Oyente: Making Smart Contracts Smarter

Loi Luu, Duc-Hiep Chu, Hrishi Olickel
Prateek Saxena, Aquinas Hobor
National University of Singapore, Yale-NUS College

Programming securely is hard

"Security can be no stronger than its weakest link"

Programming Secure Smart Contracts is Harder

- Smart contracts !=normal programs
  - Self-executed
  - One-shot programs
    - Cannot patch
- New language
  - Solidity != JavaScript
  - Serpent != Python

382

I think TheDAO is getting drained right now

89d • ledgerwatch • self.ethereum

33

Etherdice is down for maintenance. We are having troubles with our smart contract and will probably need to invoke

King of the Ether Throne

An Ethereum ÐApp (a "contract"), living on the blockchain, that will make you a King or Queen, might grant you riches, and will immortalize your name.

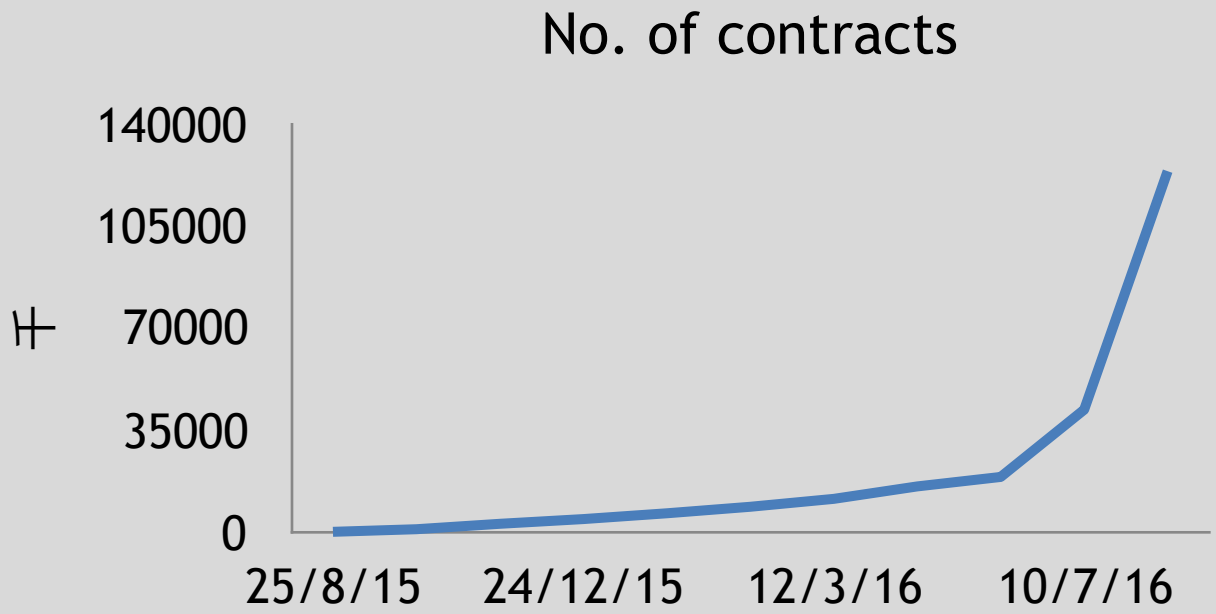
Important Notice

A SERIOUS ISSUE has been identified that can cause monarch compensation payments to not be sent.

DO NOT send payments to the contract previously referenced on this page, or attempt to claim the throne. Refunds will CERTAINLY NOT be made for any payments made after this issue was identified on 2016-02-07.

Questions?

- Are there other bugs?
  - Apart from call-stack and reentrancy?
- How many contracts are vulnerable?

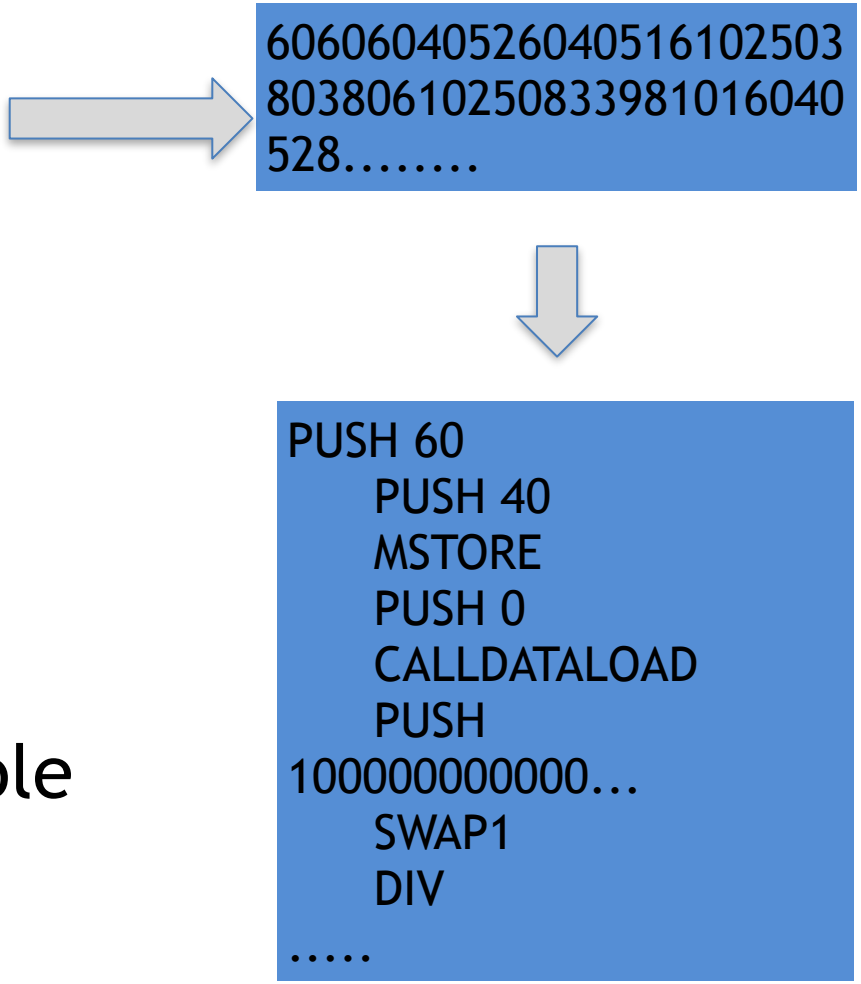


Challenges

- Contracts code are not always available

```
contract Greetings {
    string greeting;
    function Greetings (string _greeting) public {
        greeting = _greeting;
    }

    /* main function */
    function greet() constant returns (string) {
        return greeting;
    }
}
```



- Too many contracts
  - Manual analysis is impossible

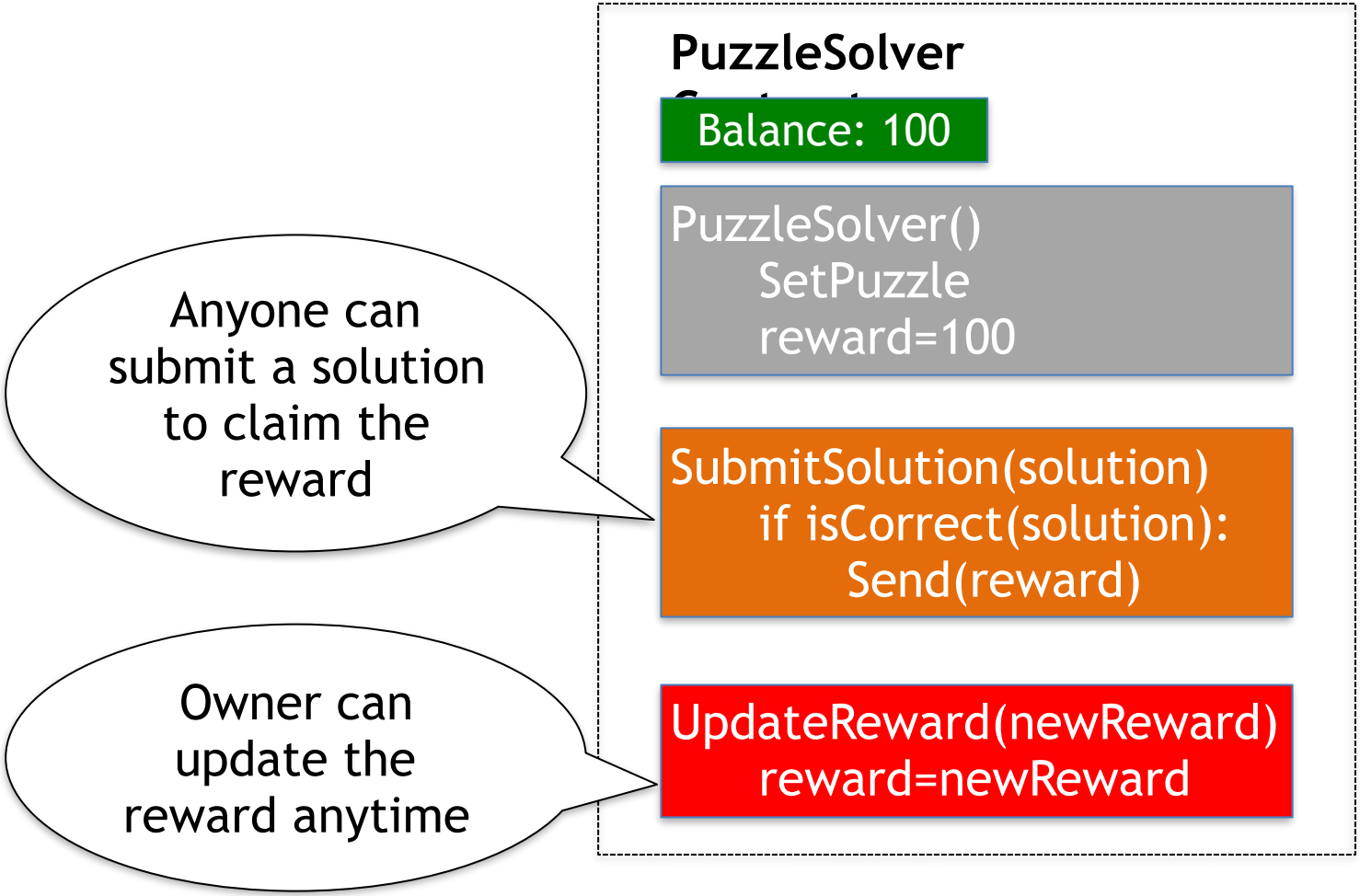
Contribution

- Identify New Smart Contract Bugs
  - Transaction Ordering Dependence (TOD)
  - Timestamp Dependence
- Oyente: An analyzer for smart contracts
  - Use symbolic execution
  - Detect all popular bugs
    - TOD
    - Timestamp dependence
    - Reentrancy
    - Mishandling exceptions (e.g. send)
  - Flags 8836/ 19366 contracts as vulnerable
    - As of May 2016

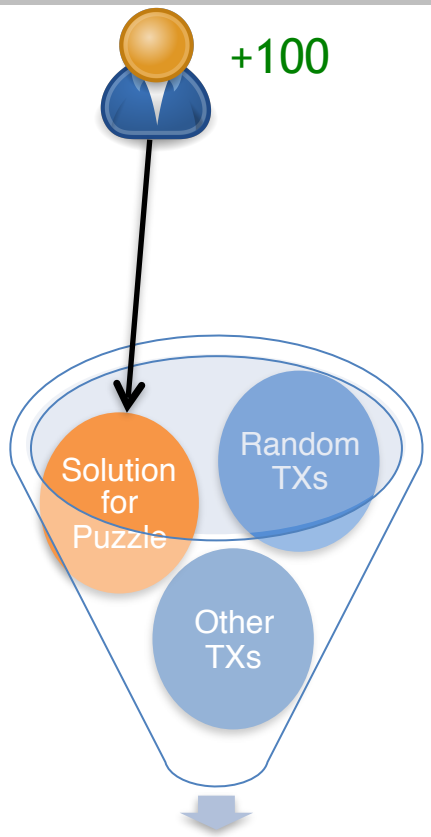
New Smart Contract Bugs

Transaction Ordering Dependence

Example: Puzzle Solver



Scenario 1: SubmitSolution is trigerred



Miners

Block

- Random TXs
- SubmitSolution
- Other TXs

PuzzleSolver Contract

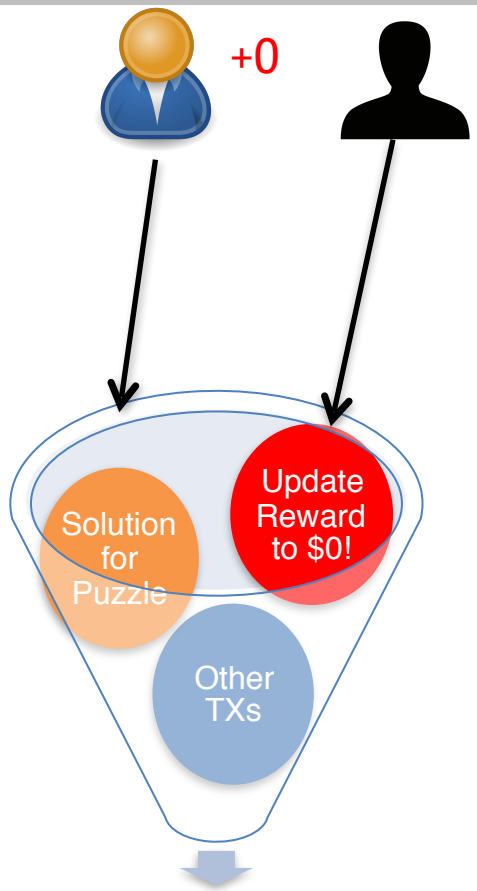
Balance: 0

```
PuzzleSolver()
    SetDifficulty
    reward=100
```

```
SubmitSolution(solution)
    if isCorrect(solution):
        Send(reward)
```

```
UpdateReward(newReward)
    reward=newReward
```

Scenario 2: Both SubmitSolution and UpdateReward are triggered



Miners

Block

- UpdateReward = 0
- SubmitSolution
- Other TXs

PuzzleSolver Contract

Balance: 0

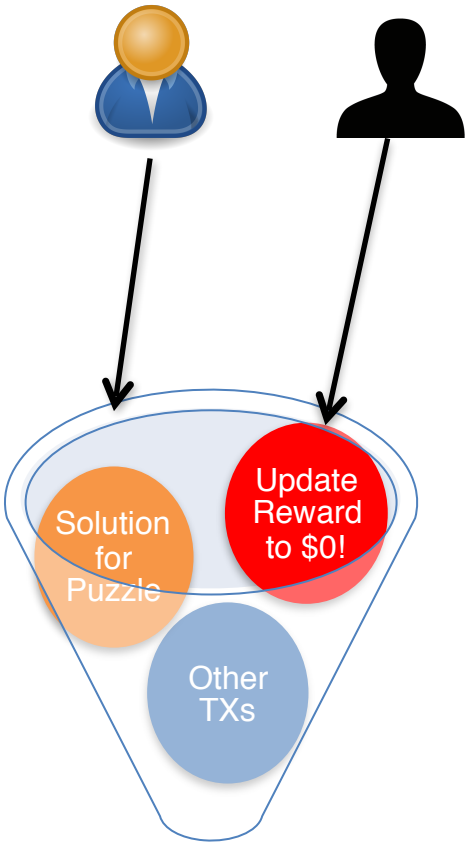
```
PuzzleSolver()
    SetDifficulty
    reward=100
```

```
SubmitSolution(solution)
    if isCorrect(solution):
        Send(reward)
```

```
UpdateReward(newReward)
    reward=newReward
```

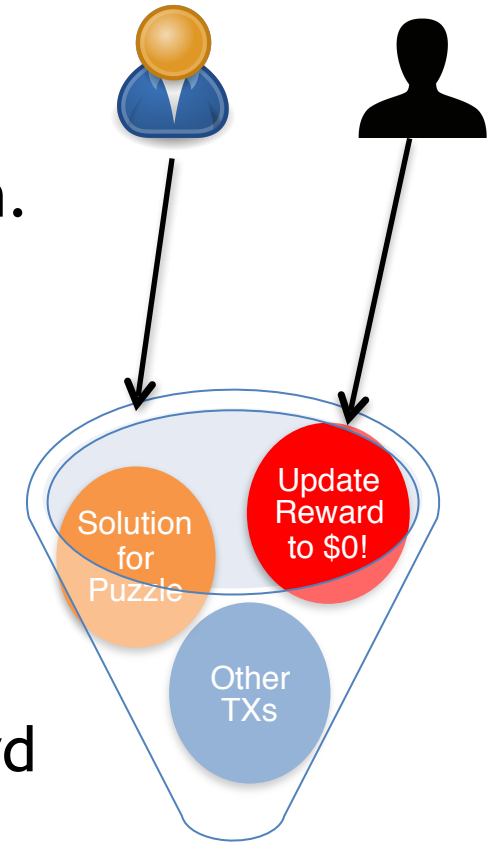
Transaction Ordering Dependence

- Observed state != execution state
  - The expectation of the state of the contract may not be true during execution.
  - Miners decide the order of TXs
- Can be coincidence
  - Two transactions happen at the same time



Transaction Ordering Dependence

- Observed state != execution state
  - The expectation of the state of the contract may not be true during execution.
  - Miners decide the order of TXs
- Can be coincidence
  - Two transactions happen at the same time
- Can be malicious
  - Saw the targeted TX from the victim
  - Submit the second TX to update the reward
  - Both TXs enter the race



New Smart Contract Bugs

Timestamp Dependence

Contract: TheRun

randomness = **F**(timestamp)

```
contract theRun {
  uint private LastPayout = 0;
  uint256 salt =  block.timestamp;
  function random() returns (uint256 result){
    uint256 y = salt * block.number/(salt%5);
    uint256 seed = block.number/3 + (salt%300)
                   + LastPayout +y;

    //h = the blockhash of the seed-th last block
    uint256 h = uint256(block.blockhash(seed));

    //random number between 1 and 100
    return uint256(h % 100) + 1;
  }
  ...
}
```

Contract: PonziGovernmentMental

```
function lendGovernmentMoney(address buddy)
    returns (bool) {
    ...
    if (lastTimeOfNewCredit + TWELVE_HOURS >
        block.timestamp) {
        msg.sender.send(amount);
        // Sends jacpot to the last creditor
        creditorAddresses[nCreditors - 1]
            .send(profitFromCrash);
        owner.send(this.balance);
    ...
    }
}
```

Timestamp can be manipulated

- Miners can vary the block timestamp

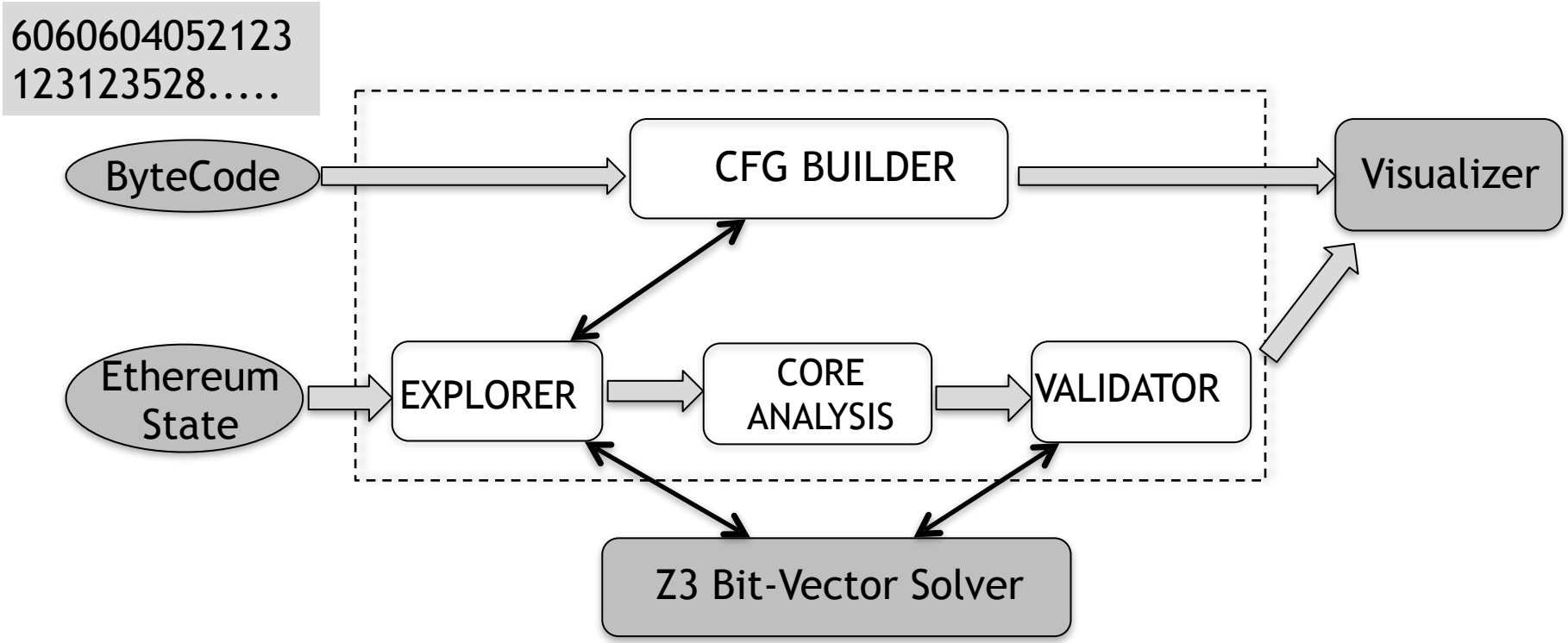
```
block.timestamp <= now + 900 &&
block.timestamp >= parent.timestamp
```

- Bias the output of contract execution to their benefit
  - Timed puzzles, time-based RNGs

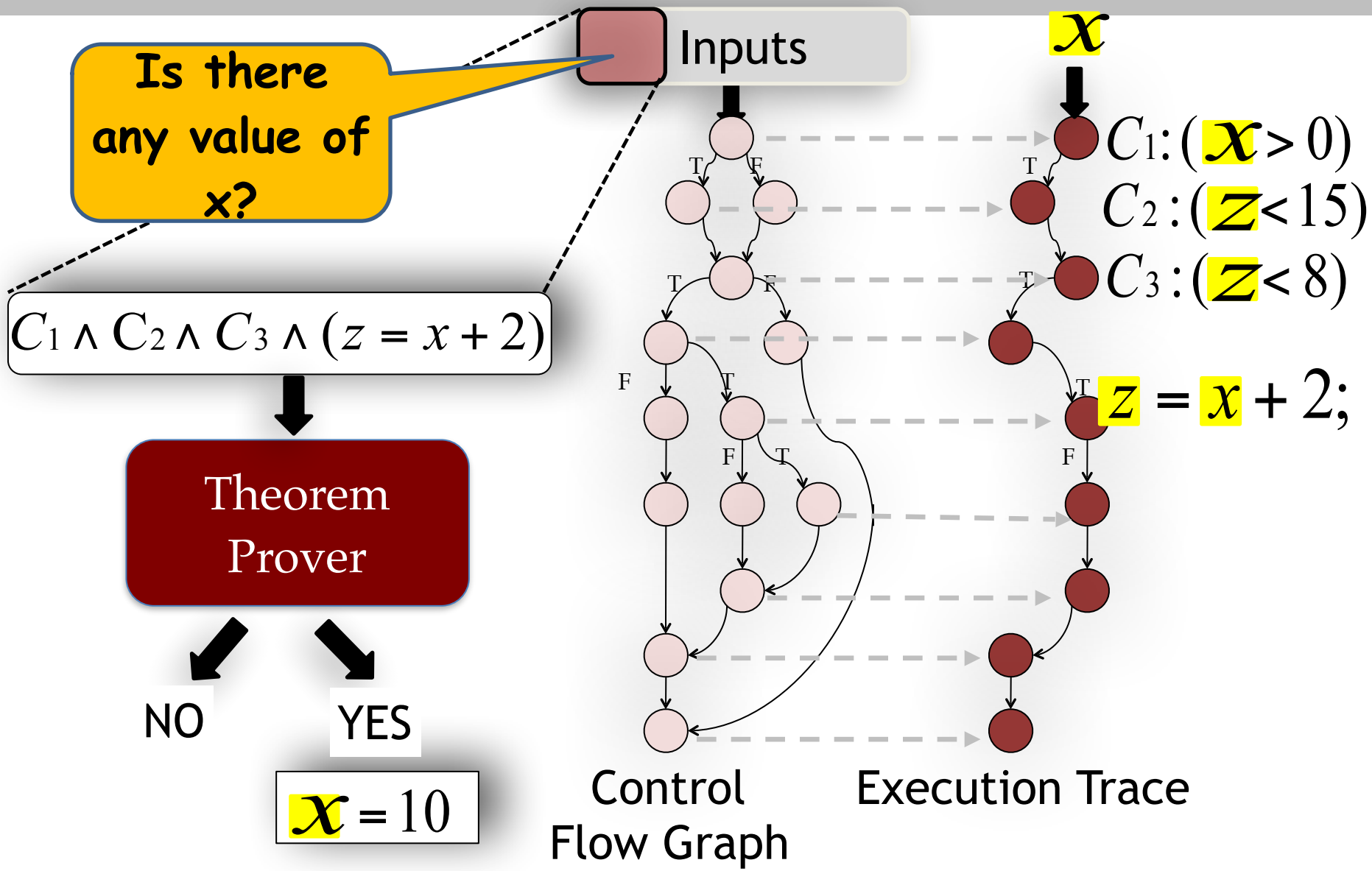
Oyente: An Analyzer for Smart Contracts

Architecture

- Based on symbolic execution
- Have separate modules
  - Can add more analysis separately

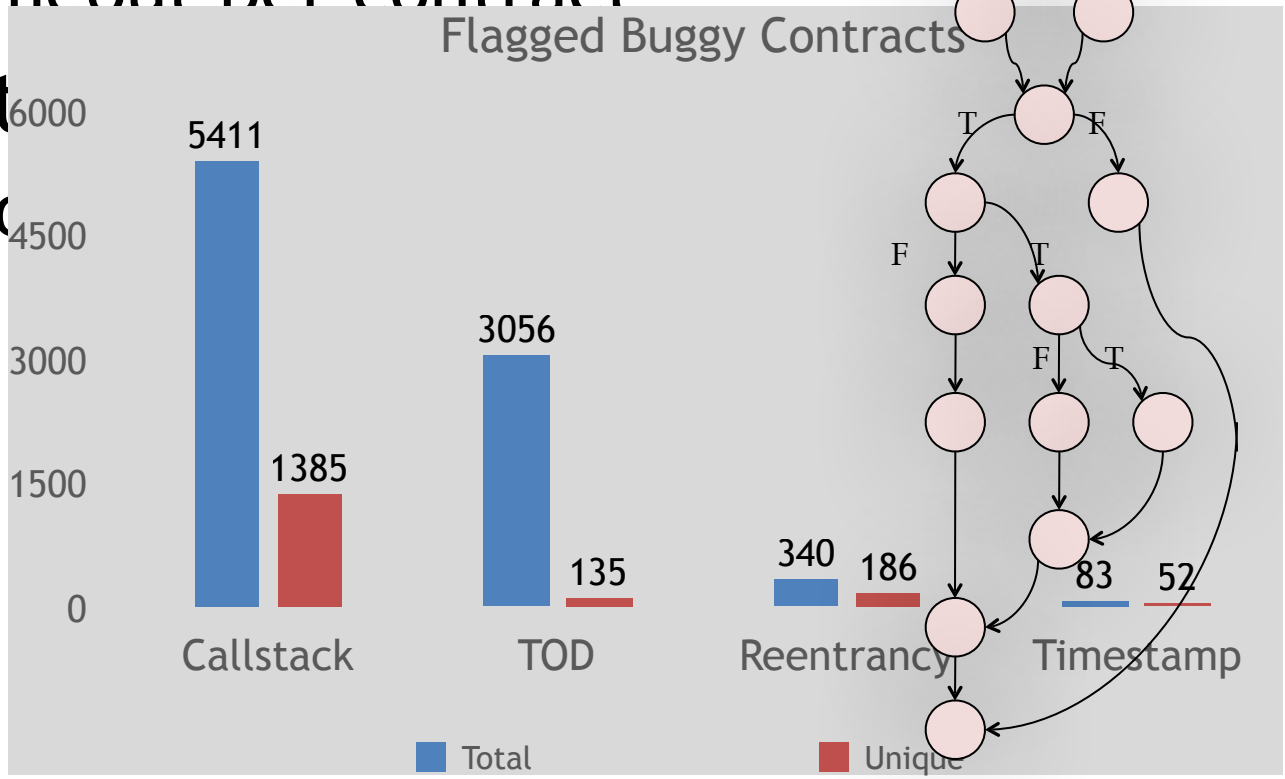


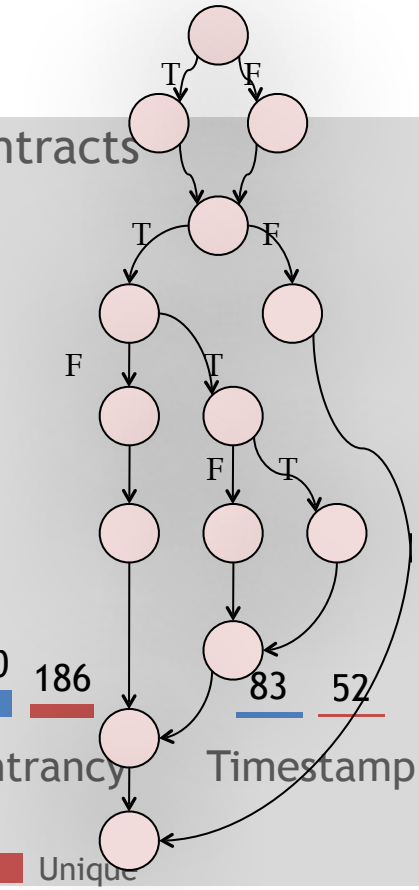
Symbolic Execution



What Can Oyente Do?

- Detect Bugs In Existing Smart Contracts
  - Run with 19, 366 contracts
  - 30 mins timeout per contract
- Test generat
  - Cover all p





Oyente is Open Source

- https://github.com/ethereum/oyente
- Future work
  - Support more opcodes
  - Handle loops
  - Combine static and dynamic symbolic executions

More in the papers

- Solutions for all bugs
  - Semantic changes
- Details of Oyente's design
- Some interesting statistics
  - All smart contracts
  - Evaluation results

Thanks!

loiluu@comp.nus.edu.sg

loi_luu