Zeus:
Analyzing Safety of Smart Contracts

Philippe Schlattner
Overview

Introduction to the Safety of Smart Contracts

ZEUS – A Framework to formally verify Smart Contracts

Evaluation and Results of the ZEUS Prototype & Conclusion
Overview

Introduction to the Safety of Smart Contracts

ZEUS – A Framework to formally verify Smart Contracts

Evaluation and Results of the ZEUS Prototype & Conclusion
Introduction to the Safety of Smart Contracts

History

- „Smart Contract“: Introduced in 1996 by Nick Szabo

What is a Smart Contract?

- Contract law and related business practices between strangers on the Internet

Safety of Smart Contracts

- Enabled by blockchains

- Computation on a blockchain or distributed ledger

Motivations

Related Work
Introduction to the Safety of Smart Contracts

What is a Smart Contract?

- An immutable, self-executing program
- Encodes interaction between multiple parties
- Enable public visible, trackable and irreversible transactions without a third party
- Reside across a distributed, decentralized blockchain network
Introduction to the Safety of Smart Contracts

What is a Smart Contract?

Safety of Smart Contracts

Motivations

Related Work
Introduction to the Safety of Smart Contracts

Safety of Smart Contracts can be divided in 2 key issues:

- **Correctness** as adherence to safe programming practices.

- **Fairness** as adherence to agreed upon higher-level business logic.
Introduction to the Safety of Smart Contracts

Broad classes of correctness and fairness issues:

- **Correctness issues** due to incorrect use of constructs or programming paradigms

- **Fairness issues** due to syntactically correct contracts that do not implement the desired higher-level business logic.
Introduction to the Safety of Smart Contracts

Correctness issue – Reentrancy Attack (e.g. TheDAO bug)

```solidity
contract Wallet {
    mapping(address => uint) private userBalances;
    function withdrawBalance() {
        uint amountToWithdraw = userBalances[msg.sender];
        if (amountToWithdraw > 0) {
            msg.sender.call(userBalances[msg.sender]);
            userBalances[msg.sender] = 0;
        }
    }
}
```

```
contract AttackerContract {
    function () {
        Wallet wallet;
        wallet.withdrawBalance();
    }
}
```
Correctness issue – Unchecked send

Solidity allows 2300 gas upon a send call

- prizePaidOut set to true without payout
Introduction to the Safety of Smart Contracts

Correctness issue – Integer overflow

```
(1) uint payout = balance/participants.length;
(2) for (var i = 0; i < participants.length; i++)
(3)     participants[i].send(payout);
```

- i can hold only integers up to 255
- After 255 wrap around
Introduction to the Safety of Smart Contracts

**Fairness issue** – Unchecked resources

```
(1) contract Wallet {
(2)   uint256 balance;
(3)   ... // initialize balance
(4)   function checkAndPay(bytes32 sol, address dest, uint amt) {
(5)     balance -= amt;
(6)     if (solution != correct) { throw; }
(7)     dest.send(amt);
(8)   }
```

- No payout for a correct solution
Introduction to the Safety of Smart Contracts

**Fairness issue** – Variable mixup

- There are 2 different variables:
  - payoutCursor_Id
  - payotCursor_Id
Introduction to the Safety of Smart Contracts

**Fairness issue** – An unfair auction house contract

```solidity
function placeBid(uint auctionId) Payable
returns (bool success) {
    Auction a = auctions[auctionId];
    if (a.currentBid >= msg.value) throw;
    uint bidIdx = a.bids.length++;
    Bid b = a.bids[bidIdx];
    b.bidder = msg.sender;
    b.amount = msg.value;
    ...
    BidPlaced(auctionId, b.bidder, b.amount);
    return true;
}
```

- 2 different art of auction by law:
  - With reserve: Seller is allowed to bid
  - Without reserve: Seller is **not** allowed to bid
Introduction to the Safety of Smart Contracts

Other Smart Contract Tester/Verification Tools

Securify:
- From the Software Reliability Lab, ETH Zurich
- Beta available online

Oyente:
- Based on symbolic execution only
- Not sound, nor complete
- Don’t handle fairness issues

Bhargavan et al.:
- Verify smart contracts in a subset of Solidity
- Requires manual proofs
Overview

Introduction to the Safety of Smart Contracts

ZEUS – A Framework to formally verify Smart Contracts

Evaluation and Results of the ZEUS Prototype & Conclusion
Overview

Introduction to the Safety of Smart Contracts

ZEUS – A Framework to formally verify Smart Contracts

Evaluation and Results of the ZEUS Prototype & Conclusion
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder
2) Policy Embedding
3) Source Code Translation
4) Verification
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder

2) Policy Embedding

3) Source Code Translation

4) Verification

• Fairness criteria in a five-tuple XACML style policy

- <Subject>
  • Participants that have to be tracked

- <Object>
  • Assets with which the participants interact

- <Operation>
  • Set of side affecting operations API invocations, ...

- <Condition>
  • Predicate

- <Result>
  • Is the interaction between subject + object governed by condition permitted or not
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder

2) Policy Embedding

3) Source Code Translation

4) Verification

Example

Solidity code

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

Policy

```
Policy

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

No balance check, thus not correct
ZEUS – A Framework to formally verify Smart Contracts

- Policy embedded inside the smart contract
- Place assertions at the right places

Where to place the assertion
- **<Subject>**
- **<Object>**
- **<Operation>**
- **<Condition>**
- **<Result>**

Determine predicate
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder

2) Policy Embedding

3) Source Code Translation

4) Verification

Example

function transfer() {
  assert(msg.value <= balance);
  msg.sender.send(msg.value);
  balance = balance - msg.value;
}
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder
   - Conversion to the LLVM IR

2) Policy Embedding
   - Encodes execution semantics of the blockchain platform

3) Source Code Translation

4) Verification

Zeus provides the first Solidity to LLVM translator.

- Decoupling of the backend from the source language
- Enable verification versatility
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder

2) Policy Embedding

3) Source Code Translation

4) Verification

- Assert failure calls the verifier’s error function

Example
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder

2) Policy Embedding

3) Source Code Translation

4) Verification

Verifier that works atop LLVM bitcode

Verifier checks whether all the assertions hold
ZEUS – A Framework to formally verify Smart Contracts

Overview

1) Policy Builder

2) Policy Embedding

3) Source Code Translation

4) Verification

Example

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

⇒ Assertion failure
Overview

Introduction to the Safety of Smart Contracts

ZEUS – A Framework to formally verify Smart Contracts

Evaluation and Results of the ZEUS Prototype & Conclusion
Overview

Introduction to the Safety of Smart Contracts

ZEUS – A Framework to formally verify Smart Contracts

Evaluation and Results of the ZEUS Prototype & Conclusion
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info's about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion

Prototype

Policy Builder:
- Extracts information from the AST nodes in `solc`
- Taint analysis + user assistance to retrieve the policy
- 500 LOC in C++

Source Code Translator:
- LLVM APIs for the translation to bitcode
- 3000 LOC in C++

Verifier:
- Off-the-shelf model checker Seahorn, using CHCs over LLVM bitcode
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

- Analyzing 1524 unique Smart Contracts (from a set of 22,493)
- Timeout 1min
- Comparison to Oyente
- Ground truth annotated by hand for 7 bug classes

Info’s about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion
### Evaluation and Results of the ZEUS Prototype & Conclusion

#### Setup & Method

#### Info’s about the Dataset

<table>
<thead>
<tr>
<th>Category</th>
<th>#Contracts</th>
<th>LOC (K) Source</th>
<th>LOC (K) LLVM</th>
<th>#Send/Call</th>
<th>#Ext. Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAO</td>
<td>140</td>
<td>2.8</td>
<td>24.3</td>
<td>252</td>
<td>350</td>
</tr>
<tr>
<td>Game</td>
<td>244</td>
<td>23.3</td>
<td>609.2</td>
<td>851</td>
<td>16</td>
</tr>
<tr>
<td>Token</td>
<td>290</td>
<td>25.2</td>
<td>385.9</td>
<td>311</td>
<td>271</td>
</tr>
<tr>
<td>Wallet</td>
<td>72</td>
<td>10.8</td>
<td>105.9</td>
<td>186</td>
<td>6</td>
</tr>
<tr>
<td>Misc.</td>
<td>778</td>
<td>47.6</td>
<td>924.3</td>
<td>1102</td>
<td>498</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>
<td><strong>2702</strong></td>
<td><strong>1141</strong></td>
</tr>
</tbody>
</table>

#### Results & Performance

#### ZEUS’s Versatility

#### Conclusion
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info’s about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info's about the Dataset

Results & Performance

ZEUS's Versatility

Conclusion
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info's about the Dataset

Results & Performance

ZEUS's Versatility

Conclusion

Results

<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 Alarms</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>
</tr>
<tr>
<td>Unchkd. send</td>
<td>1191</td>
<td>324</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0.20</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>
</tr>
<tr>
<td>Int. overflow</td>
<td>378</td>
<td>1095</td>
<td>18</td>
<td>33</td>
<td>40</td>
<td>0</td>
<td>2.72</td>
</tr>
<tr>
<td>Tx. State Dep.</td>
<td>1513</td>
<td>8</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Blk. State Dep.</td>
<td>1266</td>
<td>250</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Tx. Order Dep.</td>
<td>894</td>
<td>607</td>
<td>13</td>
<td>10</td>
<td>16</td>
<td>0</td>
<td>1.07</td>
</tr>
</tbody>
</table>

<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 Alarms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reentrancy</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>Unchkd. send</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>798</td>
<td>15</td>
<td>226</td>
<td>485</td>
<td>2</td>
<td>84</td>
<td>0.25</td>
</tr>
<tr>
<td>Int. overflow</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>Tx. State Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blk. State Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tx. Order Dep.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info’s about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion

Performance

(c) Verification constraints generated.

(d) Verification time in minutes.
Evaluation and Results of the ZEUS Prototype & Conclusion

- Trials with Fabric blockchain as front-end, with GO language
- Trials with SMACK as backend verifier

→ Both suggest that it is easy to extend Zeus
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info’s about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion

• Smart contracts are not safe
  • Immutability and faithful execution ensured by blockchain
  • But correctness and fairness is not guaranteed (95% of the analyzed contracts had safety issues)

• Zeus is a solution to the formal verification of smart contracts
  • Works at scale (analyzed over 22 000 contracts)
  • Sound (no false negatives, few false positives)
  • Fast verification (97% of the contracts in under a minute)
Thank you!
Any questions?
Evaluation and Results of the ZEUS Prototype & Conclusion

Performance

Setup & Method

Info’s about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info’s about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion

(d) Frequently used LLVM operations.
Evaluation and Results of the ZEUS Prototype & Conclusion

Setup & Method

Info’s about the Dataset

Results & Performance

ZEUS’s Versatility

Conclusion