
Asem Ghaleb and Karthik Pattabiraman

July 22nd, 2020
Smart contracts
Motivation: Smart contracts

- Cannot be updated
- Transactions are immutable
- Financial nature (incentive for attackers)

(2016) The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

(2017) Yes, this kid really just deleted $300 MILLION by messing around with Ethereum’s smart contracts

(2019) Ethereum Classic's '51% Attack,' $1 Million Loss, Raise Concerns About Security
Our goal

- Code vulnerabilities are still reported frequently [1]
- No evaluation methodology of static analyzers

A systematic approach for evaluating efficacy of smart contract static analysis tools on detecting bugs

Contributions

- Systematic approach: SolidiFI
- Evaluated 6 static analyzers
- Analysis of the analyzers’ false negatives and false positives

Tools failed to detect several bugs and reported high false positives
Research challenges

- Solidity; different from traditional languages
- Injecting bugs into all potential locations
- Injecting exploitable vulnerabilities
Bug model

- Code snippets which lead to vulnerabilities
- Injecting bugs claimed to be detected
- Playing the role of developers rather attackers
- Injecting distinct bugs as possible

```c
if (startTime+5 == block.timestamp)
    { //code }

uint _vtime = block.timestamp;
if (startTime+5 == _vtime)
    { //code }
```
Bug injection

```solidity
contract MyWallet {

    address owner;
    mapping(address => uint256) balances;

    constructor () public {
        owner = msg.sender;
    }

    function sendTo(address payable receiver, uint8 amount) public {
        require(msg.sender == owner);
        (bool success) = receiver.send(amount);
        if (!success)
            // revert();
    }

    function bug_reEntrancy ( uint256 _Amt ) public {
        require(balances[msg.sender] >= _Amt);
        (bool success,) = msg.sender.call.value(_Amt)("");
        require(success);
        balances[msg.sender] -= _Amt;
    }
}
```

SolidiFI works on AST-level of the source code

Code transformation

Security weakening

code snippet injection

6 static analysis tools

(Oyente, Securify, Mythril, Smartcheck, Manticore, Slither)

50 Smart Contracts representative of Etherscan (39-741 loc) ~ Most Etherscan contracts size <1000 loc

Different functionalities and syntactic elements

RQ1: False negatives of the evaluated tools?
RQ2: False positives of the evaluated tools?
RQ3: Injected bugs can be activated?
Experimental setup

- 7 common bug classes considered by the tools
- 9,369 distinct bugs
- Timeout: 15 minutes per smart contract

<table>
<thead>
<tr>
<th>Bug Type</th>
<th>Oyente</th>
<th>Securify</th>
<th>Mythril</th>
<th>SmartCheck</th>
<th>Manticore</th>
<th>Slither</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-entrancy</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Timestamp dependency</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Unchecked send</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unhandled exceptions</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>TOD</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer over/underflow</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Use of tx.origin</td>
<td></td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
RQ1: False negatives of the evaluated tools

<table>
<thead>
<tr>
<th>Security bug</th>
<th>Injected bugs</th>
<th>Oyente</th>
<th>Security</th>
<th>Mytril</th>
<th>SmartCheck</th>
<th>Manticore</th>
<th>Sither</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-entrancy</td>
<td>1343</td>
<td>1008</td>
<td>232</td>
<td>1085</td>
<td>1343</td>
<td>1250</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(844)</td>
<td>(232)</td>
<td>(805)</td>
<td>(106)</td>
<td>(1108)</td>
<td></td>
</tr>
<tr>
<td>Timestamp dependency</td>
<td>1381</td>
<td>1381</td>
<td>NA</td>
<td>810</td>
<td>902</td>
<td>NA</td>
<td>537</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(886)</td>
<td>(810)</td>
<td>(810)</td>
<td>(341)</td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>Unchecked send</td>
<td>1266</td>
<td>NA</td>
<td>499</td>
<td>389</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(499)</td>
<td>(389)</td>
<td>(389)</td>
<td></td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>Unhandled exceptions</td>
<td>1374</td>
<td>1052</td>
<td>673</td>
<td>756</td>
<td>1325</td>
<td>NA</td>
<td>457</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(918)</td>
<td>(571)</td>
<td>(756)</td>
<td>(1170)</td>
<td>(128)</td>
<td></td>
</tr>
<tr>
<td>TOD</td>
<td>1336</td>
<td>1199</td>
<td>263</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1199)</td>
<td>(263)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integer over/underflow</td>
<td>1333</td>
<td>898</td>
<td>NA</td>
<td>1069</td>
<td>1072</td>
<td>1196</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(898)</td>
<td>(932)</td>
<td>(1072)</td>
<td>(1127)</td>
<td>(1127)</td>
<td></td>
</tr>
<tr>
<td>Use of tx.origin</td>
<td>1336</td>
<td>NA</td>
<td>NA</td>
<td>445</td>
<td>1239</td>
<td>NA</td>
<td>✔</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(445)</td>
<td>(1120)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- None of the tools detect all bugs
- Many undetected corner cases
- Misidentification is high as well
Misidentification of bugs: Example

Oyente Scan report

Injected Reentrancy bug

Buggy contract

Reported as TOD bug
RQ2: False positives of the evaluated tools

Challenges:
- Lack of ground truth
- Large number of bugs

Approach:

Assuming a bug reported by the majority of the tools cannot be false positive

<table>
<thead>
<tr>
<th>Reported Reentrancy</th>
<th>100</th>
<th>Filtered</th>
<th>40</th>
<th>Manually inspected</th>
<th>20</th>
<th>Indeed FPs</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reported by majority</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FPs = Filtered X Indeed FPs</td>
<td></td>
</tr>
</tbody>
</table>

FPs = 40 × 80% = 32

Risk: There might be false positives reported by the majority
False positive results

- All tools reported false positives (2 to 801)
- High false positives for tools with low false negatives (e.g., Slither)
- Some cases are truly bizarre

```java
String public symbol = "CRE";
```
RQ3: Activating the undetected bugs

Goal: Checking exploitability of the undetected bugs

- Selected 5 undetected bugs for each bug type
- All bugs were exploitable
- No much effort to exploit bugs (within minutes)
Threats to validity

● External:
  ● 50 smart contracts

● Internal:
  ● Evaluating 6 tools
  ● 7 bug types

● Results measurement:
  ● Unexploitable bugs in practice
  ● True bugs counted as false positives
Summary

Goal: A systematic approach for evaluating static analyzers

- Introduced SolidiFI, for evaluating smart contract static analyzers
- Static analyzers suffer high false-negatives and false-positives
- Analyzers that detect bugs with low false positives are needed

Source code: https://github.com/DependableSystemsLab/SolidiFI
Artifact: https://github.com/DependableSystemsLab/SolidiFI-benchmark

Asem Ghaleb, PhD Candidate at University of British Columbia
aghaleb@ece.ubc.ca