Cyberscope

Audit Report

RAI

June 2025

SHA256

799fd63a3dee150976955cd0093c8cc221b1a1a5a8ca37b41adcde27787be4f2

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
Medium	ICM	Inefficient Cooldown Mechanism	Unresolved
Minor / Informative	AOI	Arithmetic Operations Inconsistency	Unresolved
Minor / Informative	MMN	Misleading Method Naming	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	RSRS	Redundant SafeMath Require Statement	Unresolved
Minor / Informative	UTPD	Unverified Third Party Dependencies	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

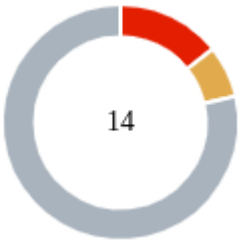
Audit Updates

Initial Audit	01 May 2025

Source Files

Filename	SHA256
RaiToken.sol	799fd63a3dee150976955cd0093c8cc221b1a1a5a8ca37b41adcde27787be4f2

Findings Breakdown



Critical	2
Medium	1
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	1	0	0	0
Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	RaiToken.sol#L3905
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it through the `feeReceiver`. If the `feeReceiver` is a malicious contract it may lead to transactions reverting. As a result, the contract may operate as a honeypot.

```
if (sellFee > 0) {
amount = amount - sellFee;
_balances[feeReceiver] += sellFee;

emit Transfer(sender, feeReceiver, sellFee);
emit FeeTaken(sender, feeReceiver, false, amount, sellFee);
IFeeReceiver(feeReceiver).triggerSwap(sellFee);
}
```

In addition, the owner may stop all buys and sell by revoking the `COOLING_PROTECTOR` from the `mainPair`. As a result all transaction will fail.

```
function _transfer(address sender, address recipient, uint256 amount) internal
virtual override {
if( (sender == mainPair || recipient == mainPair) // isTrade
&& !hasRole(COOLING_PROTECTOR,sender)
&& !hasRole(COOLING_PROTECTOR,recipient)){
revert("CoolingPeriod");
}
...
}
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` 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	RaiToken.sol#L3859
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 `setRatio` function with a high percentage value.

```
function setRatio(uint8 ratioType,uint256 ratio) external onlyDefaultAdmin
{
require(ratio <= PRECISION, "Exceeds precision");
if(ratioType ==0){
buyFeeRatio = ratio;
} else {
sellFeeRatio = ratio;
}
emit FeeRatioChanged(ratioType,ratio);
}
```

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	Minor / Informative
Location	RaiToken.sol#L3787
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 account_, uint256 amount_) external onlyVault {
_mint(account_, amount_);
}

modifier onlyVault() {
require(hasRole(MINT, msg.sender), "VaultOwned: caller is not the
Vault");
_;
}
```

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.

ICM - Inefficient Cooldown Mechanism

Criticality	Medium
Location	RaiToken.sol#L3888
Status	Unresolved

Description

The contract implements a clause for the case of buys and sells. Specifically, in the case of such a transaction, both the user and the `mainPair` must possess the role of the `COOLING_PROTECTOR`. This is an inefficient approach to implement such a mechanism as it would require all users to be whitelisted at all times. The current implementation effectively prevents users from finalizing their transactions.

```
if( (sender == mainPair || recipient == mainPair) // isTrade
&& !hasRole(COOLING_PROTECTOR,sender)
&& !hasRole(COOLING_PROTECTOR,recipient)){
revert("CoolingPeriod");
}
```

Recommendation

The team is advised to revise the implementation of the cooldown mechanism to ensure consistency of transfers for all users.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	RaiToken.sol#L3921
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
_balances[recipient] = _balances[recipient].add(amount);
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	RaiToken.sol#L3791
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. In this case, the contract defines the `onlyVault` modifier and grants minting rights to the deployer, which may not correspont to a vault contract.

```
modifier onlyVault() {
require(hasRole(MINT, msg.sender), "VaultOwned: caller is not the
Vault");
_;
}
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	RaiToken.sol#L3859
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.

```
function setFeeReceiver(address _feeReceiver) external onlyDefaultAdmin {
feeReceiver = _feeReceiver;
_setupRole(INTERN_SYSTEM,feeReceiver);
_setupRole(COOLING_PROTECTOR,feeReceiver);
}

function setMintRole(address account) external onlyDefaultAdmin returns
(bytes32) {
_setupRole(MINT, account);
return MINT;
}
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	RaiToken.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, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	RaiToken.sol#L3463
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns (uint256)
{
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	RaiToken.sol#L3916
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
IFeeReceiver(feeReceiver).triggerSwap(sellFee);
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	RaiToken.sol#L3825,3826
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.

```
uint256 public buyFeeRatio=0;
```

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	RaiToken.sol#L3738,3810,3859
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.

```
contract RAIToken is ERC20Token {
    using SafeMath for uint256;

    address public mainPair;
    address public feeReceiver = 0x3EE90695ADbfD84bEdf710Aab2a17E718B311235;//
DAO contract
    string private nameToken ="RAI"
Role(address account
```

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/stable/style-guide.html#naming-conventions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	RaiToken.sol#L3857,3860
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.

```
feeReceiver = _feeReceiver;
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	RaiToken.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 ^0.8.20;
```

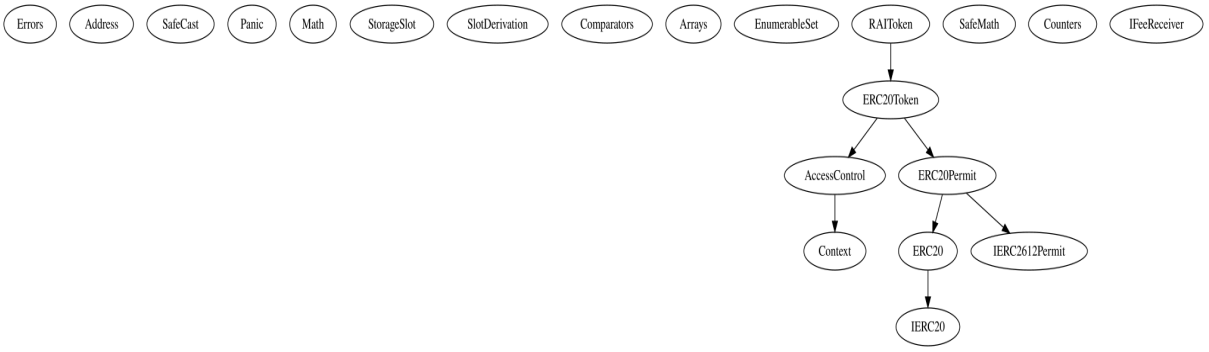
Recommendation

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.

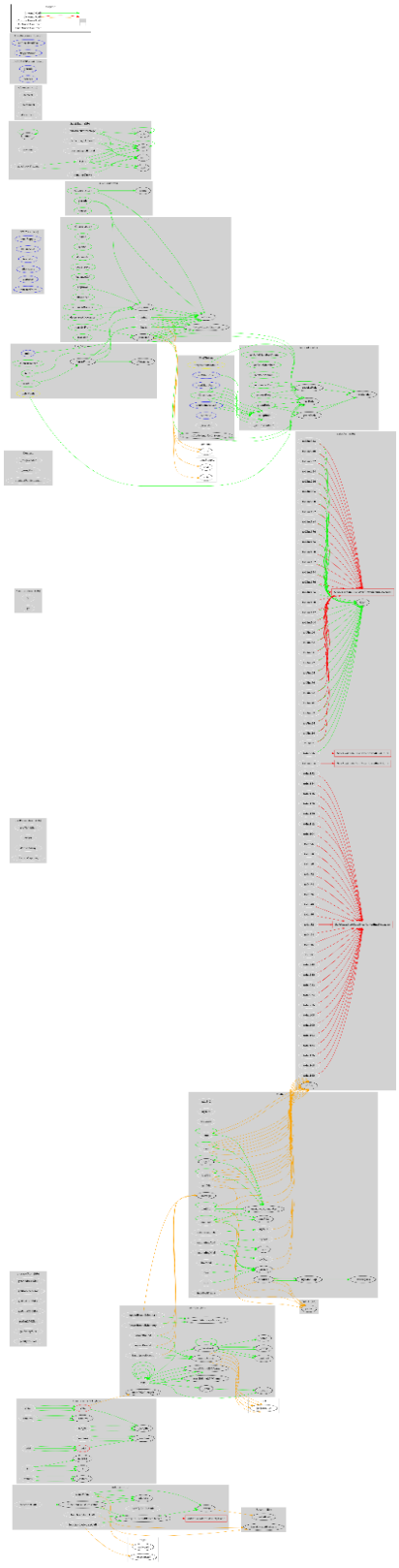
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
RAIToken	Implementation	ERC20Token		
		Public	✓	ERC20Token
	setMainPair	External	✓	onlyDefaultAdmin
	setFeeReceiver	External	✓	onlyDefaultAdmin
	setMintRole	External	✓	onlyDefaultAdmin
	setRatio	External	✓	onlyDefaultAdmin
	_transfer	Internal	✓	
	_isTradeAndNotInSystem	Internal		

Inheritance Graph



Flow Graph



Summary

RAI 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 stop transactions, manipulate the fees and mint 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

cyberscope.io