Cyberscope

Audit Report

RAFL

April 2024

SHA256 c36c91172f0dfb90efb7b0f8bae035e58505c5b596039a89143c6c59547ec842

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	IFE	Ineffective Fee Exclusion	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	DFD	Disproportional Fees Distribution	Unresolved
Minor / Informative	DVO	Duplicate Variable Optimization	Unresolved
Minor / Informative	EOO	Execution Order Optimization	Unresolved
Minor / Informative	GO	Gas Optimization	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	PBV	Percentage Boundaries Validation	Unresolved
Minor / Informative	RFO	Redundant Function Override	Unresolved
Minor / Informative	RSW	Redundant Storage Writes	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L05	Unused State Variable	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **4**
Review **6**
Audit Updates 6
Source Files 6
Findings Breakdown **7**
ST - Stops Transactions 8
Description 8
Recommendation 9
ELFM - Exceeds Fees Limit 10
Description 10
Recommendation 11
MT - Mints Tokens 12
Description 12
Recommendation 13
IFE - Ineffective Fee Exclusion 14
Description 14
Recommendation 14
CR - Code Repetition 15
Description 15
Recommendation 16
DFD - Disproportional Fees Distribution 17
Description 17
Recommendation 17
DVO - Duplicate Variable Optimization 18
Description 18
Recommendation 18
EOO - Execution Order Optimization 19
Description 19
Recommendation 19
GO - Gas Optimization 20
Description 20
Recommendation 20
IDI - Immutable Declaration Improvement 21
Description 21
Recommendation 21
MEE - Missing Events Emission 22
Description 22

Recommendation 22
PBV - Percentage Boundaries Validation 23
Description 23
Recommendation 23
RFO - Redundant Function Override 24
Description 24
Recommendation 24
RSW - Redundant Storage Writes 25
Description 25
Recommendation 25
L04 - Conformance to Solidity Naming Conventions 26
Description 26
Recommendation 27
L05 - Unused State Variable 28
Description 28
Recommendation 28
L09 - Dead Code Elimination 29
Description 29
Recommendation 30
L16 - Validate Variable Setters 31
Description 31
Recommendation 31
L17 - Usage of Solidity Assembly 32
Description 32
Recommendation 32
L19 - Stable Compiler Version 33
Description 33
Recommendation 33
Functions Analysis 34
Inheritance Graph 37
Flow Graph 38
Summary 39
Disclaimer 40
About Cyberscope 41

Review

Badge Eligibility	Must Fix Criticals

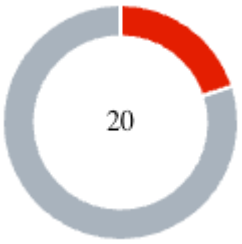
Audit Updates

Initial Audit	11 Apr 2024

Source Files

Filename	SHA256
ReflectiveERC20.sol	2fc4a8462a586d98c9237b71c725efecf265cbbddf58cd0afa7edb9ca593d6eb
RAFLToken.sol	c36c91172f0dfb90efb7b0f8bae035e58505c5b596039a89143c6c59547ec842
lib/LibCommon.sol	f09868b4d38dff45900eb0c57921d597389d106b01fda5e2ac0eb885c9544065

Findings Breakdown



Critical	4
Medium	0
Minor / Informative	16

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	0	0	0	0
Minor / Informative	16	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	RAFLToken.sol#L389 ReflectiveERC20.sol#L152
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the `maxTokenAmountPerAddress` to a very small value.

```
if (isMaxAmountOfTokensSet()) {
  if (balanceOf(to) + amountToTransfer > maxTokenAmountPerAddress) {
    revert DestBalanceExceedsMaxAllowed(to);
  }
}
```

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the `isReflective` to true and the `deflationBPS` greater than 0.

```
function _burn(address account, uint256 value) internal override {
  if (isReflective) {
    revert BurningNotEnabled();
  } else {
    super._burn(account, value);
  }
}
```

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the `tFeeBPS` to a large value.

```
uint256 tFee = calculateFee(tAmount);
uint256 tTransferAmount = tAmount - tFee;
```

Recommendation

The contract could embody a check for not allowing setting the `maxTokenAmountPerAddress` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	RAFLToken.sol#L298
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setReflectionConfig` function with a high percentage value.

```
function setReflectionConfig(uint256 _feeBPS) external onlyOwner {
  if (!isReflective()) {
    revert TokenIsNotReflective();
  }
  super._setReflectionFee(_feeBPS);

  emit ReflectionConfigSet(_feeBPS);
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

MT - Mints Tokens

Criticality	Critical
Location	RAFLToken.sol#L445
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address to, uint256 amount) external onlyOwner {
  if (!isMintable()) {
    revert MintingNotEnabled();
  }
  if (isMaxAmountOfTokensSet()) {
    if (balanceOf(to) + amount > maxTokenAmountPerAddress) {
      revert DestBalanceExceedsMaxAllowed(to);
    }
  }
  if (isMaxSupplySet()) {
    if (totalSupply() + amount > maxTotalSupply) {
      revert TotalSupplyExceedsMaxAllowedAmount();
    }
  }

  super._mint(to, amount);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

IFE - Ineffective Fee Exclusion

Criticality	Critical
Location	RAFLToken.sol#L420 ReflectiveERC20.sol#L195,196
Status	Unresolved

Description

The contract maintains a list of addresses that should be excluded from the fee mechanism, providing flexibility and customization options for certain users. However, the inherited `transfer` and `transferFrom` functions internally deduct fees from the transferred amount, regardless of whether the involved addresses are excluded from fees or not. This inconsistency between excluded addresses and internal fee deduction could lead to confusion among authorized users and diminish the code quality and trustworthiness of the contract.

```
if ( isExcludedFromFee(from) || isExcludedFromFee(to) ) {
  return super.transferFrom(from, to, amount);
}

uint256 tFee = calculateFee(tAmount);
uint256 tTransferAmount = tAmount - tFee;
```

Recommendation

To ensure consistency and clarity in the fee mechanism, it is recommended to align the internal fee deduction logic with the list of addresses excluded from fees. By aligning the internal fee deduction logic with the list of excluded addresses and enhancing the contract documentation, users can have a clearer understanding of the fee mechanism, improving trust and confidence in the contract's functionality.

CR - Code Repetition

Criticality	Minor / Informative
Location	RAFLToken.sol#L378,415
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
uint256 taxAmount = _taxAmount(from, amount);
uint256 deflationAmount = _deflationAmount(amount);
uint256 amountToTransfer = amount - taxAmount - deflationAmount;

// Bypass taxes for whitelisted wallets (both sender, and receiver)
if ( isExcludedFromFee(from) || isExcludedFromFee(to) ) {
  return super.transferFrom(from, to, amount);
}

if (isMaxAmountOfTokensSet()) {
  if (balanceOf(to) + amountToTransfer > maxTokenAmountPerAddress) {
    revert DestBalanceExceedsMaxAllowed(to);
  }
}

if (taxAmount != 0) {
  _transferNonReflectedTax(from, taxAddress, taxAmount);
}

if (deflationAmount != 0) {
  _burn(from, deflationAmount);
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

DFD - Disproportional Fees Distribution

Criticality	Minor / Informative
Location	ReflectiveERC20.sol#L196 RAFLToken.sol#L380
Status	Unresolved

Description

The smart contract applies fees on token transfers. However, instead of sharing the initial fee amounts proportionally, it first calculates the tax and deflation fee from the transferred amount and then applies the remaining fees to the reduced amount. This approach leads to a disproportional distribution of fees, potentially impacting the intended distribution model and investor expectations.

```
uint256 taxAmount = _taxAmount(msg.sender, amount);
uint256 deflationAmount = _deflationAmount(amount);
uint256 amountToTransfer = amount - taxAmount - deflationAmount;
...
uint256 tFee = calculateFee(tAmount);
uint256 tTransferAmount = tAmount - tFee;
```

Recommendation

The team is adviced to review and revise the fee distribution mechanism to ensure proportional allocation according to the defined tokenomics. Consider implementing a consistent approach that distributes fees based on the initial transfer amount without prior deduction for specific fees.

DVO - Duplicate Variable Optimization

Criticality	Minor / Informative
Location	RAFLToken.sol#L14 ReflectiveERC20.sol#L12
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 contract introduces two different variables that are used for the same purpose. Specifically, the `MAX_BPS_AMOUNT` and `BPS_DIVISOR` variables are used by the contract as denominators for calculating fee amounts. Both variables have the same value. As a result, the contract introduces redundancy and consumes more storage.

```
uint256 private constant MAX_BPS_AMOUNT = 10_000;
uint256 private constant BPS_DIVISOR = 10_000;
```

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.

EOO - Execution Order Optimization

Criticality	Minor / Informative
Location	RAFLToken.sol#L415,416,417
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 contract implements a fee mechanism on each transaction, excluding authorized addresses. However, the fee calculations are executed at the start of the transfer flow, even when the sender or recipient are exempt from fees. This approach results in gas inefficiency, as unnecessary calculations are performed for exempt addresses, consuming additional gas without providing any benefit.

```
uint256 taxAmount = _taxAmount(from, amount);
uint256 deflationAmount = _deflationAmount(amount);
uint256 amountToTransfer = amount - taxAmount - deflationAmount;
```

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.

GO - Gas Optimization

Criticality	Minor / Informative
Location	RAFLToken.sol#L201
Status	Unresolved

Description

Gas optimization refers to the process of reducing the amount of gas required to execute a transaction. Gas is the unit of measurement used to calculate the fees paid to miners for including a transaction in a block on the blockchain.

The contract modifies the state of certain variables when the provided argument is different than their current state. However, in the case where the argument matches the currect state of the variable, the contract will not modify the state but the caller of the function will still be charged with gas.

```
function unexcludeFromFee(address _addressToWhitelist) external onlyOwner
{
  if ( feeExclusionList[_addressToWhitelist] ) {
    delete feeExclusionList[_addressToWhitelist];
  }
}
```

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.

The contract could modify these segments to use the `require` function provided by Solidity. By doing so, if the condition is not met, the transaction will revert and the caller will not be charged for executing the function.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	RAFLToken.sol#L164
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
maxTotalSupply
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	RAFLToken.sol#L197
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
feeExclusionList[_addressToWhitelist] = true;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	RAFLToken.sol#L298
Status	Unresolved

Description

The contract utilizes variables for percentage-based calculations that are required for its operations. These variables are involved in multiplication and division operations to determine proportions related to the contract's logic. If such variables are set to values beyond their logical or intended maximum limits, it could result in incorrect calculations. This misconfiguration has the potential to cause unintended behavior or financial discrepancies, affecting the contract's integrity and the accuracy of its calculations.

```
function setReflectionConfig(uint256 _feeBPS) external onlyOwner {
  if (!isReflective()) {
    revert TokenIsNotReflective();
  }
  super._setReflectionFee(_feeBPS);

  emit ReflectionConfigSet(_feeBPS);
}
```

Recommendation

To mitigate risks associated with boundary violations, it is important to implement validation checks for variables used in percentage-based calculations. Ensure that these variables do not exceed their maximum logical values. This can be accomplished by incorporating `require` statements or similar validation mechanisms whenever such variables are assigned or modified. These safeguards will enforce correct operational boundaries, preserving the contract's intended functionality and preventing computational errors.

RFO - Redundant Function Override

Criticality	Minor / Informative
Location	RAFLToken.sol#L475,482
Status	Unresolved

Description

The contract employs custom `renounceOwnership` and `transferOwnership` functions, which are defined as public functions overriding the base implementation. These functions delegate the inherited `super.renounceOwnership()` and `super.transferOwnership(newOwner)` methods respectively. The custom implementation in this contract does not modify or extend the functionality of the inherited methods. Consequently, the inclusion of these override functions is redundant, as it merely replicates the existing functionality without any additional benefits or modifications.

```
function renounceOwnership() public override onlyOwner {
  super.renounceOwnership();
}
function transferOwnership(address newOwner) public override onlyOwner {
  super.transferOwnership(newOwner);
}
```

Recommendation

It is recommended to remove the custom functions from the contract. This action will streamline the contract by eliminating unnecessary code, reducing potential confusion, and adhering more closely to the standard implementation of the ERC20 contract. By relying on the inherited functions, the contract can maintain clarity and efficiency, ensuring that it conforms to standard practices.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	RAFLToken.sol#L197
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
feeExclusionList[_addressToWhitelist] = true;
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ReflectiveERC20.sol#L38,221 RAFLToken.sol#L174,196,201,208,298,311,331,332,357
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.

```
function _tTotal() public view virtual returns (uint256) {
    return totalSupply();
  }
uint256 _amount
address _taxAddress
address _addressToWhitelist
address _whitelistedAddress
uint256 _feeBPS
uint256 _maxTaxBPS
uint256 _taxBPS
uint256 _deflationBPS
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	ReflectiveERC20.sol#L15
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
mapping(address => uint256) private _tOwned
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	ReflectiveERC20.sol#L140,152,165,229 lib/LibCommon.sol#L25,60,95
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 _mint(address account, uint256 value) internal override {
    if (isReflective) {
      revert MintingNotEnabled();
    } else {
      super._mint(account, value);
    }
...
function _burn(address account, uint256 value) internal override {
    if (isReflective) {
      revert BurningNotEnabled();
    } else {
      super._burn(account, 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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	RAFLToken.sol#L151,159,349
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.

```
taxAddress = _taxAddress
initialTokenOwner = tokenOwner
```

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	lib/LibCommon.sol#L27,43,79,108
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 {
  // Transfer the ETH and check if it succeeded or not.
  if iszero(call(gas(), to, amount, 0, 0, 0, 0)) {
    // Store the function selector of `ETHTransferFailed()`.
    // bytes4(keccak256(bytes("ETHTransferFailed()"))) = 0xb12d13eb
    mstore(0x00, 0xb12d13eb)
    // Revert with (offset, size).
    revert(0x1c, 0x04)
  }
}
...
```

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	ReflectiveERC20.sol#L3 RAFLToken.sol#L3 lib/LibCommon.sol#L2
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 ^0.8.17;
```

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.

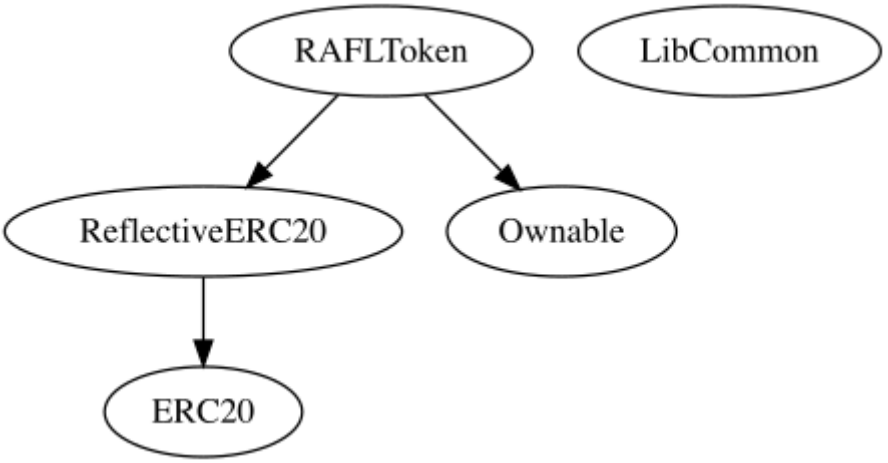
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ReflectiveERC20	Implementation	ERC20		
	_tTotal	Public		-
		Public	✓	ERC20
	balanceOf	Public		-
	transferFrom	Public	✓	-
	transfer	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setReflectionFee	Internal	✓	
	tokenFromReflection	Public		-
	_transferReflected	Private	✓	
	_reflectFee	Private	✓	
	calculateFee	Private		
	_transferNonReflectedTax	Internal	✓	
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		

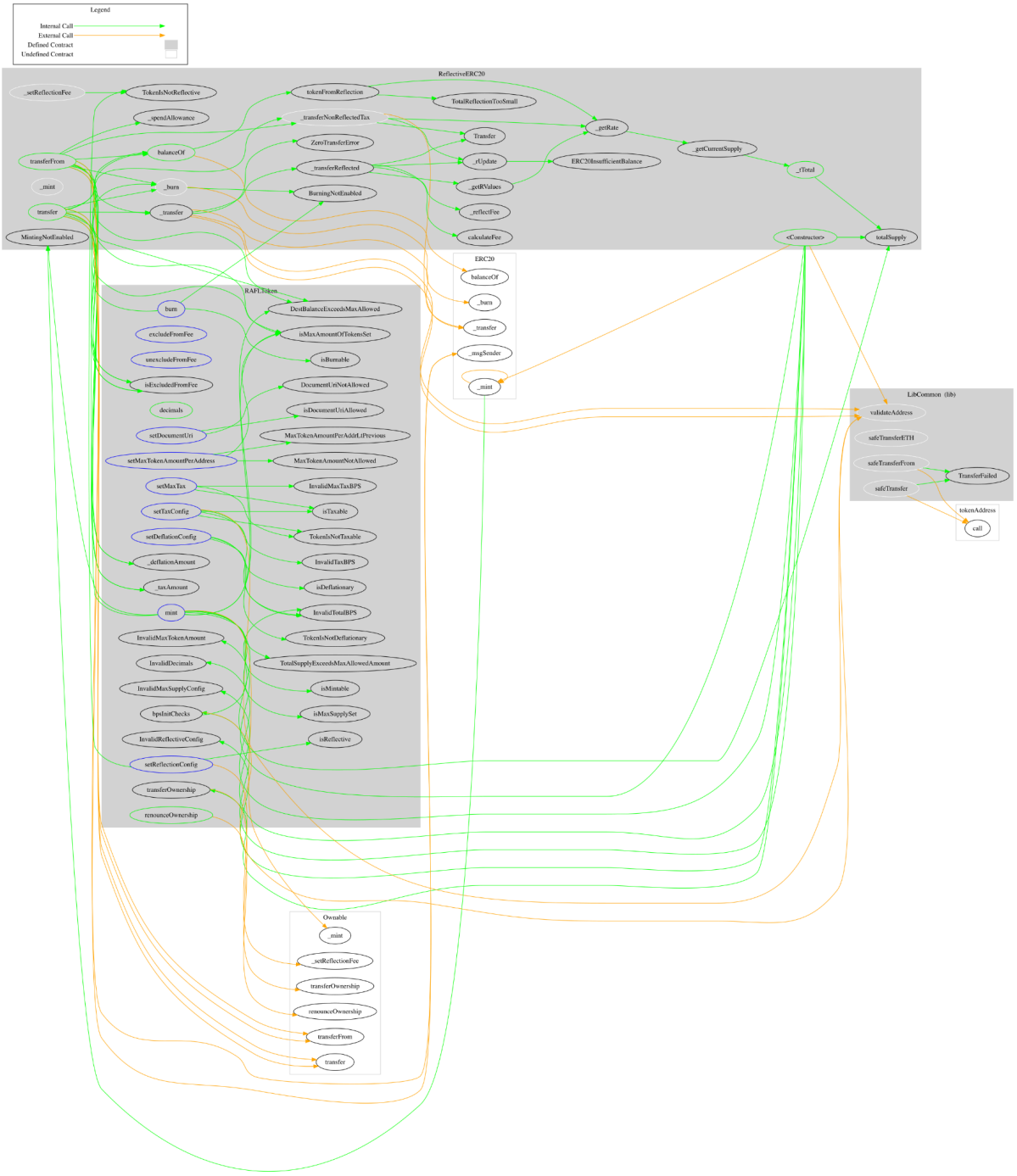
	_rUpdate	Private	✓	
RAFLToken	Implementation	ReflectiveERC20, Ownable		
		Public	✓	ReflectiveERC20
	bpsInitChecks	Private		
	excludeFromFee	External	✓	onlyOwner
	unexcludeFromFee	External	✓	onlyOwner
	isExcludedFromFee	Public		-
	isMintable	Public		-
	isBurnable	Public		-
	isMaxAmountOfTokensSet	Public		-
	isMaxSupplySet	Public		-
	isDocumentUriAllowed	Public		-
	decimals	Public		-
	isTaxable	Public		-
	isDeflationary	Public		-
	isReflective	Public		-
	setDocumentUri	External	✓	onlyOwner
	setMaxTokenAmountPerAddress	External	✓	onlyOwner
	setReflectionConfig	External	✓	onlyOwner
	setMaxTax	External	✓	onlyOwner
	setTaxConfig	External	✓	onlyOwner
	setDeflationConfig	External	✓	onlyOwner

	transfer	Public	✓	-
	transferFrom	Public	✓	-
	mint	External	✓	onlyOwner
	burn	External	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_taxAmount	Internal		
	_deflationAmount	Internal		
LibCommon	Library			
	safeTransferETH	Internal	✓	
	validateAddress	Internal		
	safeTransferFrom	Internal	✓	
	safeTransfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ReflectiveERC20
_setReflectionFee
TokenIsNotReflective
tokenFromReflection
TotalReflectionTooSmall
_spendAllowance
_transferNonReflectedTax
_getRate
Transfer
transferFrom
balanceOf
ZeroTransferError
_getCurrentSupply
_rUpdate
ERC20InsufficientBalance
_mint
_burn
_transferReflected
_rTotal
transfer
_transfer
BurningNotEnabled
_getRValues
MintingNotEnabled
_reflectFee
calculateFee
<Constructor>
totalSupply
ERC20
balanceOf
_burn
_transfer
_msgSender
_mint
RAFLToken
burn
DestBalanceExceedsMaxAllowed
excludeFromFee
isMaxAmountOfTokensSet
unexcludeFromFee
isBurnable
isExcludedFromFee
DocumentUriNotAllowed
decimals
isDocumentUriAllowed
setDocumentUri
MaxTokenAmountPerAddrLtPrevious
setMaxTokenAmountPerAddress
MaxTokenAmountNotAllowed
setMaxTax
InvalidMaxTaxBPS
setTaxConfig
isTaxable
setDeflationConfig
TokenIsNotTaxable
_deflationAmount
InvalidTaxBPS
_taxAmount
isDeflationary
mint
InvalidTotalBPS
InvalidMaxTokenAmount
TokenIsNotDeflationary
InvalidDecimals
TotalSupplyExceedsMaxAllowedAmount
InvalidMaxSupplyConfig
isMintable
bpsInitChecks
isMaxSupplySet
InvalidReflectiveConfig
isReflective
setReflectionConfig
transferOwnership
renounceOwnership
LibCommon (lib)
validateAddress
safeTransferETH
safeTransferFrom
TransferFailed
safeTransfer
tokenAddress
call
Ownable
_mint
_setReflectionFee
transferOwnership
renounceOwnership
transferFrom
transfer

Summary

RAFL contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions, manipulating the fees, and minting tokens. if the contract owner abuses the mint functionality, then the contract will be highly inflated. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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