Cyberscope

Audit Report

RAIFI Staking

June 2025

SHA256

ca0a1e7ef15df81b8c455ab2222c1ee6b8af4c6b84c21b19e00867545b75958b

Audited by © cyberscope

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	11 May 2025
Corrected Phase 2	11 Jun 2025

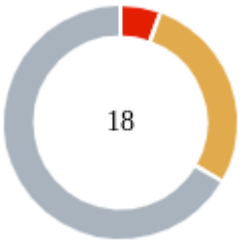
Source Files

Filename	SHA256
Staking.sol	ca0a1e7ef15df81b8c455ab2222c1ee6b8af4c6b84c21b19e00867545b75958b

Contract Readability Comment

The audit scope is to identify security vulnerabilities, validate the business logic, and recommend potential optimizations. The codebase is incomplete, with key functionalities missing, references to non-existent functions, and non-functional or broken logic. As such, the project cannot be considered production-ready. Furthermore, the contract does not adhere to core Solidity principles related to gas efficiency, code readability, and appropriate use of data structures. The development team is strongly advised to re-evaluate the business logic and align the implementation with established Solidity best practices to ensure both security and maintainability. Even if the identified issues are addressed and rectified, the contract would remain far from production-ready due to its convoluted and incomplete nature. It is worth noting that, although automated tools provide valuable assistance, expert knowledge remains essential for the development of reliable and secure smart contracts.

Findings Breakdown



Critical	1
Medium	5
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	5	0	0	0
Minor / Informative	12	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	ETM	Excessive Token Mint	Unresolved
Medium	IGU	Inconsistent Gons Update	Unresolved
Medium	MBD	Missing Bonus Deposit	Unresolved
Medium	MCS	Missing Code Segments	Unresolved
Medium	PGA	Potential Griefing Attack	Unresolved
Medium	UTPD	Unverified Third Party Dependencies	Unresolved
Minor / Informative	MEM	Misleading Error Message	Unresolved
Minor / Informative	MDA	Misrepresented Distribution Amount	Unresolved
Minor / Informative	MAC	Missing Access Control	Unresolved
Minor / Informative	MEM	Missing Error Messages	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved

●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

ETM - Excessive Token Mint

Criticality	Critical
Location	Staking.sol#L665
Status	Unresolved

Description

The contract has the authority to mint tokens. This is possible by calling the `rebase` function. Specifically, the method mints to the `yieldVestingContract` contract an amount of Rai tokens that is nearly twice the circulating supply of sRai tokens. As a result, the supply of Rai will be highly inflated.

```
uint staked = circulatingsRaiSupply();
```

```
function circulatingsRaiSupply() public view returns (uint256) {
uint256 totalSupply_sRAI = IERC20(sRAI).totalSupply();
uint256 stakingRAIBalance = IERC20(sRAI).balanceOf(address(this));
uint256 warmupBalance = IERC20(sRAI).balanceOf(warmupContract);
return totalSupply_sRAI.sub(stakingRAIBalance).sub(warmupBalance);
}
```

```
uint256 mintBonus=staked + staked.mul(90).div(100);
//2. Min RAI and send to yieldVestingContract
IRaiToken(RAI).mint(yieldVestingContract, mintBonus);
```

Recommendation

The team should revise the implementation of the rebase method to ensure that token minting is consistent. Enforcing minting through predetermined algorithms with clearly defined bounds will enhance both consistency and user trust.

IGU - Inconsistent Gons Update

Criticality	Medium
Location	Staking.sol#L569
Status	Unresolved

Description

The contract implements the `stake` function, enabling users to deposit an `_amount` of `RAI` into the contract. The contract maintains a `warmupInfo` record for each user, tracking the cumulative `gons` contributed through successive calls to the stake function. This is a redundant and potentially misleading operation, since the staked tokens do not necessarily support a rebasing mechanism.

```
gons: info.gons.add(_amount),
```

Recommendation

The team is advised to revise the current implementation to eliminate redundancies and misleading operations. This can be accomplished by refactoring the code base to eliminate references to rebasing parameters.

MBD - Missing Bonus Deposit

Criticality	Medium
Location	Staking.sol#L724
Status	Unresolved

Description

The contract implements the `insBonus` function, which allows the manager to increase the `totalBonus` value by a specified `_amount`. However, this function does not enforce any checks on the bonus logic nor does it perform any actual token transfers. As a result, the `totalBonus` variable can be arbitrarily inflated.

```
function insBonus( uint _amount ) external onlyManager(){
totalBonus = totalBonus.add( _amount );
}
```

Recommendation

The team is advised to revise the implementation to ensure that updates to `totalBonus` reflect real, auditable value changes. Consider integrating validation mechanisms and token transfer logic to prevent arbitrary or adjustments to internal state variables.

MCS - Missing Code Segments

Criticality	Medium
Location	Staking.sol#L722
Status	Unresolved

Description

The contract lacks critical components necessary for its functionality. In its current state, it cannot achieve its intended design due to these missing elements.

```
if ( distributor != address(0) ) {
IDistributor( distributor ).distribute();
}
```

Recommendation

It is required that all contract functionalities are fully developed to ensure viable and consistent operation.

PGA - Potential Griefing Attack

Criticality	Medium
Location	Staking.sol#L570
Status	Unresolved

Description

The contract provides functionality for users to stake assets on behalf of a `_recipient`. Upon acceptance of a stake, the contract extends the expiry by a predefined warmupPeriod. This design permits griefing attacks, where users may stake minimal amounts for recipients, repeatedly prolonging the recorded expiry for other users to manipulate the system and potentially extract value.

```
expiry: epoch.number.add( warmupPeriod )
```

Recommendation

The team is recommended to establish stringent access controls to guarantee that only eligible users can make impactful modifications to the contract's state.

UTPD - Unverified Third Party Dependencies

Criticality	Medium
Location	Staking.sol#L578
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. In particular the contract calls the `updateTotalContributionValue` method from a `contributionValueRewards` contract, which however is not a known implementation.

```
IContributionValueRewardsContract(contributionValueRewards).updateTotalContrib
utionValue(msg.sender,_amount);
```

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.

MEM - Misleading Error Message

Criticality	Minor / Informative
Location	Staking.sol#L642
Status	Unresolved

Description

The `require` statement reverts with the message `NotEnoughInterest`, which is misleading, as the contract does not implement an interest mechanism. The message also lacks clarity on the actual reason for failure, which may lead to user confusion.

```
require(total > principalAmt && total - principalAmt >= _amount
,"NotEnoughInterest");
```

Recommendation

The team is advised to revise the error message to more accurately reflect the actual condition being checked. Clear and context-appropriate revert messages improve contract transparency, user experience, and debugging efficiency.

MDA - Misrepresented Distribution Amount

Criticality	Minor / Informative
Location	Staking.sol#L690
Status	Unresolved

Description

The contract maintains a `rebaseHistory` mapping that stores key information related to the rebase process. Among this data, it includes a `distributeAmount` object, which does not maintain the actual amount distributed.

```
uint distributeAmount =
epoch.distribute.mul(rebasePercentage).div(100);
```

```
rebaseHistory[epoch.number] = RebaseInfo({
timestamp: block.timestamp,
distributeAmount: epoch.distribute,
circulatingSupply_sRAI: staked
});
```

Recommendation

The team is advised to monitor the current implementation to ensure it accurately reflects the actual amount distributed. Clear and accurate state tracking enhances transparency and system reliability.

MAC - Missing Access Control

Criticality	Minor / Informative
Location	Staking.sol#L586
Status	Unresolved

Description

The contract fails to implement adequate access controls, permitting third-party users to disrupt the `claim` process. As a result, an unauthorized user can unintentionally withdraw funds from another user's warmup and delete their `warmupInfo`.

```
function claim ( address _recipient ) public {
...
}
```

Recommendation

The `claim` function should implement stringent access controls to guarantee that only authorized users or the user themselves can execute the action.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	Staking.sol#L523,525
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_RAI != address(0))
require(_sRAI != address(0))
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Staking.sol#L665,734,773,776,783,786
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 rebase() public {...}
function setContract( CONTRACTS _contract, address _address ) external
onlyManager() {...}
function setWarmup( uint _warmupPeriod ) external onlyManager() {...}
function updateEpoch( uint _epochLength,uint _firstEpochNumber ,uint
_firstEpochBlock,uint _minRebase,uint _maxRebase  ) external onlyManager()
{...}
function setRebasePercentage( uint _rebasePercentage ) external onlyManager()
{...}
unction transferPrincipalOwnership(address from, address to, uint256 amount)
external {...}
```

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.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	Staking.sol#L559
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
function stake( uint _amount, address _recipient ) external returns ( bool )
{
IERC20( RAI ).transferFrom( msg.sender, address(this), _amount );
....
deposit: info.deposit.add( _amount ),
...
}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Staking.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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Staking.sol#L443,477,559,586,626,724,734,773,776,783
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 RAIProfit_
address public RAI
uint _amount
address _recipient
bool _trigger
bool _isPrincipal
CONTRACTS _contract
address _address
uint _warmupPeriod
uint _epochLength
uint _firstEpochBlock
uint _firstEpochNumber
uint _maxRebase
uint _minRebase

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Staking.sol#L725,774,784
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
totalBonus = totalBonus.add( _amount )
warmupPeriod = _warmupPeriod
rebasePercentage = _rebasePercentage
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Staking.sol#L527,537,539,540,541,542,543
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.

```
warmupContract=_warmupContract
releasePool = _releasePool
contributionValueRewards=_contributionValueRewards
community=_community
yieldVestingContract = _yieldVestingContract
usdtToken=_usdtToken
daoContract=_daoContract
```

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	Staking.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 ^0.8.0;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Staking.sol#L562,602,637,647
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20( RAI ).transferFrom( msg.sender, address(this), _amount )
IERC20( RAI ).transfer( msg.sender, info.deposit )
IERC20(RAI).transfer(staker, _amount)
IERC20(usdtToken).transferFrom(staker,daoContract, burnAmt)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
IERC20	Interface			
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Address	Library			
	isContract	Internal		

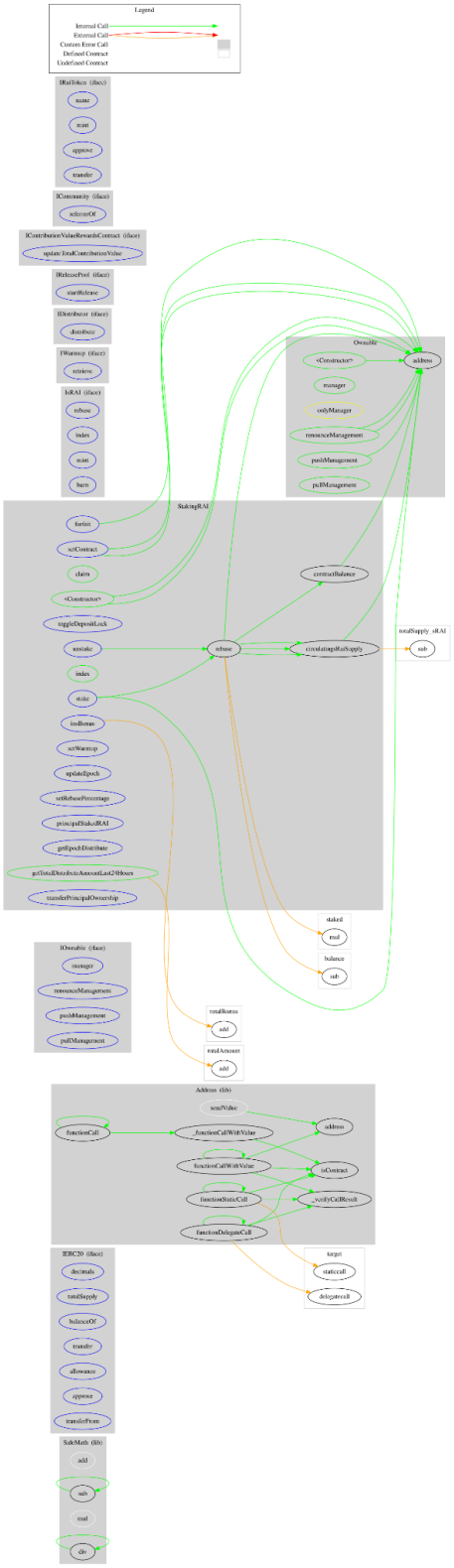
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
IOwnable	Interface			
	manager	External		-
	renounceManagement	External	✓	-
	pushManagement	External	✓	-
	pullManagement	External	✓	-
Ownable	Implementation	IOwnable		
		Public	✓	-
	manager	Public		-
	renounceManagement	Public	✓	onlyManager
	pushManagement	Public	✓	onlyManager
	pullManagement	Public	✓	-

IsRAI	Interface			
	rebase	External	✓	-
	index	External		-
	mint	External	✓	-
	burn	External	✓	-
IWarmup	Interface			
	retrieve	External	✓	-
IDistributor	Interface			
	distribute	External	✓	-
IReleasePool	Interface			
	startRelease	External	✓	-
IContributionValueRewardsContract	Interface			
	updateTotalContributionValue	External	✓	-
ICommunity	Interface			
	referrerOf	External		-
IRaiToken	Interface			
	name	External		-
	mint	External	✓	-

	approve	External	✓	-
	transfer	External	✓	-
StakingRAI	Implementation	Ownable		
		Public	✓	-
	stake	External	✓	-
	claim	Public	✓	-
	forfeit	External	✓	-
	toggleDepositLock	External	✓	-
	unstake	External	✓	-
	index	Public		-
	rebase	Public	✓	-
	contractBalance	Public		-
	circulatingsRaiSupply	Public		-
	insBonus	External	✓	onlyManager
	setContract	External	✓	onlyManager
	setWarmup	External	✓	onlyManager
	updateEpoch	External	✓	onlyManager
	setRebasePercentage	External	✓	onlyManager
	principalStakedRAI	External		-
	getEpochDistribute	External		-
	getTotalDistributeAmountLast24Hours	Public		-
	transferPrincipalOwnership	External	✓	-

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IRaiToken (iface)
name
symbol
approve
transfer
ICommunity (iface)
referrerOf
IContributionValueRewardsContract (iface)
updateTotalContributionValue
IReleasePool (iface)
startRelease
IDistributor (iface)
distribute
IWarmup (iface)
retrieve
IsRAI (iface)
rebase
index
mint
burn
Ownable
<Constructor>
address
manager
onlyManager
resourceManagement
poolManagement
pidIMangement
StakingRAI
forfeit
setContract
claim
<Constructor>
contractBalance
toggleDepositLock
unstake
rebase
circulatingRaiSupply
totalSupply_sRAI
sub
index
stake
totalBonus
setWarmup
updateEpoch
setRebasePercentage
principalStakedRAI
getEpochDistribute
getTotalDistributeAmountLast24Hours
transferPrincipalOwnership
staked
mul
balance
sub
IOwnable (iface)
manager
resourceManagement
poolManagement
pidIMangement
totalBonus
add
totalAmount
add
Address (lib)
sendValue
functionCall
_functionCallWithValue
address
functionCallWithValue
isContract
functionStaticCall
_verifyCallResult
functionDelegateCall
target
staticcall
delegatecall
IERC20 (iface)
decimals
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
SafeMath (lib)
add
sub
mul
div

Summary

RAIFI contract implements a staking mechanism. This audit investigates security issues, business logic concerns and potential improvements.

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