Making Smart Contracts Smarter

Loi Luu
National University of Singapore
loiluu@comp.nus.edu.sg

Duc-Hiep Chu
National University of Singapore
hiepcd@comp.nus.edu.sg

Hrishi Olickel
Yale-NUS College
hrishi.olickel@yale-nus.edu.sg

Prateek Saxena
National University of Singapore
prateeks@comp.nus.edu.sg

Aquinas Hobor
Yale-NUS College&
National University of Singapore
hobor@comp.nus.edu.sg

Vulnerabilities in Smart Contracts ,

suggestions to prevent/overcome them,

and detection using Oyente.

https://eprint.iacr.org/2016/633.pdf
See also https://eprint.iacr.org/2016/1007.pdf

MOTIVATION

19,366/1,459,999
contract within first blocks
8,833/19,366
was flagged by oyente
1,682/8,833
was “specially” flagged by oyente
175/1,682
was checked via their source code
10/(175-10)
false positive =6.4%

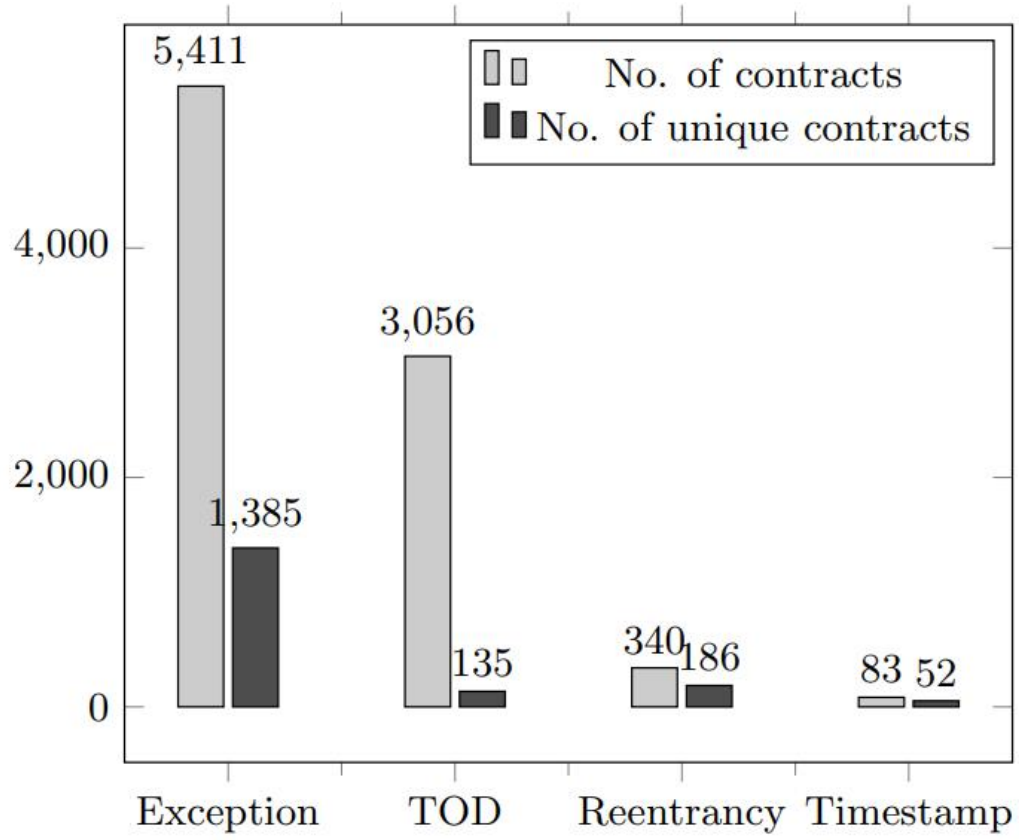


Figure 12: Number of buggy contracts per each security problem reported by OYENTE.

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() //constructor{
9     owner = msg.sender;
10    reward = msg.value;
11    locked = false;
12    diff = bytes32(11111); //pre-defined difficulty
13  }
14
15  function(){ //main code, runs at every invocation
16    if (msg.sender == owner){ //update reward
17      if (locked)
18        throw;
19      owner.send(reward);
20      reward = msg.value;
21    }
22    else
23      if (msg.data.length > 0){ //submit a solution
24        if (locked) throw;
25        if (sha256(msg.data) < diff){
26          msg.sender.send(reward); //send reward
27          solution = msg.data;
28          locked = true;
29        }}}}
```

Figure 3: A contract that rewards users who solve a computational puzzle.

TRANSACTION ORDERING DEPENDENCE

- Programmers / users think transactions are immediately, by the order they sent to the Ethereum net
- Even order within block isn't known and cannot be ensured
- Blockchain is distributed system, based on the consensus principal, you can't enforce order(/synchronize) as you wish. (no 1 authority)
- "Protocol" Solution: "Guard condition"

$$g \equiv (reward == R)$$

TIMESTEP DEPENDENCE

```
1 contract theRun {
2   uint private Last_Payout = 0;
3   uint256 salt =  block.timestamp;
4   function random returns (uint256 result){
5     uint256 y = salt * block.number/(salt%5);
6     uint256 seed = block.number/3 + (salt%300)
7                    + Last_Payout +y;
8     //h = the blockhash of the seed-th last bloc
9     uint256 h = uint256(block.blockhash(seed));
10    //random number between 1 and 100
11    return uint256(h % 100) + 1;
12  }}
```

TIMESTEP DEPENDENCE

- Programmers / users don’t take in account the influence of the miners on the created block (timestep can be manipulate, not so “random”)
- Miners can choose timestep as they wish,

  + - 900 sec , and it still be “fine” to the consensus
- Any miner has its own Discretion to the timestep in block it creates
- The seed for random() must be inferred from the blockchain, but also be “secret” as much as could
- time() must be inferred from the blockchain
- “Educational” Solution:

  $use\ block's\ index\ instead$

MISHANDLED EXCEPTIONS

```
1 contract KingOfTheEtherThrone {
2   struct Monarch {
3     // address of the king.
4     address ethAddr;
5     string name;
6     // how much he pays to previous king
7     uint claimPrice;
8     uint coronationTimestamp;
9     }
10  Monarch public currentMonarch;
11  // claim the throne
12  function claimThrone(string name) {
13    /.../
14    if (currentMonarch.ethAddr != wizardAddress)
15      currentMonarch.ethAddr.send(compensation);
16    /.../
17    // assign the new king
18    currentMonarch = Monarch(
19        msg.sender, name,
20        valuePaid, block.timestamp);
21  }}
```

MISHANDLED EXCEPTIONS

```
1// ID on sale, and enough money
2if(d.price > 0 &&  msg.value >= d.price){
3     if(d.price > 0)
4          address(d.owner).send(d.price);
5     d.owner = msg.sender;// Change the ownership
6     d.price = price;      // New price
7     d.transfer = transfer;  // New transfer
8     d.expires = block.number + expires;
9     DomainChanged( msg.sender, domain, 0 );
10}
```

Figure 18: EtherID contract, which allows users to register, buy and sell any ID. This code snippet handles buy requests from users.

MISHANDLED EXCEPTIONS

- Programmers don’t check return value to see if exception was thrown but suppressed
- “Safe” and “simple” code can make user lose money by mistake, or by intentional hacker attack
- Programmers / users aren’t familiar with Solidity rules, and EVM, which exception handling are complex
- When writing contract programmers think no exception will be thrown inside their contract execution if it written good
- “Partial” Solution: solidity compiler
- “Educational” Solution: check for return value,

  maybe Solidity should introduce try/catch syntax

REENTRANCY VULNERABILITY

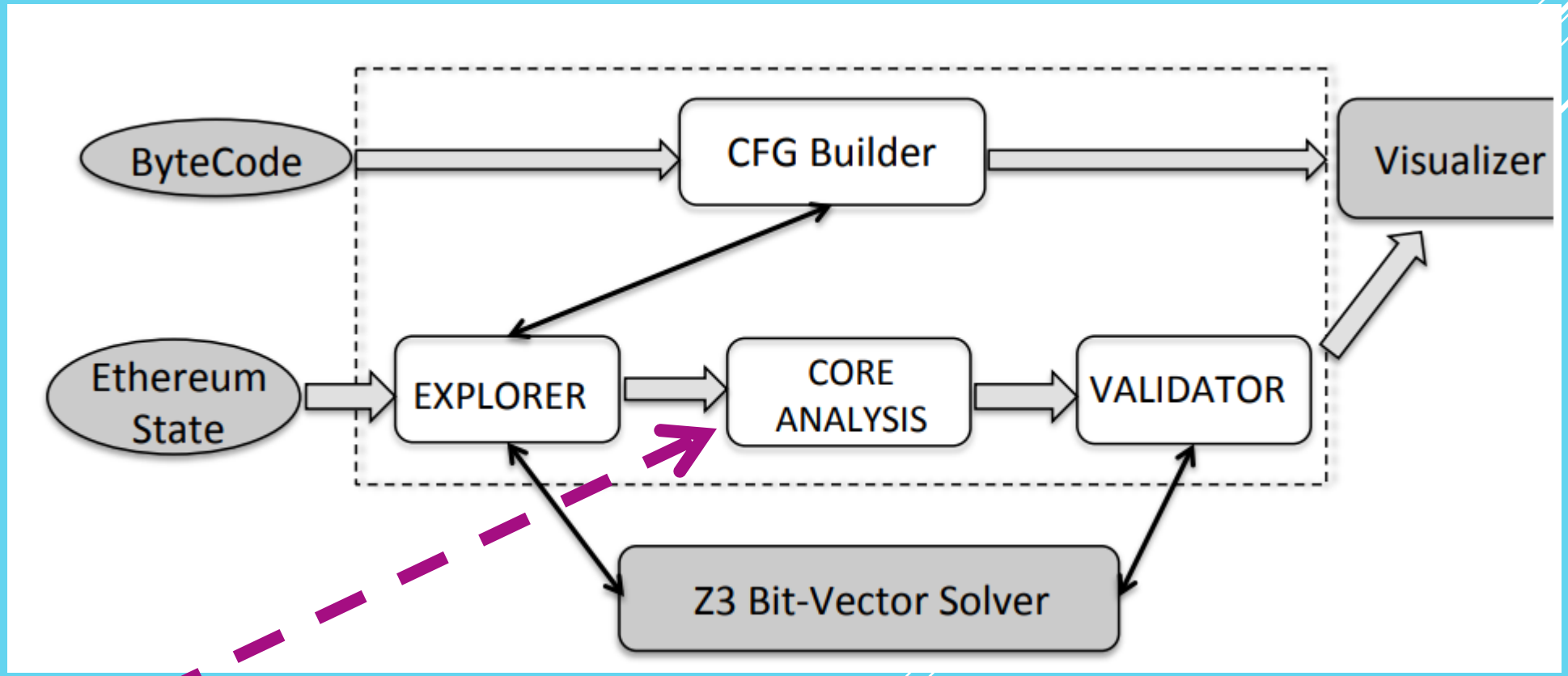
```
1 contract SendBalance {
2  mapping (address => uint) userBalances;
3  bool withdrawn = false;
4  function getBalance(address u) constant returns(uint){
5   return userBalances[u];
6  }
7  function addToBalance() {
8   userBalances[msg.sender] += msg.value;
9  }
10 function withdrawBalance(){
11  if (!(msg.sender.call.value(
12    userBalances[msg.sender])())) { throw; }
13  userBalances[msg.sender] = 0;
14  }}
```

Figure 7: An example of the reentrancy bug. The contract implements a simple bank account.

REENTRANCY VULNERABILITY

- Programmers don’t take in account that any function call outside their contract, can be risk
- Programmers / users don’t think their contract can be invoked because of their own contract in “innocent send() command”
- Contract which didn’t take in account that address can be for another contract
- “Educational” Solution: put all transaction right before the function will return

OYENTE



Based on math representation to evm bytecode's instructions , EtherLite

SUMMARIZE:

- Examples, suggested solution to:
  - Transaction ordering dependence
  - Timestep dependence
  - Mishandled Exceptions
  - Reentrancy vulnerability
- Oyente

QUESTIONS?