Analysis of Blockchain Smart Contracts: Techniques and Insights

Shinhae Kim and Sukyoung Ryu

#IEEESecDev  https://secdev.ieee.org/2020
Basic Concept of a Blockchain

> A blockchain is a chain of “blocks”

- a data structure that stores transactions among parties

> Each block has the hash of previous block data
  - Modifying the block data is computationally infeasible
Basic Concept of a Blockchain

> A blockchain is built on top of a **P2P network**
  * Each node has a **copy of the chain**

> When a **new transaction** is made..
Basic Concept of a Blockchain

> A blockchain is built on top of a **P2P network**
  • Each node has a **copy of the chain**

> When a **new transaction** is made..

1. A node forms a block with the transaction

```
Transaction #1
Transaction #2
...
“Alice transferred a dollar to Bob”
...
Transaction #N
```
Basic Concept of a Blockchain

> A blockchain is built on top of a P2P network
  * Each node has a copy of the chain

> When a new transaction is made...

1. A node forms a block with the transaction

   - Transaction #1
   - Transaction #2
   - "Alice transferred a dollar to Bob"
   - ...
Basic Concept of a Blockchain

> A blockchain is built on top of a **P2P network**
  - Each node has a **copy of the chain**

> When a **new transaction** is made..

1. A node forms a block with the transaction
2. The node broadcasts the block
3. Other nodes append the block as the tail of the chain
Basic Concept of a Smart Contract

> A smart contract is a **program deployed** on the blockchain

- When invoked, it **automatically makes transactions**

```solidity
function play() payable {
    assert(msg.value == TICKET_AMOUNT);
    pot += msg.value;

    var random = uint(block.blockhash(block.number))
                 + block.timestamp + block.difficulty
                 + block.number;

    if (mulmod(random, 1, 2) == 0) {
        bank.transfer(FEE_AMOUNT);
        msg.sender.transfer(pot - FEE_AMOUNT);
        pot = 0;
    }
}
```

“Bob transferred FEE_AMOUNT to the bank”

“<name> transferred the jackpot to Bob”
Smart Contract Vulnerabilities

> Smart contracts enabled blockchain applicable to various domains

> However, smart contracts have suffered from vulnerabilities

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary Writes to Storage</td>
<td>Any users can overwrite values in storage slots [31].</td>
</tr>
<tr>
<td>Block Variable Dependency</td>
<td>Contracts use block hash or timestamp to generate random values [18, 19].</td>
</tr>
<tr>
<td>Careless Handling of tx.origin</td>
<td>tx.origin returns the address of a transaction-initiated account [27].</td>
</tr>
<tr>
<td>Inability to Transfer Ether</td>
<td>Contracts can receive Ether but do not provide a way for users to withdraw [23].</td>
</tr>
<tr>
<td>DoS: External Call Failures</td>
<td>A callee may throw an exception. For example, send assigns 2300 units of gas only [18].</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected Fallback Function</td>
<td>Fallback executions take place in case of send or no matching function signature [18].</td>
</tr>
<tr>
<td>Integer Division</td>
<td>Dividing integers rounds down the quotient, always resulting in an integer [41].</td>
</tr>
<tr>
<td>Integer Overflow/Underflow</td>
<td>Arithmetic operations can result in over/underflow [113].</td>
</tr>
<tr>
<td>Private Information Leakage</td>
<td>Private fields do not guarantee confidentiality as transaction records are public [18].</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>An external malicious callee can re-enter the body of its caller [18, 19, 20].</td>
</tr>
<tr>
<td>Usage of call.value</td>
<td>call.value forwards all remaining gas [41].</td>
</tr>
<tr>
<td>Return Value Bypassing</td>
<td>In case of exceptions, the built-in functions send and call return false [18, 19].</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Order Dependency</td>
<td>Minors can control the order of deploying transactions in a block [18, 19, 20].</td>
</tr>
<tr>
<td>Transfer to an Orphan Address</td>
<td>Accounts lose Ether if they send it to a non-existing address [26].</td>
</tr>
<tr>
<td>Unauthorized Ether Withdrawal</td>
<td>Invoking Ether transfers is possible by any users [31].</td>
</tr>
<tr>
<td>Unauthorized SELFDESTRUCT</td>
<td>Any users can destroy a contract if no authorization exists for SELFDESTRUCT [31].</td>
</tr>
<tr>
<td>Function without Visibility</td>
<td>Functions without visibility declarations are public by default [19].</td>
</tr>
<tr>
<td>Unbounded Mass Operations</td>
<td>Iterating a loop can cause an out-of-gas exception if its iteration number is too high [32].</td>
</tr>
</tbody>
</table>

a list of “exploitable-vulnerabilities”
Smart Contract Vulnerabilities

<table>
<thead>
<tr>
<th>Reentrancy</th>
<th>An external malicious callee can re-enter the body of its caller [18, 19, 20].</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELEGATECALL</td>
<td>DELEGATECALL passes the caller context to its callee [19].</td>
</tr>
</tbody>
</table>

```solidity
11  mapping (address => uint) private balances;
12
13  function withdraw() public {
14      uint amount = balances[msg.sender];
15      (bool success, ) = msg.sender.call.value(amount)("*");
16      require(success);
17      balances[msg.sender] = 0;
18  }
```

- The adversary stole **$50 million** from the DAO contract (June, 2016)
- Ethereum blockchain was forked into Ethereum and Ethereum Classic
Smart Contract Vulnerabilities

| Reentrancy | An external malicious callee can re-enter the body of its caller [18, 19, 20]. |

| DELEGATECALL | DELEGATECALL passes the caller context to its callee [19]. |

```solidity
9 contract Wallet {
10     address _walletLibrary;
11     address owner;
12
13     function Wallet(address _owner) {
14         _walletLibrary = ${hardcoded address};
15         _walletLibrary.delegatecall(bytes4(sha3("initWallet(address)")), _owner);
16     }
17
18     function () payable {
19         if (msg.value > 0)
20             Deposit(msg.sender, msg.value);
21         else if (msg.data.length > 0)
22             _walletLibrary.delegatecall(msg.data);
23     }
```

1. Adversary calls `initWallet(msg.sender)`
2. The wallet delegates the call to the library
3. The library resets `owner` to adversary
4. Adversary withdraws all of the balance

- The adversary stole **$31 million** from the wallet contract (July, 2017)
Contribution of this Research

> Such vulnerabilities led to massive research on **smart contract analysis**

And this paper is …

the first **comprehensive survey** of research on **smart contract analysis**
Overview of this Research

391 academic papers

67 “analysis-related” papers
Overview of this Research

391 academic papers

67 “analysis-related” papers

Analysis of Blockchain Smart Contracts: Techniques and Insights
Research Trends of Smart Contract Analysis

Static analysis has been more dominant than dynamic analysis:

- Smart contracts need to be audited before deployment
  - Smart contracts cannot be modified once deployed
- Static analysis can reason over all possible paths

> Smart contracts are often opaque (i.e., no source code available)
  - Bytecode loses some information such as variable types
> Dynamic analysis has no false positives
Research Trends of Smart Contract Analysis

- The **most dominant techniques** are the following:

  - **Symbolic execution**
    - Smart contracts often have **small code sizes**

  - **Model checking**
    - Smart contract **execution environments** need to be considered as well
      (nuXmv, Corral, Uppaal, MCK model checkers)

  - **Formal verification**
    - The **highest level of correctness** can be guaranteed
      (Isabelle/HOL, F*, Coq, K framework)
A large portion of work is to detect vulnerabilities.

> Vulnerabilities can directly result in financial losses.

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arbitrary Writes to Storage</td>
<td>Any users can overwrite values in storage slots [31].</td>
</tr>
<tr>
<td>Block Variable</td>
<td>Contracts use block hash or timestamp to generate random values [18, 39].</td>
</tr>
<tr>
<td>Careless Handling of tx.origin</td>
<td>tx.origin returns the address of a transaction-related account [27].</td>
</tr>
<tr>
<td>DelegateCall</td>
<td>DELEGECALL passes the caller context to its callee [19].</td>
</tr>
<tr>
<td>DelegationEther</td>
<td>Contracts can receive Ether but do not provide a way for users to withdraw [23].</td>
</tr>
<tr>
<td>Delegate Call</td>
<td>A callee may throw an exception. For example, send assigns 2300 units of gas only [18].</td>
</tr>
<tr>
<td>Unexpected FallBack</td>
<td>FallBack exceptions take place in case of send or no matching function signature [18].</td>
</tr>
<tr>
<td>Integer Division</td>
<td>Dividing integers rounds down the quotient, always resulting in an integer [41].</td>
</tr>
<tr>
<td>Integer Overflow/Underflow</td>
<td>Arithmetic operations can result in overflow/underflow [115].</td>
</tr>
<tr>
<td>Private Information Leakage</td>
<td>Private fields do not guarantee confidentiality as transaction records are public [18].</td>
</tr>
<tr>
<td>Reentrancy</td>
<td>An external malicious caller can re-enter the body of its caller [18, 19, 20].</td>
</tr>
<tr>
<td>Usage of call-value</td>
<td>call.value forwards all remaining gas [41].</td>
</tr>
<tr>
<td>Return Value Bypassing</td>
<td>In case of exceptions, the built-in functions send and call return false [18, 19].</td>
</tr>
<tr>
<td>Transaction Order</td>
<td>Miners can control the order of deploying transactions in a block [18, 19, 20].</td>
</tr>
<tr>
<td>Transfer to an Orphan Address</td>
<td>Accounts lose Ether if they send it to a non-existing address [20].</td>
</tr>
<tr>
<td>Unauthorized Ether</td>
<td>Unauthorized Ether transfers are possible by any users [31].</td>
</tr>
<tr>
<td>Unauthorized SELFDESTRUCT</td>
<td>Unauthorized SELFDESTRUCT exists for SELFDESTRUCT [31].</td>
</tr>
<tr>
<td>Function without Visibility</td>
<td>Functions without visibility declarations are public by default [19].</td>
</tr>
<tr>
<td>Unbound Mass Operations</td>
<td>Renting a loop can cause an out-of-gas exception if its iteration number is too high [32].</td>
</tr>
</tbody>
</table>
Insights from the Survey

Open Challenges
- Ambiguity in **Contract Behaviors**
- Unstable **Language Semantics**
- Lack of Clear **Property Definition**

Research Directions
- New **Language Design**
- **Type-based Approaches**
- **Machine Learning** Techniques
- Rising **Platforms** and **Languages**
Ambiguity in Contract Behaviors

> Miners can control the **order of deploying transactions**

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction Order Dependency</td>
<td>Miners can control the order of deploying transactions in a block [18, 19, 20].</td>
</tr>
</tbody>
</table>

> Smart contracts often rely on **off-chain sources** through oracles

```
12  function callback(bytes32 key, string callbackData) external payable only_dAppBridge {
13       usd_btc_rate = callbackData;
14       bytes32 newkey = setURLTimeout("callback", 240,
15           "https://api.coindesk.com/v1/bpi/currentprice.json",
16           ","", "bpi.USD.rate");
17    }
18  function startTesting() public {
19      if(msg.sender == owner) {
20         bytes32 newkey = setURLTimeout("callback", 0,
21           "https://api.coindesk.com/v1/bpi/currentprice.json",
22           ","", "bpi.USD.rate");
23      }
```

> Most contracts do not have their **source code available**
Unstable Language Semantics

> Solidity frequently releases **new versions** and undergoes **semantic changes**
> Other languages often do not have **language specifications**

```
pragma solidity ^0.4.11;

contract ReactExample {
    address private owner;

    function ReactExample() public {
        owner = msg.sender;
    }

    function kill() public {
        require(msg.sender == owner);
        selfdestruct(owner);
    }

    function () public payable {
        revert();
    }
}
```

```
pragma solidity ^0.5.0;

contract ReactExample {
    address payable private owner;

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

    function kill() public {
        require(msg.sender == owner);
        selfdestruct(owner);
    }

    function () external payable {
        revert();
    }
}
```

Solidity before ver. 0.5.0  
Solidity after ver. 0.5.0
Lack of Clear Property Definition

> Existing analyzers do not have **clear definitions** of vulnerabilities
  
  - The detected “patterns” do not necessarily imply vulnerabilities

```solidity
  function enter() {
    uint amount = msg.value;
    if ((amount < contribution)) {
      msg.sender.send(msg.value);
      return;
    }
    addParticipant(msg.sender, inviter);
    address next = inviter;
    uint rest = amount;
    uint level = 1;
    while ((next != top) && (level < 7)) {
      uint toSend = rest/2;
      next.send(toSend);
      ...
    }
    next.send(rest);
    Tree[next].totalPayout += rest;
  }
```

**Two paths** that have different Ether flows exist?

**YES!**

Transaction Order Dependency vulnerability
Lack of Clear Property Definition

> Existing analyzers do not have **clear definitions** of vulnerabilities
  
  - The detected “**patterns**” do not necessarily imply vulnerabilities

```solidity
function enter() {
    uint amount = msg.value;
    if ((amount < contribution)...) {
        msg.sender.send(msg.value);
        return;
    }
    addParticipant(msg.sender, inviter);
    address next = inviter;
    uint rest = amount;
    uint level = 1;
    while ((next != top) && (level < 7)) {
        uint toSend = rest/2;
        next.send(toSend);
        //...
    }
    next.send(rest);
    Tree[next].totalPayout += rest;
}
```

**Two paths** that have **different Ether flows** exist?

**YES!**

- **Transaction Order Dependency** vulnerability

- **False Positive!**
Promising Research Directions

> New **Language Design**

- Languages with **fully specified semantics** are amenable to analysis

> **Type-based Approaches**

- Most statically-typed languages do not support **strong type checking**
- Enriching existing type systems can **discover vulnerabilities**

```
11 contract Bob {
12    function bar(Alice alice){
13       alice.foo(42);
14    }
15 }
```

**Compiler before ver. 0.5.0** does not check whether `alice` is an address of `Alice` contract
Promising Research Directions

> **Machine Learning** Techniques

  • Machine learning may improve the *scalability and accuracy* of smart contract analysis

> **Rising Platforms and Languages**

  • Blockchain ecosystem is still *evolving very fast*
  • Research on rising platforms and languages is *at an early stage*

  e.g. Hyperledger Fabric, EOS (platform)
  e.g. Serpent, Vyper (language)
Thanks for your Listening!