Ethereum Bugs through the Lens of Formal Verification

Shelly Grossman, Ittai Abraham, Guy Golan-Gueta, Y. Michalevsky, Noam Rinetzky, Mooly Sagiv and Yoni Zohar
whoami

- Phd in security from Stanford

- CTO and co-founder @Anjuna (www.anjuna.io)
Smart Contracts

- Smart Contracts are actual computer programs that can be stored on a blockchain
- Executed by miners that form a consensus on the resulting state post execution
- Simple contracts can be implemented on top of blockchains like **bitcoin** or **litecoin**
- **ethereum**: a flexible and popular smart contracts framework over its own blockchain
Powerful but tricky

The $280M Ethereum’s Parity bug.
A critical security vulnerability in Parity multi-sig wallet got triggered on 6th November—paralyzing wallets created after the 20th July.

As you may have read, Parity issued a security advisory today to inform its users and developers about a bug that got “accidentally deployed” which resulted in freezing more than $280M worth of ETH, belonging to Parity’s Founder & Ethereum former core developer, Joseph Woods.

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".
The DAO Bug

Object DAO
Map<Object, int> credit
int balance
Invariant (sum o: credit[o]) = balance

Method withdrawAll(Object o)
2: if (oCredit > 0)
   // 2.5: credit[o] = 0
3: this.balance -= oCredit
4: o.pay(oCredit)
5: credit[o] = 0

Method deposit(Object o, int amount)
6: credit[o] += amount
7: this.balance += amount

Expected invariant: sum of credits should equal the balance
The DAO Bug

Object GoodClient
   Object Dao, int balance
   Method init(Object dao)
      1: this.Dao = dao
   Method pay(int profit)
      2: this.balance += profit
   Method depositCredit(Object dao, int amount)
      3: Dao.deposit(this, amount)
   Method getCredit(Object dao)
      4: Dao.withdrawAll(this)

Doesn’t violate the invariant

Object Attacker
   Object Dao, bool stop, int balance
   Method init(Object dao)
      1: Dao = dao
      2: stop = false
   Method pay(int profit)
      3: this.balance += profit
      4: if (!stop)
         5: stop = true
      6: Dao.withdrawAll(this)
      7: stop = false

Violates the invariant
The DAO Bug

Attack Execution Trace

\[ D.c[G] = 100 \]
\[ D.c[A] = 100 \]
\[ D.b = 200 \]
\[ A.b = 0 \]
\[ A.s = false \]

1. \[ w \{ \]
2. \[ p \{ \]
3. \[ w \{ \]
4. \[ p \{ \]
5. \[ p \} \]
6. \[ w \} \]
7. \[ p \} \]
8. \[ w \} \]
The DAO Bug
The DAO Bug

Object DAO
Map<Object, int> credit
int balance
Invariant \( \text{sum } o: \text{credit}[o] \) = balance

Method withdrawAll(Object o)
2: if (oCredit > 0)
// 2.5: credit[o] = 0
3: this.balance -= oCredit
4: o.pay(oCredit)
5: credit[o] = 0

Method deposit(Object o, int amount)
6: credit[o] += amount
7: this.balance += amount

Adding this line fixes the bug
Callbacks are Dangerous
ECF Contracts

- **ECF** - Effectively Callback Free
- **dECF** - Dynamically Effective Callback Free
  A contract is dECF if there exists an equivalent execution of the contract without callbacks.
- **sECF** - Statically Effectively Callback Free
  A contract is sECF if all possible executions are ECF.

The DAO contract is not ECF
Dynamic Verification

- sECF for Ethereum contracts is not generally decidable - since the EVM is Turing Complete

- sECF is statically decidable for finite state space
  - Decidable for objects that can be modeled using Pushdown Automata (PDA)

- Instead, we implemented dynamic verification
Dynamic Verification

- **Segment**: represents the maximal sequence of adjacent transitions pertaining to the same object.

- A segment $t=(R,W,D,Idx)$ contains info. about read $R(t)$ and write-sets $W(t)$, depth of invocation $D(t)$ and the segment execution index.

- **Commutative Segments**: Segments $t_1$ and $t_2$ commute, if

\[
R(t_1) \cap W(t_2) = \emptyset \wedge R(t_2) \cap W(t_1) = \emptyset
\]
Segments

Method withdrawAll(Object o)
  2: if (oCredit > 0)
      // 2.5: credit[o] = 0
  3: this.balance -= oCredit
  4: o.pay(oCredit)
  5: credit[o] = 0

Object Attacker
  Object Dao, bool stop, int balance
  Method init(Object dao)
    1: Dao = dao
    2: stop = false
  Method pay(int profit)
    3: this.balance += profit
    4: if (!stop)
        5: stop = true
    6: Dao.withdrawAll(this)
    7: stop = false
Algorithm

- **Invocation Order Constraint (IOC):** specifies invocation ordering as a graph
- Add IOC-s based on discovered commutativity of segments

*Theorem.* If the IOC-s graph contains no cycles, the execution is ECF.
Evaluation

- Prototype implementation of detection algorithm based on geth Ethereum client
- 2.2GHz Intel Xeon E5 x 2 (22 cores each, 2 threads / core), 256 GB RAM
- Ran on the entire blockchain from July 30, 2015 to March 30, 2017
- CPU overhead: 3.5%. Memory overhead: 17%
- < 0.01% of executions were non-ECF
- Only 10 out of 100M executions were legitimately non-ECF

<table>
<thead>
<tr>
<th>Blockchain</th>
<th>Date</th>
<th>Contracts</th>
<th>Executions</th>
<th>Callbacks</th>
<th>Non-ECF (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethereum</td>
<td>30.VII.2015-30.III.2017</td>
<td>138,457</td>
<td>81,097,421</td>
<td>128,670</td>
<td>3,315 (0.004%)</td>
</tr>
<tr>
<td>Ethereum</td>
<td>30.III.2017-23.VI.2017</td>
<td>203,859</td>
<td>15,311,650</td>
<td>155,662</td>
<td>6 (&lt;0.001%)</td>
</tr>
<tr>
<td>Eth. Classic</td>
<td>30.VII.2015-29.VI.2017</td>
<td>91,191</td>
<td>32,494,464</td>
<td>81,731</td>
<td>2,288 (0.007%)</td>
</tr>
</tbody>
</table>
# non-ECF Contracts

<table>
<thead>
<tr>
<th>Name</th>
<th>Contract address</th>
<th>Execs.</th>
<th>Execs. w. obs.</th>
<th>Non-ECF</th>
<th>Stack depth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ethereum Network (ETH)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vulnerable. Fixed.</td>
<td>0xd654bdd32fc99471455e...</td>
<td>924</td>
<td>143</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>DAO</td>
<td>0xbb9bc244d798123fde78...</td>
<td>274,820</td>
<td>103,064</td>
<td>3,296</td>
<td>2-146</td>
</tr>
<tr>
<td>DAO related</td>
<td>0x34a5451ef61a567ee088...</td>
<td>91</td>
<td>8</td>
<td>1</td>
<td>46</td>
</tr>
<tr>
<td>Dark DAO</td>
<td>0x304a554a310c7e546dfe...</td>
<td>13,223</td>
<td>2,812</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>DAO-like Exercise</td>
<td>0x59752433dbe28f5aa59b...</td>
<td>15</td>
<td>6</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Ambisafe</td>
<td>0x97361ea911d6348cf2af...</td>
<td>44</td>
<td>42</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Validity Labs training</td>
<td>0xbf78025535c98f4c605f...</td>
<td>25</td>
<td>22</td>
<td>3</td>
<td>3-9</td>
</tr>
<tr>
<td>Ambisafe</td>
<td>0x232f3a7723137ced12bc...</td>
<td>144</td>
<td>142</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Ambisafe</td>
<td>0x7c525c4e3b273a3afc4b...</td>
<td>35</td>
<td>33</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>Ethereum Classic Network (ETC)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C1</td>
<td>0xd654bdd32fc99471455e...</td>
<td>850</td>
<td>143</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>C2</td>
<td>0xbb9bc244d798123fde78...</td>
<td>195,428</td>
<td>86,573</td>
<td>805</td>
<td>2-146</td>
</tr>
<tr>
<td>C3</td>
<td>0x34a5451ef61a567ee088...</td>
<td>18</td>
<td>9</td>
<td>1</td>
<td>46</td>
</tr>
<tr>
<td>C4</td>
<td>0x304a554a310c7e546dfe...</td>
<td>14,150</td>
<td>3,064</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>C10</td>
<td>0xf4c64518ea10f995918a...</td>
<td>428</td>
<td>177</td>
<td>11</td>
<td>42-122</td>
</tr>
<tr>
<td>C11</td>
<td>0xb136707642a4ea12fb4b...</td>
<td>2,582</td>
<td>305</td>
<td>201</td>
<td>17-20</td>
</tr>
<tr>
<td>C12</td>
<td>0xe0da70933f4c7849fc0...</td>
<td>5,330</td>
<td>3,992</td>
<td>1,259</td>
<td>12-57</td>
</tr>
</tbody>
</table>
**non-ECF Contracts**

Object C
```
Object Sender
Method call(data, sender)
    if (Sender != nil) throw
    Sender = sender; ret = this.do(data); Sender = nil

Method do(data)
    ... // read Sender
```

Valid Ambisafe contracts with measures to prevent DAO-like vulnerability. This programming pattern is inherently non-ECF.
Related Tools

• Oyente: Solidity verification tool

• Porosity: EVM bytecode decompiler
Conclusion

• In principal, automatic verification requires reasoning about the whole code

• We explored modularity, and..

• Obtained a general result that applies to objects in other programming languages
Any Questions?