Advances in Automated Smart Contract Vulnerability Detection

AUGUST 22th, 2019

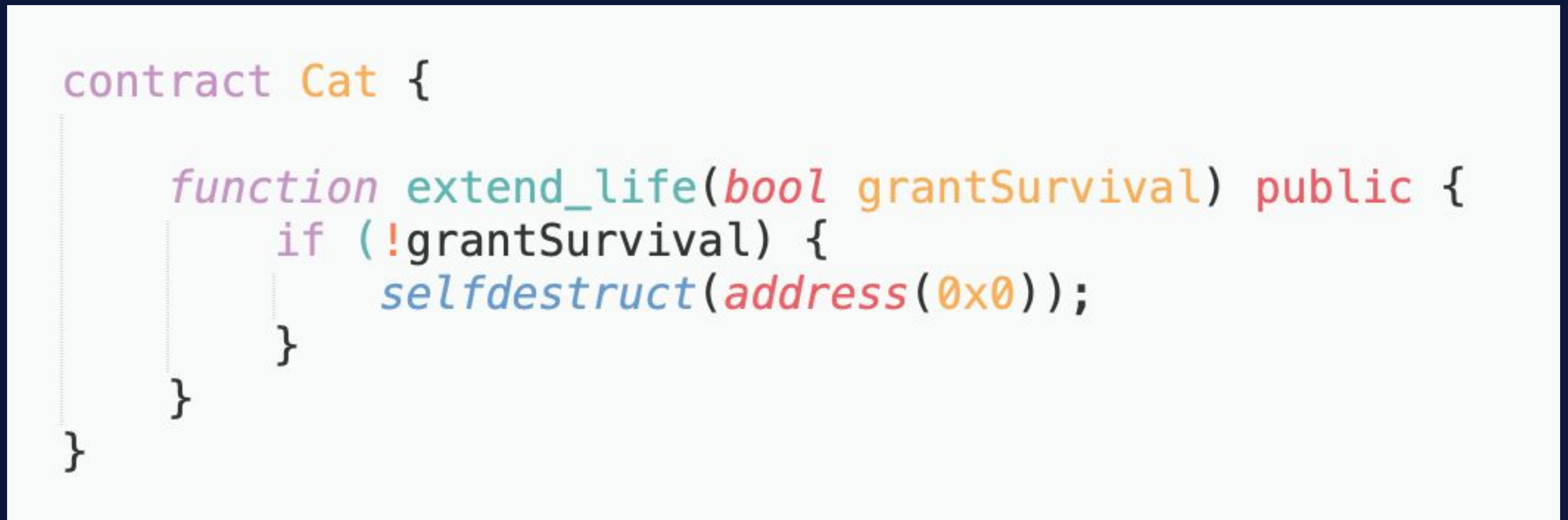
CONSENSYS Diligence

MythX

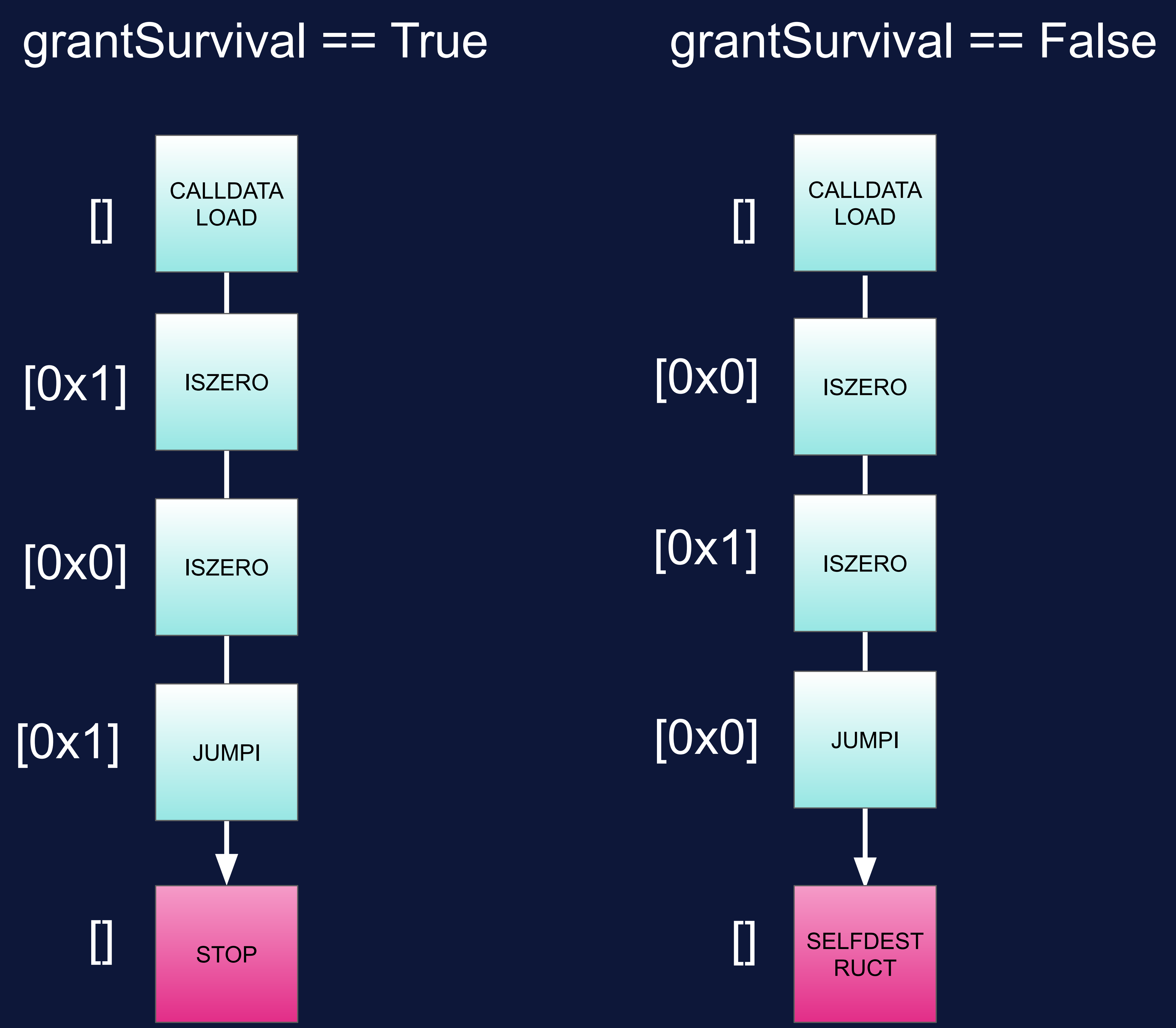
In this Talk

- Addressing Challenges Symbolic Execution
  - Eliminate False Positives
  - Increase Performance
- Verifying Invariants
- Tool Demos
  - Mythril
    - https://www.github.com/ConsenSys/mythril
  - MythX
    - https://mythx.io

Symbolic Execution (1)

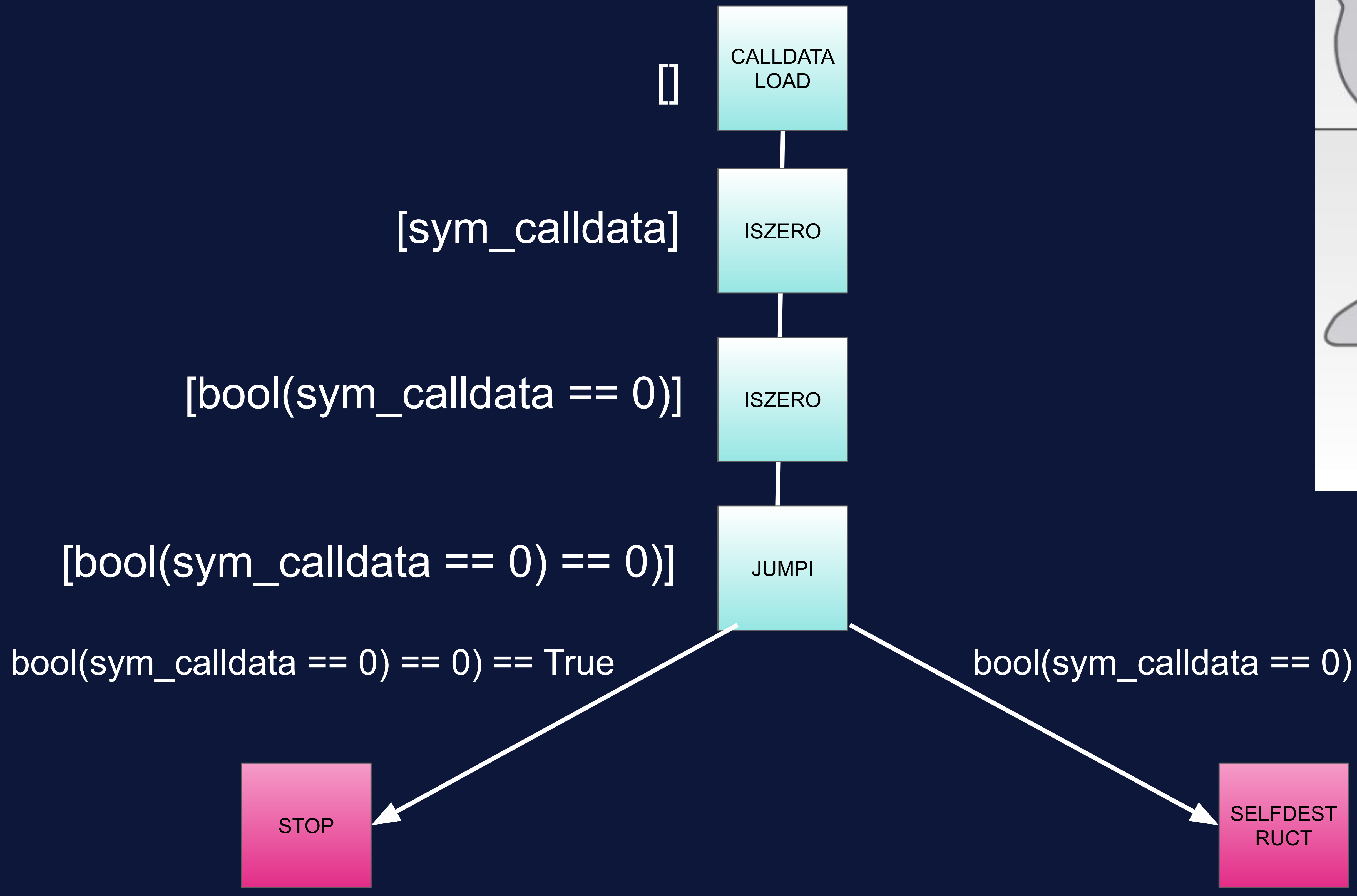


```
contract Cat {

    function extend_life(bool grantSurvival) public {
        if (!grantSurvival) {
            selfdestruct(address(0x0));
        }
    }
}
```



Symbolic Execution (2)

Symbolic Calldata



[] CALLDATA LOAD

[sym_calldata] ISZERO

[bool(sym_calldata == 0)] ISZERO

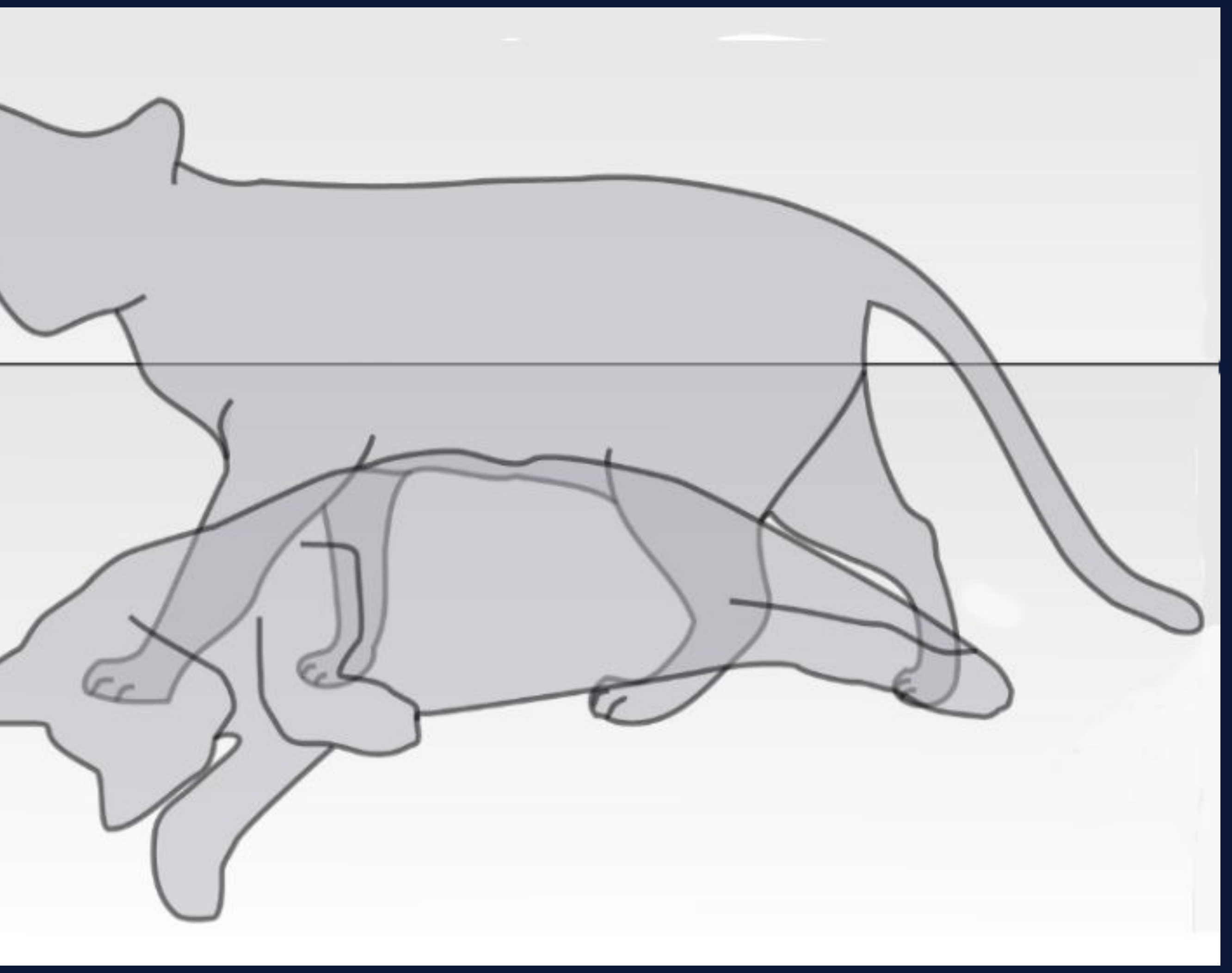
[bool(sym_calldata == 0) == 0)] JUMPI

bool(sym_calldata == 0) == 0) == True

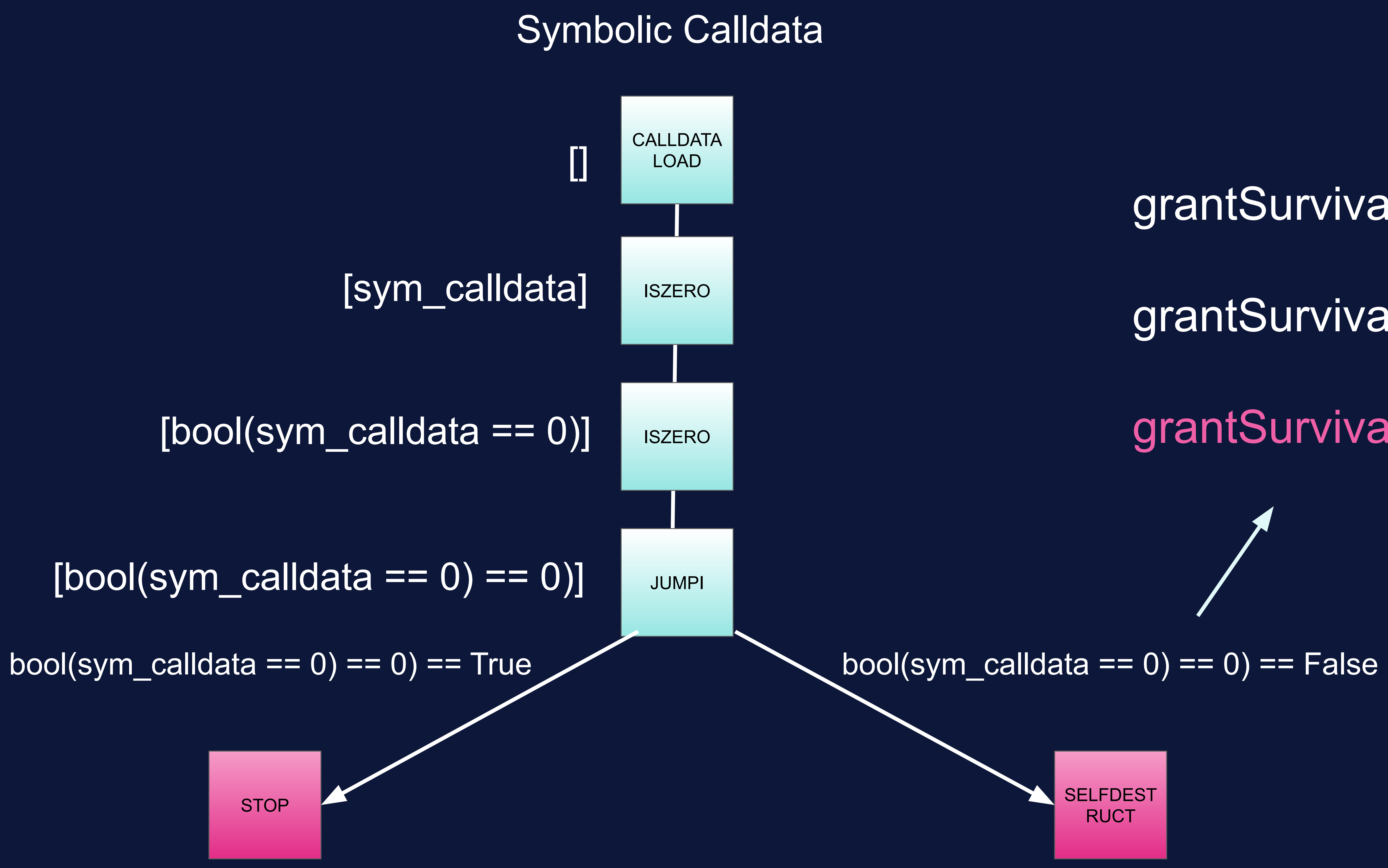
bool(sym_calldata == 0) == 0) == False

STOP

SELFDEST RUCT



How to Kill the Cat?



Symbolic Calldata

[] CALLDATA LOAD

[sym_calldata] ISZERO

[bool(sym_calldata == 0)] ISZERO

[bool(sym_calldata == 0) == 0)] JUMPI

bool(sym_calldata == 0) == 0) == True

bool(sym_calldata == 0) == 0) == False

STOP

SELFDEST RUCT

grantSurvival = ((0 == 0) == 0) == True

grantSurvival = (True == False) == True

grantSurvival = False

Mythril Basic Usage

```
$ pip3 install mythril
```

```
$ myth analyze <solidity_file>[:contract_name]
```

```
$ myth analyze -a <address>
```

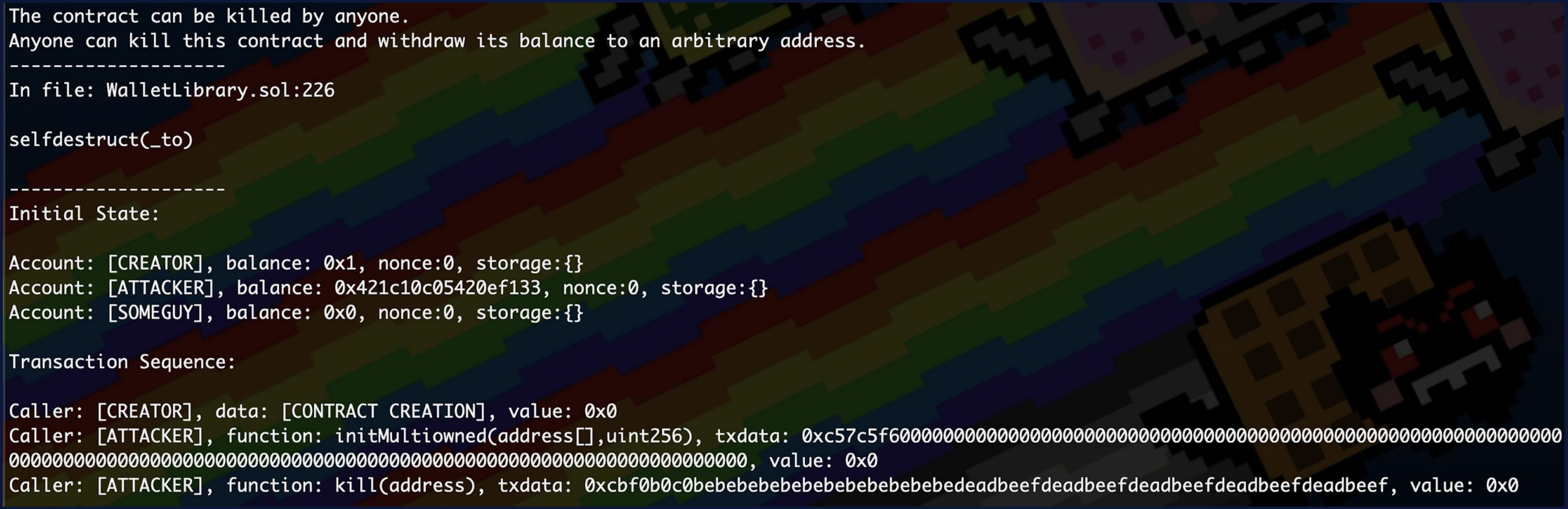
(Demo)

Classic Example

```
                                             OOOOOOOO
  O             OO    O       OO             O
  OO           OO  O   O    O  OO           OO
   O           OO   O   O   O  OO           O
   O          OO           O    OO          O
   OO         OO          O     OO         OO
    O         OO        OO      OO         O
    OO         OO     OO       OO         OO
      O        OO              OO         O
      OOOOOOOO  OO            OO  OOOOOOOO

“I accidentally killed it*”
```

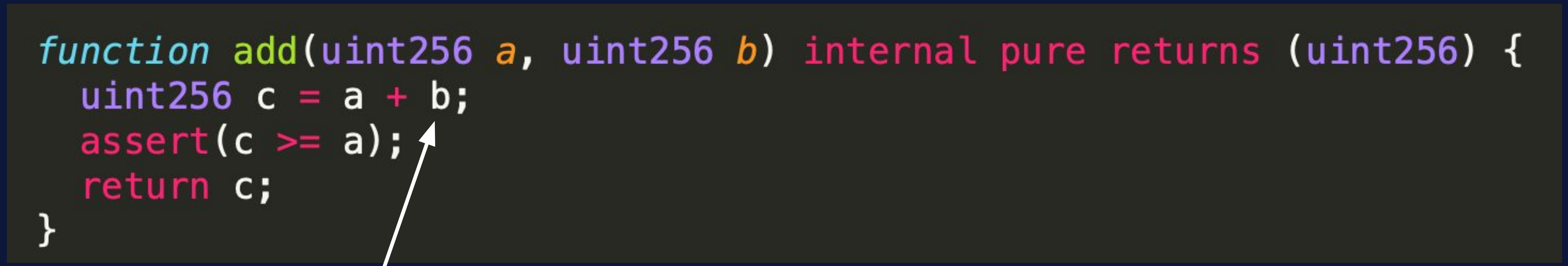
* Parity WalletLibrary

Demo: WalletLibrary



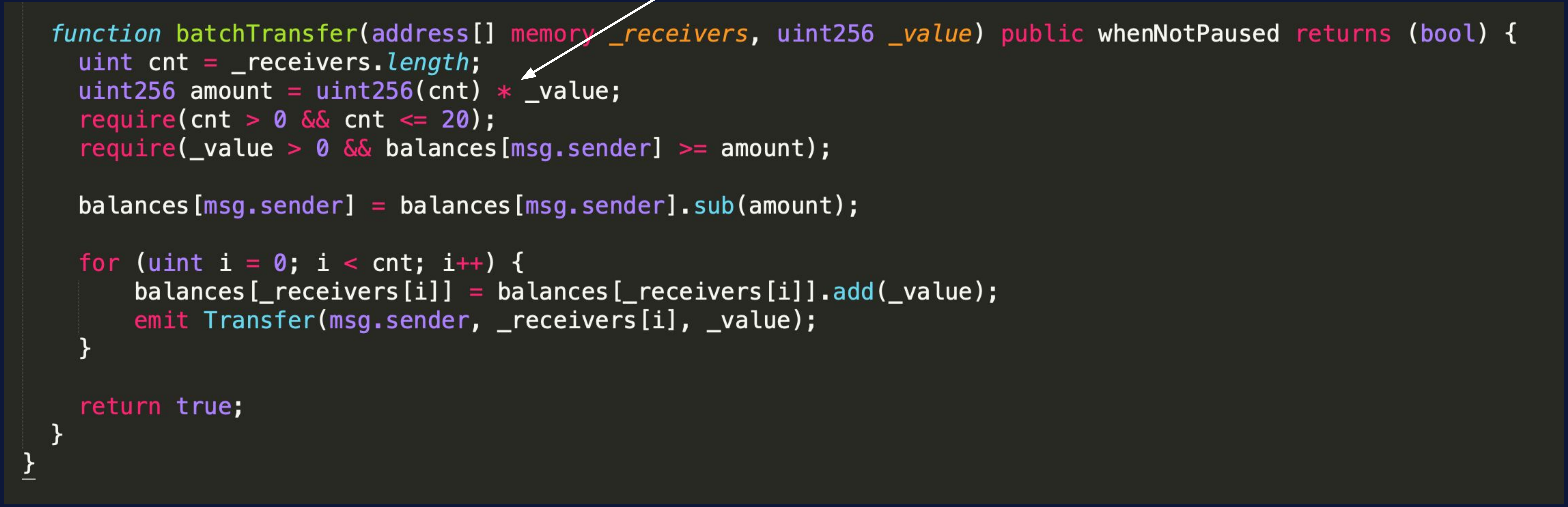
```
The contract can be killed by anyone.
Anyone can kill this contract and withdraw its balance to an arbitrary address.
--------------------
In file: WalletLibrary.sol:226

selfdestruct(_to)

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

Account: [CREATOR], balance: 0x1, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x421c10c05420ef133, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x0, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x0
Caller: [ATTACKER], function: initMultiowned(address[],uint256), txdata: 0xc57c5f60000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [ATTACKER], function: kill(address), txdata: 0xcbf0b0c0bebebebebebebebebebebebebebebebedeadbeefdeadbeefdeadbeefdeadbeef, value: 0x0
```

Challenge: Spurious Issues



```
function add(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a + b;
  assert(c >= a);
  return c;
}
```

“batchOverflow”:
MUL overflow
escapes to storage



```
  function batchTransfer(address[] memory _receivers, uint256 _value) public whenNotPaused returns (bool) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;
    require(cnt > 0 && cnt <= 20);
    require(_value > 0 && balances[msg.sender] >= amount);

    balances[msg.sender] = balances[msg.sender].sub(amount);

    for (uint i = 0; i < cnt; i++) {
        balances[_receivers[i]] = balances[_receivers[i]].add(_value);
        emit Transfer(msg.sender, _receivers[i], _value);
    }

    return true;
  }
}
```

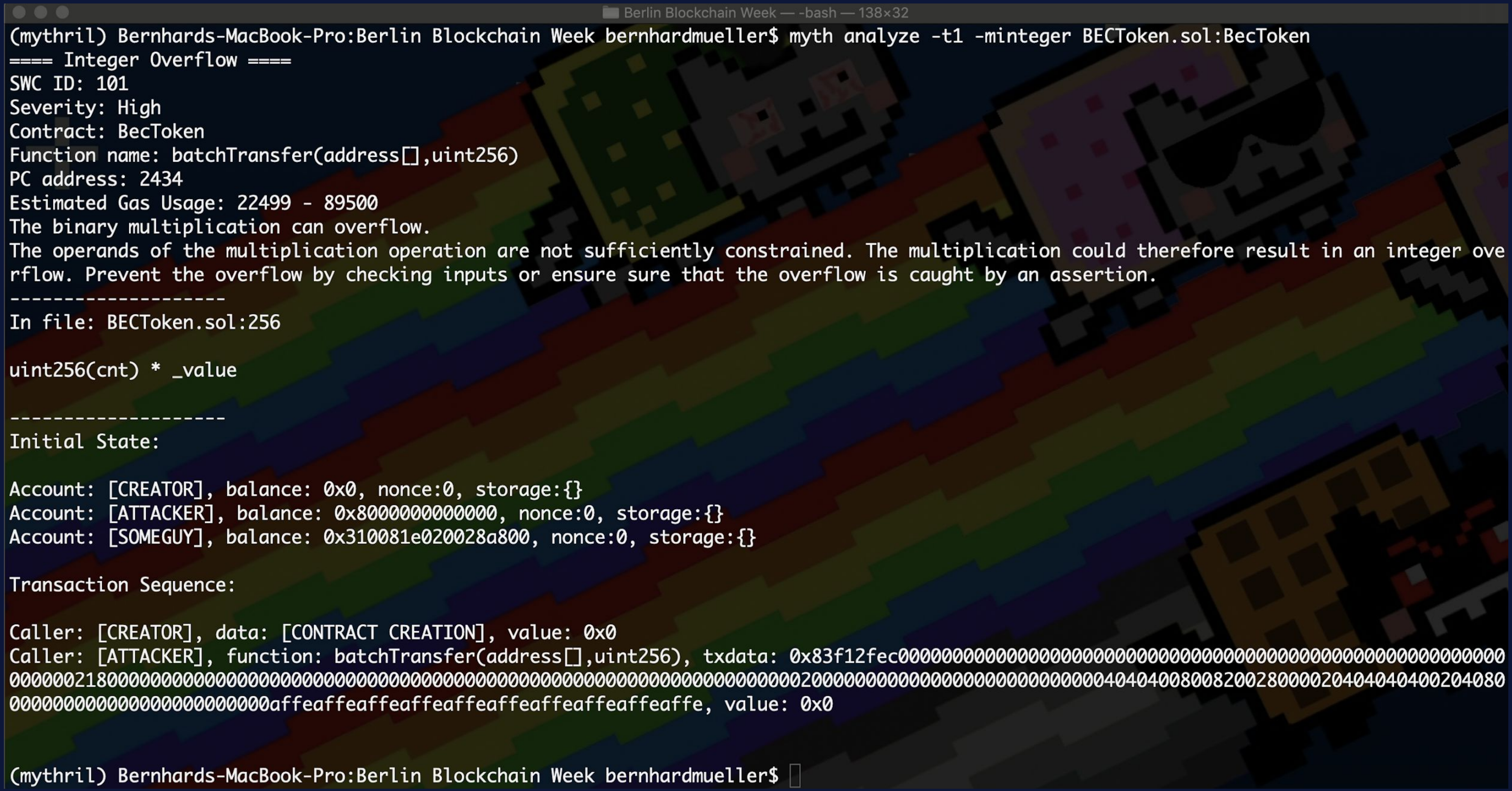
From SafeMath:
Overflow caught by
assert/require

Integer Overflow Detection

- How we determine relevance of over/underflows:
  - Taint symbolic expressions created by arithmetic instruction
  - Check whether the result affects state somewhere along a path (control flow or write to storage)
  - When a STOP is reached, check whether the expression can overflow AND the STOP state is reachable if it does

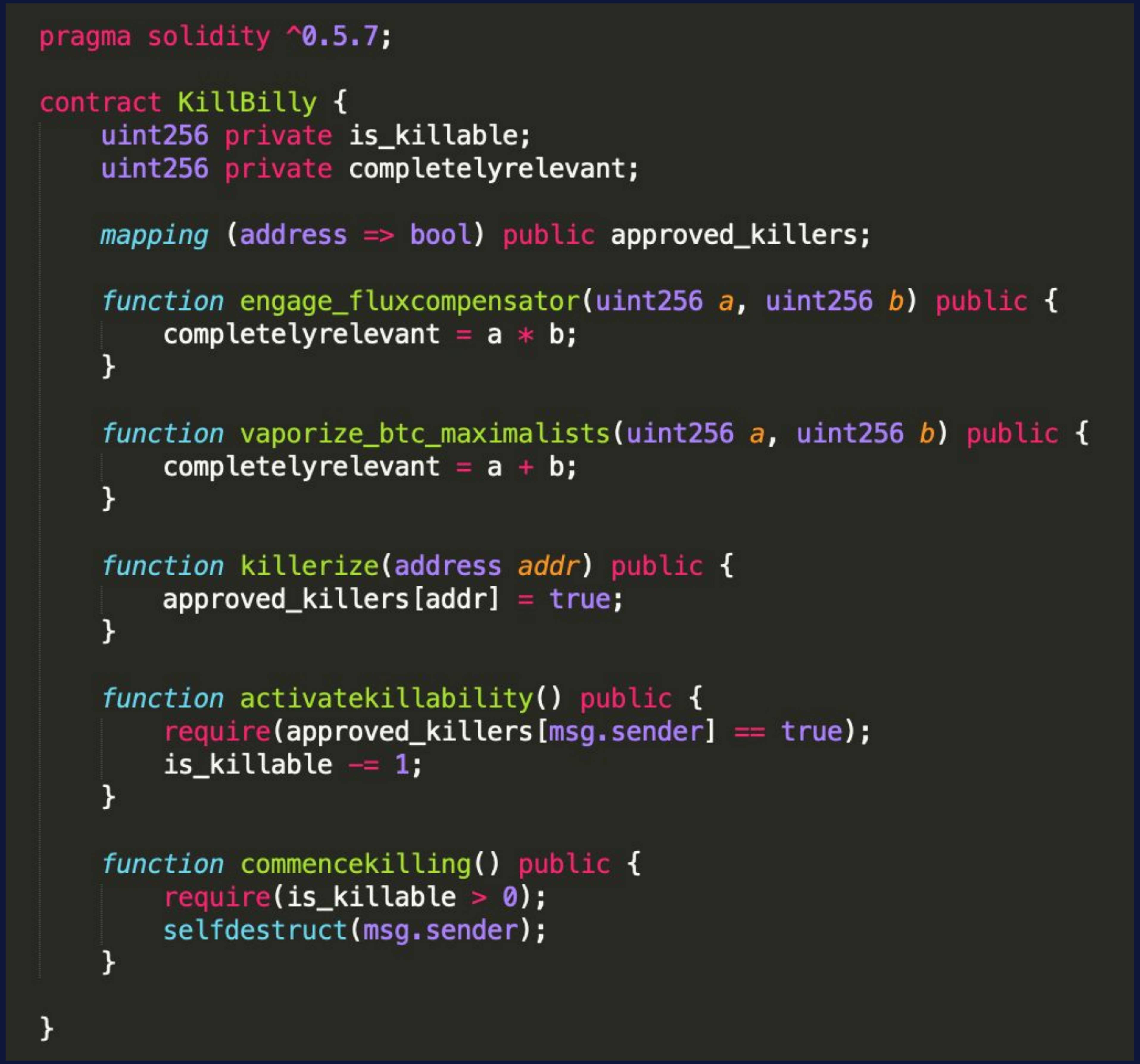
Demo: Beautychain

Demo: Beautychain

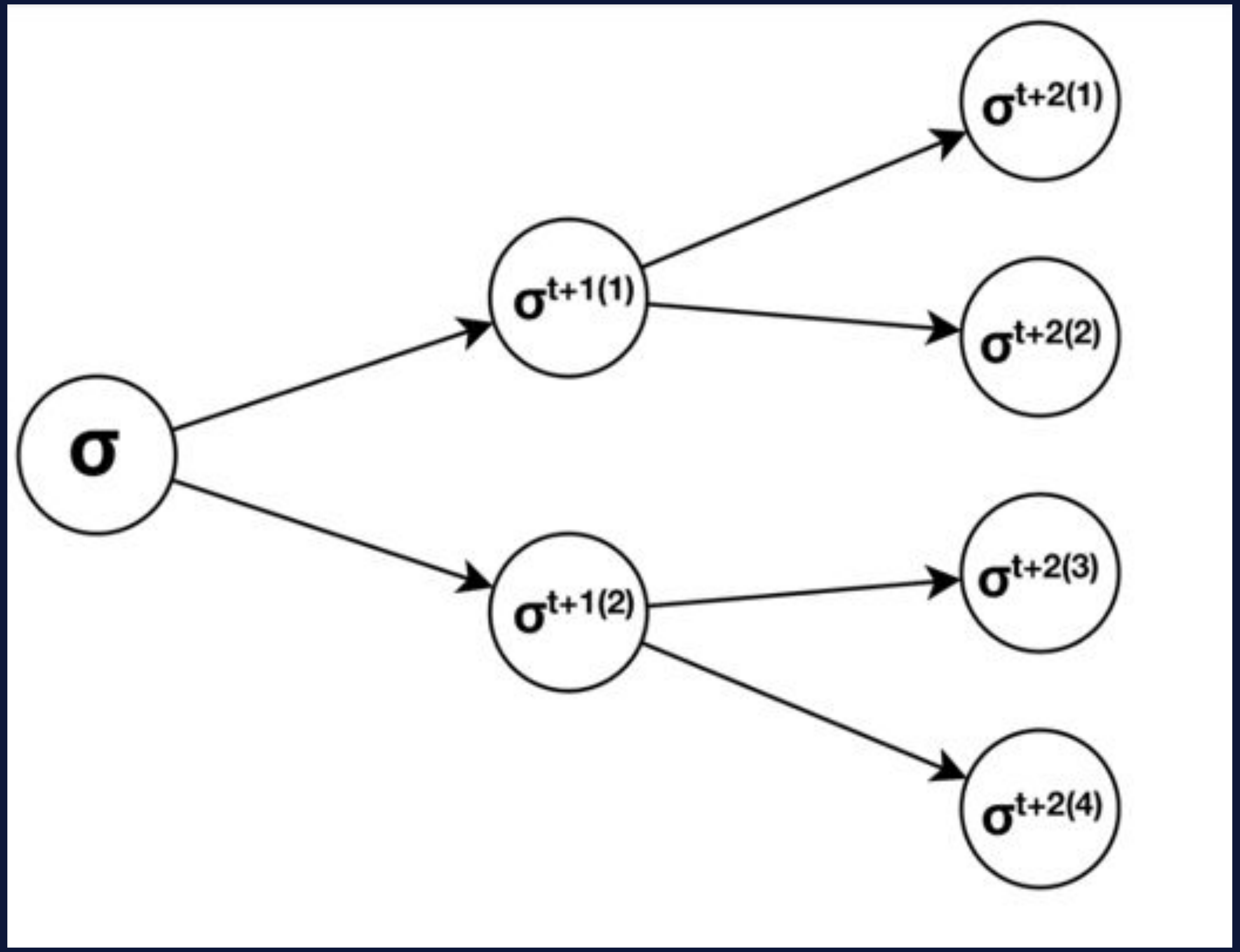


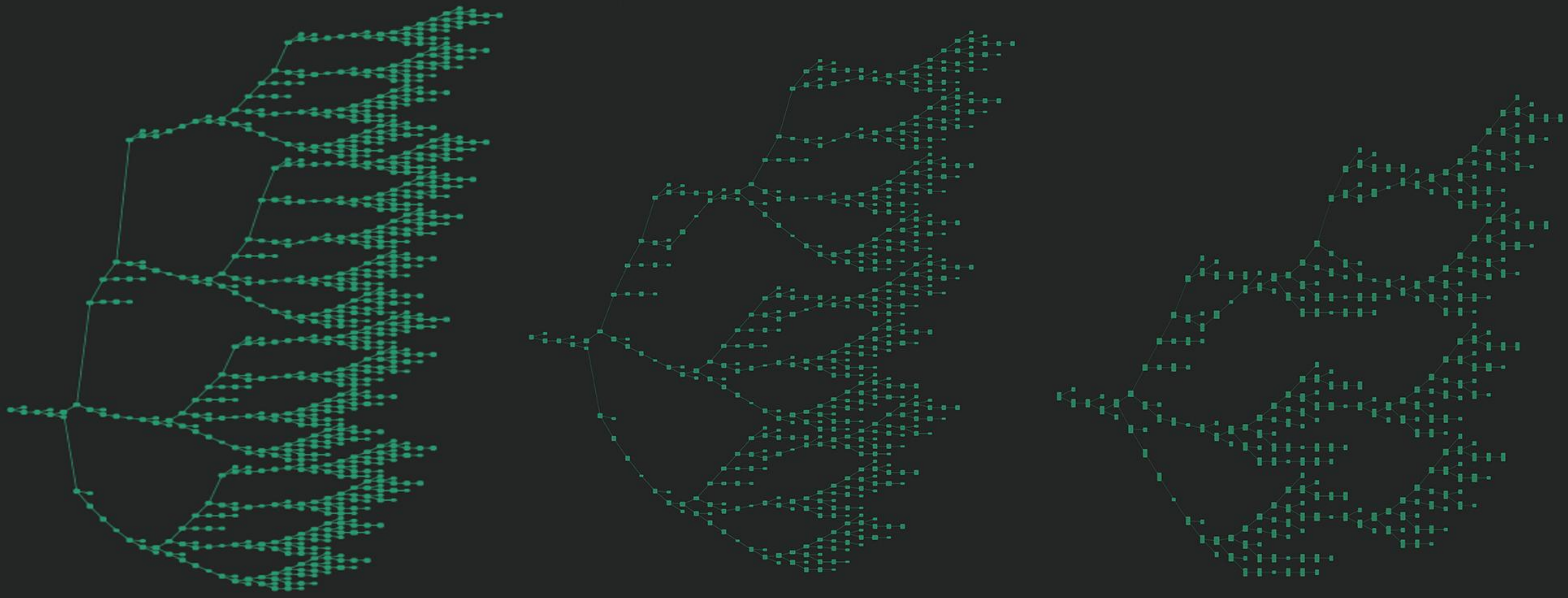
```
(mythril) Bernhards-MacBook-Pro:Berlin Blockchain Week bernhardmueller$ myth analyze -t1 -minteger BECToken.sol:BecToken
==== Integer Overflow ====
SWC ID: 101
Severity: High
Contract: BecToken
Function name: batchTransfer(address[],uint256)
PC address: 2434
Estimated Gas Usage: 22499 - 89500
The binary multiplication can overflow.
The operands of the multiplication operation are not sufficiently constrained. The multiplication could therefore result in an integer ove
rflow. Prevent the overflow by checking inputs or ensure sure that the overflow is caught by an assertion.
--------------------
In file: BECToken.sol:256

uint256(cnt) * _value

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

Account: [CREATOR], balance: 0x0, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x8000000000000, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x310081e020028a800, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x0
Caller: [ATTACKER], function: batchTransfer(address[],uint256), txdata: 0x83f12fec00000000000000000000000000000000000000000000000000000000
000000218000000000000000000000000000000000000000000000000000000000000000000000200000000000000000000004040400800820028000020404040400204080
000000000000000000000000affeaffeaffeaffeaffeaffeaffeaffeaffeaffe, value: 0x0


(mythril) Bernhards-MacBook-Pro:Berlin Blockchain Week bernhardmueller$ 
```

Challenge: State Space Explosion



```
pragma solidity ^0.5.7;

contract KillBilly {
    uint256 private is_killable;
    uint256 private completelyrelevant;

    mapping (address => bool) public approved_killers;

    function engage_fluxcompensator(uint256 a, uint256 b) public {
        completelyrelevant = a * b;
    }

    function vaporize_btc_maximalists(uint256 a, uint256 b) public {
        completelyrelevant = a + b;
    }

    function killerize(address addr) public {
        approved_killers[addr] = true;
    }

    function activatekillability() public {
        require(approved_killers[msg.sender] == true);
        is_killable -= 1;
    }

    function commencekilling() public {
        require(is_killable > 0);
        selfdestruct(msg.sender);
    }

}
```





no pruning
8,807 states

prune pure functions
5,636 states (-36%)

dynamic pruning
3,355 states (-62%)

Mythril v0.21.12
State space graph for 3 transactions
killbilly.sol - https://gist.github.com/b-mueller/8fcf3b8a2c0f0b691ecc0ef3e245c1c7

Mythril Pruning Algorithms

- Prune unreachable paths given concrete initial state
- Prune pure functions (STOP state == initial state)
- Dynamic pruning. Execute a path only if:
  - It is newly discovered
  - A state variable that was modified in the previous transaction is read somewhere along the path
  - Somewhere along this path, a state variable is written to that we know is being read elsewhere

teEther uses a similar method: https://www.usenix.org/node/217465

Pruning Effectiveness

Fully execute 63 samples from the smart contract weakness registry

https://smartcontractsecurity.github.io/SWC-registry/

	Base	Prune Pure Funcs	Dynamic Pruning	Speedup
1 TX	297s	N/A	N/A	N/A
2 TX	2,346s	1,919s	1,152s	103.5%
3 TX	9,943s	6,072s	2,242s	343.49%
4 TX	too long	13,312s	7,440s	> 400%

Customizing the analysis

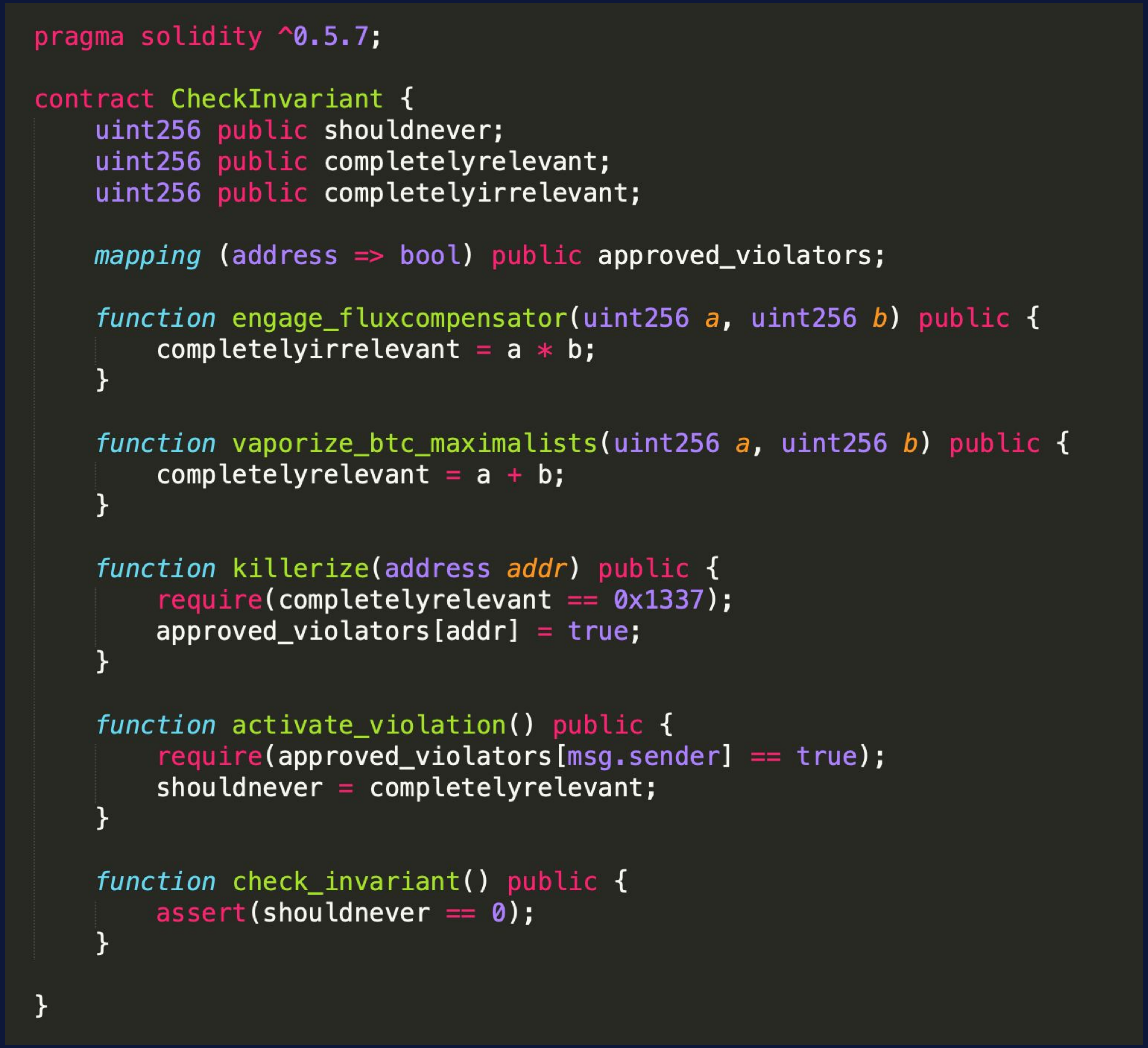
```
$ myth -m exceptions analyze -t4 --execution-timeout 3600 <solidity_file>
```

Only check for exceptions

Exhaustively execute 4 transactions

Terminate after 1 hour and return results

Demo: Multi-Tx



```
pragma solidity ^0.5.7;

contract CheckInvariant {
    uint256 public shouldnever;
    uint256 public completelyrelevant;
    uint256 public completelyirrelevant;

    mapping (address => bool) public approved_violators;

    function engage_fluxcompensator(uint256 a, uint256 b) public {
        completelyirrelevant = a * b;
    }

    function vaporize_btc_maximalists(uint256 a, uint256 b) public {
        completelyrelevant = a + b;
    }

    function killerize(address addr) public {
        require(completelyrelevant == 0x1337);
        approved_violators[addr] = true;
    }

    function activate_violation() public {
        require(approved_violators[msg.sender] == true);
        shouldnever = completelyrelevant;
    }

    function check_invariant() public {
        assert(shouldnever == 0);
    }

}
```

Invariant Checking Cheat Sheet:

1. Write assertion

2. Run:

   $ myth analyze -t <num_transactions> -mexceptions <solidity_file>

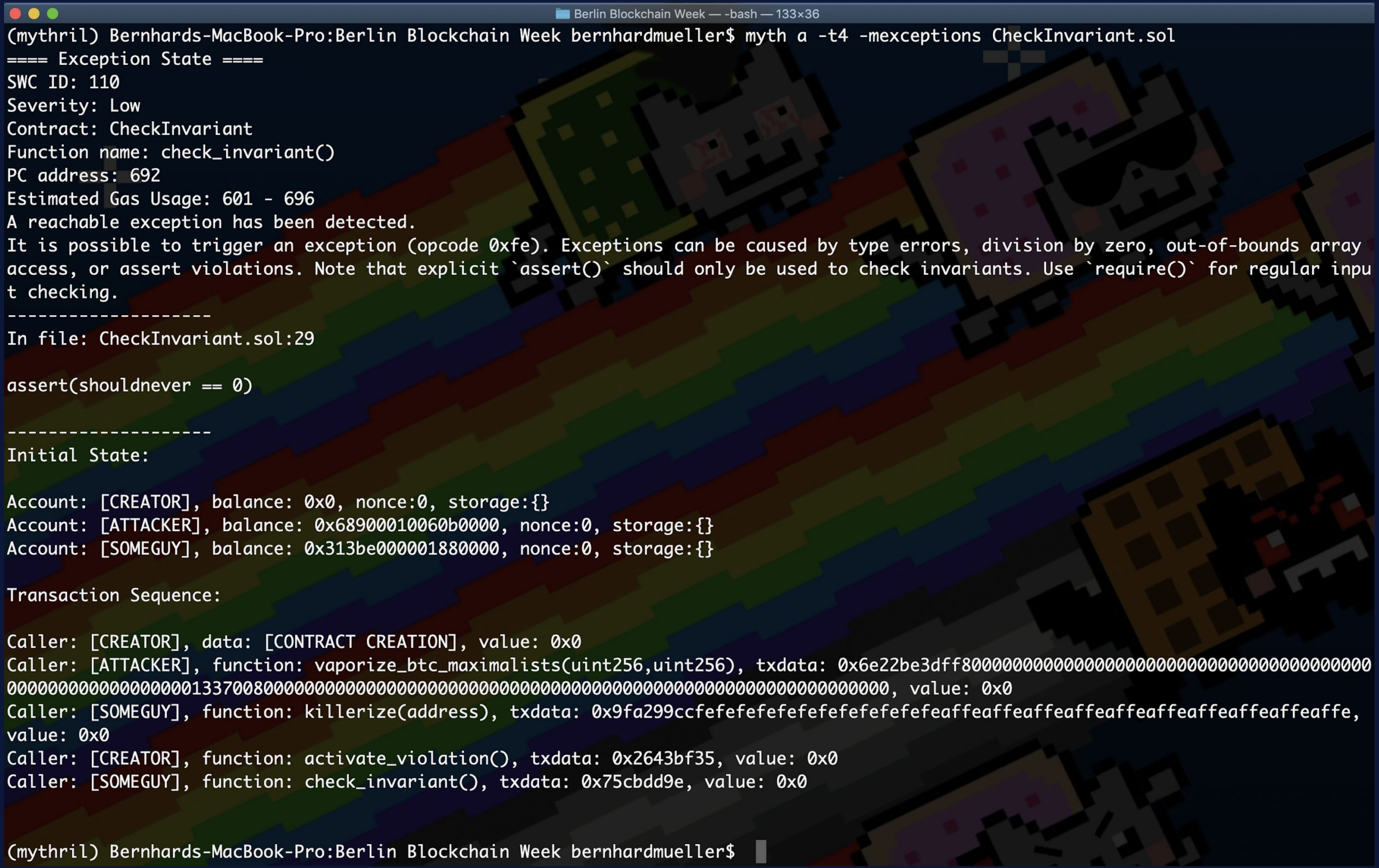
   (initializes state with constructor)

   OR

   $ myth analyze -t <num_transactions> -mexceptions -a <contract_address>

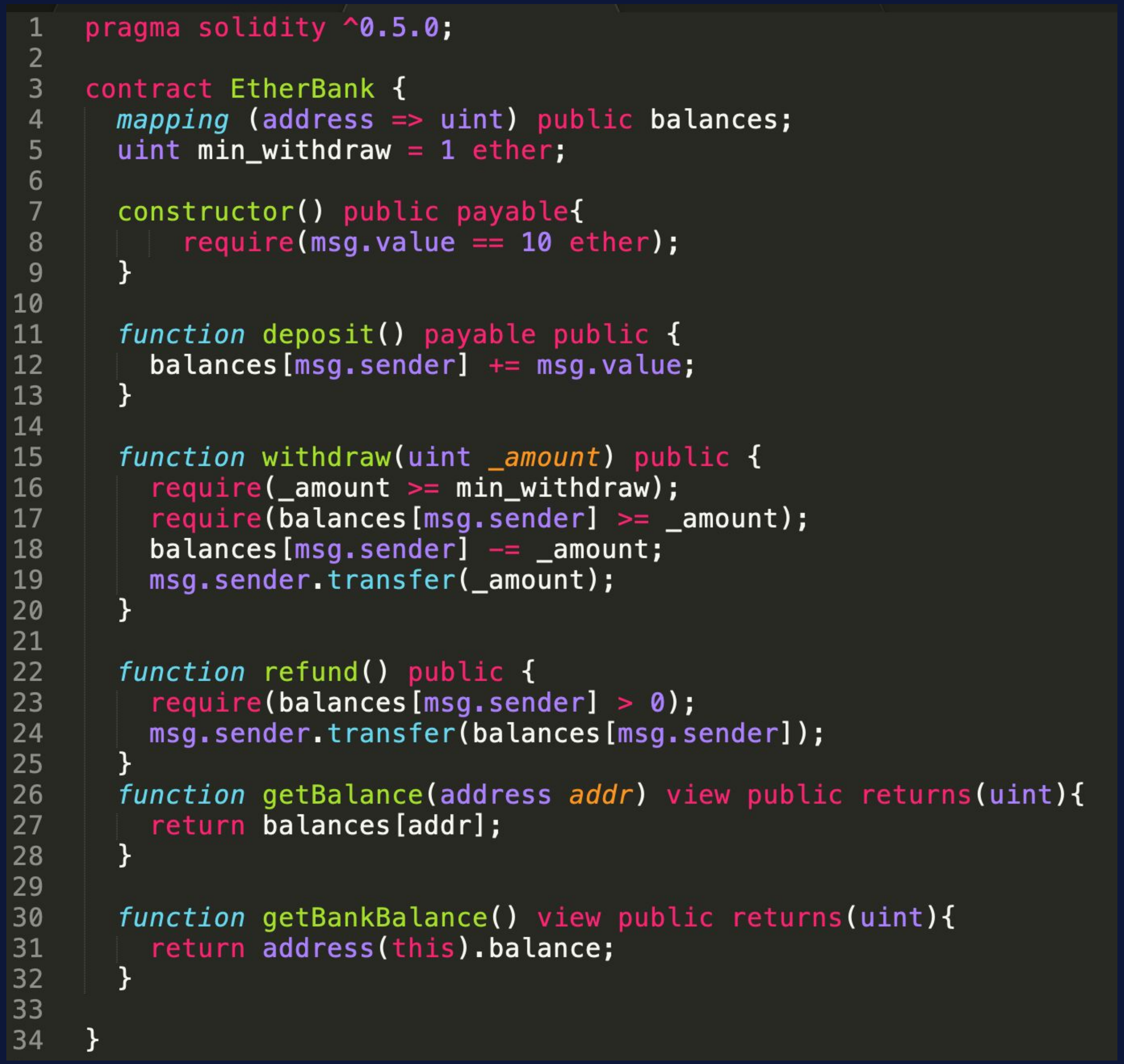
   (loads state & dependencies from node)

Demo: Multi-Tx

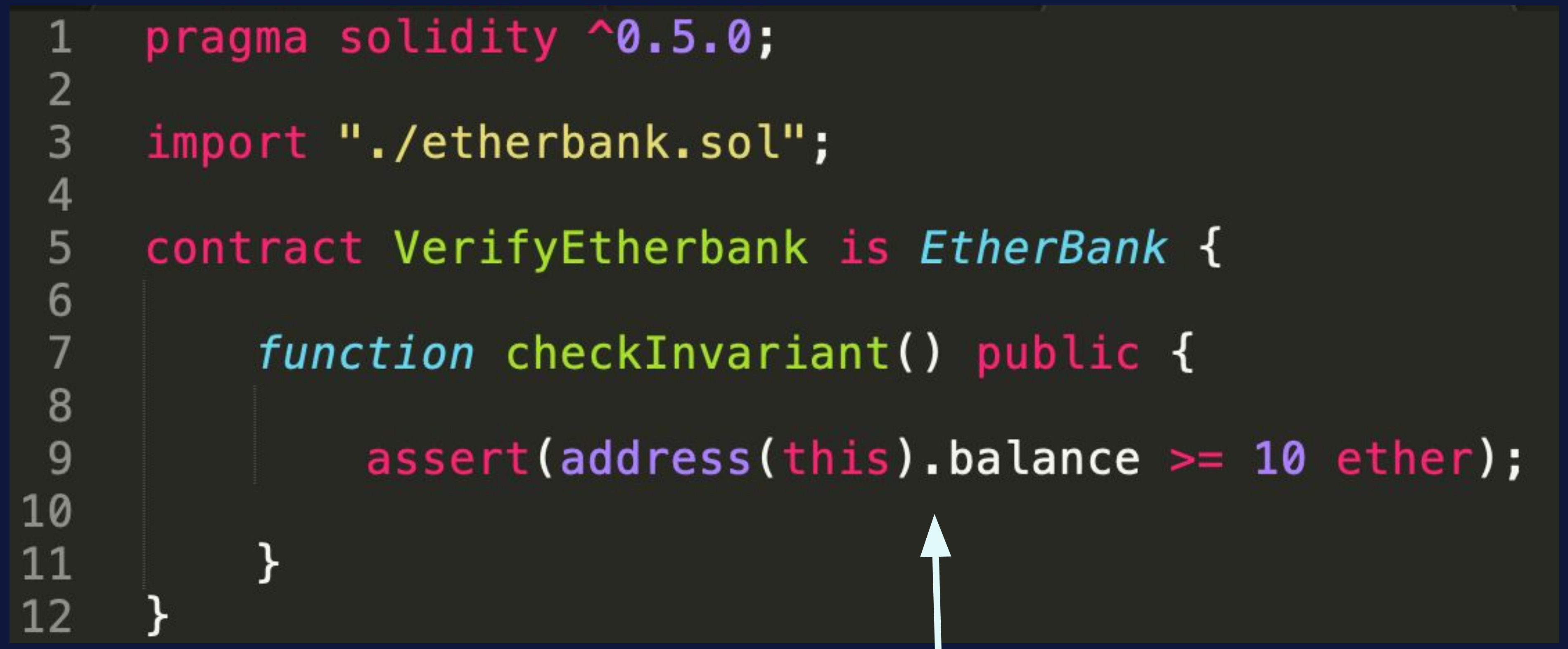


```
(mythril) Bernhards-MacBook-Pro:Berlin Blockchain Week bernhardmueller$ myth a -t4 -mexceptions CheckInvariant.sol
==== Exception State ====
SWC ID: 110
Severity: Low
Contract: CheckInvariant
Function name: check_invariant()
PC address: 692
Estimated Gas Usage: 601 - 696
A reachable exception has been detected.
It is possible to trigger an exception (opcode 0xfe). Exceptions can be caused by type errors, division by zero, out-of-bounds array
access, or assert violations. Note that explicit `assert()` should only be used to check invariants. Use `require()` for regular inpu
t checking.
--------------------
In file: CheckInvariant.sol:29

assert(shouldnever == 0)

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

Account: [CREATOR], balance: 0x0, nonce:0, storage:{}
Account: [ATTACKER], balance: 0x68900010060b0000, nonce:0, storage:{}
Account: [SOMEGUY], balance: 0x313be000001880000, nonce:0, storage:{}

Transaction Sequence:

Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x0
Caller: [ATTACKER], function: vaporize_btc_maximalists(uint256,uint256), txdata: 0x6e22be3dff8000000000000000000000000000000000000000
00000000000000000133700800000000000000000000000000000000000000000000000000000000000000, value: 0x0
Caller: [SOMEGUY], function: killerize(address), txdata: 0x9fa299ccfefefefefefefefefefefefeaffeaffeaffeaffeaffeaffeaffeaffeaffeaffe,
value: 0x0
Caller: [CREATOR], function: activate_violation(), txdata: 0x2643bf35, value: 0x0
Caller: [SOMEGUY], function: check_invariant(), txdata: 0x75cbdd9e, value: 0x0


(mythril) Bernhards-MacBook-Pro:Berlin Blockchain Week bernhardmueller$ 
```

Verifying an Invariant



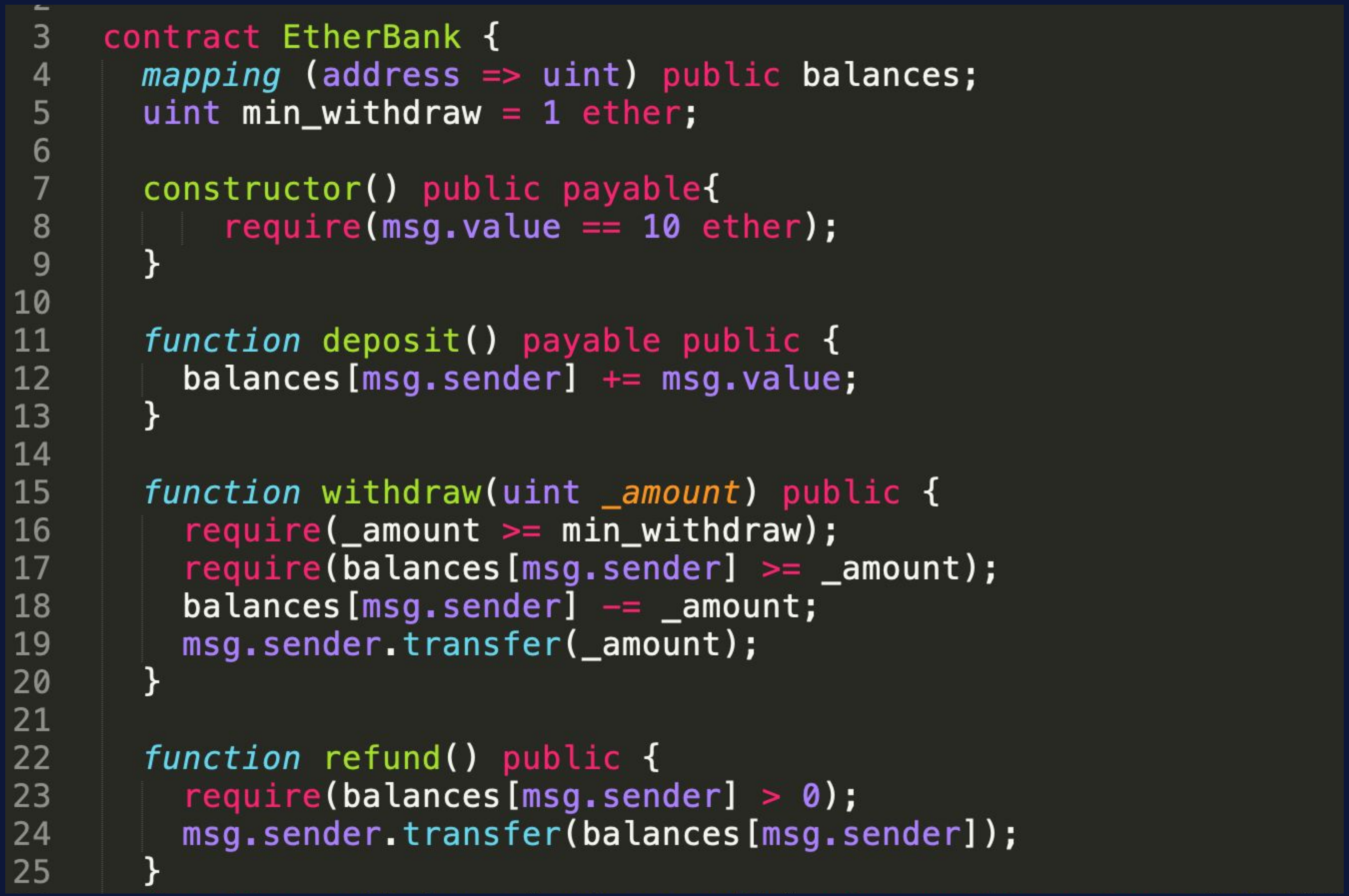
```
pragma solidity ^0.5.0;

contract EtherBank {
  mapping (address => uint) public balances;
  uint min_withdraw = 1 ether;

  constructor() public payable{
      require(msg.value == 10 ether);
  }

  function deposit() payable public {
    balances[msg.sender] += msg.value;
  }

  function withdraw(uint _amount) public {
    require(_amount >= min_withdraw);
    require(balances[msg.sender] >= _amount);
    balances[msg.sender] -= _amount;
    msg.sender.transfer(_amount);
  }

  function refund() public {
    require(balances[msg.sender] > 0);
    msg.sender.transfer(balances[msg.sender]);
  }
  function getBalance(address addr) view public returns(uint){
    return balances[addr];
  }

  function getBankBalance() view public returns(uint){
    return address(this).balance;
  }

}
```



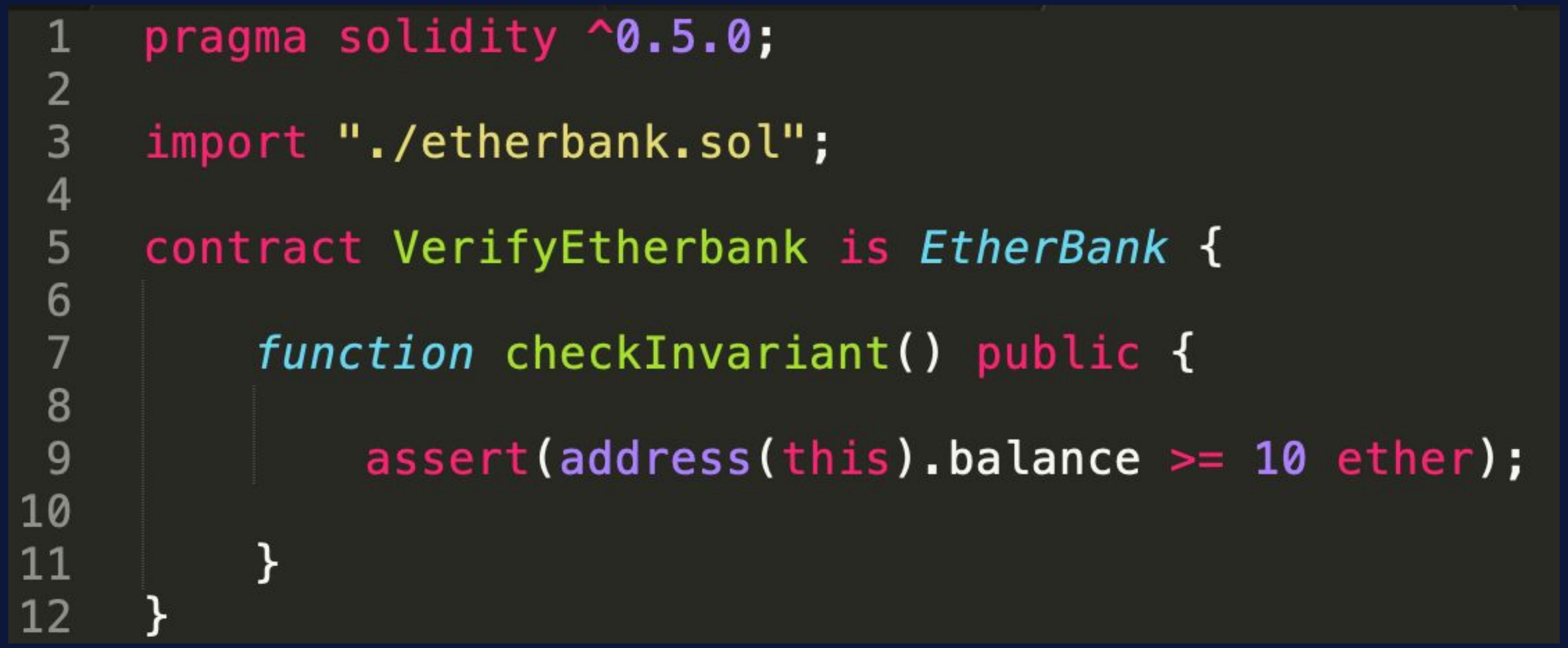
```
pragma solidity ^0.5.0;

import "./etherbank.sol";

contract VerifyEtherbank is EtherBank {

    function checkInvariant() public {

        assert(address(this).balance >= 10 ether);

    }
}
```

This is supposed to always hold

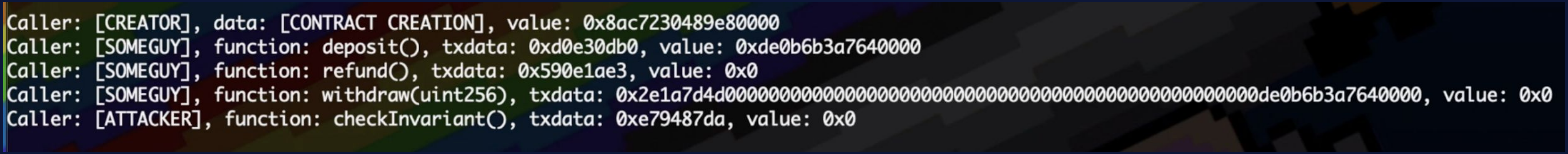
Verifying an Invariant



```
contract EtherBank {
  mapping (address => uint) public balances;
  uint min_withdraw = 1 ether;

  constructor() public payable{
      require(msg.value == 10 ether);
  }

  function deposit() payable public {
    balances[msg.sender] += msg.value;
  }

  function withdraw(uint _amount) public {
    require(_amount >= min_withdraw);
    require(balances[msg.sender] >= _amount);
    balances[msg.sender] -= _amount;
    msg.sender.transfer(_amount);
  }

  function refund() public {
    require(balances[msg.sender] > 0);
    msg.sender.transfer(balances[msg.sender]);
  }
```



```
pragma solidity ^0.5.0;

import "./etherbank.sol";

contract VerifyEtherbank is EtherBank {

    function checkInvariant() public {

        assert(address(this).balance >= 10 ether);

    }
}
```



```
Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x8ac7230489e80000
Caller: [SOMEGUY], function: deposit(), txdata: 0xd0e30db0, value: 0xde0b6b3a7640000
Caller: [SOMEGUY], function: refund(), txdata: 0x590e1ae3, value: 0x0
Caller: [SOMEGUY], function: withdraw(uint256), txdata: 0x2e1a7d4d0000000000000000000000000000000000000000000000000de0b6b3a7640000, value: 0x0
Caller: [ATTACKER], function: checkInvariant(), txdata: 0xe79487da, value: 0x0
```

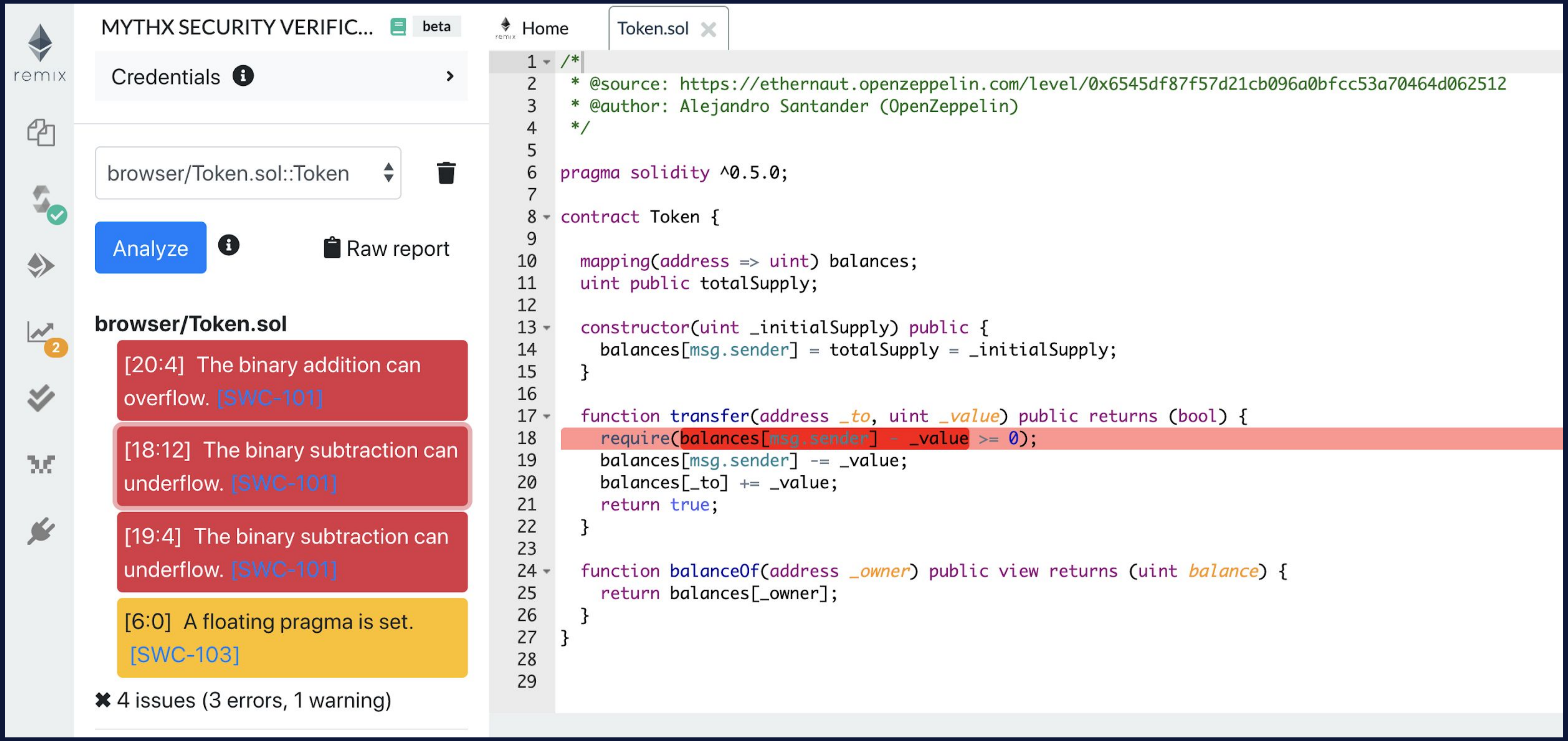
MythX Security API

- Does everything that Mythril does and a lot more
- Linting + data flow analysis + symbolic execution + input fuzzing
- Using the CLI:

```
$ npm install sabre-mythx
$ sabre <solidity-file> <contract-name>
```

Integration with Remix

- Open “Plugin manager” and activate “MythX Security Verification”



browser/Token.sol

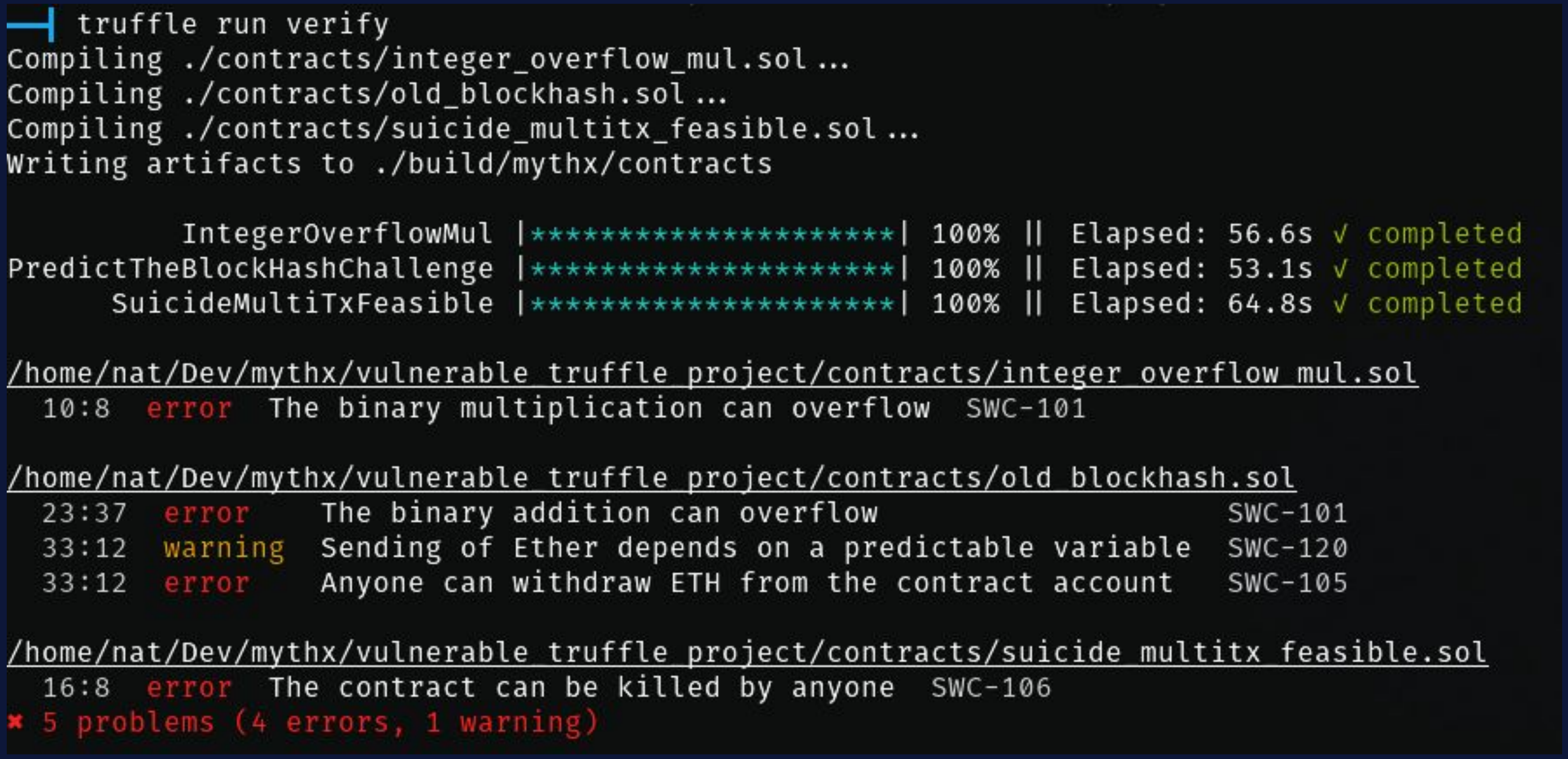
- [20:4] The binary addition can overflow. [SWC-101]
- [18:12] The binary subtraction can underflow. [SWC-101]
- [19:4] The binary subtraction can underflow. [SWC-101]
- [6:0] A floating pragma is set. [SWC-103]

✖ 4 issues (3 errors, 1 warning)

```
 1  /*
 2   * @source: https://ethernaut.openzeppelin.com/level/0x6545df87f57d21cb096a0bfcc53a70464d062512
 3   * @author: Alejandro Santander (OpenZeppelin)
 4   */
 5
 6  pragma solidity ^0.5.0;
 7
 8  contract Token {
 9
10    mapping(address => uint) balances;
11    uint public totalSupply;
12
13    constructor(uint _initialSupply) public {
14      balances[msg.sender] = totalSupply = _initialSupply;
15    }
16
17    function transfer(address _to, uint _value) public returns (bool) {
18      require(balances[msg.sender] - _value >= 0);
19      balances[msg.sender] -= _value;
20      balances[_to] += _value;
21      return true;
22    }
23
24    function balanceOf(address _owner) public view returns (uint balance) {
25      return balances[_owner];
26    }
27  }
28
29
```

Integration with Truffle

```
$ npm install truffle-security
$ truffle run verify
```



```
    truffle run verify
Compiling ./contracts/integer_overflow_mul.sol...
Compiling ./contracts/old_blockhash.sol...
Compiling ./contracts/suicide_multitx_feasible.sol...
Writing artifacts to ./build/mythx/contracts

          IntegerOverflowMul |*********************| 100% || Elapsed: 56.6s √ completed
PredictTheBlockHashChallenge |*********************| 100% || Elapsed: 53.1s √ completed
      SuicideMultiTxFeasible |*********************| 100% || Elapsed: 64.8s √ completed

/home/nat/Dev/mythx/vulnerable_truffle_project/contracts/integer_overflow_mul.sol
  10:8  error  The binary multiplication can overflow  SWC-101

/home/nat/Dev/mythx/vulnerable_truffle_project/contracts/old_blockhash.sol
  23:37  error    The binary addition can overflow                    SWC-101
  33:12  warning  Sending of Ether depends on a predictable variable  SWC-120
  33:12  error    Anyone can withdraw ETH from the contract account   SWC-105

/home/nat/Dev/mythx/vulnerable_truffle_project/contracts/suicide_multitx_feasible.sol
  16:8  error  The contract can be killed by anyone  SWC-106
✖ 5 problems (4 errors, 1 warning)
```

Try our tools!

Many awesome plugins:
Truffle, Visual Studio Code,
Embark, Github,...

- Mythril
  - https://www.github.com/ConsenSys/mythril

Or write your own tools and
earn revenue share!

- MythX
  - https://mythx.io
  - https://www.github.com/b-mueller/awesome-mythx-tools

- Visit the Security Helpdesk at Factory Görlitzer Park

Possible Optimizations (WIP)

- Parallelization
- State merging
  - Merge path constraints and world state by disjunction (c1 v c2)
  - Used by Manticore
- Function summaries
  - Store constraints imposed on state when executing paths (“summary”)
  - In subsequent runs, apply summary via conjunction instead of re-executing the same code
  - Pakala uses a comparable approach
- (...)

Further Reading

- Introduction to Mythril and Symbolic Execution (Joran Honig)
  - https://medium.com/@joran.honig/introduction-to-mythril-classic-and-symbolic-execution-ef59339f259b
- Smashing Smart Contracts
  - https://github.com/b-mueller/smashing-smart-contracts
- teether: Gnawing at Ethereum to Automatically Exploit Smart Contracts (J. Krupp, C. Rossow)
  - https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-krupp.pdf