters **args** –

Returns

mythril.laser.smt.bitvec_helper.**Extract**(*high: int*, *low: int*, *bv: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bitvec.BitVec

Create an extract expression.

Parameters

- **high** –
- **low** –
- **bv** –

Returns

mythril.laser.smt.bitvec_helper.**If**(*a: Union[mythril.laser.smt.bool.Bool, bool]*, *b: Union[mythril.laser.smt.array.BaseArray, mythril.laser.smt.bitvec.BitVec, int]*, *c: Union[mythril.laser.smt.array.BaseArray, mythril.laser.smt.bitvec.BitVec, int]*) → Union[mythril.laser.smt.bitvec.BitVec, mythril.laser.smt.array.BaseArray]

Create an if-then-else expression.

Parameters

- **a** –
- **b** –
- **c** –

Returns

mythril.laser.smt.bitvec_helper.**LShR**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*)

mythril.laser.smt.bitvec_helper.**SRem**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bitvec.BitVec

Create a signed remainder expression.

Parameters

- **a** –
- **b** –

Returns

mythril.laser.smt.bitvec_helper.**Sum**(**args*) → mythril.laser.smt.bitvec.BitVec

Create sum expression.

Returns

mythril.laser.smt.bitvec_helper.**UDiv**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bitvec.BitVec

Create an unsigned division expression.

Parameters

- **a** –
- **b** –

Returns

mythril.laser.smt.bitvec_helper.**UGE**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bool.Bool

Create an unsigned greater than or equal to expression.

Parameters

- **a** –
- **b** –

Returns

mythril.laser.smt.bitvec_helper.**UGT**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bool.Bool

Create an unsigned greater than expression.

Parameters

- **a** –
- **b** –

Returns

mythril.laser.smt.bitvec_helper.**ULE**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bool.Bool

Create an unsigned less than or equal to expression.

Parameters

- **a** –
- **b** –

Returns

mythril.laser.smt.bitvec_helper.**ULT**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bool.Bool

Create an unsigned less than expression.

Parameters

- **a** –
- **b** –

Returns

mythril.laser.smt.bitvec_helper.**URem**(*a: mythril.laser.smt.bitvec.BitVec*, *b: mythril.laser.smt.bitvec.BitVec*) → mythril.laser.smt.bitvec.BitVec

Create an unsigned remainder expression.

Parameters

- **a** –
- **b** –

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

substitute(*original_expression*, *new_expression*)

Parameters

- **original_expression** –
- **new_expression** –

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

mythril.laser.smt.function module

class mythril.laser.smt.function.**Function**(*name: str, domain: List[int], value_range: int*)

Bases: object

An uninterpreted function.

mythril.laser.smt.model module

class mythril.laser.smt.model.**Model**(*models: List[z3.z3.ModelRef] = None*)
Bases: object

The model class wraps a z3 model

This implementation allows for multiple internal models, this is required for the use of an independence solver which has models for multiple queries which need an uniform output.

decls() → List[z3.z3.ExprRef]
Get the declarations for this model

eval(*expression: z3.z3.ExprRef*, *model_completion: bool = False*) → Union[None, z3.z3.ExprRef]
Evaluate the expression using this model

Parameters

- **expression** – The expression to evaluate
- **model_completion** – Use the default value if the model has no interpretation of the given expression

Returns The evaluated expression

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()

Submodules

mythril.laser.execution_info module

class mythril.laser.execution_info.**ExecutionInfo**
Bases: abc.ABC

as_dict()
Returns a dictionary with the execution info contained in this object

The returned dictionary only uses primitive types.

Module contents

6.1.7 mythril.mythril package

Submodules

mythril.mythril.mythril_analyzer module

class mythril.mythril.mythril_analyzer.**MythrilAnalyzer**(*disassembler: mythril.mythril.mythril_disassembler.MythrilDisasse cmd_args: argparse.Namespace, strategy: str = 'dfs', address: Optional[str] = None*)
Bases: object

The Mythril Analyzer class Responsible for the analysis of the smart contracts

dump_statespace(*contract: mythril.ethereum.evmcontract.EVMContract = None*) → str
Returns serializable statespace of the contract :param contract: The Contract on which the analysis should be done :return: The serialized state space

fire_lasers(*modules: Optional[List[str]] = None*, *transaction_count: Optional[int] = None*) → mythril.analysis.report.Report

Parameters

- **modules** – The analysis modules which should be executed
- **transaction_count** – The amount of transactions to be executed

Returns The Report class which contains the all the issues/vulnerabilities

graph_html(*contract: mythril.ethereum.evmcontract.EVMContract = None*, *enable_physics: bool = False*, *phrackify: bool = False*, *transaction_count: Optional[int] = None*) → str

Parameters

- **contract** – The Contract on which the analysis should be done
- **enable_physics** – If true then enables the graph physics simulation
- **phrackify** – If true generates Phrack-style call graph

- **transaction_count** – The amount of transactions to be executed

Returns The generated graph in html format

mythril.mythril.mythril_config module

class mythril.mythril.mythril_config.**MythrilConfig**
Bases: object

The Mythril Analyzer class Responsible for setup of the mythril environment

static init_mythril_dir() → str
Initializes the mythril dir and config.ini file :return: The mythril dir's path

set_api_from_config_path() → None
Set the RPC mode based on a given config file.

set_api_infura_id(*id*)

set_api_rpc(*rpc: str = None*, *rpctls: bool = False*) → None
Sets the RPC mode to either of ganache or infura :param rpc: either of the strings - ganache, infura-mainnet, infura-rinkeby, infura-kovan, infura-ropsten, infura-goerli, avalanche, arbitrum, bsc, optimism, polygon

set_api_rpc_infura() → None
Set the RPC mode to INFURA on Mainnet.

set_api_rpc_localhost() → None
Set the RPC mode to a local instance.

mythril.mythril.mythril_disassembler module

class mythril.mythril.mythril_disassembler.**MythrilDisassembler**(*eth: Optional[mythril.ethereum.interface.rpc.clie = None*, *solc_version: str = None*, *solc_settings_json: str = None*, *enable_online_lookup: bool = False*, *solc_args=None*)
Bases: object

The Mythril Disassembler class Responsible for generating disassembly of smart contracts:

- Compiles solc code from file/onchain
- Can also be used to access onchain storage data

get_state_variable_from_storage(*address: str*, *params: Optional[List[str]] = None*) → str
Get variables from the storage :param address: The contract address :param params: The list of parameters param types: [position, length] or ["mapping", position, key1, key2, ...]

or [position, length, array]

Returns The corresponding storage slot and its value

static hash_for_function_signature(*func: str*) → str
Return function nadmes corresponding signature hash :param func: function name :return: Its hash signature

load_from_address(*address: str*) → Tuple[str, mythril.ethereum.evmcontract.EVMContract]
Returns the contract given it's on chain address :param address: The on chain address of a contract :return: tuple(address, contract)

load_from_bytecode(*code: str*, *bin_runtime: bool = False*, *address: Optional[str] = None*) → Tuple[str, mythril.ethereum.evmcontract.EVMContract]
Returns the address and the contract class for the given bytecode :param code: Bytecode :param bin_runtime: Whether the code is runtime code or creation code :param address: address of contract :return: tuple(address, Contract class)

load_from_foundry()

load_from_solidity(*solidity_files: List[str]*) → Tuple[str, List[mythril.solidity.soliditycontract.SolidityContract]]

Parameters **solidity_files** – List of solidity_files

Returns tuple of address, contract class list

mythril.mythril.mythril_disassembler.**format_warning**(*message*, *category*, *filename*, *lineno*, *line=''*)

Module contents

6.1.8 mythril.plugin package

Submodules

mythril.plugin.discovery module

class mythril.plugin.discovery.**PluginDiscovery**
Bases: object

PluginDiscovery class

This plugin implements the logic to discover and build plugins in installed python packages

build_plugin(*plugin_name: str*, *plugin_args: Dict[KT, VT]*) → mythril.plugin.interface.MythrilPlugin
Returns the plugin for the given plugin_name if it is installed

get_plugins(*default_enabled=None*) → List[str]
Gets a list of installed mythril plugins

Parameters **default_enabled** – Select plugins that are enabled by default

Returns List of plugin names

init_installed_plugins()

installed_plugins

is_installed(*plugin_name: str*) → bool
Returns whether there is python package with a plugin with plugin_name

mythril.plugin.interface module

class mythril.plugin.interface.**MythrilCLIPlugin**(***kwargs*)
Bases: *mythril.plugin.interface.MythrilPlugin*

MythrilCLIPlugin interface

This interface should be implemented by mythril plugins that aim to add commands to the mythril cli

class mythril.plugin.interface.**MythrilLaserPlugin**(***kwargs*)
Bases: *mythril.plugin.interface.MythrilPlugin*, *mythril.laser.plugin.builder.PluginBuilder*, abc.ABC

Mythril Laser Plugin interface

Plugins of this type are used to instrument the laser EVM

class mythril.plugin.interface.**MythrilPlugin**(***kwargs*)
Bases: object

MythrilPlugin interface

Mythril Plugins can be used to extend Mythril in different ways: 1. Extend Laser, in which case the LaserPlugin interface must also be extended 2. Extend Laser with a new search strategy in which case the SearchStrategy needs to be implemented 3. Add an analysis module, in this case the AnalysisModule interface needs to be implemented 4. Add new commands to the Mythril cli, using the MythrilCLIPlugin Interface

author = 'Default Author'

name = 'Plugin Name'

plugin_description = 'This is an example plugin description'

plugin_license = 'All rights reserved.'

plugin_type = 'Mythril Plugin'

plugin_version = '0.0.1 '

mythril.plugin.loader module

class mythril.plugin.loader.**MythrilPluginLoader**
Bases: object

MythrilPluginLoader singleton

This object permits loading MythrilPlugin's

load(*plugin: mythril.plugin.interface.MythrilPlugin*)
Loads the passed plugin

This function handles input validation and dispatches loading to type specific loaders. Supported plugin types:

- laser plugins
- detection modules

set_args(*plugin_name: str*, ***kwargs*)

exception mythril.plugin.loader.**UnsupportedPluginType**
Bases: Exception

Raised when a plugin with an unsupported type is loaded

Module contents

6.1.9 mythril.solidity package

Submodules

mythril.solidity.soliditycontract module

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

class mythril.solidity.soliditycontract.**SolcAST**(*ast*)
Bases: object

abs_path

node_type

nodes

class mythril.solidity.soliditycontract.**SolcSource**(*source*)
Bases: object

ast

contents

id

name

class mythril.solidity.soliditycontract.**SolidityContract**(*input_file*, *name=None*, *solc_settings_json=None*, *solc_binary='solc'*, *solc_data=None*)
Bases: *mythril.ethereum.evmcontract.EVMContract*

Representation of a Solidity contract.

static get_full_contract_src_maps(*ast: mythril.solidity.soliditycontract.SolcAST*) → 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(*input_file: str, data: Dict[KT, VT]*) → Dict[KT, VT]
Returns solc file indices

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

Parameters

- **address** –
- **constructor** –

Returns

static get_sources(*indices_data: Dict[KT, VT], source_data: Dict[KT, VT]*) → None
Get source indices mapping. Function not needed for older solc versions.

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** –

mythril.solidity.soliditycontract.**get_contracts_from_foundry**(*input_file*, *foundry_json*)

Parameters

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

Module contents

6.1.10 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.lock module

class mythril.support.lock.**LockFile**(*file_name*, *timeout=100*, *delay=0.05*)
Bases: object

Locks files.

acquire()
Acquires a lock when possible.

release()
Releases the lock

exception mythril.support.lock.**LockFileException**
Bases: Exception

mythril.support.model module

mythril.support.model.**get_model**
Returns a model based on given constraints as a tuple :param constraints: Tuple of constraints :param minimize: Tuple of minimization conditions :param maximize: Tuple of maximization conditions :param solver_timeout: The solver timeout :return:

mythril.support.model.**solver_worker**(*constraints*, *minimize=()*, *maximize=()*, *solver_timeout=None*)
Returns a model based on given constraints as a tuple :param constraints: Tuple of constraints :param minimize: Tuple of minimization conditions :param maximize: Tuple of maximization conditions :param solver_timeout: The timeout for solver :return:

mythril.support.opcodes module

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:

add_sigs(*file_path: str*, *solc_json*)

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.source_support module

class mythril.support.source_support.**Source**(*source_type=None*, *source_format=None*, *source_list=None*)
Bases: object

Class to handle to source data

get_source_from_contracts_list(*contracts*)
get the source data from the contracts list :param contracts: the list of contracts :return:

get_source_index(*bytecode_hash: str*) → int
Find the contract index in the list :param bytecode_hash: The contract hash :return: The index of the contract in the _source_hash list

mythril.support.start_time module

class mythril.support.start_time.**StartTime**
Bases: object

Maintains the start time of the current contract in execution

mythril.support.support_args module

class mythril.support.support_args.**Args**
Bases: object

This module helps in preventing args being sent through multiple of classes to reach any analysis/laser module

mythril.support.support_utils module

This module contains utility functions for the Mythril support package.

class mythril.support.support_utils.**LRUCache**(*size*)
Bases: object

get(*key*)

put(*key*, *value*)

class mythril.support.support_utils.**ModelCache**
Bases: object

check_quick_sat

put(*key*, *value*)

class mythril.support.support_utils.**Singleton**
Bases: type

A metaclass type implementing the singleton pattern.

mythril.support.support_utils.**get_code_hash**

Parameters **code** – bytecode

Returns Returns hash of the given bytecode

mythril.support.support_utils.**rzpad**(*value*, *total_length*)
Right zero pad value *x* at least to length *l*.

mythril.support.support_utils.**sha3**(*value*)

mythril.support.support_utils.**zpad**(*x*, *l*)
Left zero pad value *x* at least to length *l*.

Module contents

6.2 Submodules

6.3 mythril.exceptions module

This module contains general exceptions used by Mythril.

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.**IllegalArgumentError**
Bases: ValueError

The argument used does not exist

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.**SolverTimeOutException**
Bases: *mythril.exceptions.UnsatError*

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

exception mythril.exceptions.**UnsatError**
Bases: *mythril.exceptions.MythrilBaseException*

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

6.4 Module contents

CHAPTER 7

Indices and Tables

- genindex
- modindex
- search

Python Module Index

m

Index

A

B

C

D

E

F

G

H

I

N

O

P

R

S

T

U

V

W

X

Z