SOLIDITY AUDIT

Smart Contract Solidity Audit

Audit Details:

Audited project: KriptoCoin

Deployer Address: 0xC42119a7F49E73df9d756291B1DEdfb99b492AE0

Blockchain: Ethereum

July 23

Audit

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed - upon a decision of the Customer.

Introduction

Solidity Audit (Consultant) was contracted by KriptoCoin (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract . Scope The scope of the project is main net smart contracts that can be found on Etherscan:

https://etherscan.io/token/0xC42119a7F49E73df9d756291B1DEdfb99b492AE0

We have scanned this smart contract for commonly known and more specific vulnerabilities. List of the commonly known vulnerabilities that are considered:

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. A general overview is presented in AS-IS section, and all found issues can be found in the Audit overview section.

Security engineers found **2** medium, **1** informational issue during the audit.

Notice: the audit scope is limited and not include all files in the repository. Though, reviewed contracts are secure, we may not guarantee secureness of contracts that are not in the scope.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

Understanding Vulnerabilities

Re-Entrancy

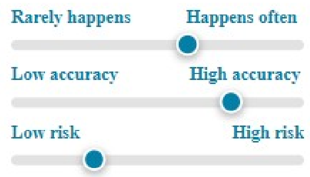
Inject fallback function calls on the function itself, so the process re-enters the function and then go to fallback function again. This loop goes on endlessly and if the function involves transferring, then it will end up with an empty wallet

Parity Multisig Bug

Contract A has functions that rely on contract B, contract A cannot guarantee that contract B is in good shape. Happens particularly in Parity contracts.

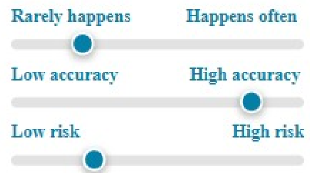
Transaction-ordering Dependency

The order of transactions getting verified can be manipulated by the miners. If multiple transactions are submitted within the short period of time, it is possible that the later one gets verified before the prior. Thus create problems or conflicts such as a race condition.

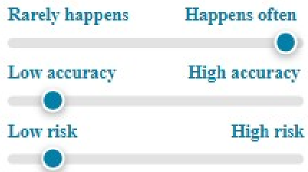


Timestamp Dependency

The timestamp is a controllable variable, it is easy to exploit as a factor of the random number. Ex: attackers can send an attack at a specific calculated time, then the randomness of an RNG is eliminated.

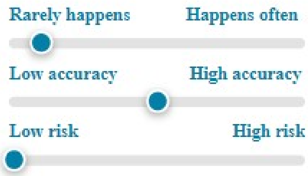


Integer Overflow/ Underflow



The unsigned integer should be taken care with boundary with boundary check, because of uint MAX + 1 = uint MIN (and vice versa). This can lead to unexpected outputs or even capital loss if it happens in a transfer function.

Callstack Depth Attack



A function recursively calls itself or another. Imagine the newer call is stack on the prior call. Until a certain depth level, the code can no more be executed.

* This issue was already fixed by Ethereum developers and is no longer able to exploit. But still be aware of potential high gas cost when running recursive calls.

```
/**
 *Submitted for verification at Etherscan.io on 2021-07-20
*/

pragma solidity ^0.4.18;

contract SafeMath {
    function safeAdd(uint a, uint b) public pure returns (uint c) {
        c = a + b;
        require(c >= a);
    }
    function safeSub(uint a, uint b) public pure returns (uint c) {
        require(b <= a);
        c = a - b;
    }
    function safeMul(uint a, uint b) public pure returns (uint c) {
        c = a * b;
        require(a == 0 || c / a == b);
    }
    function safeDiv(uint a, uint b) public pure returns (uint c) {
        require(b > 0);
        c = a / b;
    }
}

contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}

contract ApproveAndCallFallBack {
    function receiveApproval(address from, uint256 tokens, address token, bytes data) public;
}

contract Owned {
    address public owner;
    address public newOwner;

    event OwnershipTransferred(address indexed _from, address indexed _to);

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

    modifier onlyOwner {
        require(msg.sender == owner);
        _;
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        newOwner = _newOwner;
    }
    function acceptOwnership() public {
        require(msg.sender == newOwner);
        OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}

contract KriptoCoin is ERC20Interface, Owned, SafeMath {
    string public symbol;
    string public  name;
    uint8 public decimals;
    uint public _totalSupply;

    mapping(address => uint) balances;
    mapping(address => mapping(address => uint)) allowed;

    function KriptoCoin() public {
        symbol = "KRPT";
        name = "KriptoCoin";
        decimals = 8;
        _totalSupply = 100000000000000;
        balances[0x8E1ca9c4AA5157Ad599BE2AEf70Ab25e6162515D] = _totalSupply;
        Transfer(address(0), 0x8E1ca9c4AA5157Ad599BE2AEf70Ab25e6162515D, _totalSupply);
    }

    function totalSupply() public constant returns (uint) {
        return _totalSupply  - balances[address(0)];
    }

    function balanceOf(address tokenOwner) public constant returns (uint balance) {
        return balances[tokenOwner];
    }

    function transfer(address to, uint tokens) public returns (bool success) {
        balances[msg.sender] = safeSub(balances[msg.sender], tokens);
        balances[to] = safeAdd(balances[to], tokens);
        Transfer(msg.sender, to, tokens);
        return true;
    }

    function approve(address spender, uint tokens) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        Approval(msg.sender, spender, tokens);
        return true;
    }

    function transferFrom(address from, address to, uint tokens) public returns (bool success) {
        balances[from] = safeSub(balances[from], tokens);
        allowed[from][msg.sender] = safeSub(allowed[from][msg.sender], tokens);
        balances[to] = safeAdd(balances[to], tokens);
        Transfer(from, to, tokens);
        return true;
    }

    function allowance(address tokenOwner, address spender) public constant returns (uint remaining) {
        return allowed[tokenOwner][spender];
    }

    function approveAndCall(address spender, uint tokens, bytes data) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        Approval(msg.sender, spender, tokens);
        ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);
        return true;
    }

    function () public payable {
        revert();
    }

    function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner returns (bool
success) {
        return ERC20Interface(tokenAddress).transfer(owner, tokens);
    }
}
```

Vulnerability analysis

ETH Smart Contract Audit Report

Address: 0xc42119a7f49e73df9d756291b1dedfb99b492ae0

MD5:0b48fadda05c5cd9c77d7e63da4070af Runtime:0.9s

Scored higher than 11% of similar code

Score

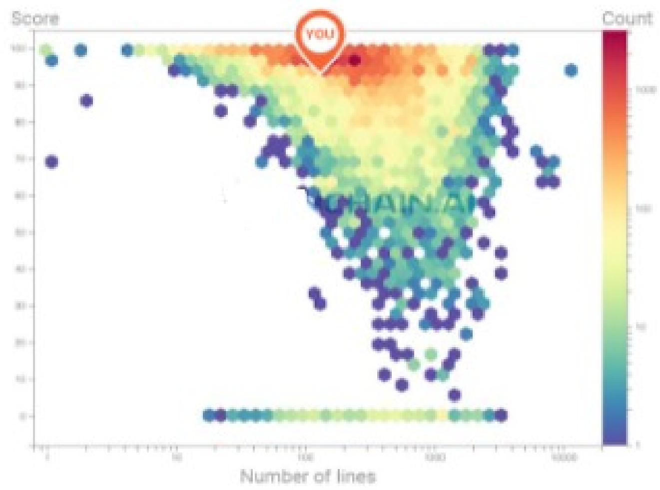
93.6

Threat Level

Elevated

Number of lines

129



Overview

Code Class	EVM Coverage
KriptoCoin	90.4%
Owned	99.2%
SafeMath	98.5%

2 Vulnerabilities Found

High Risk Medium Risk Low Risk

KriptoCoin

Integer Underflow Found near or at lines

65 66 83 117 118

Integer Overflow Found near or at lines

4 12 16 39 40 65 66 67 68 82 86 99 111 115 126 127

Recommendations

- 'constant' is deprecated. Consider using 'view' instead.
  See line(s) 23, 24, 25, 82, 86, 111
- Consider using exact language version instead.
  See line(s) 1
- Hard-coded address should be checked.
  See line(s) 78, 79
- Missing check on 'msg.data.length' could lead to short-address attack in this ERC20 transfer function.
  See line(s) 90, 103
- Underflow or overflow may happen here, consider check boundaries such as assert(n < INT_MAX).
  See line(s) 83

Vulnerability Checklist

KriptoCoin

- ⚠ Integer Underflow
- ⚠ Integer Overflow
- ✓ Parity Multisig Bug
- ✓ Callstack Depth Attack
- ✓ Transaction-Ordering Dependency
- ✓ Timestamp Dependency
- ✓ Re-Entrancy

Owned

- ✓ Integer Underflow
- ✓ Integer Overflow
- ✓ Parity Multisig Bug
- ✓ Callstack Depth Attack
- ✓ Transaction-Ordering Dependency
- ✓ Timestamp Dependency
- ✓ Re-Entrancy

SafeMath

- Integer Underflow
- Integer Overflow
- Parity Multisig Bug
- Callstack Depth Attack
- Transaction-Ordering Dependency
- Timestamp Dependency
- Re-Entrancy

Disclaimers

Solidity Audit Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). The audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.