

Smart Contract Hacking

Konstantinos Karagiannis

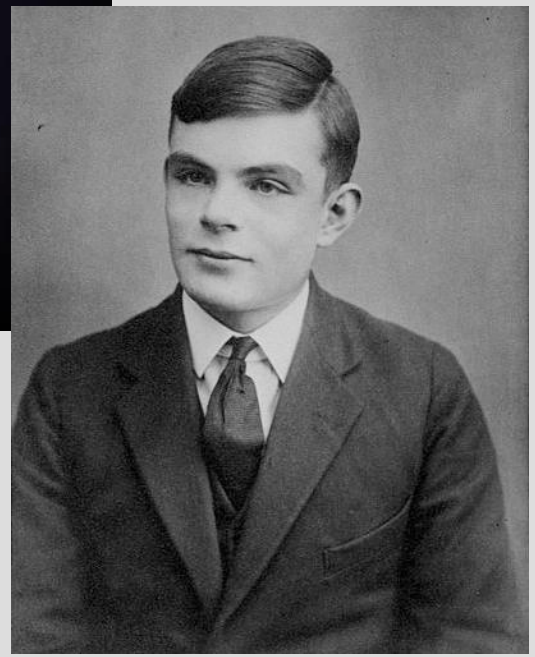
CTO, Security Consulting, BT Americas

@KonstantHacker

BT

when transactions aren't enough





- ***"The key component is this idea of a Turing-complete blockchain"***
- **--Vitalik Buterin**

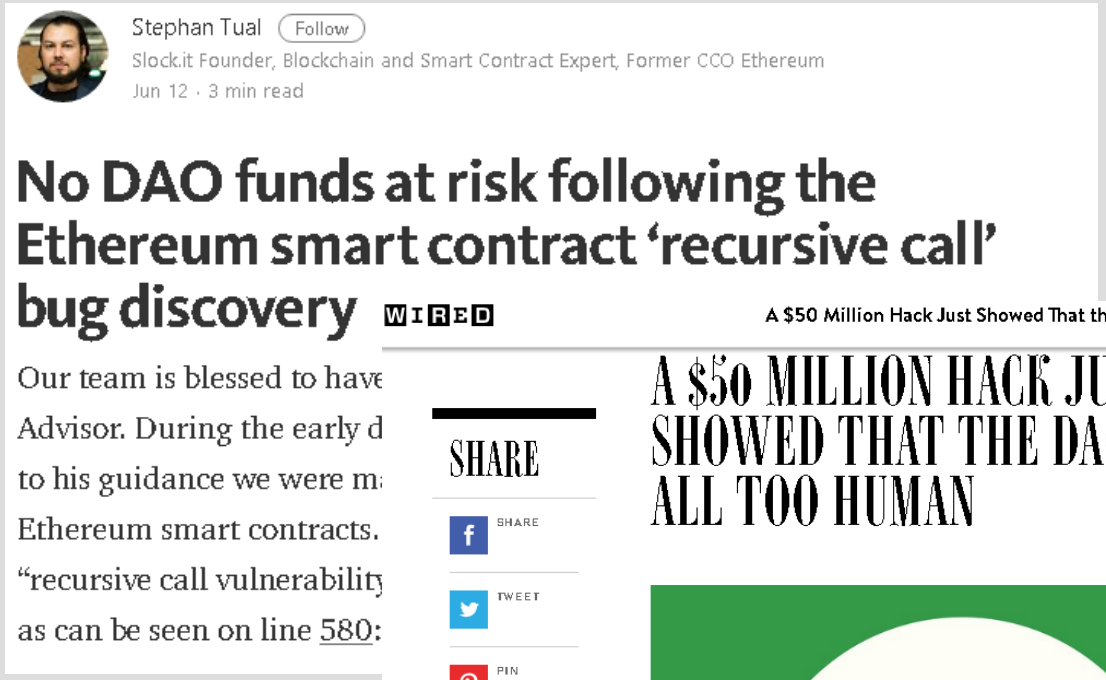
meow—putting that computing to use



smart contracts



billions, or *just* millions, of reasons



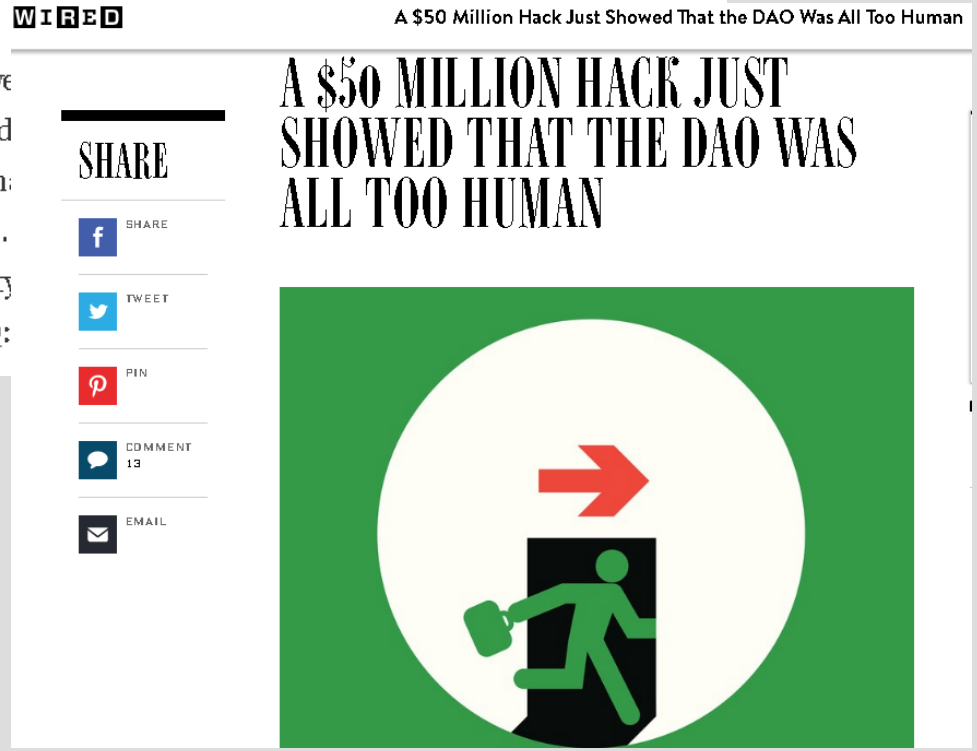
Stephan Tual Follow

Slock.it Founder, Blockchain and Smart Contract Expert, Former CCO Ethereum

Jun 12 · 3 min read

No DAO funds at risk following the Ethereum smart contract 'recursive call' bug discovery

Our team is blessed to have
Advisor. During the early d
to his guidance we were m
Ethereum smart contracts.
"recursive call vulnerability
as can be seen on line 580:



WIRED

A $50 Million Hack Just Showed That the DAO Was All Too Human

SHARE

SHARE

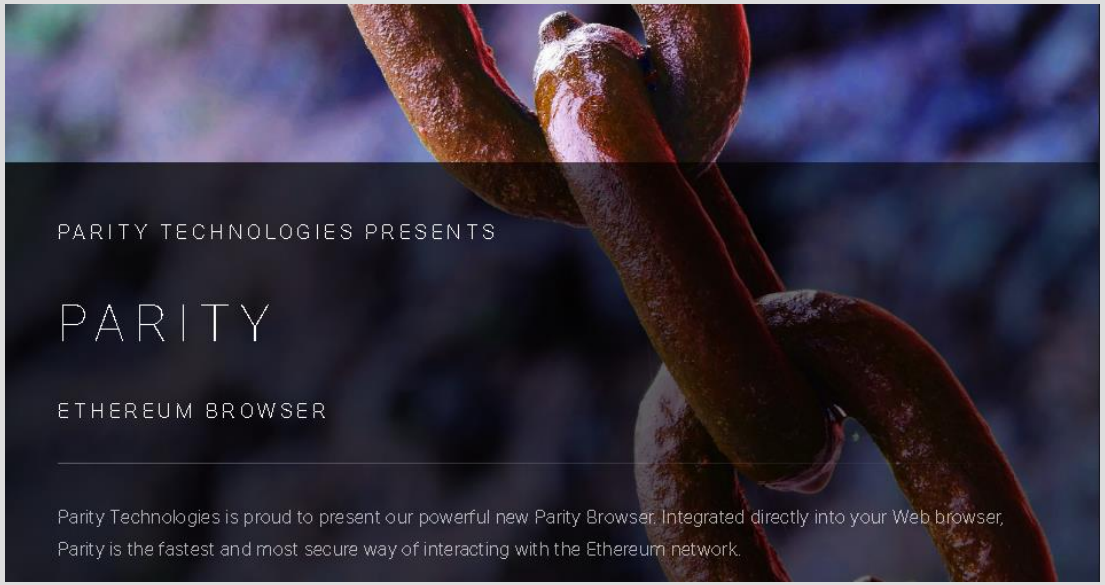
TWEET

PIN

COMMENT 13

EMAIL

A $50 MILLION HACK JUST SHOWED THAT THE DAO WAS ALL TOO HUMAN



problem isn't going away

Category	**#Candidates flagged** (*distinct*)	Candidates without source	#Validated	% of true positives
Prodigal	**1504** (*438*)	1487	1253	*97*
Suicidal	**1495** (*403*)	1487	1423	*99*
Greedy	**31,201** (*1524*)	31,045	1083	*69*
Total	34,200 (2,365)	34,019	3,759	89

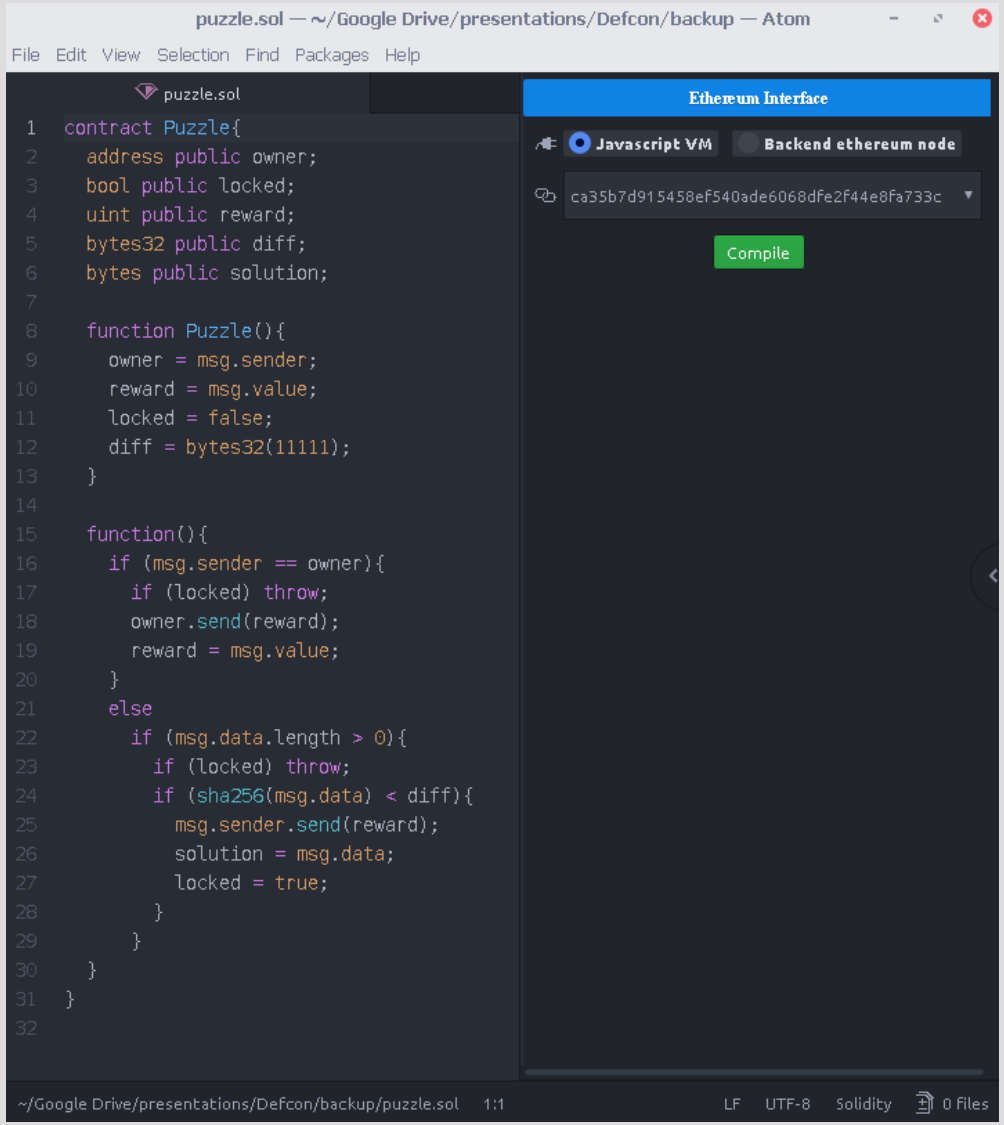
Table 1: Final results using invocation depth 3 at block height BH. Column 1 reports number of flagged contracts, and the distinct among these. Column 2 shows the number of flagged which have no source code. Column 3 is the subset we sampled for concrete validation. Column 4 reports true positive rates; the total here is the average TP rate weighted by the number of validated contracts.

Solidity

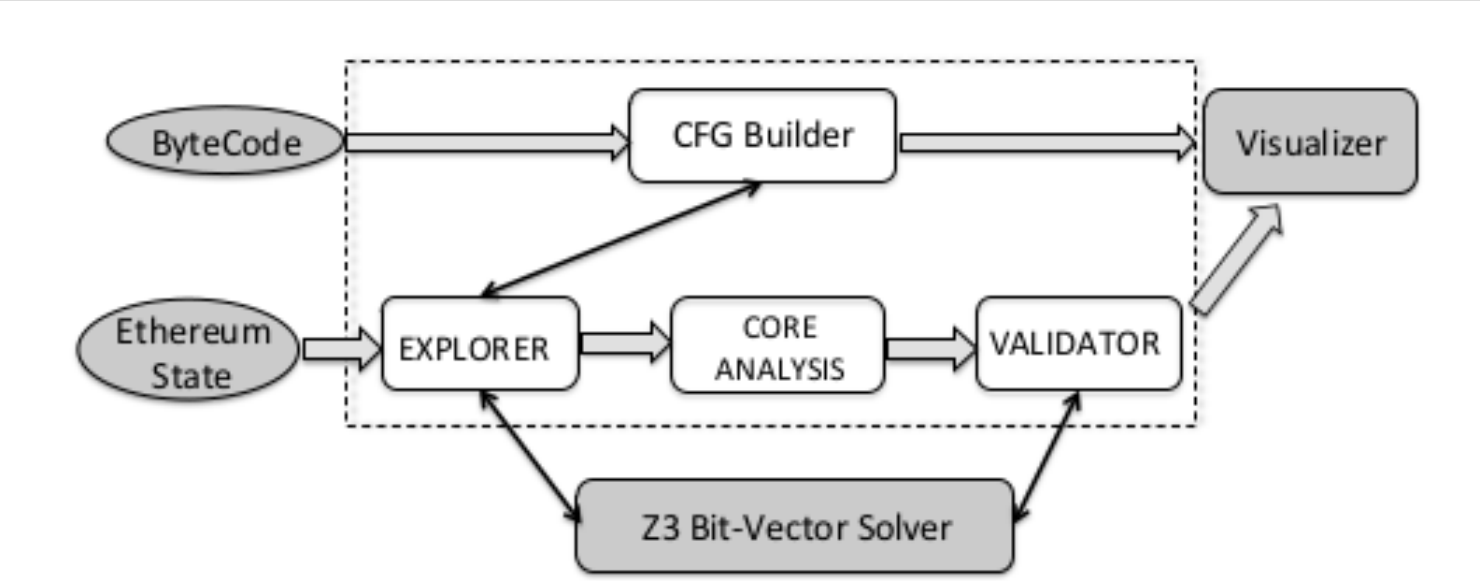


dev tools

- .sol files > bytecode > blockchain
- Atom with plugins:
  - language-ethereum
  - etheratom
- Remix: browser based



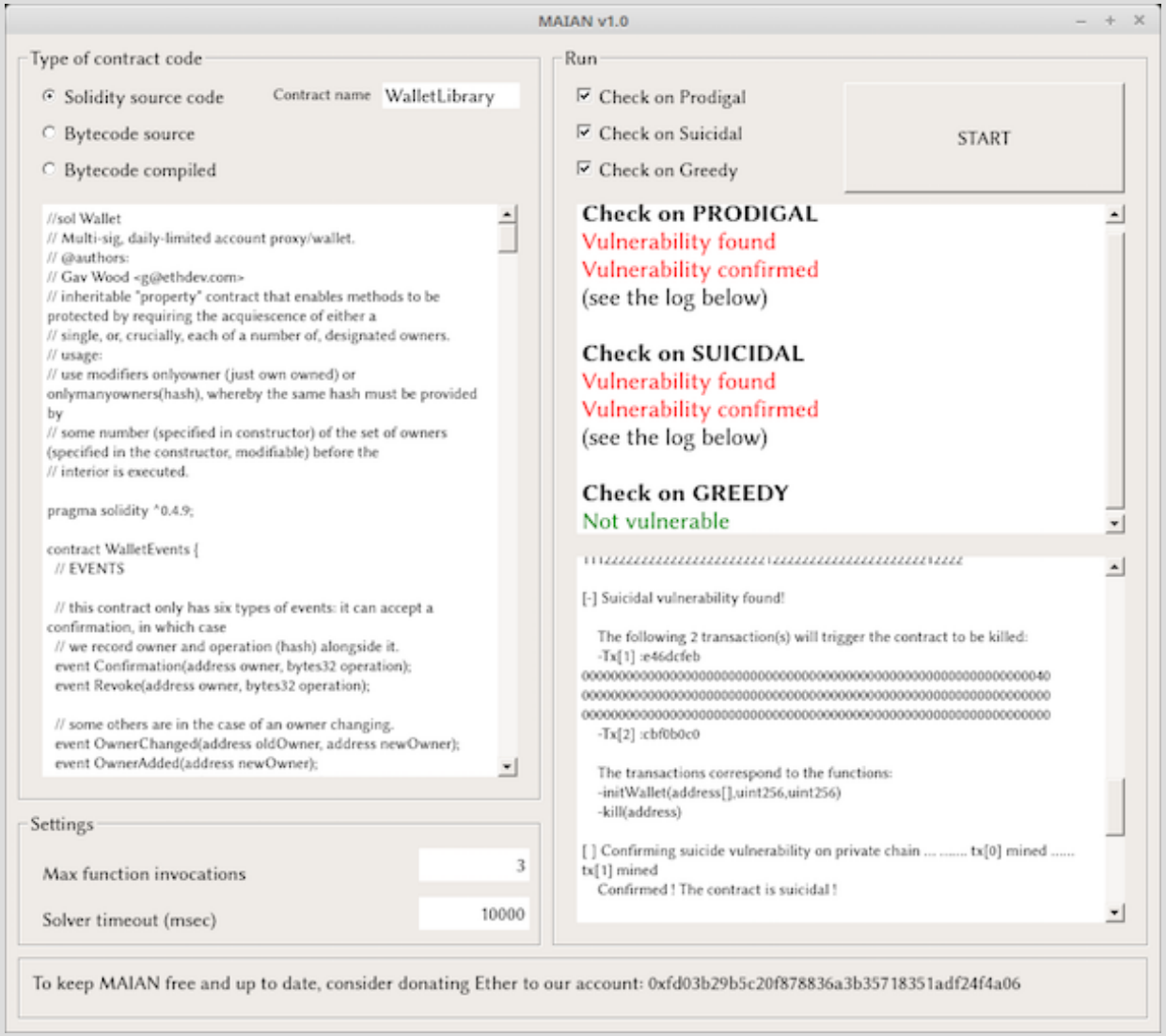
oyente and Manticore




```
root@01562f60ae20:/oyente/oyente# python oyente.py -s hackertest.sol
INFO:root:contract hackertest.sol:greeter:
INFO:symExec:    ============ Results ===========
INFO:symExec:      EVM Code Coverage:                      99.5%
INFO:symExec:      Parity Multisig Bug 2:                  False
INFO:symExec:      Callstack Depth Attack Vulnerability:   False
INFO:symExec:      Transaction-Ordering Dependence (TOD):  False
INFO:symExec:      Timestamp Dependency:                   False
INFO:symExec:      Re-Entrancy Vulnerability:              False
INFO:symExec:    ====== Analysis Completed ======
```



MAIAN




```
[ ] Compiling Solidity contract from the file example_contracts/ParityWalletLibrary.sol ...  Done
[ ] Connecting to PRIVATE blockchain emptychain  ... ESTABLISHED
[ ] Deploying contract .... confirmed at address: 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract code length on the blockchain :    16530  : 0x6060604052600436106101d5760...
[ ] Contract address saved in file: ./out/WalletLibrary.address
[ ] Check if contract is SUICIDAL

[ ] Contract address    : 0x9E536236ABF2288a7864C6A1AfaA4Cb98D464306
[ ] Contract bytecode   : 6060604052600436106101d576000357c0100000000000000...
[ ] Bytecode length     : 16528
[ ] Blockchain contract: True
[ ] Debug               : False

[ ] Search with call depth: 1   : 11111111111111111111
[ ] Search with call depth: 2   : 11122222222222222222222212222222222222222222212222

[-] Suicidal vulnerability found!

    The following 2 transaction(s) will trigger the contract to be killed:
    -Tx[1] :e46dcfeb 0000000000000000000000000000000000000000000000000000000000000040 0000000000000000
0000000000000000000000000000000000000000 0000000000000000000000000000000000000000000000000000000000000000
0000000000000
    -Tx[2] :cbf0b0c0

    The transactions correspond to the functions:
    -initWallet(address[],uint256,uint256)
    -kill(address)

[ ] Confirming suicide vulnerability on private chain ... ..... tx[0] mined ........ tx[1] mined
    Confirmed ! The contract is suicidal !
```

basic methodology

- Interview devs
- Review .sol file
- Try compiling
- Dissect code flow
- Run oyente (cross fingers)
- Run Manticore
- Run MAIAN
- Manually check for following vulns...



reentrancy



```
contract ReEntrancy {

  mapping (address => uint) private expendableTokens;

  function stealTokens() public {
      uint amountToLose = expendableTokens[msg.sender];
      if (!(msg.sender.call.value(amountToLose)())) { throw; }
      expendableTokens[msg.sender] = 0;
  }
}
```

leave off the first “re-” for savings



```
contract Entrancy {

  mapping (address => uint) private expendableTokens;

  function stealTokens() public {
      uint amountToLose = expendableTokens[msg.sender];
      expendableTokens[msg.sender] = 0;
      if (!(msg.sender.call.value(amountToLose)())) { throw; }
  }
}
```

reentrancy (and irony) in the dao code

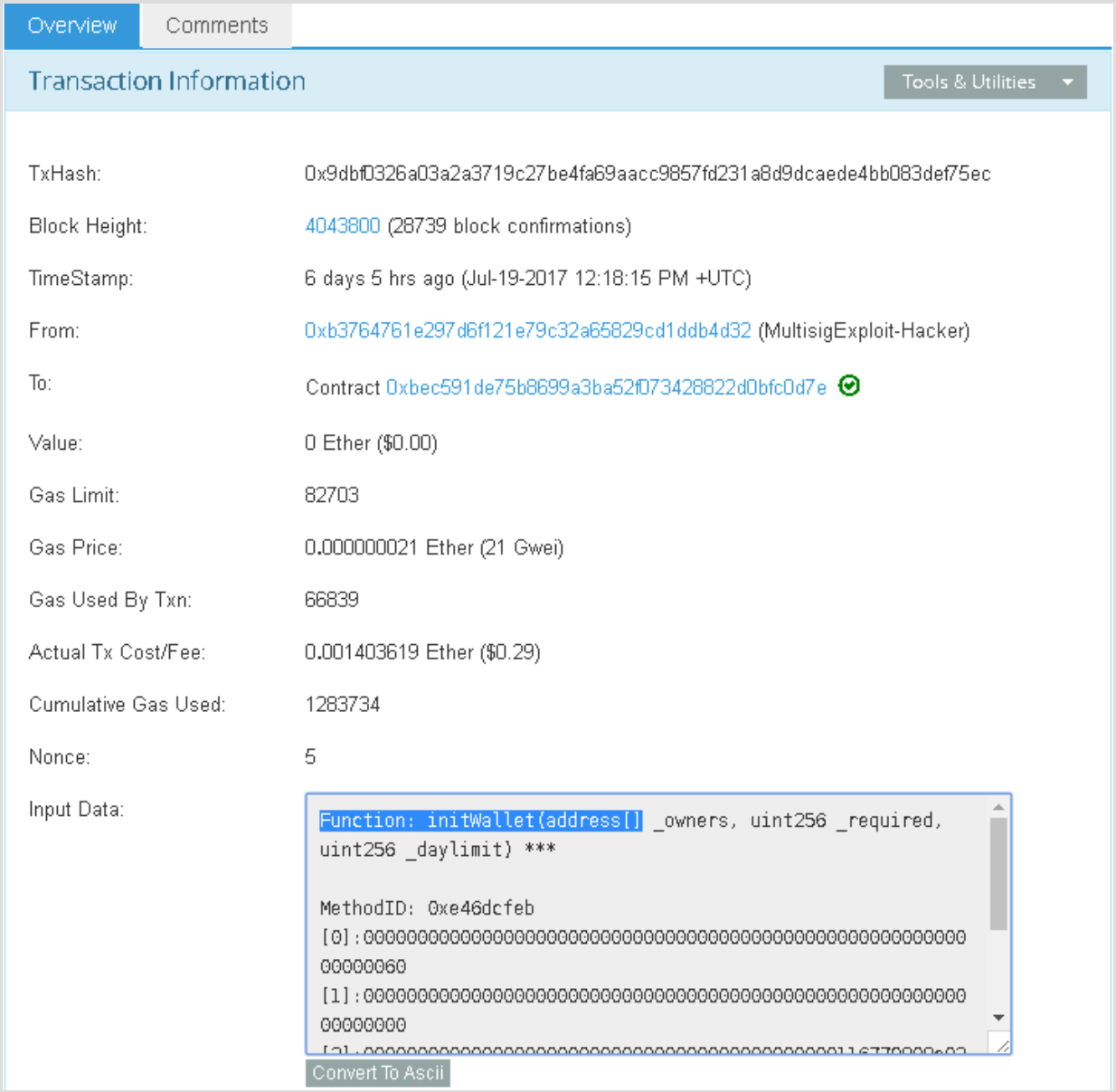
```
// Burn DAO Tokens
        Transfer(msg.sender, 0, balances[msg.sender]);
        withdrawRewardFor(msg.sender); // be nice, and get his rewards
        totalSupply -= balances[msg.sender];
        balances[msg.sender] = 0;
        paidOut[msg.sender] = 0;
        return true;
    }
```

default public – Parity wallet hack

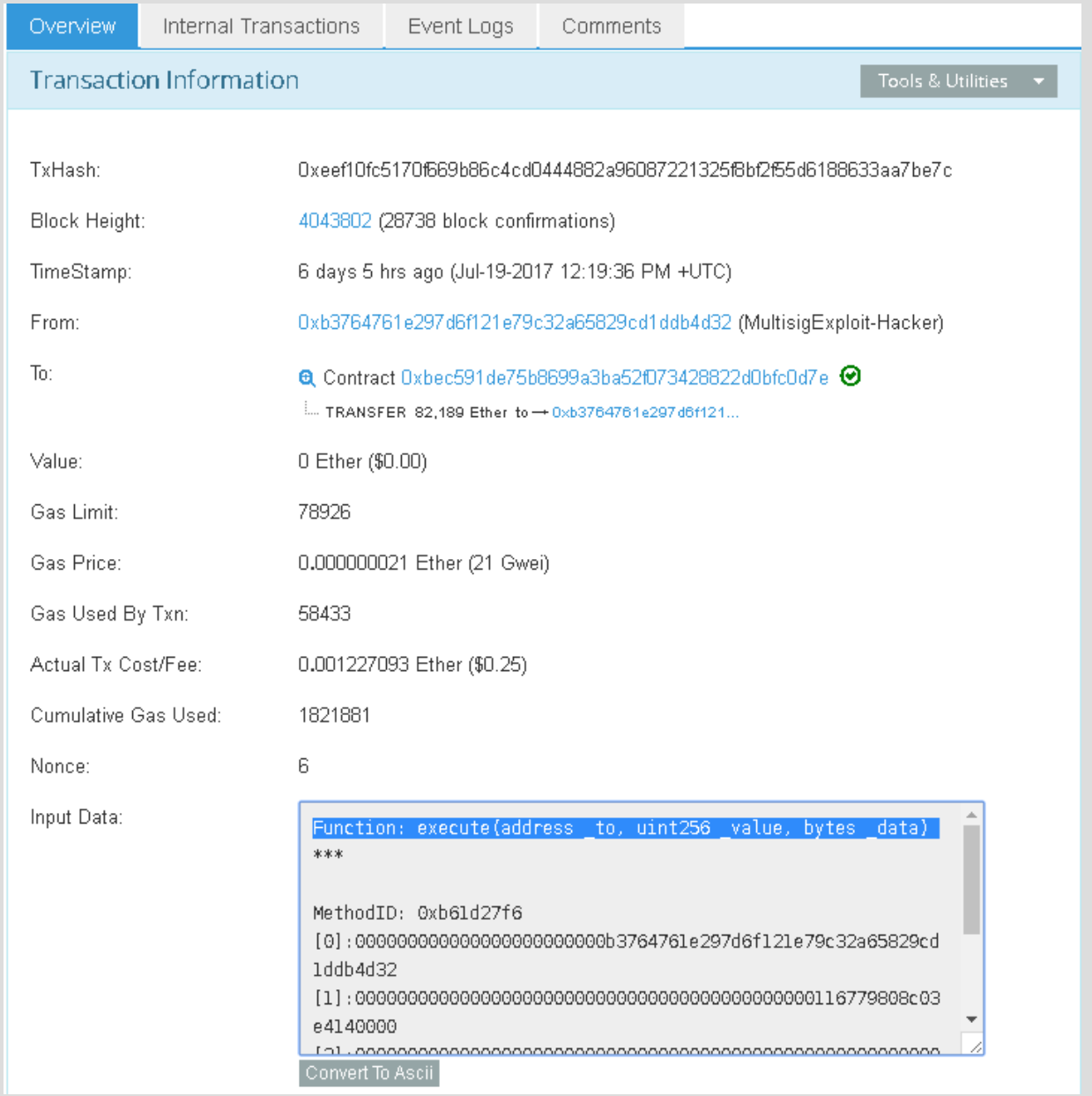
```
@@ -104,7 +104,7 @@ contract WalletLibrary is WalletEvents {
104 104
105 105     // constructor is given number of sigs required to do protected "onlymanyowners" transactions
106 106     // as well as the selection of addresses capable of confirming them.
107     -   function initMultiowned(address[] _owners, uint _required) {
    107 +   function initMultiowned(address[] _owners, uint _required) internal {
108 108       m_numOwners = _owners.length + 1;
109 109       m_owners[1] = uint(msg.sender);
110 110       m_ownerIndex[uint(msg.sender)] = 1;
@@ -198,7 +198,7 @@ contract WalletLibrary is WalletEvents {
198 198     }
199 199
200 200     // constructor - stores initial daily limit and records the present day's index.
201     -   function initDaylimit(uint _limit) {
    201 +   function initDaylimit(uint _limit) internal {
```

```
    216 +
214 217     // constructor - just pass on the owner array to the multiowned and
215 218     // the limit to daylimit
216     -   function initWallet(address[] _owners, uint _required, uint _daylimit) {
    219 +   function initWallet(address[] _owners, uint _required, uint _daylimit) only_uninitialized {
```

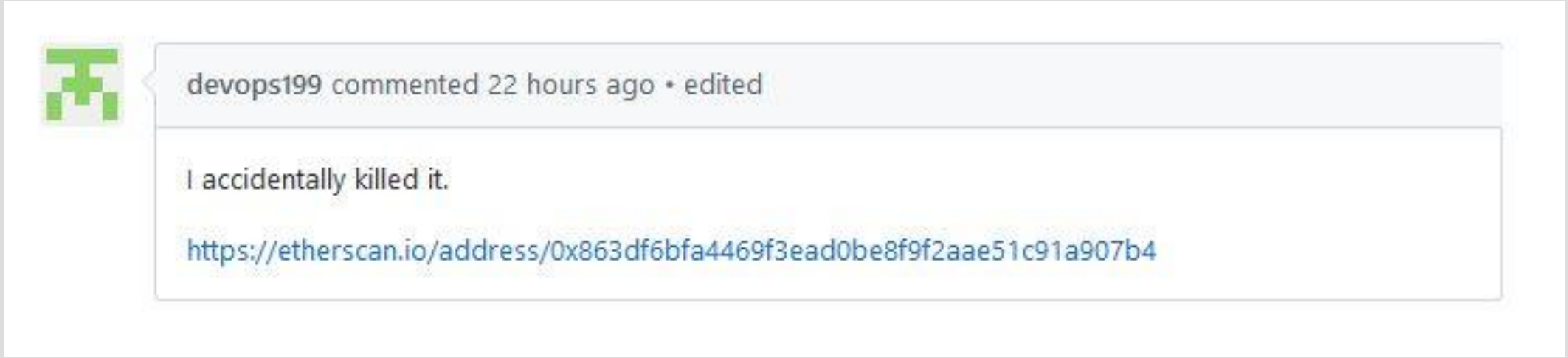
initWallet


Overview
Comments
Transaction Information
Tools & Utilities
TxHash:
0x9dbf0326a03a2a3719c27be4fa69aacc9857fd231a8d9dcaede4bb083def75ec
Block Height:
4043800 (28739 block confirmations)
TimeStamp:
6 days 5 hrs ago (Jul-19-2017 12:18:15 PM +UTC)
From:
0xb3764761e297d6f121e79c32a65829cd1ddb4d32 (MultisigExploit-Hacker)
To:
Contract 0xbec591de75b8699a3ba52f073428822d0bfc0d7e
Value:
0 Ether ($0.00)
Gas Limit:
82703
Gas Price:
0.000000021 Ether (21 Gwei)
Gas Used By Txn:
66839
Actual Tx Cost/Fee:
0.001403619 Ether ($0.29)
Cumulative Gas Used:
1283734
Nonce:
5
Input Data:
Function: initWallet(address[] _owners, uint256 _required, uint256 _daylimit) ***
MethodID: 0xe46dcfeb
[0]: 0000000000000000000000000000000000000000000000000000000000000060
[1]: 0000000000000000000000000000000000000000000000000000000000000000
Convert To Ascii

execute


Overview
Internal Transactions
Event Logs
Comments
Transaction Information
Tools & Utilities
TxHash: 0xeef10fc5170f669b86c4cd0444882a96087221325f8bf2f55d6188633aa7be7c
Block Height: 4043802 (28738 block confirmations)
TimeStamp: 6 days 5 hrs ago (Jul-19-2017 12:19:36 PM +UTC)
From: 0xb3764761e297d6f121e79c32a65829cd1ddb4d32 (MultisigExploit-Hacker)
To: Contract 0xbec591de75b8699a3ba52f073428822d0bfc0d7e
TRANSFER 82,189 Ether to 0xb3764761e297d6f121...
Value: 0 Ether ($0.00)
Gas Limit: 78926
Gas Price: 0.000000021 Ether (21 Gwei)
Gas Used By Txn: 58433
Actual Tx Cost/Fee: 0.001227093 Ether ($0.25)
Cumulative Gas Used: 1821881
Nonce: 6
Input Data:
Function: execute(address _to, uint256 _value, bytes _data)
***
MethodID: 0xb61d27f6
[0]:000000000000000000000000b3764761e297d6f121e79c32a65829cd1ddb4d32
[1]:0000000000000000000000000000000000000000000000116779808c03e4140000
Convert To Ascii

Parity multisig wallet hack 2



Parity 2 transactions

```
Function: initWallet(address[] _owners, uint256 _required, uint256
_daylimit)
MethodID: 0xe46dcfeb
[0]:0000000000000000000000000000000000000000000000000000000000000060
[1]:0000000000000000000000000000000000000000000000000000000000000000
[2]:0000000000000000000000000000000000000000000000000000000000000000
[3]:0000000000000000000000000000000000000000000000000000000000000001
[4]:000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952
```

```
Function: kill(address _to)
MethodID: 0xcbf0b0c0
[0]:000000000000000000000000ae7168deb525862f4fee37d987a971b385b96952
```

not going with the (over)flow

$2^{256} - 1$



unchecked send in king of the ether



GitHub, Inc. [US] | https://github.com/kieranelby/KingOfTheEtherThrone/blob/v0.4.0/contr

```
117         uint compensation = valuePaid - wizardCommission;
118
119         if (currentMonarch.etherAddress != wizardAddress) {
120             currentMonarch.etherAddress.send(compensation);
121         } else {
122             // When the throne is vacant, the fee accumulates for the wizard.
123         }
124
```

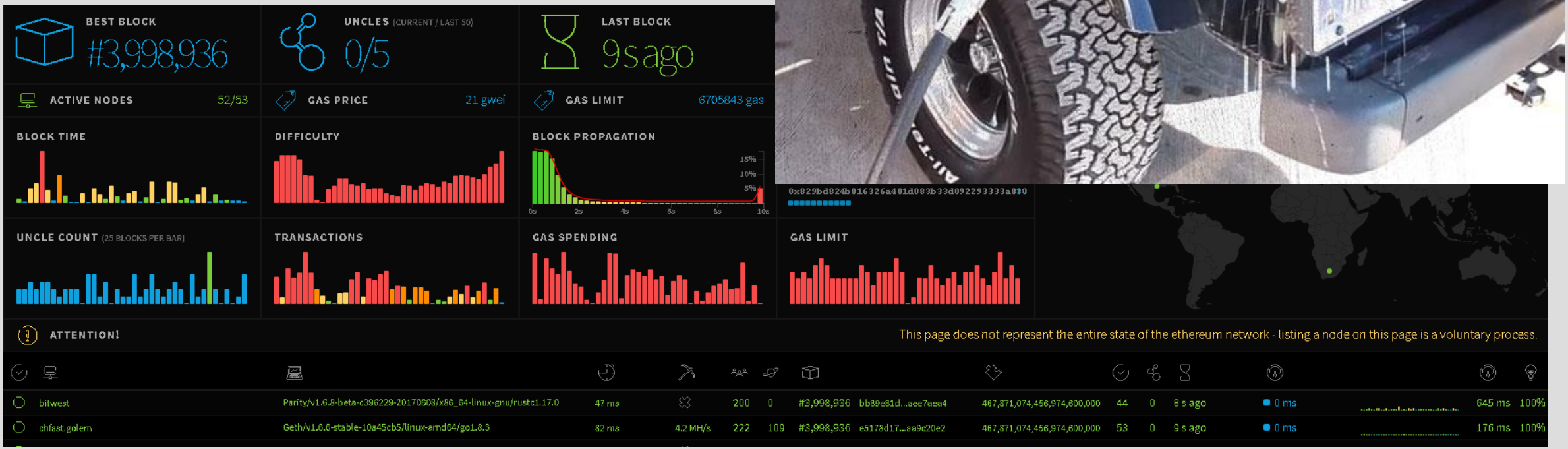
unchecked send

```
1  if (kingOfLosingDone && !( compensationSent ) ) {
2    monarch.send(500);
3    compensationSent = True;
4  }
```

```
1  if (kingOfLosingDone && !( compensationSent ) ) {
2    if (monarch.send(500))
3      compensationSent = True;
4    else throw;
5  }
```

gas limits




BEST BLOCK
#3,998,936
UNCLES (CURRENT / LAST 50)
0/5
LAST BLOCK
9s ago
ACTIVE NODES 52/53
GAS PRICE 21 gwei
GAS LIMIT 6705843 gas
BLOCK TIME
DIFFICULTY
BLOCK PROPAGATION
15%
10%
5%
0s
2s
4s
6s
8s
10s
UNCLE COUNT (25 BLOCKS PER BAR)
TRANSACTIONS
GAS SPENDING
GAS LIMIT
ATTENTION!
This page does not represent the entire state of the ethereum network - listing a node on this page is a voluntary process.
bitwest Parity/v1.6.8-beta-c396229-20170608/x86_64-linux-gnu/rustc1.17.0 47 ms 200 0 #3,998,936 bb89e81d...aee7aea4 467,871,074,456,974,600,000 44 0 8 s ago 0 ms 645 ms 100%
chfast.golem Geth/v1.6.6-stable-10a45cb5/linux-amd64/go1.8.3 82 ms 4.2 MH/s 222 109 #3,998,936 e5178d17...aa9c20e2 467,871,074,456,974,600,000 53 0 9 s ago 0 ms 176 ms 100%

withdraw don't send

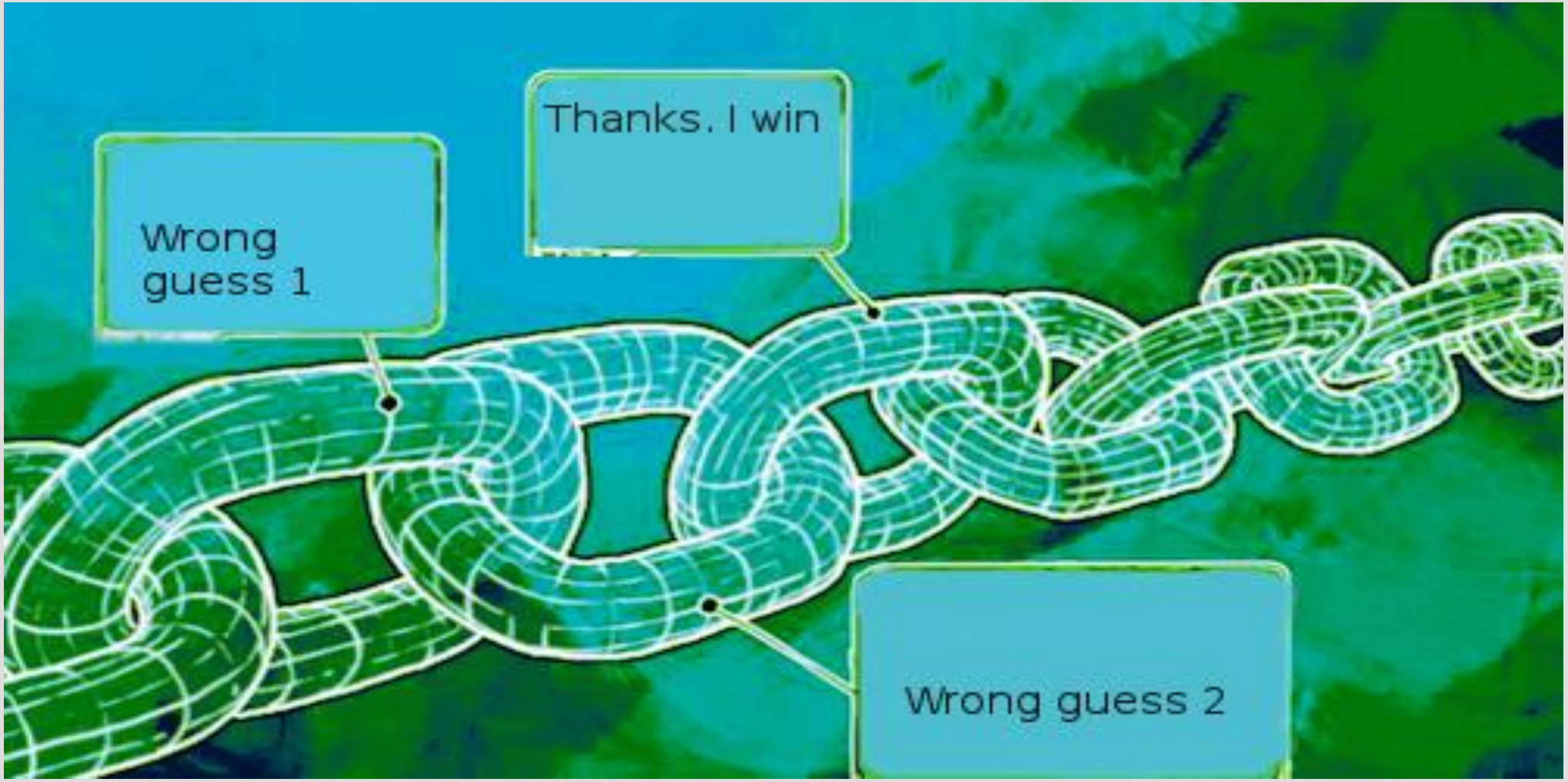


```
contract SendContract {
    address public richest;
    uint public mostSent;

    function SendContract() payable {
        richest = msg.sender;
        mostSent = msg.value;
    }

    function becomeRichest() payable returns (bool) {
        if (msg.value > mostSent) {
            richest.transfer(msg.value);
            richest = msg.sender;
            mostSent = msg.value;
            return true;
        } else {
            return false;
```

withdrawn not sent

```
contract WithdrawalContract {
    address public richest;
    uint public mostSent;

    mapping (address => uint) pendingWithdrawals;

    function WithdrawalContract() payable {
        richest = msg.sender;
        mostSent = msg.value;
    }

    function becomeRichest() payable returns (bool) {
        if (msg.value > mostSent) {
            pendingWithdrawals[richest] += msg.value;
            richest = msg.sender;
            mostSent = msg.value;
            return true;
        } else {
            return false;
        }
    }

    function withdraw() {
        uint amount = pendingWithdrawals[msg.sender];
        pendingWithdrawals[msg.sender] = 0;
        msg.sender.transfer(amount);
```

encryption



transaction-ordering dependence

```
1   contract Puzzle{
2     address public owner;
3     bool public locked;
4     uint public reward;
5     bytes32 public diff;
6     bytes public solution;
7
8     function Puzzle(){
9       owner = msg.sender;
10      reward = msg.value;
11      locked = false;
12      diff = bytes32(11111);
13    }
14
```

```
15    function(){
16      if (msg.sender == owner){
17        if (locked) throw;
18        owner.send(reward);
19        reward = msg.value;
20      }
21      else
22        if (msg.data.length > 0){
23          if (locked) throw;
24          if (sha256(msg.data) < diff){
25            msg.sender.send(reward);
26            solution = msg.data;
27            locked = true;
28          }
```

transaction-ordering dependence

```
1   contract Puzzle{
2     address public owner;
3     bool public locked;
4     uint public reward;
5     bytes32 public diff;
6     bytes public solution;
7
8     function Puzzle(){
9       owner = msg.sender;
10      reward = msg.value;
11      locked = false;
12      diff = bytes32(11111);
13    }
14
```

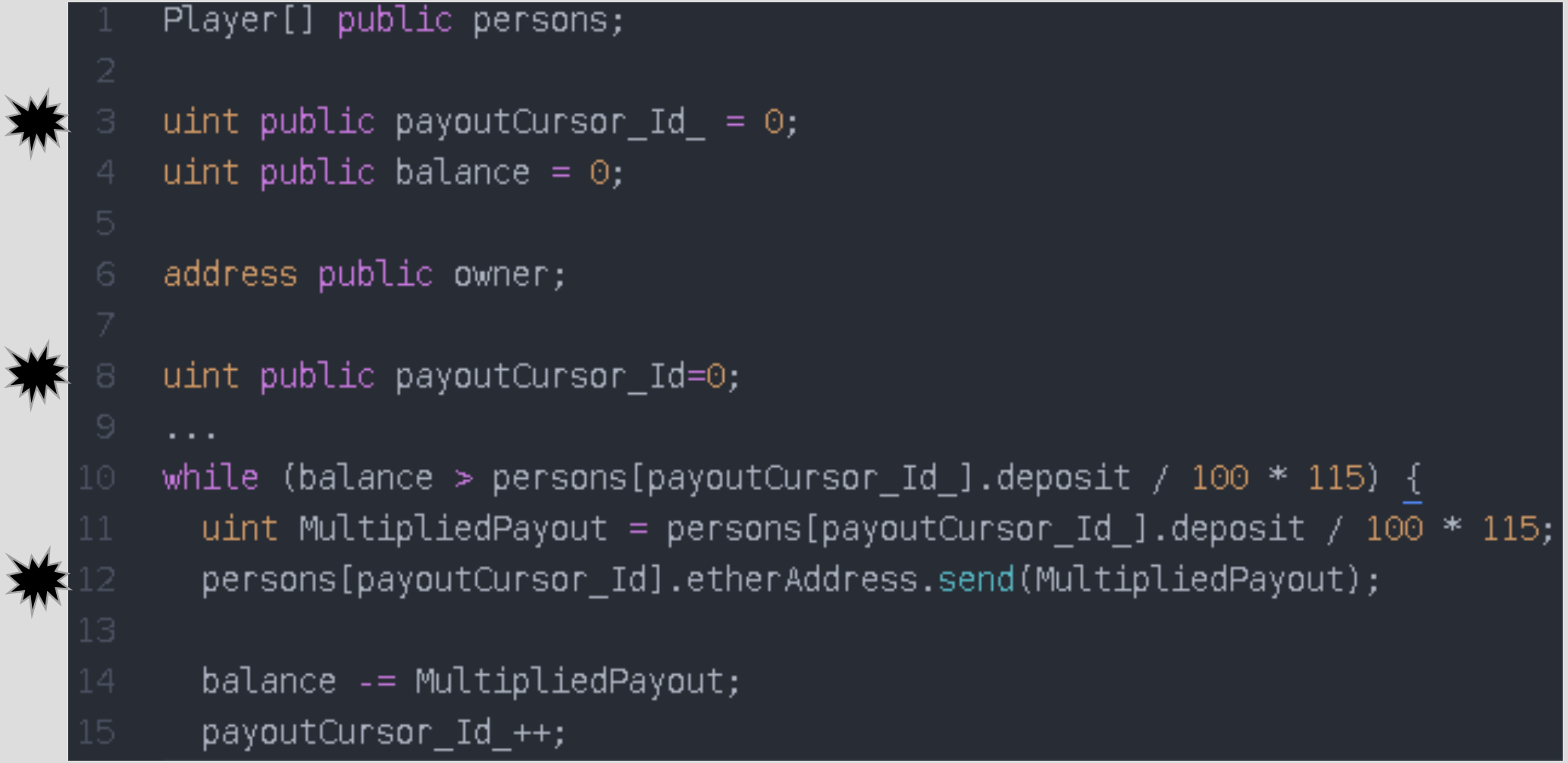
```
15    function(){
16      if (msg.sender == owner){
17        if (locked) throw;
18        owner.send(reward);
19        reward = msg.value;
20      }
21      else
22        if (msg.data.length > 0){
23          if (locked) throw;
24          if (sha256(msg.data) < diff){
25            msg.sender.send(reward);
26            solution = msg.data;
27            locked = true;
28          }
```

call-stack depth limit

```
INFO:symExec:    ============ Results ===========
INFO:symExec:      EVM Code Coverage:                    99.5%
INFO:symExec:      Parity Multisig Bug 2:                False
INFO:symExec:      Callstack Depth Attack Vulnerability: False
INFO:symExec:      Transaction-Ordering Dependence (TOD): False
INFO:symExec:      Timestamp Dependency:                 False
INFO:symExec:      Re-Entrancy Vulnerability
INFO:symExec:    ====== Analysis Completed ==
```

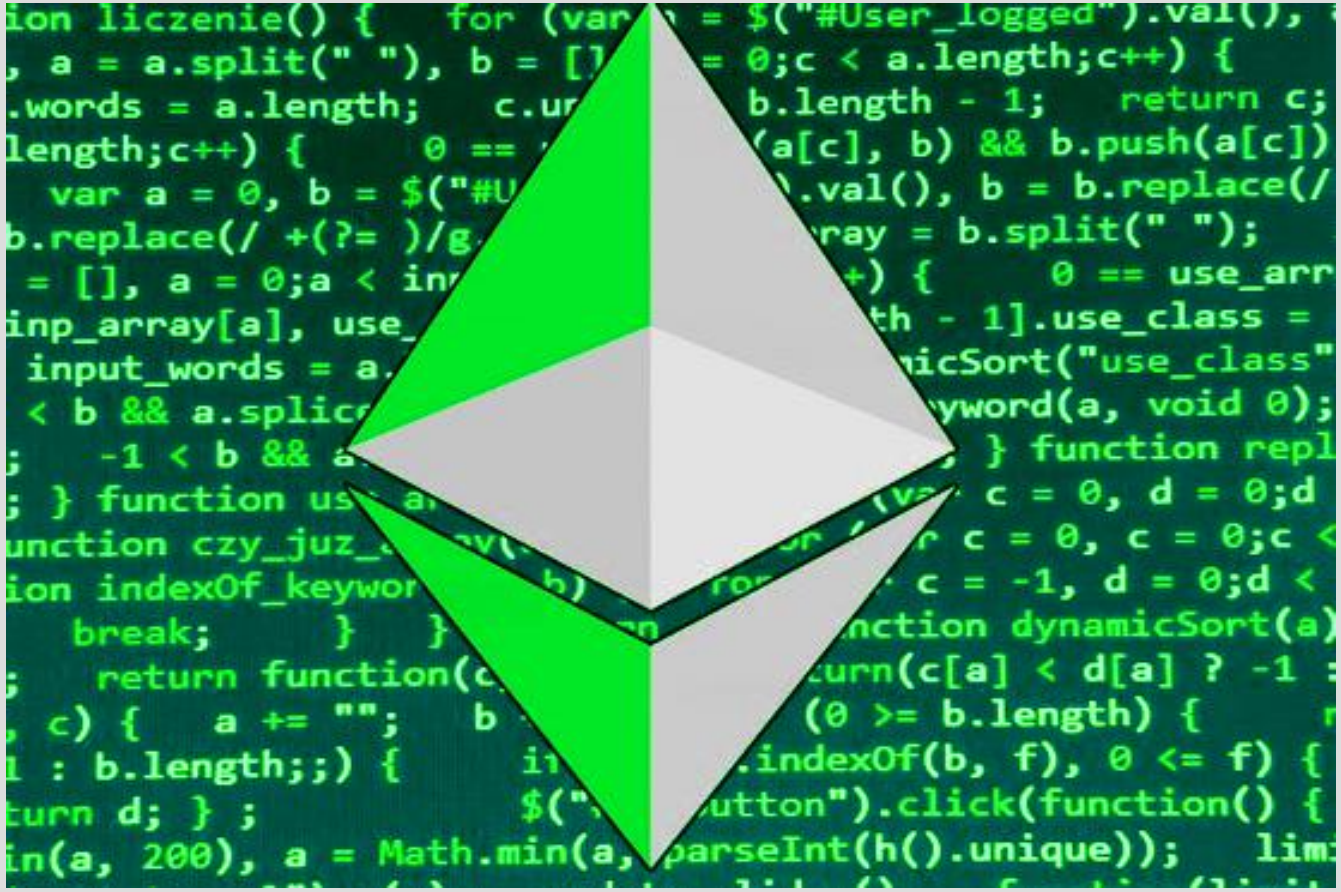


variable or function ambiguity



odds and ends

- Timestamp dependence
- Business logic flaws
- Separating public/private data



things might be getting better?



keep in touch

@KonstantHacker

