ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection

Authors: Bo Jiang*, Ye Liu*, and W.K. Chan†
*Beihang University
†City University of Hong Kong

Bo Jiang: gongbell@gmail.com
The Rise of Blockchain and Cryptocurrencies
The Rapid Growth of Ethereum Ecosystem

source: https://ethereum.stackexchange.com/questions/383/what-is-a-dapp
## Smart Contracts with Verified Source Code

### Etherscan

#### Contracts With Verified Source Codes Only

Sponsored: **Gravity Presale is Live** - Exclusive 30% Bonus until August 31. **Don't miss out!**

A Total Of 42051 verified contract source codes found

<table>
<thead>
<tr>
<th>Address</th>
<th>ContractName</th>
<th>Compiler</th>
<th>Balance</th>
<th>TxCount</th>
<th>Settings</th>
<th>DateVerified</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x9ce8b0ebb4f45b2...</td>
<td>EtheremonWorldNFT</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>1</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0x70d2b63ef6b27a...</td>
<td>ZCCToken</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>1</td>
<td>-</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0xEe7A44343dc25...</td>
<td>FILAToken</td>
<td>v0.4.21</td>
<td>0 Ether</td>
<td>3</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0x74bb7e3bdfee4...</td>
<td>FORZE</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>1</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0xe8e8c2c4b65b7c...</td>
<td>BurnableERC20Token</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>3</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0xfec8c5187aee4c2...</td>
<td>Sale</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>2</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0x5eb4b436104680...</td>
<td>YumeriumManager</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>2</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0x821285d692b585...</td>
<td>Morze</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>5</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0x05a766bc23d3c...</td>
<td>TCT</td>
<td>v0.4.12</td>
<td>0 Ether</td>
<td>30</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
<tr>
<td>0x8b807547d78e6...</td>
<td>CMR_Mining</td>
<td>v0.4.24</td>
<td>0 Ether</td>
<td>1</td>
<td>⚠️</td>
<td>9/1/2018</td>
</tr>
</tbody>
</table>
Dapp = Frontend + Smart Contracts

source: https://ethereum.stackexchange.com/questions/383/what-is-a-dapp
Attacks on Smart Contracts

- Managing so much wealth with code will surely attract hackers...

- **Attack on “The DAO bug”**
  - $60 million US loss

- **First round attack of parity wallet bug**
  - $60 million US loss

- **Second round attack of parity wallet bug**
  - $150 million frozen in terms of Ether

- **Attack on Integer Overflow Vulnerability**
  - BEC coin on Ethereum withdrawn by attackers
  - Market value cleared
The Dangerous Delegate Call in Parity Wallet Contract

```solidity
contract Wallet {
    function() payable { // fallback function
        if (msg.value > 0)
            Deposit(msg.sender, msg.value);
        else if (msg.data.length > 0)
            _walletLibrary.delegatecall(msg.data);
    }
}

contract WalletLibrary {
    function initWallet(address[] _owners, uint _required, uint _daylimit) {
        initDaylimit(_daylimit);
        initMultiowned(_owners, _required);
    }
}
```
Contents

- Motivation
- Typical Smart Contract Vulnerabilities
- The Design of ContractFuzzer Tool
- Experimental Evaluation
- Related Work
- Conclusion
<table>
<thead>
<tr>
<th>Vulnerability Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gasless Send</td>
<td>send() triggers out of gas exception due to expensive fallback &amp; sender keep ether wrongfully</td>
</tr>
<tr>
<td>Exception Disorder</td>
<td>Inconsistent error propagation of low-level calls</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>Nonreentrant function invoked in reentrant manner</td>
</tr>
<tr>
<td>Timestamp Dependency</td>
<td>Relying timestamp to decide ether transfer</td>
</tr>
<tr>
<td>Block Number Dependency</td>
<td>Relying Block Number to decide ether transfer</td>
</tr>
<tr>
<td>Dangerous DelegateCall</td>
<td>The argument of delegatecall can be provide by msg.data</td>
</tr>
<tr>
<td>Freezing Ether</td>
<td>Smart contracts can receive ether but cannot send ether except through delegatecall</td>
</tr>
</tbody>
</table>
Contents

- Motivation
- Typical Smart Contract Vulnerabilities
- The Design of ContractFuzzer Tool
- Experimental Evaluation
- Related Work
- Conclusion
Overview of ContractFuzzer

0. Offline EVM Instrumentation

contracts pool crawled

2. ABI signature analysis

contracts indexed by signature

3. Input Generation

Fuzzing Inputs

1. Static Analysis

ABI and bytecode of contract under test

ABI arguments & signatures of functions used by contract

4. Smart Contracts Fuzzing

Execution Log

5. Vulnerability Analysis

Fuzzing Results

Testnet with instrumented EVMs and deployed smart contracts

Open Sourced: https://github.com/gongbell/ContractFuzzer
Overview of ContractFuzzer

1. Offline EVM Instrumentation
   - contracts pool crawled
   - contracts indexed by signature

2. ABI signature analysis
   - ABI signature analysis
   - contracts indexed by signature

3. Input Generation
   - ABI and bytecode of contract under test
   - ABI arguments & signatures of functions used by contract

4. Smart Contracts Fuzzing
   - Execution Log
   - Fuzzing Results

5. Vulnerability Analysis
   - Testnet with instrumented EVMs and deployed smart contracts

Open Sourced: https://github.com/gongbell/ContractFuzzer
Overview of ContractFuzzer

0. Offline EVM Instrumentation

1. Static Analysis
   - ABI and bytecode of contract under test
   - ABI arguments & signatures of functions used by contract
   - Testnet with instrumented EVMs and deployed smart contracts

2. ABI signature analysis
   - contracts indexed by signature

3. Input Generation
   - Fuzzing Inputs
   - ABI arguments & signatures of functions used by contract

4. Smart Contracts Fuzzing

5. Vulnerability Analysis
   - Execution Log
   - Fuzzing Results

Open Sourced: https://github.com/gongbell/ContractFuzzer
Input Generation for ABI Signature

1. ABI/Bytecode Analysis
2. Parameter Generation for Basic Types: int, uint, bytes, fixed array, non-fixed-size inputs
3. Parameter Generation for Address Type
How to generate inputs for address type?

1. Build a map with the contract pool: `<function selector, contract address>`
2. Analyze the bytecode of each ABI function of the smart contract, extract function selectors used in call()
3. Build a private candidate smart contract pool for each ABI function
Input Generation for Reentrancy Vulnerability

```solidity
contract BountyHunt{
    ...
    function claimBounty() preventTheft { 
        uint balance = bountyAmount[msg.sender];
        if (msg.sender.call.value(balance)()) {
            totalBountyAmount -= balance;
            bountyAmount[msg.sender] = 0;
        }
    }
}

contract AttackerAgent{
    ...
    function AgentCall(address contract_addr, bytes msg_data) {
        call_contract_addr = contract_addr;
        call_msg_data = msg_data;
        contract_addr.call(msg_data);
    }
    function() payable {
        call_contract_addr.call(call_msg_data);
    }
}
```
## Test Oracle Design and EVM Instrumentation

<table>
<thead>
<tr>
<th>Vulnerability Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gasless Send</td>
<td><code>send()</code> call returns <code>ErrOutOfGas</code> during execution</td>
</tr>
<tr>
<td>Exception Disorder</td>
<td>For a chain of nested calls: the root call doesn’t throw exception while at least one of its nested calls throws exception</td>
</tr>
<tr>
<td>Reentrancy</td>
<td><code>ReentrancyCall ∧ CallAgentWithValue</code></td>
</tr>
<tr>
<td>Timestamp Dependency</td>
<td><code>TimestampOp ∧ (SendCall ∨ EtherTransfer)</code></td>
</tr>
<tr>
<td>Block Number Dependency</td>
<td><code>BlockNumOp ∧ (SendCall ∨ EtherTransfer)</code></td>
</tr>
<tr>
<td>Dangerous DelegateCall</td>
<td><code>delegateCall</code> on function extracted from input</td>
</tr>
<tr>
<td>Freezing Ether</td>
<td><code>balance &gt; 0 ∧ no transfer/send/call/suicide</code></td>
</tr>
</tbody>
</table>
Motivation

Typical Smart Contract Vulnerabilities

The Design of ContractFuzzer Tool

Experimental Evaluation

Related Work

Conclusion
Experiment Setup

- **Subject Programs**: 6991 out of 9960 smart contracts
- **Environment**: Two dockers, one for contract fuzzer tool and one for testnet with geth client version 1.7.0

**Procedure**:

1. generate $6k$ calls for each ABI function
   - 3 types of account: creator account, external account, `AttackAgent`
   - 2 `call.value()`: with & without value transfer
2. merge all calls into a pool for a smart contract
3. Randomly select calls from pool for fuzzing
4. Analyzing and report results
### Summary of Vulnerabilities Detected

<table>
<thead>
<tr>
<th>Vulnerability Type</th>
<th>Number of Vulnerabilities</th>
<th>Percentage</th>
<th>True Positive Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gasless Send</td>
<td>138</td>
<td>2.06%</td>
<td>100%</td>
</tr>
<tr>
<td>Exception Disorder</td>
<td>36</td>
<td>0.54%</td>
<td>100%</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>14</td>
<td>0.21%</td>
<td>100%</td>
</tr>
<tr>
<td>Timestamp Dependency</td>
<td>152</td>
<td>2.27%</td>
<td>96.05%</td>
</tr>
<tr>
<td>Block Number Dependency</td>
<td>82</td>
<td>1.23%</td>
<td>96.34%</td>
</tr>
<tr>
<td>Dangerous Delegatecall</td>
<td>7</td>
<td>0.10%</td>
<td>100%</td>
</tr>
<tr>
<td>Freezing Ether</td>
<td>30</td>
<td>0.45%</td>
<td>100%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>459</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Comparison of ContractFuzzer and Oyente

<table>
<thead>
<tr>
<th>Vulnerability Type</th>
<th>ContractFuzzer</th>
<th>Oyente</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No.</td>
<td>FP</td>
</tr>
<tr>
<td>Statistics</td>
<td>152</td>
<td>6</td>
</tr>
<tr>
<td>Timestamp Dependency</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>
A False Negative Case of ContractFuzzer

```
|   | contract BDSM_Crowdsale {
|---|-------------------------------
| 2 | ...                          |
| 3 | uint public startICO_20_December = 1513728060; //2017.12.20 |
| 4 | uint public stopICO_20_March = 1521504060; //2018.3.20 |
| 5 | function () payable {       |
| 6 |     if (now < startICO_20_December) { |
| 7 |         msg.sender.transfer(msg.value); |
| 8 |     }                           |
| 9 | ...                          |
```
A False Negative Case of Oyente

```
function buy(uint8 _bomb) public payable {
    ...
    int _random =
        uint(keccak256(block.blockhash(block.number - 1), msg.gas, tx.gasprice, block.timestamp)) % bomb.chance + 1;
    if (_random == 1){
        bomb.owner.transfer(...)
        ceoAddress.transfer(...)
    }
}
```

Cryptographic functions hard for SMT solver
Case Studies-ICO Wrongfully Holding Ether

```
1 contract CrowdSalePreICO {
2     ...
3     function() payable stopInEmergency onlyStarted notFinalized{
4         ...
5         uint contribution = msg.value;
6         if (safeAdd(totalDepositedEthers, msg.value) > hardCapAmount)
7             contribution = safeSub(hardCapAmount, totalDepositedEthers);
8     }
9     uint excess = safeSub(msg.value, contribution);
10    if (excess > 0){
11        msg.sender.send(excess);
12    }\)
```

Send may fail, error is hidden due to exception disorder, but send does not check for error.
Case Studies – Timestamp Dependency

```solidity
contract SlotMachine {
    function() {
        uint nr = now; // now is the block timestamp
        uint y = nr & 3;
        if(y==1) { wins[1]++; win = (msg.value * 2) + (msg.value / 2); }
        earnings += int(msg.value);
        if(win > 0) {
            bool res = msg.sender.send(win);
            earnings -= int(win);
        }
    }
}
```
Case Studies – The Wallet Smart Contracts Frozen by Parity Bug

```
1  contract Wallet is WalletEvents {
2      ...
3      function() payable {
4          if (msg.value > 0)
5              Deposit(msg.sender, msg.value);
6          else if (msg.data.length > 0)
7              _walletLibrary.delegatecall(msg.data);
8      }
9      function hasConfirmed(bytes32 _operation, address _owner)...
10         return _walletLibrary.delegatecall(msg.data);
11      }
12      address constant _walletLibrary = 0xcafecafecafecafecafecafecafecafecafecafecafe;
13  }
```
Contents

- Motivation
- Typical Smart Contract Vulnerabilities
- The Design of ContractFuzzer Tool
- Experimental Evaluation
- Related Work
- Conclusion
Related Work

Oyente
- Luu et al. @ ACM CCS, 2016.
- Symbolic execution based verification

MAIAN
- Nikolic et al. @ arxiv.org
- Symbolic execution tool on execution traces

Echidna Fuzzer
- https://github.com/trailofbits/echidna
- Fuzzing based on Unit testing framework
- No support for security testing of smart contracts
Contents

- Motivation
- Typical Smart Contract Vulnerabilities
- The Design of ContractFuzzer Tool
- Experimental Evaluation
- Related Work
- Conclusion
Conclusion

In this work, we proposed ContractFuzzer, the first comprehensive fuzzing framework to detect smart contract vulnerabilities.

We proposes the test oracles for detecting 7 typical smart contract security vulnerabilities.

Our experiment with 6991 real world smart contracts shows ContractFuzzer are effective to detect security vulnerabilities with very high precision.

In total, 459 vulnerabilities detected during Fuzzing. Open Sourced: https://github.com/gongbell/ContractFuzzer
Future Work

- Explore more effective input generation strategies
- Evaluate the tool in terms of coverage
- Reduce false negatives
Thank you!
Thank you!
Thank you!