EthPloit: From Fuzzing to Efficient Exploit Generation against Smart Contracts

Qingzhao Zhang\textsuperscript{1,2}, Yizhuo Wang\textsuperscript{1}, Juanru Li\textsuperscript{1}, Siqi Ma\textsuperscript{3}

\textsuperscript{1}Shanghai Jiao Tong University, China
\textsuperscript{2}University of Michigan, America
\textsuperscript{3}Data 61, CSIRO, Australia

SANER'20, London ON. Canada, February 21, 2020
1 Background

2 Motivation

3 EthPloit Fuzzer

4 Evaluation

5 Conclusion
Overview of Ethereum

Ethereum is the second-largest blockchain system

In General

- A programmable blockchain
- A platform for decentralized applications.

In Detail

- A transaction-based state machine
- The heart is Ethereum Virtual Machine (EVM)
- Based on Turing-complete programming language (Solidity)
Overview of Ethereum

Ethereum is the second-largest blockchain system

In General

- A programmable blockchain
- A platform for decentralized applications.

In Detail

- A transaction-based state machine
- The heart is Ethereum Virtual Machine (EVM)
- Based on Turing-complete programming language (Solidity)
Smart Contract

Contract Code

- Source code written in Solidity
- Compiled by Solc to get bytecode
- Bytecode run on EVM

Contract Action

- Created by External Owned Account
- Executed on incoming transactions
Transaction

**Basic Fields**

- **From:** Sender's Address
- **To:** Receiver's Address
- **Value:** Amount of Currency
- **Data:** Various situations
  - Empty (just transfer currency)
  - Init code of contract
  - **Called function with arguments**

**Simulate a scene**

- Call a function of contract
- Run the code

**Result**

- Change the balance
- Update the storage
  - State variable
Exploitation of Smart Contract

What is the exploitation

- From attacker to target contract
- A sequence of transactions

Categories of exploitation

According to the cause of damages:

- Balance Increment
- Self-destruction
- Code Injection

![Diagram of exploit process]
Exploitable Vulnerabilities

Unchecked Transfer Value

- Misuse of `this.balance`
- Unlimited profit

Vulnerable Access Control

- Missing & misuse of check
  - before sensitive operation

Exposed Secret

- Newly identified vulnerability
- Previous tools cannot exploit

```solidity
function withdraw() notOnPause public {
    if (block.timestamp >= x.c(msg.sender) + 10 minutes) {
        uint _payout = (x.d(msg.sender) . mul(x.getInterest(msg.sender)) . div(10000)) . mul(
            block.timestamp . sub(x.c(msg.sender)) . div(1 days);
        x.updateCheckpoint(msg.sender);
    }
    if (_payout > 0)
        msg.sender.transfer(_payout);
}
```
Exploitable Vulnerabilities

Unchecked Transfer Value
- Misuse of `this.balance`
- Unlimited profit

Vulnerable Access Control
- Missing & misuse of check
  - before sensitive operation

Exposed Secret
- Newly identified vulnerability
- Previous tools cannot exploit

```
contract Game {
    address questionSender;
    string public question;
    bytes32 responseHash;

    function Try(string _response) external payable{
        require(msg.sender == tx.origin);
        if(responseHash == keccak256(_response) &
            msg.value>1 ether){
            msg.sender.transfer(this.balance);
        }
    }

    function StartGame(string _question,string _response) public payable {
        if(responseHash==0x0){
            responseHash = keccak256(_response);
            question = _question;
            questionSender = msg.sender;
        }
    }
}
```
Exploitable Vulnerabilities

Unchecked Transfer Value

- Misuse of `this.balance`
- Unlimited profit

Vulnerable Access Control

- Missing & misuse of check
  - before sensitive operation

Exposed Secret

- Newly identified vulnerability
- Previous tools cannot exploit

---

**Attackers inspect the secret from the data of previous transactions**

**Attackers break the secret checking to gain profit**

```solidity
contract Game {
    address questionSender;
    string public question;
    bytes32 responseHash;

    function Game() public {
        questionSender = msg.sender;
        question = _question;
        responseHash = keccak256(_response);  
        msg.sender.transfer(this.balance);  
    }
```
Goal of the Work

Unchecked Transfer Value
Vulnerable Access Control
Exposed Secret

Vulnerabilities Detected

Vulnerabilities Exploited

Fuzzing ➔ Efficient Exploit Generation
Challenges of Exploit Generation

Challenge-1: Unsolvable Constraint

< Situation in smart contract >

Condition restricting sensitive operations
- Involve complicated operation like hash

```
function Try(string _response) external payable{
    require(msg.sender == tx.origin);
    if (responseHash == keccak256(_response) && msg.value > 1 ether){
        msg.sender.transfer(this.balance);
    }
}
```

< Previous solution >

Previous tools (e.g., Teether, Mythril) rely on SMT solver
- Cannot solve cryptographic constraint
- Ignore the runtime value
  - not stored in contract state
Challenges of Exploit Generation

Challenge-2: Blockchain Effects

Blockchain effects of blockchain system affect the execution of smart contracts

- E.g., blockchain properties

```solidity
function withdraw() notOnPause public {
    if (block.timestamp >= x.c(msg.sender) + 10 minutes) {
        uint _payout = (x.d(msg.sender).mul(x.getInterest(
            msg.sender)).div(10000)).mul(block.timestamp,
            sub(x.c(msg.sender))).div(365*24*60*60);
        x.updateCheckpoint(msg.sender);
    }
    if (_payout > 0)
        msg.sender.transfer(_payout);
}
```

Previous tools have difficulties on manipulating blockchain effect:

- Lack of considering the syntax of blockchain properties
  e.g., invalid timestamp
- Ignore the possibility of call reverting, thus lose coverage
  e.g., Teether, ContractFuzzer
Our Solution

Fuzzing

EthPloit: a smart contract specific fuzzer

Feedback of runtime value

Record the runtime values of arguments and variables
- Create a blank seed set
- Update the seed set
- Use for the next generation

Indicated information:
- Execution history
  - e.g., the hash image
- State of the contract
  - i.e., the state variable

Manipulation of blockchain execution

By instrumenting the execution environment
Contents

1 Background

2 Motivation

3 EthPloit Fuzzer

4 Evaluation

5 Conclusion
Workflow of EthPloit

1. Dependencies
2. ABI
3. Taint Analyzer
4. Solidity Compiler
5. Bytecode

Test Case Generator:
- Taint Relation Graph
- Function Selection
- Argument Generation
- Property Generation

Instrumented EVM:
- Block Property Configuration
- Revert-call Configuration

Exploit Detector:
- Critical Instruction Coverage
- Feedback Construction

Coverage Guider:
- Function Reward
- Valuable Seeds

Feedback Handler:
- Function Distribution
- Dynamic Seed Set

Fuzzing Iteration
Taint Analyzer

**Knowledge of dependencies** of modifying contract state improves *fuzzing efficiency*

**EthPloit** applies *static taint analysis* to discover dependencies of modifying contract states:
- Generate control flow graph
- Label taint sources and sinks
- Perform taint propagation

**Extract variable-level dependencies**
- **Variable-Data** Dependency
- **Variable-Control** Dependency

```solidity
function test(uint value) public {
    require(balance[msg.sender] >= value);
    balance[msg.sender] -= value;
    msg.sender.transfer(value);
}
```
1 Taint Analyzer

Knowledge of dependencies of modifying contract state improves fuzzing efficiency

*EthPloit* applies **static taint analysis** to discover dependencies of modifying contract states:
- Generate control flow graph
- Label taint sources and sinks
- Perform taint propagation

Extract variable-level dependencies
- **Variable-Data** Dependency
- **Variable-Control** Dependency

**Variable-Data**
- `msg.sender taint balance`
- `msg.sender value taint balance`

**Variable-Control**
- `msg.sender ctrl transfer`
- `msg.sender value ctrl transfer`
- `msg.sender balance ctrl transfer`
Test Case Generator

Optimize the test case by analyzing how inputs affect the execution of exploits

Taint Relation Graph

Function Selection

- Add suitable functions into a set of candidates
- Select function from candidates based on probability distribution

Arguments Generation

- From pseudo-random generator
- From dynamic seed set

Blockchain Properties Generation

Based on Instrumented EVM Environment
3 Instrumented EVM Environment

**EthPloit environment**

- Based on remix-debugger
- Deploy contract
- Execute transaction
- **Extract full execution trace**

Compared to private Ethereum chain

- More light-weight
- More flexible for configure

**Three instrumentations**

- **Configure accounts**
  - For each test case

- **Configure block properties**
  - For each execution of transaction

- **Force external calls to revert**
  - For each external call
  - Revert the 2nd execution of call
4 Trace Analyzers

**Coverage Guider**

- Measure the progress of exploit-oriented fuzzing
- Construct feedback as rewards

**Critical instruction coverage**

**Feedback construction**
- Seed feedback
- Function distribution feedback

\[ P(f) = c_0 + \frac{N_c}{N_t} (c_1 - c_0) \]

**Exploit Detector**

**Balance Increment oracle**
- If attackers’ balance is *increased*

**Self-Destruction oracle**
- If the opcode **SELFDESTRUCTION** is found

**Code Injection oracle**
- If opcodes **CALLCODE, DELEGATECALL** are found
- If destination is controlled by attackers

Workflow
Feedback Handler

Dynamic Seed Strategy

Aim to **guide** the test case generator to produce proper function arguments

For the whole process of fuzzing
- Perform more mutation based on interesting cases
- Select **global seeds** which have a lifetime during fuzzing one contract
- All arguments of interesting cases causing coverage increment

For each test case
- Make use of connections among transactions
- Select **local seeds** after each execution of transaction:
  - Previous arguments
  - State variables
  - I/O of complicated calls
  - Constant values
Workflow of EthPloit

1. Taint Analyzer
2. Test Case Generator
   - Taint Relation Graph
   - Function Selection
   - Argument Generation
   - Property Generation
3. Instrumented EVM
   - Block Property Configuration
   - Revert-call Configuration
4. Exploit Detector
   - Critical Instruction Coverage
   - Feedback Construction
5. Coverage Guider
   - Feedback Handler
     - Function Reward
     - Valuable Seeds
   - Dynamic Seed Set
Environment

Dataset
Totally 45,308 contracts

Environment
Two 3.60GHz Xeon CPUs with 128GB RAM

Fuzzing Configuration
▪ Maximum test cases as 1,000
▪ Maximum length as 3 for each case

Comparison
Teether[1] and MAIAN[2] with a timeout of 5 minutes


Evaluation of Contract Exploit

EthPloit
- Totally generated 644 exploits
- No false positive, verified using real-world EVM
- 600 Balance Increment, 59 Self-destruction, 4 Code Injection

Teether / MAIAN
- unable to analyze 5,123 contracts and 102 contracts
- Teether generated 14 false positive
- MAIAN cannot exploit lots of vulnerable contracts
Evaluation of Contract Exploit

Summary of exploits generated based on triggered vulnerabilities

<table>
<thead>
<tr>
<th>Tools</th>
<th>Exposed Secret</th>
<th></th>
<th>Unchecked Transfer Value</th>
<th></th>
<th>Bad Access Control</th>
<th>Others</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cryptographic Checks</td>
<td>Others</td>
<td>Total</td>
<td>Unlimited Profit</td>
<td>Misused this.balance</td>
<td>Others</td>
<td>Total</td>
</tr>
<tr>
<td>ETHPLOIT</td>
<td>104</td>
<td>8</td>
<td>112</td>
<td>144</td>
<td>181</td>
<td>26</td>
<td>351</td>
</tr>
<tr>
<td>teether</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>25</td>
<td>6</td>
<td>61</td>
</tr>
<tr>
<td>MAIAN</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>31</td>
<td>143</td>
<td>16</td>
<td>190</td>
</tr>
</tbody>
</table>

EthPloit

- For Exposed Secret, **104** out of **112** exploits have **cryptographic checks** in the execution path
- For Unchecked Transfer Value, **144** out of **351** exploits are caused by **Unlimited Profit**

Comparison

- EthPloit has huge advantage over teether and MAIAN
  - Especially in exploiting **Exposed Secret** and **Unchecked Transfer Value**
Evaluation of Contract Exploit

Summary of exploits generated based on two typical vulnerabilities

<table>
<thead>
<tr>
<th>Tools</th>
<th>Cryptographic Checks</th>
<th>Unlimited Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>EthPloit</td>
<td>104</td>
<td>144</td>
</tr>
<tr>
<td>Teether</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>MAIAN</td>
<td>0</td>
<td>31</td>
</tr>
</tbody>
</table>

Dynamic Seed Strategy:
- Fetch secret value
- Solve hash checks

Instrumented EVM Environment:
- Simulate block properties
- Exploit lottery games
  - Block properties as random seed
### Impact of Vulnerabilities Identified

#### Information of typical contracts exploited by EthPloit

<table>
<thead>
<tr>
<th>Contract</th>
<th>Address</th>
<th>#Tx</th>
<th>Highest Balance</th>
<th>Vulnerability</th>
<th>Exploit results</th>
<th>Number of Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>TestR</td>
<td>0xaf53...</td>
<td>6</td>
<td>0.5 ETH, $269.2</td>
<td>Exposed Secret</td>
<td>×/√</td>
<td>Normal: 13.0, No EVM: 18.1, No Seeds: -, No Taint: 8.9</td>
</tr>
<tr>
<td>BLITZ_GAME</td>
<td>0x35b5...</td>
<td>4</td>
<td>6.0 ETH, $572.6</td>
<td>Exposed Secret</td>
<td>×/×</td>
<td>49.6, No EVM: 50.0, No Seeds: -, No Taint: 169.0</td>
</tr>
<tr>
<td>Who_Wants...</td>
<td>0xc6e2...</td>
<td>10</td>
<td>4.0 ETH, $546.3</td>
<td>Exposed Secret</td>
<td>×/×</td>
<td>46.2, No EVM: 28.0, No Seeds: -, No Taint: 61.5</td>
</tr>
<tr>
<td>Game</td>
<td>0xe37b...</td>
<td>6</td>
<td>3.0 ETH, $445.9</td>
<td>Exposed Secret</td>
<td>×/×</td>
<td>50.2, No EVM: 37.8, No Seeds: -, No Taint: 65.5</td>
</tr>
<tr>
<td>GPUMining</td>
<td>0xa965...</td>
<td>346</td>
<td>1.2 ETH, $712.3</td>
<td>Unchecked Transfer Value</td>
<td>×/×</td>
<td>188.1, No EVM: 660.6, No Seeds: 319.7, No Taint: 332.9</td>
</tr>
<tr>
<td>HRKD</td>
<td>0xa070...</td>
<td>307</td>
<td>50.1 ETH, $11k</td>
<td>Unchecked Transfer Value</td>
<td>×/×</td>
<td>48.4, No EVM: -, No Seeds: 29.2, No Taint: 20.1</td>
</tr>
<tr>
<td>Slotthereum</td>
<td>0xb43b...</td>
<td>76</td>
<td>0.4 ETH, $92.4</td>
<td>Unchecked Transfer Value</td>
<td>×/×</td>
<td>52.9, No EVM: 87.4, No Seeds: 214.6, No Taint: 57.2</td>
</tr>
<tr>
<td>Divs4D</td>
<td>0x3983...</td>
<td>161</td>
<td>4.1 ETH, $905.3</td>
<td>Unchecked Transfer Value</td>
<td>×/×</td>
<td>10.7, No EVM: -, No Seeds: 18.9, No Taint: 29.1</td>
</tr>
<tr>
<td>DailyRoi</td>
<td>0x77e4...</td>
<td>4,488</td>
<td>397.1 ETH, $87k</td>
<td>Unchecked Transfer Value</td>
<td>×/×</td>
<td>11.6, No EVM: -, No Seeds: 10.3, No Taint: 10.7</td>
</tr>
<tr>
<td>Dividend</td>
<td>0xe3ac...</td>
<td>47</td>
<td>140.5 ETH, $66k</td>
<td>Unchecked Transfer Value</td>
<td>×/√</td>
<td>134.7, No EVM: 47.8, No Seeds: -, No Taint: 333.3</td>
</tr>
<tr>
<td>HOTTO</td>
<td>0x612f...</td>
<td>132</td>
<td>1.1 ETH, $320.1</td>
<td>Bad Access Control</td>
<td>×/√</td>
<td>18.2, No EVM: 23.8, No Seeds: -, No Taint: 15.3</td>
</tr>
<tr>
<td>Crypto...Network</td>
<td>0x781f...</td>
<td>52K</td>
<td>1.3 ETH, $541.8</td>
<td>Bad Access Control</td>
<td>×/√</td>
<td>28.8, No EVM: 40.0, No Seeds: 21.4, No Taint: 89.7</td>
</tr>
</tbody>
</table>

**Exposed Secret exploited in total:** 32 contract, lost **37.3 ETH**, about **$6,485**

**Unchecked Transfer Value & Vulnerable Access Control affect lots of widely used contracts, e.g., DailyRoi:**

4,888 transactions, maximum balance of **397.1 ETH ($87k)**
Evaluation of Core Techniques

Benchmarks: newly discovered 554 exploitable contracts

Four different configuration of EthPloit:
- 1. Without EVM instrumentation
- 2. Without dynamic seed strategy
- 3. Without taint constraints
- Baseline: All techniques are enabled

Benchmark is tested for 10 times under each configuration, respectively
Evaluation of Core Techniques

Number of generated exploits under various configuration

- **Without dynamic seed strategy**: EthPloit misses **most** Exposed Secret
- **Without EVM instrumentation**: EthPloit misses **69** Unchecked Transfer Value
Evaluation of Core Techniques

- Use the number of test cases to represent fuzzing efficiency
- The overall fuzzing efficiency is damaged when taint analysis is removed
- With taint constraints, over 90% exploits can be found in 100 test cases
Contents

1 Background

2 Motivation

3 EthPloit Fuzzer

4 Evaluation

5 Conclusion
Conclusion

**Design EthPloit**

1. Automatically generate exploits of contracts
2. Deploy light-weight approaches to solve:
   - Unsolvable Constraints
   - Blockchain Effects
3. Fuzz 45,308 contracts in real world
4. Introduce a new vulnerability: Exposed Secret
In memory of medical staff who bravely fight COVID

During the new coronavirus infection in 2020:
- Li Wenliang and 8 other doctors died of illness
- More than 3,000 health workers infected

Pay the highest respect to all the medical staff!
Thank you & Question?