Blockchain Security
Smart Contract Analysis

Yue Duan
Outline

- Research paper:
  - Making Smart Contracts Smarter
  - Securify: Practical Security Analysis of Smart Contracts
Making Smart Contracts Smarter

Loi Luu, Duc-Hiep Chu, Hrishi Olickel Prateek Saxena, Aquinas Hobor

CCS 2016
Programming Secure Smart Contracts is Hard

- Smart Contract != normal programs
  - self-executed
  - one-shot programs
- New language
  - solidity != Javascript
  - serpent != python
I think TheDAO is getting drained right now

89d · ledgerwatch · self.ethereum

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 dApp (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.
Apart from call-stack and reentrancy, are there other bugs?
How many contracts are vulnerable?
Challenges

- Source-code may not be always available

- Too many contracts
  - manual analysis is impractical
Contributions

- Identify new bugs
  - transaction ordering dependence
  - timestamp dependence

- Oyente: smart contract analyzer
  - symbolic execution tool
  - detect all popular bugs
    - TOD
    - timestamp dependence
    - reentrancy
    - mishandling exceptions (e.g. send)
Transaction Ordering Dependence

PuzzleSolver Contract

- Balance: 100
- PuzzleSolver()
  - SetPuzzle
  - reward=100
- SubmitSolution(solution)
  - if isCorrect(solution):
    - Send(reward)
- UpdateReward(newReward)
  - reward=newReward

Anyone can submit a solution to claim the reward

Owner can update the reward anytime
Scenario 1

Miners

Block
Random TXs
Other TXs

PuzzleSolver Contract

Balance: 100
PuzzleSolver()
SetPuzzle
reward=100
SubmitSolution(solution)
if isCorrect(solution):
Send(reward)
UpdateReward(newReward)
reward=newReward
Scenario 2

Miners

Solution for Puzzle
Random TXs
Other TXs

PuzzleSolver Contract

Balance: 0

PuzzleSolver()
SetDifficulty
reward=100

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

UpdateReward(newReward)
reward=newReward

Random TXs
SubmitSolution
Other TXs
Scenario 3

Solution for Puzzle

Miners

Other TXs

+0

Update Reward to $0!

SubmitSolution

UpdateReward = 0

Other TXs

Block

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

- Can be malicious
  - Saw the targeted TX from the victim
  - Submit the second TX to update the reward
  - Both TXs enter the race

Solution for Puzzle

Update Reward to $0!

Other TXs
Timestamp Dependence

```
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;
    }
    ...
}
```

randomness = F(timestamp)
function lendGovernmentMoney(address buddy) 
    returns (bool) {

    if (lastTimeOfNewCredit + TWELVE_HOURS > block.timestamp) {
        msg.sender.send(amount);
        // Sends jackpot to the last creditor
        creditorAddresses[nCreditors - 1].send(profitFromCrash);
        owner.send(this.balance);
    }
}
Timestamp is not reliable

- Miners can vary the block timestamp
  
  \[ \text{block.timestamp} \leq \text{now} + 900 \land \text{block.timestamp} \geq \text{parent.timestamp} \]

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

- Based on symbolic execution
- Modularized architecture

Diagram:
- ByteCode
- Ethereum State
- CFG BUILDER
- EXPLORER
- CORE ANALYSIS
- VALIDATOR
- Z3 Bit-Vector Solver
- Visualizer
Is there any value of $x$?

$$C_1 \land C_2 \land C_3 \land (z = x + 2)$$

Theorem Prover

$C_1: (x > 0)$

$C_2: (z < 15)$

$C_3: (z < 8)$

$z = x + 2$
Evaluation

- Detect buys in real-world smart contracts
  - Run with 19,366 contracts
  - 30 mins timeout
Securify: Practical Security Analysis of Smart Contracts

Petar Tsankov, Andrei Dan Dana, Drachsler-Cohen, Arthur Gervais,
Florian Bünzli, Martin Vechev

ACM CCS 2018
Motivation

All possible contract behaviors

Security violations
Motivation

Truffle
Testing
Report true bugs
Can miss bugs

Oyente, Mythril, MAIAN
Dynamic (symbolic) analysis
Report true bugs
Can miss bugs

WANTED: Automated Verifier
Can report false alarms
No missed bugs

Bug finding
Verification
Key insight

- When contracts satisfy/violate a **property**, they often also satisfy/violate **a much simpler property**

```plaintext
function withdraw() {
    uint amount = balances[msg.sender];
    msg.sender.call.value(amount)();
    balances[msg.sender] = 0;
}
```

**Security property**
No state changes after call instructions

- **Hard to verify in general**

**Compliance pattern**
No writes to storage may follow call instructions

- **Easier to check automatically**

**Violation pattern**
A write to storage must follow call instructions

- Verifies 91% of all deployed contracts
Classifying Behaviors
Securify: A Practical Verifier

Decompile

EVM Bytecode

Intermediate Representation

00: \texttt{x = Balance}  
02: \texttt{y = 0x20}  
04: \texttt{If (x == 0x00)}  
06: \texttt{MStore(y, x)}  
08: \texttt{z = y}  
0a: \texttt{goto 0x42}  

Infer facts

Semantic Representation

\begin{align*}
\text{MemTag}(0x20, \text{Balance}) \\
\text{MemTag}(0x40, \text{Const}) \\
\text{VarTag}(z, \text{Const}) \\
\text{VarTag}(k, \text{Gas}) \\
\text{Assign}(s, 0x20) \\
\text{Call}(s, k) \\
\end{align*}

Check patterns

TOTAL issues

Transaction Recording

Gas

Recursive Call

Insecure Coding Patterns

Untrusted Exception

Results of phantom reads should not be used to manipulate the ledger.

Simulark ID
From EVM to CFG over SSA

- Decompiling EVM bytecode
  - Convert into SSA format
  - Perform partial evaluation
    - Resolve jump dest, memory/storage offsets
  - Identify and inline methods
  - Construct CFG
Facts Inference

- Many properties can be checked on the contracts’ dependency graph

<table>
<thead>
<tr>
<th>Flow dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MayFollow(l, l')$</td>
<td>The instruction at label $l$ may follow that at label $l'$</td>
</tr>
<tr>
<td>$MustFollow(l, l')$</td>
<td>The instruction at label $l$ must follow that at label $l'$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data dependencies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$MayDepOn(x, t)$</td>
<td>The value of $x$ may depend on tag $t$</td>
</tr>
<tr>
<td>$DetBy(x, t)$</td>
<td>For different values of $t$ the value of $x$ is different.</td>
</tr>
</tbody>
</table>
Facts Inference

- Example: \textit{MayFollow}

\begin{align*}
\text{MayFollow}(i,j) & \leftarrow \text{Follow}(i,j) \\
\text{MayFollow}(i,j) & \leftarrow \text{Follow}(i,k), \text{MayFollow}(k,j)
\end{align*}
Deriving MayDepOn

1: x := Balance
2: Mstore(0x20, x)
3: y := MLoad(0x20)
4: z := x + y

MayDepOn(x, t) ← Assign(x, t)
MayDepOn(x, t) ← Op(_, x, x'), MayDepOn(x', t)
MayDepOn(x, t) ← MLoad(l, x, o), isConst(l, o), MemTag(l, o, t)
MayDepOn(x, t) ← MLoad(l, x, o), ¬isConst(l, o), MemTag(l, __, t)

MemTag(l, o, t) ← MStore(l, o, x), isConst(o), MayDepOn(x, t)
MemTag(l, T, t) ← MStore(l, o, x), ¬isConst(o), MayDepOn(x, t)
MemTag(l, o, t) ← Follows(l, l'), MemTag(l', o, t), ¬MStore(l, o, __)
Pattern Check: the DAO attack

Security property: No state changes after call instructions

Compliance pattern: 
\[ \text{Call}(l, \_\_\_, \_\_\_): \neg \exists \text{Store}(l', \_\_\_, \_\_\_). \text{MayFollow}(l, l') \]

Violation pattern: 
\[ \text{Call}(l, \_\_\_, \_\_\_): \exists \text{Store}(l', \_\_\_, \_\_\_). \text{MustFollow}(l, l') \]
Pattern Check: Unrestricted Writes

Security property: No storage offset is writable by all users

Compliance pattern

Violation pattern

SSStore(_, x, _): DetBy(x, Caller)

SSStore(l, x, _): ¬MayDepOn(x, Caller) ∧ ¬MayDepOn(l, Caller)
Evaluation

- Dataset
  - 80 open-source smart contracts

- Baseline
  - Oyente
  - Mythril

- Experiment
  - Run contracts using the three tools
  - Manually inspect each reported vulnerability
Evaluation
Thank you!

Questions?