Cyberscope

Audit Report

Katheer

April 2023

Network BSC

Address 0x1482288A53c704f598A65D69Ee503aDA6cE96749

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Findings Breakdown 4
Roles 5
Analysis 6
Diagnostics 7
RF - Redundant Function 8
Description 8
Recommendation 8
MMN - Misleading Method Naming 9
Description 9
Recommendation 9
RSML - Redundant SafeMath Library 10
Description 10
Recommendation 10
RSK - Redundant Storage Keyword 11
Description 11
Recommendation 11
L02 - State Variables could be Declared Constant 12
Description 12
Recommendation 12
L04 - Conformance to Solidity Naming Conventions 13
Description 13
Recommendation 14
L08 - Tautology or Contradiction 15
Description 15
Recommendation 15
L09 - Dead Code Elimination 16
Description 16
Recommendation 16
L11 - Unnecessary Boolean equality 17
Description 17
Recommendation 17
L13 - Divide before Multiply Operation 18
Description 18
Recommendation 18
L15 - Local Scope Variable Shadowing 19

Description 19
Recommendation 19
L16 - Validate Variable Setters 20
Description 20
Recommendation 20
L17 - Usage of Solidity Assembly 21
Description 21
Recommendation 21
L19 - Stable Compiler Version 22
Description 22
Recommendation 22
Functions Analysis **23**
Inheritance Graph **30**
Flow Graph **31**
Summary **32**
Disclaimer **33**
About Cyberscope **34**

Review

Contract Name	KatheerToken
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://bscscan.com/address/0x1482288a53c704f598a65d69ee503ada6ce96749
Address	0x1482288a53c704f598a65d69ee503ada6ce96749
Network	BSC
Symbol	KAH
Decimals	6
Total Supply	10.000.000.000

Audit Updates

Initial Audit	23 Apr 2023

Source Files

Filename	SHA256
katheertoken-imports.sol	0c34c11f446ac0cca9edcde22c02a6bece79422100a2ac286ab75d1eb48d3c69
katheertoken.sol	1fc942c7f1d534dfaf9d4907fb2390e0667d45b99dcc3ac22a9517dadc11b512

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

Roles

The contact roles consist of the owner and the manager role.

The `owner` is responsible:

- Renounce or transfer ownership.
- Lock or unlock the owner's ownership.
- Exldude or include users from rewards and fees.

The `manager` has the authority to configure the manager role.

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	RF	Redundant Function	Unresolved
	MMN	Misleading Method Naming	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSK	Redundant Storage Keyword	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L08	Tautology or Contradiction	Unresolved
	L09	Dead Code Elimination	Unresolved
	L11	Unnecessary Boolean equality	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved
	L16	Validate Variable Setters	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved

RF - Redundant Function

Criticality	Minor / Informative
Location	katheertoken.sol#L632,645
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The function `_takeFeeToETH` is redundant since it calls the `_takeFee` method.

```
function _takeFee(uint256 amount, uint256 currentRate, uint256 fee,
address recipient, uint256 index) private {

    uint256 tAmount = amount.mul(fee).div(FEES_DIVISOR);
    uint256 rAmount = tAmount.mul(currentRate);

    _reflectedBalances[recipient] =
_reflectedBalances[recipient].add(rAmount);
    if(_isExcludedFromRewards[recipient])
        _balances[recipient] = _balances[recipient].add(tAmount);

    _addFeeCollectedAmount(index, tAmount);
}


function _takeFeeToETH(uint256 amount, uint256 currentRate, uint256
fee, address recipient, uint256 index) private {
    _takeFee(amount, currentRate, fee, recipient, index);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	katheertoken.sol#L645
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The `_takeFeeToETH` function does not transfer ETH.

```
function _takeFeeToETH(uint256 amount, uint256 currentRate,
uint256 fee, address recipient, uint256 index) private {
    _takeFee(amount, currentRate, fee, recipient, index);
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	katheertoken-imports.solkatheertoken.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, and overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	katheertoken.sol#L73
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Fee storage
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	katheertoken.sol#L31,32,34,36
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
address internal developmentAddress =
0xD07884EA2340321A22238ad7b915617796d0c925
address internal marketingAddress =
0x1D1D74D6cCca51d5403C58F982bB3358F42aD246
address internal burnAddress =
0x000000000000000000000000000000000000dEaD
address internal tipToTheDev =
0xD07884EA2340321A22238ad7b915617796d0c925
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	katheertoken.sol#L23,25,27,29,105,106,107,109,110,399,400katheertoken-imports.sol#L146
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
uint256 internal _reflectedSupply = (MAX - (MAX %
TOTAL_SUPPLY))
uint256 internal constant maxTransactionAmount = TOTAL_SUPPLY /
200
uint256 internal constant maxWalletBalance = TOTAL_SUPPLY / 1
uint256 internal constant numberOfTokensToSwapToLiquidity =
TOTAL_SUPPLY / 200
mapping (address => uint256) internal _reflectedBalances
mapping (address => uint256) internal _balances
mapping (address => mapping (address => uint256)) internal
_allowances
mapping (address => bool) internal _isExcludedFromFee
mapping (address => bool) internal _isExcludedFromRewards
IPancakeV2Router internal _router
address internal _pair
function WETH() external pure returns (address);
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	katheertoken.sol#L74
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require( index >= 0 && index < fees.length,
"FeesSettings._getFeeStruct: Fee index out of bounds")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	katheertoken.sol#L53,58,87katheertoken-imports.sol#L36,37,41,42,43,44,50,53,58,61,66
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _addFee(FeeType name, uint256 value, address
recipient) private {
        fees.push( Fee(name, value, recipient, 0 ) );
        sumOfFees += value;
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	katheertoken-imports.sol#L116
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(_renounced == false, "Cannot unlock renounced
ownership")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	katheertoken.sol#L335,339,370,371,625,626,634,635
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 tBurn = amount.mul(fee).div(FEES_DIVISOR)
uint256 rBurn = tBurn.mul(currentRate)
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	katheertoken.sol#L310,327,570,602
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
uint256 sumOfFees = _getSumOfFees(sender, amount)
uint256 sumOfFees
Env _env
FeeType name
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	katheertoken-imports.sol#L137
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
_manager = newManager
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	katheertoken-imports.sol#L36,69
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	katheertoken.sol#L4katheertoken-imports.sol#L4
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
SafeMath	Library			

	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-

	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	getUnlockTime	Public		-
	lock	Public	✓	onlyOwner
	unlock	Public	✓	-
Manageable	Implementation	Context		
		Public	✓	-
	manager	Public		-
	transferManagement	External	✓	onlyManager
IPancakeV2Factory	Interface			
	createPair	External	✓	-
IPancakeV2Router	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Tokenomics	Implementation			
		Public	✓	-
	_addFee	Private	✓	

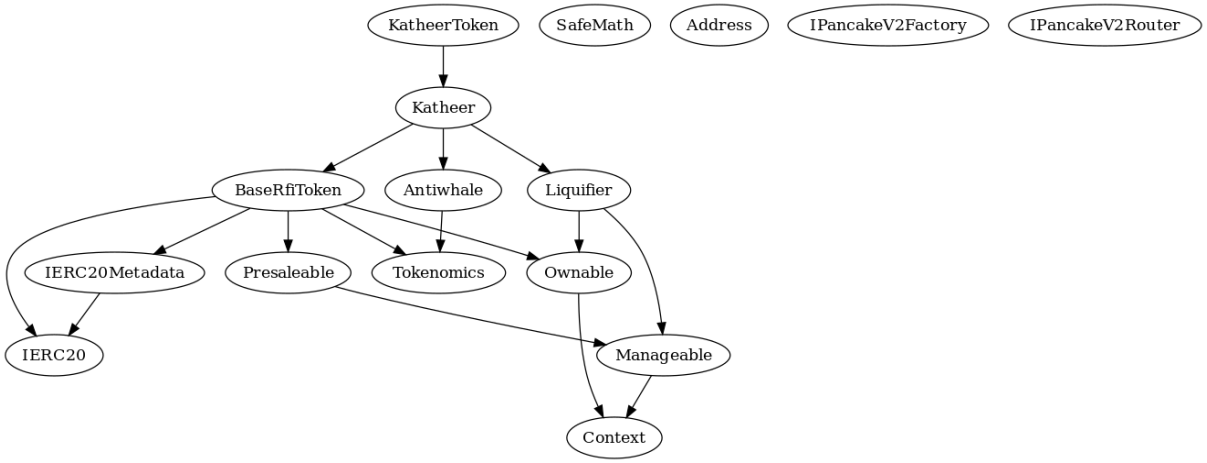
	_addFees	Private	✓	
	_getFeesCount	Internal		
	_getFeeStruct	Private		
	_getFee	Internal		
	_addFeeCollectedAmount	Internal	✓	
	getCollectedFeeTotal	Internal		
Presaleable	Implementation	Manageable		
	setPreseableEnabled	External	✓	onlyManager
BaseRfiToken	Implementation	IERC20, IERC20Metadata, Ownable, Presaleable, Tokenomics		
		Public	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	Public		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
	burn	External	✓	-

	_burnTokens	Internal	✓	
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	External		-
	reflectionFromToken	External		-
	tokenFromReflection	Internal		
	excludeFromReward	External	✓	onlyOwner
	_exclude	Internal	✓	
	includeInReward	External	✓	onlyOwner
	setExcludedFromFee	External	✓	onlyOwner
	isExcludedFromFee	Public		-
	_approve	Internal	✓	
	_isUnlimitedSender	Internal		
	_isUnlimitedRecipient	Internal		
	_transfer	Private	✓	
	_transferTokens	Private	✓	
	_takeFees	Private	✓	
	_getValues	Internal		
	_getCurrentRate	Internal		
	_getCurrentSupply	Internal		
	_beforeTokenTransfer	Internal	✓	
	_getSumOfFees	Internal		
	_isV2Pair	Internal		

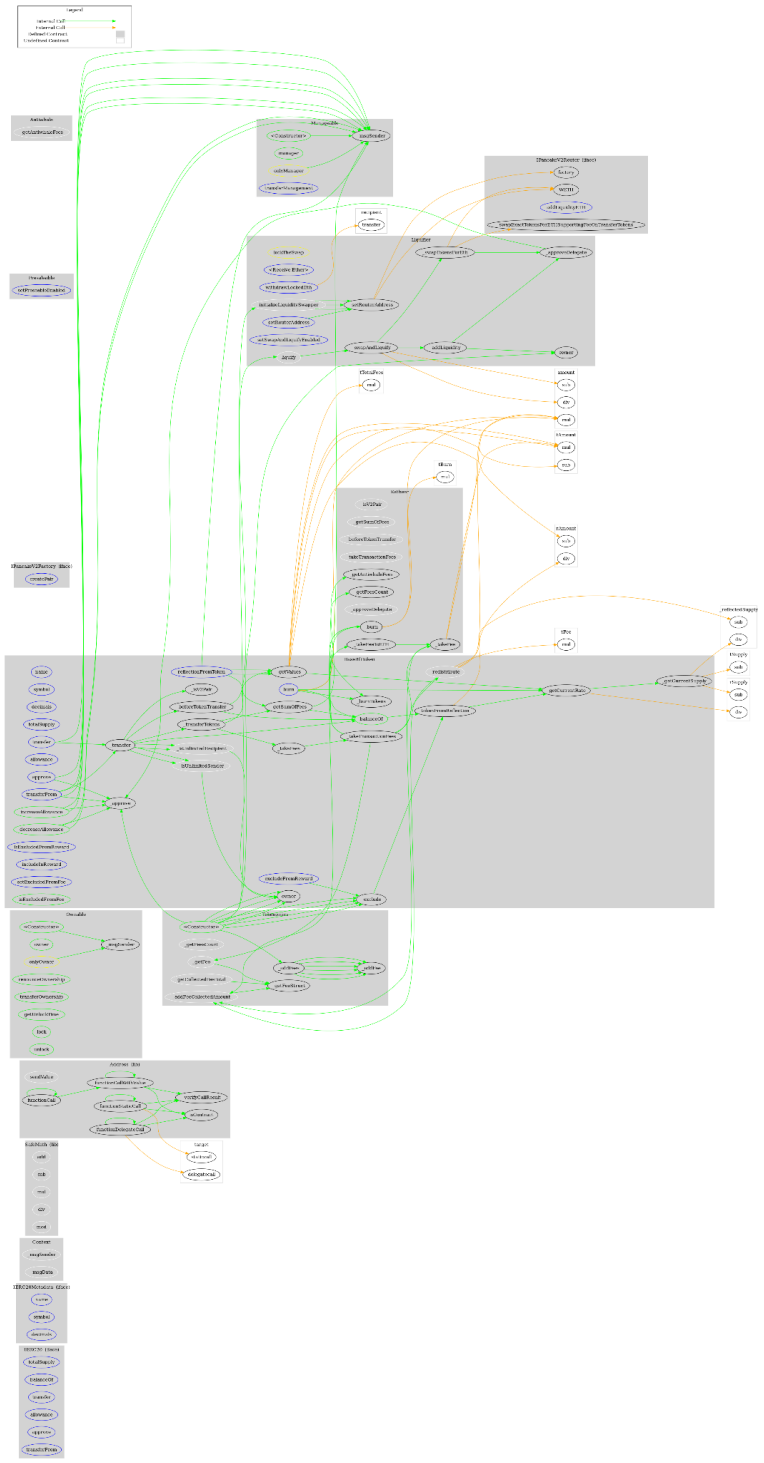
	_redistribute	Internal	✓	
	_takeTransactionFees	Internal	✓	
Liquifier	Implementation	Ownable, Manageable		
		External	Payable	-
	initializeLiquiditySwapper	Internal	✓	
	liquify	Internal	✓	
	_setRouterAddress	Private	✓	
	_swapAndLiquify	Private	✓	lockTheSwap
	_swapTokensForEth	Private	✓	
	_addLiquidity	Private	✓	
	setRouterAddress	External	✓	onlyManager
	setSwapAndLiquifyEnabled	External	✓	onlyManager
	withdrawLockedEth	External	✓	onlyManager
	_approveDelegate	Internal	✓	
Antiwhale	Implementation	Tokenomics		
	_getAntiwhaleFees	Internal		
Katheer	Implementation	BaseRfiToken, Liquifier, Antiwhale		
		Public	✓	-
	_isV2Pair	Internal		
	_getSumOfFees	Internal		

	_beforeTokenTransfer	Internal	✓	
	_takeTransactionFees	Internal	✓	
	_burn	Private	✓	
	_takeFee	Private	✓	
	_takeFeeToETH	Private	✓	
	_approveDelegate	Internal	✓	
KatheerToken	Implementation	Katheer		
		Public	✓	Katheer

Inheritance Graph



Flow Graph



Summary

Katheer contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Katheer is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors or critical issues.

The Contract Owner and the manager can access some admin functions that can not be used in a malicious way to disturb the users' transactions. Additionally, the contract has an owner lock mechanism. There is also a limit of max 5% fee.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. 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 Cyberscope's prior written consent. 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 regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document 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 represents an extensive assessment 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 Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope'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 Cyberscope are subject to dependencies and are 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.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io