

CORE RISE

Security Assessment

www.corerise.finance

TABLE OF CONTENTS

Summary

Overview

Project Summary

Audit Summary

Vulnerability Summary

Audit Scope

Findings

01 : Centralization Risk in `_hasBeenLiqAdded` Function

02 : CentralizationRisk in Contract `RISE CORE`

03 : CentralizationRisk in Contract function `getMultipliedFee()`

04 : `SetAllowance()`

05 : Initial Token Distribution

06 : `autoLiquidityReceiver`

07 : UNLOCKED COMPILER VERSION

08 : Lack of Error Message

09 : Redundant Code

10 : Typos In The Contract

Appendix

Disclaimer

About

Summary

This report has been prepared to discover issues and vulnerabilities in the source code of the project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques. The auditing process pays special attention to the following considerations:

Testing the smart contracts against both common and uncommon attack vectors. Assessing the codebase to ensure compliance with current best practices and industry standards. Ensuring contract logic meets the specifications and intentions of the client.

Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders. Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

Enhance general coding practices for better structures of source codes; Add enough unit tests to cover the possible use cases; Provide more comments per each function for readability, especially contracts that are verified in public; Provide more transparency on privileged activities once the protocols live.

Project Summary

Project Name	RISE CORE - (http://corerise.finance)
Platform	CORE DAO
Language	Solidity
Codebase	https://scan.coredao.org/address/0x3cDf6D8E12464d5a4dd575CDC614C930d8C8D6a3
Commit	55pp76887612fd345ftf5b698fh4576h7lg10i472sdnfj4nb

Audit Summary

Delivery Date	March 6, 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	Core Token

Vulnerability Summary

Vulnerability Level	Total	⚠ Pending	⊗ Declined	ℹ Acknowledged	🔄 Partially Resolved	✅ Resolved
🔴 Critical	0	0	0	0	0	0
🟠 Major	2	0	0	2	0	0
🟡 Medium	2	0	0	2	0	2
🟠 Minor	0	0	0	3	0	0
🟡 Informational	1	0	0	6	0	1
🟢 Discussion	0	0	0	0	0	0

Audit Scope

4 files audited ● 3 files with Acknowledged findings ● 1 file with Resolved findings

ID	Repo	Commit	File	SHA256 Checksum
● APB	coredao-org/dao-contracts	1a550d5	contracts/AirdropPool.sol	6b107e9cb0aea38712d01b7e40a7ed18aeb91a0e2aa7dd1ccd880b266c0dc728
● COR	coredao-org/dao-contracts	1a550d5	contracts/CORESales.sol	0323126fc721ae5ea737ad33aeb2568a26b5eeb2fd011006f1cfce1dee7e648
● TVB	coredao-org/dao-contracts	1a550d5	contracts/TeamVesting.sol	939147a2ac8de592227101e2fb9aeeb4c2d00eefac4679d96eab4f4eef78c635
● SMB	coredao-org/dao-contracts	1a550d5	contracts/lib/SafeMath.sol	6eeb4a240710a44001fb885f5000495e673c553a0c19e878265a523ad66095ea

Overview

External Dependencies

The contract serves as the underlying entity to interact with third-party protocols (token-wrapping). The scope of the audit treats third-party entities as blackboxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

Privileged Functions

The contract contains the following privileged functions that are restricted by role with the modifier. They are used to modify the contract configurations and address attributes. We grouped these functions below.

- function `getLiquidityBacking()`
- function `getMultipliedFee()`
- function `allowance()`
- function `approve()`
- function `marketingFeeReceiver()`
- function `isOverLiquified()`
- function `owner()`
- function `MASK()`
- function `setSwapAndLiquifyEnabled()`
- function `setRouterAddress()`

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the Timelock contract.

01 | Centralization Risk in Function

Description

The `marketingWallet()` function calls the `coreV2Router.addLiquidityETH` function with the `to()` address specified as `owner()` for acquiring the generated LP tokens from the corresponding pool. As a result, over time the `_owner` address will accumulate a significant portion of LP tokens. If `_owner` is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project as a whole.

Recommendation

We advise `to()` the address of the `coreV2Router.addLiquidityETH()` function call to be replaced by the `contract()` itself, i.e. `address(this)`, and to restrict the management of the LP tokens within the scope of the contract's business logic. This will also protect the LP tokens from being stolen if the `_owner()` account is compromised. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract based accounts with enhanced security practices, f.e. `Multisignature wallets()`.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, i.e. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key;
- Introduction of a DAO / governance / voting module to increase transparency and user involvement

02 | Centralization Risk in Contract

Description

In the contract `CoinTokens()`, the role `_owner()` has the authority over the following function:

- `marketingWallet()` ; the owner of the contract has set `marketingWallet()` and can set upto 100% fee
- `buyFeeRate()` : the owner of the contract can increase the `buyfees to 100%`
- `setLiquidityFeePercent()` : the owner of the contract can set the percentage of liquidity fee.
- `setMaxTxPercent()` : the owner of the contract can set the maximum transaction amount.
- `setRouterAddress()` : the owner of the contract can set any arbitrary address as the router address.
- `setNumTokensSellToAddToLiquidity()` : the owner of the contract can set the threshold to trigger liquidity-adding process.

Any compromise to the `_owner()` account may allow the hacker to take advantage of this and modify the significant state of the contract, thus introducing centralization risk.

03 | CentralizationRisk in Contract **function** **getMultipliedFee()**

Description

This report has been prepared for RISE CORE to discover issues and vulnerabilities in the source code of the RISE CORE project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed utilizing Manual Review and Static Analysis techniques.

```
function shouldTakeFee(address sender) internal view returns (bool) {
    return !isFeeExempt[sender];
}

function getTotalFee(bool selling) public view returns (uint256) {
    if(launchedAt + 1 >= block.number){ return feeDenominator.sub(1); }
    if(selling){ return getMultipliedFee(); }
    return totalFee;
}

function getMultipliedFee() public view returns (uint256) {
    if (launchedAtTimestamp + 1 days > block.timestamp) {
        return totalFee.mul(18000).div(feeDenominator);
    } else if (buybackMultiplierTriggeredAt.add(buybackMultiplierLength) > block.timestamp) {
        uint256 remainingTime = buybackMultiplierTriggeredAt.add(buybackMultiplierLength).sub(block.timestamp);
        uint256 feeIncrease = totalFee.mul(buybackMultiplierNumerator).div(buybackMultiplierDenominator).sub(totalFee);
        return totalFee.add(feeIncrease.mul(remainingTime).div(buybackMultiplierLength));
    }
    return totalFee;
}
```

The security assessment resulted in findings that **function getMultipliedFee()**. We recommend addressing these findings to ensure a medium level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective.

04 | SetAllowance()

Description

setAllowance() currently poses a risk of a race condition. Consider the scenario

- operator() is allowed to applyFunds() up to 1000
- dao wants to increase the allowance by 100 and calls setAllowance(1100)
- However before this transaction was executed operator calls applyFunds(1000)
- After the allowance is increased operator calls applyFunds(1100)
- In total operator has spent which was not expected by dao

Recommendation

To prevent a possible race condition we recommend introducing increaseAllowance() and decreaseAllowance()

05 | Initial Token Distribution

Category	Severity	Location	Status
Logical Issue	● Medium	projects/contract.sol (98ba012): 817	① Acknowledged

```
uint256 liquidityFee = 0;  
uint256 buybackFee = 200;  
uint256 reflectionFee = 0;  
uint256 marketingFee = 300;  
uint256 totalFee = 500;  
uint256 feeDenominator = 10000;
```


Description

All of the tokens are sent to the contract deployer when deploying the contract. This could be a centralization risk as the deployer can distribute those tokens without obtaining the consensus of the community.

Recommendation

We recommend the team to be transparent regarding the initial token distribution process.

06 | Lack of Return Value Handling

Category	Severity	Location	Status
Volatile Code	Minor	projects/contract.sol (98ba012): 843	 Acknowledged

Description

The return values of function `returnValue()` are properly handled.

```
function takeFee(address sender, address receiver, uint256 amount) internal returns (uint256) {
    uint256 feeAmount = amount.mul(getTotalFee(receiver == pair)).div(feeDenominator);

    _balances[address(this)] = _balances[address(this)].add(feeAmount);
    emit Transfer(sender, address(this), feeAmount);

    return amount.sub(feeAmount);
}

function shouldSwapBack() internal view returns (bool) {
    return msg.sender != pair
        && !inSwap
        && swapEnabled
        && _balances[address(this)] >= swapThreshold;
}

function swapBack() internal swapping {
    uint256 dynamicLiquidityFee = isOverLiquified(targetLiquidity, targetLiquidityDenominator) ? 0 : liquidityFee;
    uint256 amountToLiquify = swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
    uint256 amountToSwap = swapThreshold.sub(amountToLiquify);

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = WBNB;
    uint256 balanceBefore = address(this).balance;
```

Recommendation

We recommend using variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

07 | UNLOCKED COMPILER VERSION

Language Specific	<div> <div></div> Informational </div>	contracts/AirdropPool.sol (base): <u>1</u> ; contracts/CORESales.sol (base): <u>1</u> ; contracts/TeamVesting.sol (base): <u>1</u> ; contracts/lib/SafeMath.sol (base): <u>1</u>	<div> <div></div> Resolved </div>
-------------------	--	--	---

Description

The contracts cited have an unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This in turn leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We recommend the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version v0.8.0 the contract should contain the following line

```
pragma solidity ^0.8.0;

/**
 * SAFEMATH LIBRARY
 */
library SafeMath {

    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            uint256 c = a + b;
            if (c < a) return (false, 0);
            return (true, c);
        }
    }

    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
        unchecked {
            if (b > a) return (false, 0);
            return (true, a - b);
        }
    }
}
```

08 | Lack of Error Message

Category	Severity	Location	Status
Coding Style	● Informational	projects/contract.sol (98ba012): 560	📄 Acknowledged

Description

The require statement can be used to check for conditions and throw an exception if the condition is not met. It is better to provide a string message containing details about the error that will be passed back to the caller.

Recommendation

We advise refactoring the linked codes as below:

```
interface IDEXFactory {
    function createPair(address tokenA, address tokenB) external returns (address pair);
}

interface IDEXRouter {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);

    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
}
```

09 | Redundant Code

Category	Severity	Location	Status
Logical Issue	● Informational	projects/contract.sol (98ba012): 862	① Acknowledged

Description

The condition! `_isExcluded[sender] & !_isExcluded[recipient]` can be included in `else` .

Recommendation

The following code can be removed:

```
function approveMax(address spender) external returns (bool) {
    return approve(spender, type(uint256).max);
}

function transfer(address recipient, uint256 amount) external override returns (bool) {
    return _transferFrom(msg.sender, recipient, amount);
}

function transferFrom(address sender, address recipient, uint256 amount) external override returns (bool) {
    if(_allowances[sender][msg.sender] != type(uint256).max){
        _allowances[sender][msg.sender] = _allowances[sender][msg.sender].sub(amount, "Insufficient Allowance");
    }

    return _transferFrom(sender, recipient, amount);
}
```

10 | Typos In The Contract

Category	Severity	Location	Status
Coding Style	● Informational	projects/contract.sol (98ba012): 470, 670	ⓘ Acknowledged

Description

There are several typos in the code and comments.

1. In the following code snippet, `tokensIntoLiquidity()` should be `tokensIntoLiquidity()`

```
contract CoreRise is IBEP20, Auth {
    using SafeMath for uint256;

    uint256 public constant MASK = type(uint128).max;
    address BUSD = 0x9Ebab27608bD64AFf36f027049aECC69102a0D1e;
    address public WBNB = 0x40375C92d9FAf44d2f9db9Bd9ba41a3317a2404f;
    address DEAD = 0x00000000000000000000000000000000dEaD;
    address ZERO = 0x000000000000000000000000000000000000;
    address DEAD_NON_CHECKSUM = 0x00000000000000000000000000000000dEaD;

    string constant _name = "Core Rise";
    string constant _symbol = "CRISE";
    uint8 constant _decimals = 9;

    uint256 _totalSupply = 1_000_000_000_000_000 * (10 ** _decimals);
    uint256 public _maxTxAmount = _totalSupply.div(20); // 5%

    mapping (address => uint256) _balances;
    mapping (address => mapping (address => uint256)) _allowances;

    mapping (address => bool) isFeeExempt;
    mapping (address => bool) isTxLimitExempt;
    mapping (address => bool) isDividendExempt;

    uint256 liquidityFee = 0;
    uint256 buybackFee = 200;
    uint256 reflectionFee = 0;
    uint256 marketingFee = 300;
    uint256 totalFee = 500;
    uint256 feeDenominator = 10000;

    address public autoLiquidityReceiver;
    address public marketingFeeReceiver;

    uint256 targetLiquidity = 25;
    uint256 targetLiquidityDenominator = 100;
```

2. `recieve()` should be `recieve()` `_swapping()` should be `_swapping()` in the line of comment `//to _recieve ETH from coreV2Router when swaping()` .

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different requirements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without FusionTech prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts FusionTech to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. FusionTech's position is that each company and individual are responsible for their own due diligence and continuous security.

FusionTech's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by FusionTech is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, FusionTech HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, FusionTech SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, FusionTech MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, FusionTech PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER FusionTech NOR ANY OF FusionTech’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. FusionTech WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT FusionTech’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2022 by leading academics in the field of Computer Science, FusionTech is going to be a leading blockchain security company that serves to verify the security and correctness of smart contracts KYC and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

**FUSION TECH**