# SMART CONTRACT AUDIT

ZOKYO.

October 5th, 2021 | v. 1.0

## PASS

Zokyo Security team has concluded that the given smart contracts passed security audit. The findings are presented in the document

# TECHNICAL SUMMARY

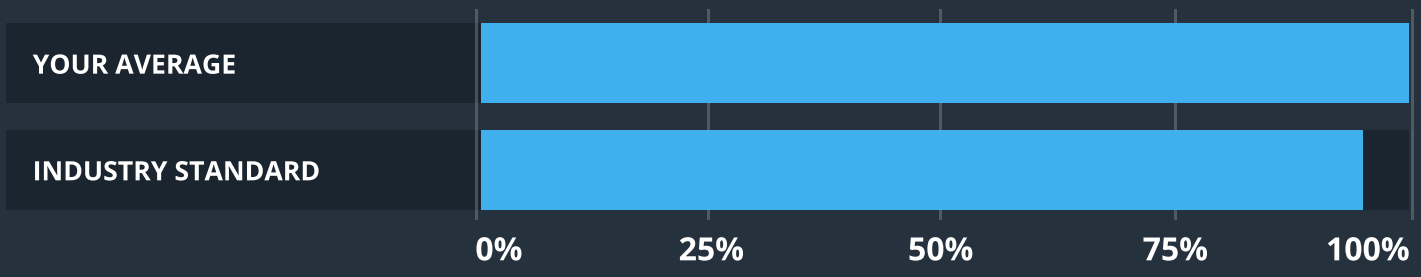
This document outlines the overall security of the Raiinmaker smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Raiinmaker smart contract codebase for quality, security, and correctness.

## Contract Status


There were no critical issues found during the audit.

## Testable Code




The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Raiinmaker team put in place a bug bounty program to encourage further and active analysis of the smart contract.

# TABLE OF CONTENTS

# AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Raiinmaker repository.

**Repository:**
https://github.com/Raiinmaker/raiinmaker-contracts/commit/6105ef4bebbb4aecbda9b86351e84804faa5747d

**Last commit:**
f0b0d9ed5f8f420c5cb2abce25392511eff8d241

**Contracts:**

- CoiinMinter;
- CoiinToken;
- CoiinVault;
- CoiinVestingVault;
- RaiinmakerCampaignVault;
- SafeERC20;
- ECDSA.

**Throughout the review process, care was taken to ensure that the token contract:**

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

| | |
|---|---|
| 1 | Due diligence in assessing the overall code quality of the codebase. |
| 2 | Cross-comparison with other, similar smart contracts by industry leaders. |
| 3 | Testing contract logic against common and uncommon attack vectors. |
| 4 | Thorough, manual review of the codebase, line-by-line. |

## SUMMARY

Zokyo team has conducted a security audit of the given codebase. The contracts provided for an audit are well written and structured. All the findings within the auditing process are presented in this document.

There were no critical issues found during the auditing process. Among the findings, there are 3 issues with the high severity and a couple of medium, low and informational issues.  All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

Taking into account the fact that mostly all of the issues were resolved and evaluating the contracts from the operational and security standpoints, we can give the score of 90% to the provided codebase.

## STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

### Critical

The issue affects the ability of the contract to compile or operate in a significant way.

### High

The issue affects the ability of the contract to compile or operate in a significant way.

### Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

### Low

The issue has minimal impact on the contract’s ability to operate.

### Informational

The issue has no impact on the contract’s ability to operate.

## COMPLETE ANALYSIS

### MultiSig can transfer all tokens to it’s address with function kill() at CoiinVault.sol

HIGH | UNRESOLVED

Function kill allows the multisig address to transfer all deposited tokens to its address.

**Recommendation:**
Remove token transfer from function or share additional information on why this function needs to be included.

### Missed interface import at SafeERC20.sol

HIGH | RESOLVED

Contract SafeERC20.sol has import import "../interfaces/IERC20Burnable.sol" which not included in repo. In all other contracts where needed SafeErc20 you use it from openzeppelin library.

**Recommendation:**
If needed add an interface to the contracts folder, so the contract can be compiled or remove the contract if you don’t need to use it.

## Requires don’t give access to mint function at CoiinMinter.sol

HIGH | RESOLVED

In constructor of CoiinMinter.sol you define variable lastMintAmount equal to 1000000 * 1e18, but in function mint you have requires:

```
// _amount cannot be less than minMintMonthly
require(_amount >= coiin.minMintMonthly(), "amount cannot be less than min mint amount");

// _amount cannot exceed maxMintMonthly
require(_amount <= coiin.maxMintMonthly(), "amount cannot be more than max mint amount");

// _amount cannot be more than 25% of the lastMintAmount
require(
      _amount > lastMintAmount.add(lastMintAmount.mul(25).div(100)),
      "amount cannot be more than 25% of last mint amount"
   );

// _amount cannot be less than 25% of the lastMintAmount
require(
      _amount < lastMintAmount.sub(lastMintAmount.mul(25).div(100)),
      "amount cannot be less than 25% of last mint amount"
   );
```

So you can mint in the range 1000000 * 1e18 to 25000000 * 1e18 (according to the first two requirements) but according to 4 requirement mint amount should be less than 75% of lastMintAmount and this amount would be less than minMintMonthly from CoiinToken.sol. Additionally, 3 and 4 requires have misleading comments and revert messages.

**Recommendation:**
Update requires, change comments and revert messages.

## Misleading functions names at RaiinmakerCampaignVault.sol

**MEDIUM** | RESOLVED

Function setFeeAmount() takes as a parameter address and assigns it to the variable feeAddress. Also in function missed check for zero address.

Function setFeeAddress takes as a parameter uint256 _fee and assigns it to the variable fee.

**Recommendation:**
Change functions names and add an additional checks:

```
function setFeeAddress(address _address) external onlyMultiSig {
        require(_address != address(0), "Wrong fee address");
        feeAddress = _address;
        emit FeeAddressChanged(_address);
    }
function setFeeAmount(uint256 _fee) external onlyMultiSig {
        fee = _fee;
        emit FeeAmountChanged(_fee);
    }
```

## Excessive validation in the claimVestedTokens () and removeTokenGrant () functions at CoiinVestingVault.sol

**LOW** | UNRESOLVED

```
require(_totalSupply.sub(amountVested) >= 0, "not enough tokens");
```

Underflow already checked in sub() function from SafeMath.

**Recommendation:**
Remove these require() functions.

## Extra comments in function kill() at RaiinmakerCampaignVault.sol

LOW | UNRESOLVED

```
function kill() public onlyMultiSig {
        pause();

        // uint256 amount = totalSupply();
        // _totalSupply[token] = _totalSupply[token].sub(amount);
        // _totalWithdrawn[token] = _totalWithdrawn[token].add(amount);
        // mtrToken.safeTransfer(multiSig, amount);
        // emit Withdrawn(multiSig, 0, amount);
    }
```

**Recommendation:**
Remove extra comments.

## Additional checks are required for constructor of CoiinMinter.sol

LOW | RESOLVED

There is no verification for the zero address for the coin and mint addresses, and checking of the valid timestamp.

**Recommendation:**
Add additional checks:

```
require(_firstIssuedTimeStamp >= now, "Wrong timestamp");
require(_coiinAddress != address(0), "Wrong coiin address");
require(_mintAddress != address(0), "Wrong mint address");
```

## Additional checks are required for function setMintAddress() at CoiinMinter.sol

LOW | RESOLVED

There is no verification for the zero address for the mint address.

**Recommendation:**
Add additional check:

```
require(_mintAddress != address(0), "Wrong mint address");
```

## Additional checks are required for function setMultiSig() at CoiinMinter.sol

LOW | RESOLVED

There is no verification for the zero address for the multiSig address.

**Recommendation:**
Add additional check:

```
require(_multiSig != address(0), "Wrong multiSig address");
```

## Missed revert message in function mint() at CoiinMinter.sol

LOW | RESOLVED

```
require(lastMintTimestampSec.add(minMintTimeIntervalSec) < now);
```

**Recommendation:**
Add revert message.

## Not reachable require statement at CoiinMinter.sol

LOW | RESOLVED

In function _inMintWindow() you have two requires :

```
require(now.mod(minMintTimeIntervalSec) >= mintWindowOffsetSec, "too early");
require(now.mod(minMintTimeIntervalSec)<(mintWindowOffsetSec.add
(mintWindowLengthSec)), "too late");
```

mintWindowOffsetSec is set in the constructor of the contract and is equal to 0. So there is no need for the first require and you can remove mintWindowOffsetSec from the second require.

**Recommendation:**
Remove “require” or add additional information about mintWindowOffsetSec if you want to use it with different values and function _inMintWindow().

## Additional checks are required for constructor of CoiinVault.sol

LOW | RESOLVED

There is no verification for the zero address for the _coiinToken and _withdrawSigner.

**Recommendation:**
Add additional checks:

```
require(_withdrawSigner != address(0), "Wrong signer address");
require(_coiinToken != address(0), "Wrong token address");
```

## Additional checks are required for function setWithdrawSigner() at CoiinVault.sol

LOW | RESOLVED

There is no verification for the zero address for the _withdrawSigner.

**Recommendation:**
Add additional check:

```
require(account != address(0), "Wrong signer address");
```

## Additional checks are required for function setMultiSig() at CoiinVault.sol

LOW | RESOLVED

There is no verification for the zero address for the MultiSig address.

**Recommendation:**
Add additional check:

```
require(_multiSig != address(0), "Wrong multiSig address");
```

## Additional checks are required for constructor of RaiinmakerCampaignVault.sol

LOW | RESOLVED

There is no verification for the zero address for the _feeToken and _withdrawSigner.

**Recommendation:**
Add additional checks:

```
require(_withdrawSigner != address(0), "Wrong signer address");
require(_feeToken != address(0), "Wrong token address");
```

## Additional checks are required for function setWithdrawSigner() at RaiinmakerCampaignVault.sol

LOW | RESOLVED

There is no verification for the zero address for the _withdrawSigner.

**Recommendation:**
Add additional check:

```
require(account != address(0), "Wrong signer address");
```

## Additional checks are required for function deposit() at RaiinmakerCampaignVault.sol

LOW | RESOLVED

There is no verification for the zero address for the token and amount equal to zero.

**Recommendation:**
Add additional checks:

```
require(token != address(0), "Wrong token address");
require(amount > 0, "Wrong amount");
```

## Additional check are required for function changeMultiSig() at RaiinmakerCampaignVault.sol

LOW | RESOLVED

There is no verification for the zero address for the multiSig.

**Recommendation:**
Add additional check:

```
require(_multiSig != address(0), "Wrong multiSig address");
```

## Missed revert message in constructor at CoiinVestingVault.sol

LOW | RESOLVED

```
require(address(_token) != address(0));
```

**Recommendation:**
Add revert message.

## Additional check are required for function deposit() at CoiinVestingVault.sol

LOW | RESOLVED

There is no verification for the deposit amount.

**Recommendation:**
Add additional check:

```
require(amount > 0, "Wrong amount");
```

## Additional check are required for function addTokenGrant() at CoiinVestingVault.sol

LOW | RESOLVED

There is no verification for the recipient address.

**Recommendation:**
Add additional check:

```
require(_recipient != address(0), "Wrong recipient address");
```

## Lock pragma to the specific version

LOW | RESOLVED

Lock the pragma to a specific version, since not all the EVM compiler versions support all the features, especially the latest one's which are kind of beta versions (for example you use pragma solidity >=0.5.12 at CoiinVestingVault.sol ), so the intended behavior written in code might not be executed as expected.

**Recommendation:**
Lock pragma to a specific version.

## Misleading comment at CoiinToken.sol

INFORMATIONAL | RESOLVED

The variables names are maxMintMonthly and minMintMonthly but in notice, we see the daily minting allowed:

```
/// @notice The daily allowed COIIN to be minted
uint256 public maxMintMonthly;
uint256 public minMintMonthly;
```

**Recommendation:**
Change comment or variables names.

## Additional optimization at CoiinToken.sol

INFORMATIONAL | RESOLVED

maxMintMonthly and minMintMonthly can be changed to constant as you don't change them further in the code:

```
/// @notice The daily allowed COIIN to be minted
uint256 public maxMintMonthly;
uint256 public minMintMonthly;
```

**Recommendation:**
Use constant for those variables.

## Additional optimization at CoiinMinter.sol

INFORMATIONAL | RESOLVED

minMintTimeIntervalSec, intWindowOffsetSec, mintWindowLengthSec can be changed to constant as you don't change them further in the code

```
minMintTimeIntervalSec = 4 weeks;
mintWindowOffsetSec = 0; // 12 AM UTC mint
mintWindowLengthSec = 14 days;
lastMintAmount = 1000000 * 1e18;
```

**Recommendation:**
Use constant for those variables.

| | CoiinMinter | CoiinToken | CoiinVault |
|---|---|---|---|
| Re-entrancy | Pass | Pass | Pass |
| Access Management Hierarchy | Pass | Pass | Pass |
| Arithmetic Over/Under Flows | Pass | Pass | Pass |
| Unexpected Ether | Pass | Pass | Pass |
| Delegatecall | Pass | Pass | Pass |
| Default Public Visibility | Pass | Pass | Pass |
| Hidden Malicious Code | Pass | Pass | Pass |
| Entropy Illusion (Lack of Randomness) | Pass | Pass | Pass |
| External Contract Referencing | Pass | Pass | Pass |
| Short Address/ Parameter Attack | Pass | Pass | Pass |
| Unchecked CALL Return Values | Pass | Pass | Pass |
| Race Conditions / Front Running | Pass | Pass | Pass |
| General Denial Of Service (DOS) | Pass | Pass | Pass |
| Uninitialized Storage Pointers | Pass | Pass | Pass |
| Floating Points and Precision | Pass | Pass | Pass |
| Tx.Origin Authentication | Pass | Pass | Pass |
| Signatures Replay | Pass | Pass | Pass |
| Pool Asset Security (backdoors in the underlying ERC-20) | Pass | Pass | Pass |

| | CoiinVestingVault | RaiinmakerCampaignVault |
|---|---|---|
| Re-entrancy | Pass | Pass |
| Access Management Hierarchy | Pass | Pass |
| Arithmetic Over/Under Flows | Pass | Pass |
| Unexpected Ether | Pass | Pass |
| Delegatecall | Pass | Pass |
| Default Public Visibility | Pass | Pass |
| Hidden Malicious Code | Pass | Pass |
| Entropy Illusion (Lack of Randomness) | Pass | Pass |
| External Contract Referencing | Pass | Pass |
| Short Address/ Parameter Attack | Pass | Pass |
| Unchecked CALL Return Values | Pass | Pass |
| Race Conditions / Front Running | Pass | Pass |
| General Denial Of Service (DOS) | Pass | Pass |
| Uninitialized Storage Pointers | Pass | Pass |
| Floating Points and Precision | Pass | Pass |
| Tx.Origin Authentication | Pass | Pass |
| Signatures Replay | Pass | Pass |
| Pool Asset Security (backdoors in the underlying ERC-20) | Pass | Pass |

| | RaiinmakerCampaignVault | SafeERC20 | ECDSA |
|---|---|---|---|
| Re-entrancy | Pass | Pass | Pass |
| Access Management Hierarchy | Pass | Pass | Pass |
| Arithmetic Over/Under Flows | Pass | Pass | Pass |
| Unexpected Ether | Pass | Pass | Pass |
| Delegatecall | Pass | Pass | Pass |
| Default Public Visibility | Pass | Pass | Pass |
| Hidden Malicious Code | Pass | Pass | Pass |
| Entropy Illusion (Lack of Randomness) | Pass | Pass | Pass |
| External Contract Referencing | Pass | Pass | Pass |
| Short Address/ Parameter Attack | Pass | Pass | Pass |
| Unchecked CALL Return Values | Pass | Pass | Pass |
| Race Conditions / Front Running | Pass | Pass | Pass |
| General Denial Of Service (DOS) | Pass | Pass | Pass |
| Uninitialized Storage Pointers | Pass | Pass | Pass |
| Floating Points and Precision | Pass | Pass | Pass |
| Tx.Origin Authentication | Pass | Pass | Pass |
| Signatures Replay | Pass | Pass | Pass |
| Pool Asset Security (backdoors in the underlying ERC-20) | Pass | Pass | Pass |

# CODE COVERAGE AND TEST RESULTS FOR ALL FILES

## Tests written by Zokyo Security team

As part of our work assisting Raiinmaker in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Raiinmaker contract requirements for details about issuance amounts and how the system handles these.

### Code Coverage

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

| FILE | % STMTS | % BRANCH | % FUNCS | % LINES | UNCOVERED LINES |
|---|---|---|---|---|---|
| contracts\ | 100.00 | 97.17 | 100.00 | 100.00 | |
| CoiinMinter.sol | 100.00 | 96.15 | 100.00 | 100.00 | |
| CoiinToken.sol | 100.00 | 100.00 | 100.00 | 100.00 | |
| CoiinVault.sol | 100.00 | 100.00 | 100.00 | 100.00 | |
| CoiinVestingVault.sol | 100.00 | 94.44 | 100.00 | 100.00 | |
| RaiinmakerCampaignVault.sol | 100.00 | 100.00 | 100.00 | 100.00 | |
| contracts\token\ | 100.00 | 87.50 | 100.00 | 100.00 | |
| SafeERC20.sol | 100.00 | 87.50 | 100.00 | 100.00 | |
| contracts\utils\ | 100.00 | 100.00 | 100.00 | 100.00 | |
| ECDSA.sol | 100.00 | 100.00 | 100.00 | 100.00 | |
| **All files** | **100.00** | **96.67** | **100.00** | **100.00** | |

## Test Results

**Contract: CoiinMinter**

CoiinMinter Test Cases

Initialization

- ✓ should deploy with correct first issued timeStamp (89ms)
- ✓ should deploy with correct coiin address (119ms)
- ✓ should deploy with correct mint address (103ms)
- ✓ should revert if firstIssuedTimeStamp less than current timestamp (910ms)
- ✓ cannot deploy with zero token address (271ms)
- ✓ cannot deploy with zero mint address (331ms)

setMintAddress

- ✓ should set mint address correctly (325ms)
- ✓ shouldn`t set mint address [wrong owner] (274ms)
- ✓ shouldn`t set mint address [zero address] (289ms)

setMultiSig

- ✓ should set multiSig address correctly (323ms)
- ✓ shouldn`t set multiSig address [wrong owner] (209ms)
- ✓ shouldn`t set multiSig address [zero address] (248ms)

inMintWindow

- ✓ should return true if the latest block timestamp is within the mint time window (202ms)
- ✓ should revert if latest block timestamp isn`t within the mint time window (431ms)

mint

- ✓ should revert if min mint interval not elapsed (613ms)
- ✓ should revert if amount less than min mint amount (232ms)
- ✓ should revert if amount more than max mint amount (264ms)
- ✓ should mint correctly (599ms)
- ✓ should catche event
- ✓ should revert if amount more than 125% of last mint amount (370ms)
- ✓ should revert if amount less than 75% of last mint amount (662ms)
- ✓ should revert if latest block timestamp isn`t within the mint time window (327ms)

**Contract: CoiinToken**

CoiinToken Test Cases

- ✓ should deploy with correct token name (130ms)
- ✓ should deploy with correct token symbol (107ms)
- ✓ should deploy with correct decimals (137ms)
- ✓ should deploy with correct total supply (124ms)

✓ should deploy with correct initial supply (81ms)
✓ should deploy with correct monthly minting limits (292ms)
✓ should remove minter correctl (395ms)

**Contract: RaiinmakerCampaignVault**
RaiinmakerCampaignVault Test Cases
Initialization
✓ should deploy with correct token address (81ms)
✓ should deploy with correct withdrawSigner address (104ms)
✓ should deploy with correct multiSig address (93ms)
✓ cannot deploy with zero signer address (295ms)
✓ cannot deploy with zero token address (276ms)
Deposit
✓ should be executed the deposit correctly (1287ms)
✓ cannot be executed the deposit if campaign exists (246ms)
✓ should revert if amount is zero (196ms)
✓ should revert if token address is zero (195ms)
Withdrawing
✓ should withdraw tokens correctly (522ms)
✓ cannot duplicate a transaction (240ms)
✓ cannot withdraw if withdrawal period expired (287ms)
✓ cannot withdraw more than totalSupply (254ms)
✓ cannot withdraw if request not signed by raiinmaker (552ms)
Admin Functions
Kill
✓ should kill correctly (330ms)
setFeeAddress
✓ should set fee address correctly (413ms)
✓ cannot set zero fee address (211ms)
✓ should catch event
setFeeAmount
✓ should set fee amount correctly (325ms)
✓ should catch event
changeMultiSig
✓ should set multiSig correctly (305ms)
✓ cannot set zero multiSig address (258ms)
Methods accessors and other functions
✓ should set withdrawSigner correctly (547ms)
✓ cannot set withdrawSigner [zero address] (200ms)

✓ should return totalSupply correctly (117ms)
✓ should return total withdraw correctly (124ms)

**Contract: CoiinVault**
CoiinVault Test Cases
Withdrawing
✓ should withdraw tokens correctly (1838ms)
✓ cannot duplicate a transaction (902ms)
✓ cannot withdraw if withdrawal period expired (301ms)
✓ cannot withdraw more than totalSupply (395ms)
✓ cannot withdraw if request not signed by raiinmaker (1075ms)
✓ should return total withdraw correctly (652ms)
Initialization
✓ should deploy with correct token address (82ms)
✓ should deploy with correct withdrawSigner address (136ms)
✓ should deploy with correct multiSig address (213ms)
✓ cannot deploy with zero signer address (411ms)
✓ cannot deploy with zero token address (464ms)
Deposit
✓ should be executed the deposit correctly (2285ms)
✓ cannot be executed the deposit if the contract is on pause (2294ms)
withdrawSigner
✓ should set withdrawSigner correctly (604ms)
✓ cannot set zero withdrawSigner address (224ms)
multiSig
✓ should set multiSig correctly (576ms)
✓ cannot set zero multiSig address (241ms)
totalSupply
✓ should return totalSupply correctly (208ms)
Kill
✓ should kill correctly (1075ms)

**Contract: CoiinVestingVault**
CoiinVestingVault Test Cases
✓ should return token vested per day correctly (703ms)
✓ should change multiSig correctly (622ms)
✓ cannot change multiSig [wrong address] (530ms)
✓ should return total supply correctly (331ms)

✓ should return total granted correctly (1036ms)
✓ should return total claimed correctly (2213ms)
Initialization
✓ should deploy with correct token address (295ms)
✓ cannot deploy with zero token address (326ms)
Deposit
✓ should be executed the deposit correctly (732ms)
✓ cannot make a deposit [wrong owner] (351ms)
✓ cannot make a deposit [zero amount] (246ms)
Token Grant
✓ should add token grant correctly (2018ms)
✓ should revert if recipient is zero address (285ms)
✓ cannot add more tokens than total supply (376ms)
✓ cannot add grant if cliff duration less than 30 days or more than 1 year (1097ms)
✓ cannot add tokens grant if vesting duration less than cliff duration (277ms)
✓ cannot add tokens grant if vesting duration more than 10 years (396ms)
✓ cannot add tokens grant if amount is 0 (340ms)
Grant Claim Calculation
✓ for grants created with a future start date, that hasn't been reached, returned 0, 0 (1075ms)
✓ for grants with cliff that has been reached, returned elapsed days, 0 (845ms)
✓ should make all tokens vested if vesting duration is over (701ms)
✓ for grants with cliff that has been reached and vesting duration is not over, returned elapsed days and vested amount (852ms)
Claim Vested Tokens
✓ should claim vested tokens correctly (1521ms)
✓ cannot claim if vested amount is 0 (993ms)
✓ should revert if caller is not grant recipient (231ms)
Remove Token Grant
✓ should remove token grant correctly (2398ms)

**Contract: SafeERC20**
SafeERC20 Test Cases
safeTransfer
✓ should transfer correctly (580ms)
safeTransferFrom
✓ should transfer correctly (442ms)
safeBurn
✓ should burn correctly (602ms)
callOptionalReturn

```
    ✓ should revert if token address isn`t contract (221ms)
    ✓ should revert if call is not success (219ms)

  105 passing (2m)
```

We are grateful to have been given the opportunity to work with the Raiinmaker team.

**The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.**

Zokyo's Security Team recommends that the Raiinmaker team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.