



Rabbit Core Finance

Security Assessment

www.rabbitcore.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 ``CoinToken``

03 : Contract Gains Non-withdrawable CORE via the ``swapandliquify``

04 : Regaining Ownership After Renouncing the Contract Ownership

05 : Initial Token Distribution

06 : Lack of Return Value Handling

07 : PotentialSandwich Attacks

08 : Lack of Error Message

09 : Redundant Code

10 : Typos In The Contract

11 : Function and Variable Naming Doesn't Match the Operating Environment

12 : Potential ResourceExhaustion

13 : Inconsistency BetweenComment and Code

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	Rabbit Core - (https://www.rabbitcore.finance/)
Platform	CORE DAO
Language	Solidity
Codebase	https://scan.coredao.org/address/0xf46346B737bA8E4e36d2D3A0f9593C083df92E83
Commit	87cc90787612fd4537ftf5b845nv543g7l10i472

Audit Summary

Delivery Date	March,3 2023
Audit Methodology	Static Analysis, Manual Review
Key Components	CoreToken

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	0
● Minor	3	0	0	3	0	0
● Informational	6	0	0	6	0	0
● Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
CKP	contract.sol	f79198f1e334d2889b0de0d9507c2bf3e16e6299f37d30102d9496b69c383809

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.

- `excludeFromReward() / includeInReward()`
- `excludeFromFee() / includeInFee()`
- `setTaxFeePercent()`
- `setAllowance()`
- `setLiquidityFeePercent()`
- `setMaxTxPercent()`
- `setDevWalletAddress()`
- `setSwapAndLiquifyEnabled()`
- `setRouterAddress()`
- `setNumTokensSellToAddToLiquidity()`

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 `addLiquidity()_hasLiqBeenAdded()` function calls the `uniswapV2Router.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 `uniswapV2Router.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

Category	Severity	Location	Status
Centralization / Privilege	● Major	projects/contract.sol (98ba012): 603, 640, 644, 648, 652, 656, 660, 665, 906, 912, 612, 636	① Acknowledged

Description

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

- `excludeFromReward()` / `includeInReward()` ; the owner of the contract can exclude/include an account from/in rewards.
- `excludeFromFee()` / `includeInFee()` : the owner of the contract can exclude/include an account from/in fee.
- `setTaxFeePercent()` : the owner of the contract can set the percentage of the tax fee.
- `setAllowance()` : the owner of the contract can set the Allowance.
- `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.
- `setDevWalletAddress()` : the owner of the contract can update the arbitrary address.
- `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 | Contract Gains Non-withdrawable CORE via the owner Function

Function

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

Description

The `swapAndLiquify()` function converts half of the `contractTokenBalance()` tokens to CORE. The other half of the tokens and part of the converted CORE are deposited into the corresponding pool on pancakeswap as liquidity. For every `swap&liquify()` function call, a small amount of CORE leftover in the contract. This is due to the price of drops after swapping the first half of tokens into COREs, and the other half of tokens require less than the converted CORE to be paired with it when adding liquidity. The contract doesn't appear to provide a way to withdraw those CORE, and they will be locked in the contract forever.

Recommendation

It's not ideal that more and more CORE are locked into the contract over time. The simplest solution is to add a `withdraw()` function in the contract to withdraw CORE. Other approaches that benefit the token holders can be:

- Distribute CORE to token holders proportional to the amount of token they hold.
- Use leftover CORE to buy back tokens from the market to increase the token price.

04 | Regaining Ownership After Renouncing the Contract Ownership

Category	Severity	Location	Status
Logical Issue	● Medium	projects/contract.sol (98ba012): 243	ⓘ Acknowledged

Description

Generally, renouncing the ownership should leave the contract without an owner, thereby removing any functionality that is only available owner to the owner. However, the owner of the cointoken is possible to gain ownership of the contract again even if the owner has called the function `renounceOwnership()` is possible to gain to `renounce()` the ownership. This can be achieved by performing the following operations:

- Call `lock()` Call to lock the contract. The variable `_previousowner()` to unlock the contract.
- Call `unlock()` to unlock the contract
- would be set to the current owner.
- Call `renounce()` to renounce the contract ownership. to regain ownership.
- Call `unlock()` to gain the ownership

Recommendation

We advise the client to review the logic and ensure if it is the intended design. If timelock functionality should be introduced, we recommend using the implementation of Compound finance as reference.

05 | Initial Token Distribution

Category	Severity	Location	Status
Logical Issue	● Minor	projects/contract.sol (98ba012): 497	📄 Acknowledged

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.

```
contract RabbitCore is IERC20, Ownable {
    using SafeMath for uint256;
    string private constant _name = 'Rabbit Core';
    string private constant _symbol = 'RBC';
    uint8 private constant _decimals = 6;
    uint256 private _totalSupply = 100 * (10**6) * (10 ** _decimals);
    uint256 private _maxTxAmountPercent = 10; // /1000
    uint256 private _maxWalletPercent = 20;
    mapping (address => uint256) _balances;
    mapping (address => mapping (address => uint256)) private _allowances;
    mapping (address => bool) public isExcludedFromFees;
    IRouter router;
    address public pair;
```

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 `addLiquidityETH()` are properly handled.

```
constructor(address routerAddr, address mkt) Ownable(msg.sender) {
    IRouter _router = IRouter(routerAddr);
    address _pair = IFactory(_router.factory()).createPair(address(this), _router.WETH());
    router = _router;
    pair = _pair;
    marketing = mkt;
    isExcludedFromFees[address(this)] = true;
    isExcludedFromFees[marketing] = true;
    isExcludedFromFees[msg.sender] = true;
    _balances[msg.sender] = _totalSupply;
    emit Transfer(address(0), msg.sender, _totalSupply);
}
```

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 | Potential Sandwich Attacks

Category	Severity	Location	Status
Logical Issue	● Minor	projects/contract.sol (98ba012): 832~838, 843~850	① Acknowledged

Description

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction being attacked) a transaction to purchase one of the assets and make profits by backrunning (after the transaction being attacked) a transaction to sell the asset.

The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:

```
checkTx(sender, recipient, amount);
checkMaxWallet(sender, recipient, amount);
swapCounters(sender, recipient);
checkTxLimit(sender, recipient, amount);
swapBack(sender, recipient, amount);
_balances[sender] = _balances[sender].sub(amount);
uint256 amountReceived = shouldTakeFee(sender, recipient) ? takeFee(sender, recipient, amount) : amount;
_balances[recipient] = _balances[recipient].add(amountReceived);
emit Transfer(sender, recipient, amountReceived);
```

```
function shouldSwapBack(address sender, address recipient, uint256 amount) internal view returns (bool) {
    bool aboveMin = amount >= _minTokenAmount;
    bool aboveThreshold = balanceOf(address(this)) >= swapThreshold;
    return !swapping && swapEnabled && aboveMin && !isExcludedFromFees[sender] && recipient == pair && swapTimes >= uint256(2) && aboveThreshold;
}

function swapBack(address sender, address recipient, uint256 amount) internal {
    if(shouldSwapBack(sender, recipient, amount)){swapBack(swapThreshold); swapTimes = uint256(0);}
}

function shouldTakeFee(address sender, address recipient) internal view returns (bool) {
    return !isExcludedFromFees[sender] && !isExcludedFromFees[recipient];
}

function getTotalFee(address sender, address recipient) internal view returns (uint256) {
    if(recipient == pair){return sellFee;}
    if(sender == pair){return buyFee;}
    return sellFee;
}
```

Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the aforementioned functions.

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:

```
560         _approve(_msgSender(), spender, _allowances[_msgSender()])
[spender].add(addedValue, "increase allowance overflow");
```

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:

```

861 ... else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
862     _transferStandard(sender, recipient, amount);
863 } ...

```

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()`

```
1 event SwapAndLiquify(  
2     uint256 tokensSwapped,  
3     uint256 ethReceived,  
4     uint256 tokensIntoLiquidity  
5 );
```

2. `recieve()` should be `recieve()` `_swapping()` should be `_swapping()` in the line of comment `//to _recieve ETH from uniswapV2Router when swaping()` .

Recommendation

We recommend correcting all typos in the contract.

11 | Function and Variable Naming Doesn't Match the Operating Environment

Category	Severity	Location	Status
Coding Style	● Informational	projects/contract.sol (98ba012): 1	① Acknowledged

Description

There are multiplenames inside the current contract, which can be misleading to use `uniswap()` and `ETH()` instead of `pancakeswap()` and `CORE()` if the project landing on CORE DAO.

For example, the `cointoken()` contract uses `pancakeswap()` for swapping and adding liquidity to the Pancakeswap pool but names it `uniswap()`

Recommendation

Change "Uniswap" and "ETH" to "Pancakeswap" and "CORE" in the contract respectively to match the operating environment and avoid confusion.

12 | Potential Resource Exhaustion

Category	Severity	Location	Status
Logical Issue	● Informational	projects/contract.sol (98ba012): 614, 709	① Acknowledged

Description

The `farloop()` within functions `includeInReward(address)` and `_getCurrentSupply()` takes the variable `_excluded.length()`, as the maximal iteration times. If the size of the array is very large, it could exceed the gas limit to execute the functions. In this case, the contract might suffer from DoS (Denial of Service) situation.

Recommendation

We recommend the team review the design and ensure this would not cause loss to the project.

13 | Inconsistency Between Comment and Code

Category	Severity	Location	Status
Inconsistency	● Informational	projects/contract.sol (98ba012): 230~236	📄 Acknowledged

Description

According to the comment in L238, the `lock()` function will lock the contract for a given time period. However, the code implementation will lock the contract until the given timestamp.

```
238 //Unlocks the contract for owner when _lockTime is exceeds
239 function unlock() public virtual {
240     require(_previousOwner == msg.sender, "You don't have permission to
unlock.");
241     require(block.timestamp > _lockTime, "Contract is locked.");
242     emit OwnershipTransferred(_owner, _previousOwner);
243     _owner = _previousOwner;
244 }
```

Recommendation

We recommend the team review the design and update either comments or code implementation to ensure consistent logic between code and comment.

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**