ZEUS: Analyzing Safety of Smart Contracts

Sukrit Kalra
IBM Research

Seep Goel
IBM Research

Mohan Dhawan
IBM Research

Subodh Sharma
IIT-Delhi
Smart Contracts

• Self-executing programs that encode the terms of interaction between multiple parties

• The code exists and runs on the blockchain network
Smart Contracts

• The participating entities need to ensure:

• **Correctness:** Syntactic implementation follows best practices

• **Fairness:** Code adheres to higher-level business logic
Correctness: The DAO

The New York Times

A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency

```solidity
function withdrawRewardFor(address _account) returns (bool _success) {
    uint reward = balanceOf(_account);
    reward *= rewardAccount.accumulatedInput();
    reward /= totalSupply;
    reward -= paidOut[_account];
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
```
Correctness: The DAO

The New York Times

A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency

```solidity
function withdrawRewardFor(address _account) 
    returns (bool _success) {
    uint reward = balanceOf(_account);
    reward *= rewardAccount.accumulatedInput();
    reward /= totalSupply;
    reward -= paidOut[_account];
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
```
Correctness: The DAO

The New York Times

A Hacking of More Than $50 Million Dashes Hopes in the World of Virtual Currency

function withdrawRewardFor(address _account)
    returns (bool _success) {
    uint reward = balanceOf(_account);
    reward *= rewardAccount.accumulatedInput();
    reward /= totalSupply;
    reward -= paidOut[_account];
    if (!rewardAccount.payOut(_account, reward))
        throw;
    paidOut[_account] += reward;
    return true;
}
Fairness: AuctionHouse

- By law, auction can be of two types:
  - **With Reserve:** Seller is allowed to bid
  - **Without Reserve:** Seller is *not* allowed to bid
Fairness: AuctionHouse

• By law, auction can be of two types:
  
  • **With Reserve:** Seller is allowed to bid
  
  • **Without Reserve:** Seller is **not** allowed to bid
Fairness: AuctionHouse

• By law, auction can be of two types:
  - With Reserve: Seller is allowed to bid
  - Without Reserve: Seller is not allowed to bid

Seller shouldn't be able to bid on the auction #8

Open  ericxtang opened this issue on Oct 16, 2016 · 2 comments  bug
Outline

• Overview
• Motivation
• Zeus
• Implementation
• Evaluation
• Conclusion
Motivation

The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

The DAO, the distributed autonomous organization that had collected over $150m worth of the cryptocurrency ether, has reportedly been hacked, sparking a broad market sell-off.

A leaderless organization comprised of a series of smart contracts written on the ethereum codebase, The DAO has lost 3.6m ether, which is currently sitting in a separate wallet after being split off into a separate grouping dubbed a "child DAO"
Motivation

£200 million worth of digital cryptocurrency is wiped out as bungling developer locks investors out while trying to stop hackers

- A developer was fixing a bug that let hackers steal funds from virtual wallets
- But the developer accidentally left a second flaw in its systems
- When the user tried to undo the damage by deleting the flaw in the code, this locked the funds in the wallets permanently
- The only way to reverse the issue is a 'hard-fork', but not everyone supports this
Motivation

- A developer was found.
- But the developer
  - When the user tried
    - The only way to return

The DAO, the di-

cryptocurrency

A leaderless org
codebase, The I

split off into a se
Motivation

Hackers Have Walked Off With About 14% of Big Digital Currencies

By Olga Kharif
January 18, 2018, 7:19 PM GMT+5:30

Cybercriminals compromise Bitcoin, Ether supply, blockchains

Crypto-crazed users adopt technology without weighing risks
Verification

- Verification suffers from state space explosion!
Verification

- Verification suffers from **state space explosion!**
Verification

- Verification suffers from **state space explosion**!

Solidity contracts are small
Smart Contract Verification

- Oyente (CCS ’16) uses symbolic execution for **bug detection** at the bytecode level
  - Neither sound nor complete
  - Cannot handle fairness issues
- Bhargavan *et al.* (PLAS ’16) formally verify contracts written in a **subset of solidity** using F*
  - Require manual proofs
  - Important constructs unsupported
Failed send()

for (uint i = 0; i < investors.length; i++) {
    if (investors[i].invested == minimum) {
        payout = investors[i].payout;
        if (!investors[i].address.send(payout))
            throw;
        investors[i] = newInvestor;
    }
}

- send() is used to transfer money in contracts

- A failed send() can lock contracts!
Failed send ()

```java
for (uint i = 0; i < investors.length; i++) {
    if (investors[i].invested == minimum) {
        payout = investors[i].payout;
        if (!investors[i].address.send(payout))
            throw;
        investors[i] = newInvestor;
    }
}
```

- `send()` is used to transfer money in contracts
- A failed `send()` can lock contracts!
Block State Dependence

function resetInvestment() {
  if (block.timestamp < lastInvestment + ONE_MINUTE)
    throw;

  lastInvestor.send(jackpot);
  ...
}

- Block state variables are used to generate randomness
- They can be tampered by the miner for profit!
Outline

• Overview
• Motivation
• Zeus
• Implementation
• Evaluation
• Conclusion
Our Approach
Our Approach

Correct placement of asserts
Placement of asserts

- **<Subject, Object, Operation>** determine the location

function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}

- `<Subject>`: Participants
- `<Object>`: Assets
- `<Operation>`: API Invocations
- `<Condition>`: Predicates
- `<Result>`: True or False
Placement of asserts

- `<Condition, Result>` determine the predicate

```plaintext
function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}
```
Placement of asserts

Conservative taint analysis ensures no false negatives

• `<Condition, Result>` determine the predicate
Our Approach
Our Approach

Sound over-approximation of the Solidity semantics
Soundness

\texttt{havoc(msg.value);}

\begin{verbatim}
function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}
\end{verbatim}

- \texttt{havoc(...)} expands the domain of legitimate values that a variable can take to the type-defined domain

- Ensures reasoning about all program paths
Our Approach

Protection against aggressive LLVM optimizations
LLVM Optimizations

- Optimizations introduce architecture specific functions which may not be modeled in the verifier
  - `add` replaced with `uadd.with.overflow` which is not modeled
  - **Solution:** Enforce no optimizations

- Verifier eliminates non side-affecting variables and function calls
  - `send(...) is optimized away`
  - **Solution:** Global variable for external function returns
End-To-End Example

```xml
<Subject> msg.sender </Subject>
:Object> msg.value </Object>
<Operation trigger="pre"> send </Operation>
<Condition> msg.value <= balance </Condition>
<Result> True </Result>
```

```javascript
function transfer() {
  msg.sender.send(msg.value);
  balance -= msg.value;
}
```
End-To-End Example

```javascript
function transfer() {
    assert(msg.value <= balance);
    msg.sender.send(msg.value);
    balance -= msg.value;
}
```
define void @transfer() {
  entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* %__value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br il %2, label "%75", label "%74"

  "%74":
      ; preds = %"64"
      call void @__VERIFIER_error()
      br label "%75"

  "%75":
      ; preds = %"74", %"64"
      %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
      %3 = load i160* %__sender
      %4 = call i1 %send(i160 %3, i256 %0)
      store i1 %4, i1* @sendReturnVal
      %5 = sub i256 %1, %0
      store i256 %5, i256* @balance
      ret void
}

define void @main() {
  entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
...
End-To-End Example

An assert failure is modeled as a call to the verifier’s error function
End-To-End Example

```c
define void @transfer() {
entry:
    %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
    %0 = load i256* %__value
    %1 = load i256* @balance
    %2 = icmp ule i256 %0, %1
    br il %2, label %"75", label %"74"

"74":           ; preds = %"64"
    call void @__VERIFIER_error()
    br label %"75"

"75":           ; preds = %"74", %"64"
    %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
    %3 = load i160* %__sender
    %4 = call i1 %send(i160 %3, i256 %0)
    store i1 %4, i1* @sendReturnVal
    %5 = sub i256 %1, %0
    store i256 %5, i256* @balance
    ret void
}

define void @main() {
entry:
    %0 = call i256 @__VERIFIER_NONDET()
    store i256 %0, i256* @balance
    ...
}
```

Globals are automatically havoc-ed to explore the entire data domain.
End-To-End Example

```c
#define void @transfer() {
  entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* %__value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br il %2, label "%75", label "%74"

"74": ; preds = %"64"
  call void @__VERIFIER_error()
  br label "%75"

"75": ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %__sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

#define void @main() {
  entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
}
```

The return value of `send` is stored in a global variable.
define void @transfer() {
  entry:
  %__value = getelementptr %msgRecord* @msg, i32 0, i32 4
  %0 = load i256* %__value
  %1 = load i256* @balance
  %2 = icmp ule i256 %0, %1
  br i1 %2, label %"75", label %"74"
}

"74": ; preds = %"64"
  call void @__VERIFIER_error()
  br label %"75"

"75": ; preds = %"74", %"64"
  %__sender = getelementptr %msgRecord* @msg, i32 0, i32 2
  %3 = load i160* %__sender
  %4 = call i1 @send(i160 %3, i256 %0)
  store i1 %4, i1* @sendReturnVal
  %5 = sub i256 %1, %0
  store i256 %5, i256* @balance
  ret void
}

define void @main() {
  entry:
  %0 = call i256 @__VERIFIER_NONDET()
  store i256 %0, i256* @balance
  ...
define void transfer() {
    entry:
        %value = getelementptr %msgRecord* @msg, i32 0, i32 4
        %0 = load i256* %value
        %1 = load i256* @balance
        %2 = icmp ule i256 %0, %1
        br i1 %2, label "%75", label "%74"
    "74":
        ; preds = %"64"
        call void __VERIFIER_error()
        br label "%75"
    "75":
        ; preds = %"74", %"64"
        %sender = getelementptr %msgRecord* @msg, i32 0, i32 2
        %3 = load i160* %sender
        %4 = call i1 @send(i160 %3, i256 %0)
        store i1 %4, i1* @sendReturnVal
        %5 = sub i256 %1, %0
        store i256 %5, i256* @balance
        ret void
}

define void main() {
    entry:
        %0 = call i256 __VERIFIER_NONDET()
        store i256 %0, i256* @balance
        ...
}
Outline

• Overview
• Motivation
• Zeus
  • Implementation
• Evaluation
• Conclusion
Implementation

• Policy Builder
  • Extracts information from the AST nodes in solc
  • Taint analysis to retrieve the policy tuple (~500 LOC C++)

• Solidity to LLVM Translator
  • Generates the LLVM bitcode for the contract (~3000 LOC C++)
  • LLVM passes to automatically insert assertions for correctness bugs

• Verifier
  • Off-the-shelf model checkers that work with LLVM (Seahorn, SMACK)
Outline

• Overview
• Motivation
• Zeus
• Implementation
• Evaluation
• Conclusion
Methodology

• Study over 22.4K Solidity contracts (1524 unique)

• Verifier timeout threshold of 1 min

• Correctness bugs
  • Manually ascertain ground truth for 7 bug classes

• Fairness issues and case study on Hyperledger discussed in the paper
Methodology

- Study over 22.4K Solidity contracts (1524 unique)
- Verifier timeout threshold of 1 min
- Correctness bugs
  - Manually ascertain ground truth for 7 bug classes
- Fairness issues and case study on Hyperledger discussed in the paper

<table>
<thead>
<tr>
<th>Category</th>
<th># Contracts</th>
<th>Source</th>
<th>LLVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAO</td>
<td>140</td>
<td>2.8</td>
<td>24.3</td>
</tr>
<tr>
<td>Game</td>
<td>244</td>
<td>23.3</td>
<td>609.2</td>
</tr>
<tr>
<td>Token</td>
<td>290</td>
<td>25.2</td>
<td>385.9</td>
</tr>
<tr>
<td>Wallet</td>
<td>72</td>
<td>10.8</td>
<td>105.9</td>
</tr>
<tr>
<td>Misc.</td>
<td>778</td>
<td>47.6</td>
<td>924.3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>1524</strong></td>
<td><strong>109.7</strong></td>
<td><strong>2049.6</strong></td>
</tr>
</tbody>
</table>
# Correctness

## Zeus

<table>
<thead>
<tr>
<th>Bug</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Timeout</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Timeout</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reentrancy</td>
<td>1438</td>
<td>54</td>
<td>7</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>548</td>
<td>265</td>
<td>226</td>
<td>485</td>
<td>254</td>
<td>51</td>
<td>31.24</td>
</tr>
<tr>
<td>Unchecked send</td>
<td>1191</td>
<td>324</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0.20</td>
<td>1066</td>
<td>112</td>
<td>203</td>
<td>143</td>
<td>89</td>
<td>188</td>
<td>7.56</td>
</tr>
<tr>
<td>Failed send</td>
<td>1068</td>
<td>447</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer Overflow</td>
<td>378</td>
<td>1095</td>
<td>18</td>
<td>33</td>
<td>40</td>
<td>0</td>
<td>2.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction State</td>
<td>1513</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block State</td>
<td>1266</td>
<td>250</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>798</td>
<td>15</td>
<td>226</td>
<td>485</td>
<td>2</td>
<td>84</td>
<td>0.25</td>
</tr>
<tr>
<td>Dependence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction Order</td>
<td>894</td>
<td>607</td>
<td>13</td>
<td>10</td>
<td>16</td>
<td>0</td>
<td>1.07</td>
<td>668</td>
<td>129</td>
<td>222</td>
<td>485</td>
<td>116</td>
<td>158</td>
<td>14.20</td>
</tr>
<tr>
<td>Dependence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Oyente (CCS ’16)

<table>
<thead>
<tr>
<th>Bug</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Timeout</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Timeout</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reentrancy</td>
<td>1438</td>
<td>54</td>
<td>7</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>548</td>
<td>265</td>
<td>226</td>
<td>485</td>
<td>254</td>
<td>51</td>
<td>31.24</td>
</tr>
<tr>
<td>Unchecked send</td>
<td>1191</td>
<td>324</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0.20</td>
<td>1066</td>
<td>112</td>
<td>203</td>
<td>143</td>
<td>89</td>
<td>188</td>
<td>7.56</td>
</tr>
<tr>
<td>Failed send</td>
<td>1068</td>
<td>447</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer Overflow</td>
<td>378</td>
<td>1095</td>
<td>18</td>
<td>33</td>
<td>40</td>
<td>0</td>
<td>2.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction State</td>
<td>1513</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block State</td>
<td>1266</td>
<td>250</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>798</td>
<td>15</td>
<td>226</td>
<td>485</td>
<td>2</td>
<td>84</td>
<td>0.25</td>
</tr>
<tr>
<td>Dependence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction Order</td>
<td>894</td>
<td>607</td>
<td>13</td>
<td>10</td>
<td>16</td>
<td>0</td>
<td>1.07</td>
<td>668</td>
<td>129</td>
<td>222</td>
<td>485</td>
<td>116</td>
<td>158</td>
<td>14.20</td>
</tr>
<tr>
<td>Dependence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

21st Feb 2018 NDSS 2018
## Correctness

<table>
<thead>
<tr>
<th>Bug</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Time Out</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Time Out</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reentrancy</td>
<td>1438</td>
<td>54</td>
<td>7</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>548</td>
<td>265</td>
<td>226</td>
<td>485</td>
<td>254</td>
<td>51</td>
<td>31.24</td>
</tr>
<tr>
<td>Unchecked send</td>
<td>1191</td>
<td>324</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0.20</td>
<td>1066</td>
<td>112</td>
<td>203</td>
<td>143</td>
<td>89</td>
<td>188</td>
<td>7.56</td>
</tr>
<tr>
<td>Failed send</td>
<td>1068</td>
<td>447</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer Overflow</td>
<td>378</td>
<td>1095</td>
<td>18</td>
<td>33</td>
<td>40</td>
<td>0</td>
<td>2.72</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transaction State Dependence</td>
<td>1513</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block State Dependence</td>
<td>1266</td>
<td>250</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>798</td>
<td>15</td>
<td>226</td>
<td>485</td>
<td>2</td>
<td>84</td>
<td>0.25</td>
</tr>
<tr>
<td>Transaction Order Dependence</td>
<td>894</td>
<td>607</td>
<td>13</td>
<td>10</td>
<td>16</td>
<td>0</td>
<td>1.07</td>
<td>668</td>
<td>129</td>
<td>222</td>
<td>485</td>
<td>116</td>
<td>158</td>
<td>14.20</td>
</tr>
</tbody>
</table>
## Correctness

<table>
<thead>
<tr>
<th></th>
<th>Zeus</th>
<th>Oyente (CCS ’16)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bug</strong></td>
<td>Safe</td>
<td>Unsafe</td>
</tr>
<tr>
<td></td>
<td>Safe</td>
<td>Unsafe</td>
</tr>
<tr>
<td><strong>Zeus</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dependence</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Block State</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependence</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Transaction</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Order Dependence</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Zeus has no false negatives
## Correctness

<table>
<thead>
<tr>
<th>Bug</th>
<th>Zeus</th>
<th>Oyente (CCS ’16)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Safe</td>
<td>Unsafe</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>1438</td>
<td>54</td>
</tr>
<tr>
<td>Unchecked send</td>
<td>1191</td>
<td>324</td>
</tr>
<tr>
<td>Failed send</td>
<td>1068</td>
<td>447</td>
</tr>
<tr>
<td>Integer Overflow</td>
<td>378</td>
<td>1095</td>
</tr>
<tr>
<td>Transaction State Dependence</td>
<td>1513</td>
<td>8</td>
</tr>
<tr>
<td>Block State Dependence</td>
<td>1266</td>
<td>250</td>
</tr>
<tr>
<td>Transaction Order Dependence</td>
<td>894</td>
<td>607</td>
</tr>
</tbody>
</table>
Correctness

<table>
<thead>
<tr>
<th>Bug</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Time Out</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
<th>Safe</th>
<th>Unsafe</th>
<th>No Result</th>
<th>Time Out</th>
<th>False +ve</th>
<th>False -ve</th>
<th>% False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeus</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oyente (CCS ’16)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Zeus has lesser false positives

<table>
<thead>
<tr>
<th>Dependence</th>
<th>Block State Dependence</th>
<th>Transaction Order Dependence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1266</td>
<td>894</td>
</tr>
<tr>
<td></td>
<td>250</td>
<td>607</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>798</td>
<td>1.07</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>668</td>
</tr>
<tr>
<td></td>
<td>226</td>
<td>222</td>
</tr>
<tr>
<td></td>
<td>485</td>
<td>485</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>84</td>
<td>158</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>14.20</td>
</tr>
</tbody>
</table>

21st Feb 2018  NDSS 2018
Verification Complexity

No state space explosion!
Zeus is scalable
Verification Time

CDF

Time (min)

Zeus

Oyente

21st Feb 2018

NDSS 2018
Zeus is quick!
Verified 97% contracts in less than a minute
Outline

• Overview
• Motivation
• Zeus
• Implementation
• Evaluation

• Conclusion
Conclusion

• Smart contracts are buggy
  • Faithful execution ensured by consensus
  • Correctness and Fairness not guaranteed
• Zeus is a framework enabling verification of smart contracts
  • Works at scale
    • Study over 22.4K Solidity contracts (1524 unique)
    • Around 94% contracts vulnerable to correctness bugs
  • Sound with low verification overhead
    • Zero false negatives, lesser false positives
    • Takes under 1 min to analyze 97% contracts
Thank You!

Contact: sukrit.kalra@in.ibm.com