Cyberscope

Audit Report

RAIFI Yield Vesting

June 2025

SHA256

ed2ff4153578c12cb9c1496865513327c218608fecdfd661e19adeb87212c25c

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

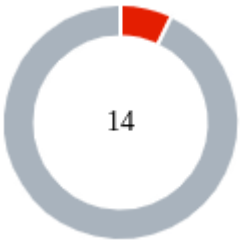
Audit Updates

Initial Audit	12 May 2025
Corrected Phase 2	18 Jun 2025

Source Files

Filename	SHA256
YieldVesting.sol	ed2ff4153578c12cb9c1496865513327c218608fecdfd661e19adeb87212c25c

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	MES	Misplaced Else Statement	Unresolved
Minor / Informative	CO	Code Optimization	Unresolved
Minor / Informative	DTR	Duplicate Token Reference	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	MTM	Missing Transfer Mechanism	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	TSI	Tokens Sufficiency Insurance	Unresolved
Minor / Informative	UF	Unused Functionalities	Unresolved
Minor / Informative	USV	Unused State Variables	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved

●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

MES - Misplaced Else Statement

Criticality	Critical
Location	YieldVesting.sol#L548
Status	Unresolved

Description

The `else` statement has been misplaced and is currently associated with the inner `if` condition checking whether `remaining == 0`. However, it is intended to handle the case where the vesting schedule is active and the remaining amount is not zero. Therefore its execution is not utilized according to the expected design.

```
if (block.timestamp >= vesting.startTime.add(vesting.duration)) {
uint256 remaining = vesting.totalReward.sub(vesting.releasedAmount);
if (remaining > 0) {
vesting.releasedAmount = vesting.totalReward;
balance = balance.add(remaining);
emit VestingReleased(user, i, remaining, vesting.rewardType);
}
else {
uint256 timeElapsed = block.timestamp.sub(vesting.startTime);
uint256 releaseableAmount =
vesting.totalReward.mul(timeElapsed).div(vesting.duration).sub(vesting.release
dAmount);
if (releaseableAmount > 0) {
vesting.releasedAmount = vesting.releasedAmount.add(releaseableAmount);
balance = balance.add(releaseableAmount);
emit VestingReleased(user, i, releaseableAmount, vesting.rewardType);
}
}
}
```

Recommendation

The team is advised to review the control flow structure and ensure the else is correctly aligned with the outer if condition to reflect the intended logic.

CO - Code Optimization

Criticality	Minor / Informative
Location	YieldVesting.sol#L568
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.

```
function getUserVestingEntries(address user, string memory _rewardTypeFilter)
external view returns (VestingEntry[] memory) {
...
}
```

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.

DTR - Duplicate Token Reference

Criticality	Minor / Informative
Location	YieldVesting.sol#L485,486
Status	Unresolved

Description

The constructor initializes both `rai` and `RAI` variables using the same address, resulting in a redundant reference to the same token. This duplication may cause confusion and increase the risk of inconsistencies in the code base.

```
rai=_rai;
RAI = IERC20(_rai);
```

Recommendation

The team is advised to retain a single reference to the token. Removing unnecessary duplication will improve code clarity and reduce future errors.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	YieldVesting.sol#L483,592
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.

```
else if ( _contract == CONTRACTS.RAI ) { // 3
rai=_address;
RAI = IERC20(_address);
} else if( _contract == CONTRACTS.BOND){
bondContractAddress=_address;
}
```

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.

MTM - Missing Transfer Mechanism

Criticality	Minor / Informative
Location	YieldVesting.sol#L519
Status	Unresolved

Description

The `addToRaiDAOFund` function updates the internal variable `raiDAOFund` by increasing it with the specified amount, however it does not perform an actual token transfer. As a result, the internal state may become inconsistent with the contract's actual token balance.

```
function addToRaiDAOFund(uint256 amount) external onlyOwner {
raiDAOFund = raiDAOFund.add(amount);
emit RaiDAOFundUpdated(amount);
}
```

Recommendation

The team is advised to revise the transfer mechanism within the function to ensure the specified amount is properly transferred to the contract, maintaining consistency between recorded balances and actual holdings.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	YieldVesting.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.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	YieldVesting.sol#L524
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
function addVestingEntry(address user, uint256 amount, uint256 duration,
string memory rewardType) external onlyAuthorized {
uint256 vestingId = nextVestingId++;
vestingEntries[user][vestingId] = VestingEntry({
totalReward: amount,
startTime: block.timestamp,
duration: duration,
releasedAmount: 0,
rewardType: rewardType
});
emit VestingAdded(user, vestingId, amount, duration, rewardType);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

UF - Unused Functionalities

Criticality	Minor / Informative
Location	YieldVesting.sol#L503,509,514,519
Status	Unresolved

Description

The contract includes methods that are not utilized during execution. The presence of such unused functions increases the overall code size and reduces readability.

```
modifier onlyPartnerBurnContract() {
require(msg.sender == partnerBurnContract, "Only Partner Burn Contract can
call this function");
_;
}
function addAllowedToken(address token) external onlyOwner {
allowedTokens[token] = true;
emit AllowedTokenAdded(token);
}

function updateTokenPrice(address token, uint256 price) external onlyOwner {
tokenPrices[token] = price;
emit TokenPriceUpdated(token, price);
}

function addToRaiDAOFund(uint256 amount) external onlyOwner {
raiDAOFund = raiDAOFund.add(amount);
emit RaiDAOFundUpdated(amount);
}
```

Recommendation

The team is advised to review and remove any redundant or unused methods to improve code clarity and maintainability.

USV - Unused State Variables

Criticality	Minor / Informative
Location	YieldVesting.sol#L491
Status	Unresolved

Description

The contract defines state variables which are not used duringthe execution flow. Unused state variables increase the contract's bytecode size and can hinder readability and maintainability.

```
vestingSpeeds[180] = 0;
vestingSpeeds[150] = 5;
vestingSpeeds[100] = 10;
vestingSpeeds[60] = 20;
vestingSpeeds[30] = 25;
vestingSpeeds[15] = 25;
```

Recommendation

The team is advised to remove redundancies to reduce deployment costs and improve overall code clarity.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	YieldVesting.sol#L465,535,564,592
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.

```
IERC20 public RAI
string memory _rewardTypeFilter
CONTRACTS _contract
address _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/stable/style-guide.html#naming-conventions.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	YieldVesting.sol#L349,362
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 _reentrancyGuardEntered() internal view returns (bool) {
        return _status == ENTERED;
    }

function _contextSuffixLength() internal view virtual returns (uint256)
{
        return 0;
    }
```

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	YieldVesting.sol#L485,487,488,489,490,594
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.

```
rai=_rai
partnerTokenWallet = _partnerTokenWallet
contributionRewardsContract = _contributionRewardsContract
partnerBurnContract = _partnerBurnContract
bondContractAddress=_bondContractAddress
partnerTokenWallet = _address
```

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	YieldVesting.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 ^0.8.20;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	YieldVesting.sol#L561
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.

```
RAI.transfer(user, balance)
```

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.

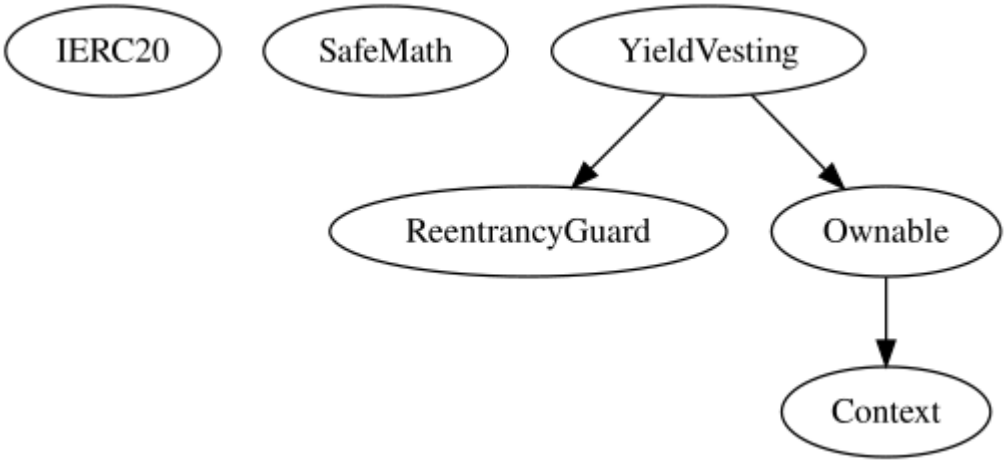
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		

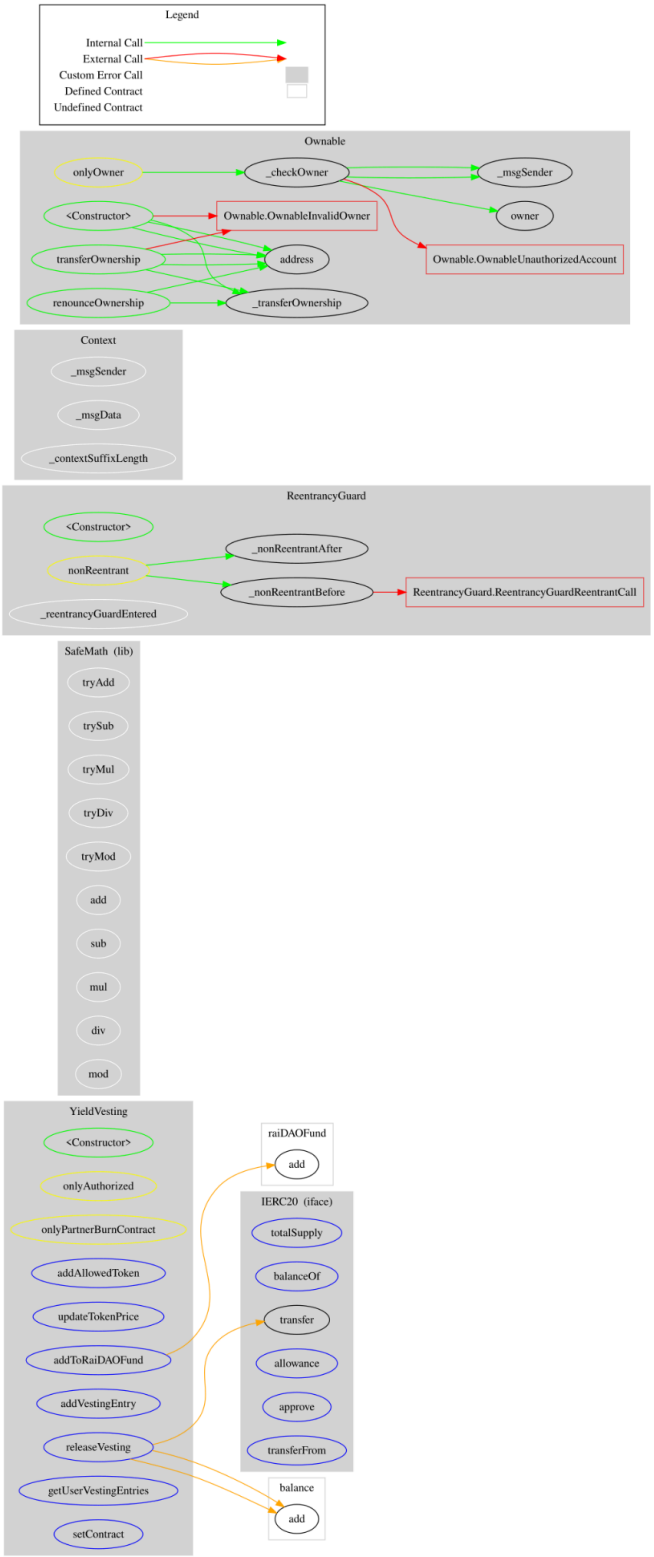
	sub	Internal		
	div	Internal		
	mod	Internal		
ReentrancyGuard	Implementation			
		Public	✓	-
	_nonReentrantBefore	Private	✓	
	_nonReentrantAfter	Private	✓	
	_reentrancyGuardEntered	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
YieldVesting	Implementation	Ownable, ReentrancyGuard		

		Public	✓	Ownable
	addAllowedToken	External	✓	onlyOwner
	updateTokenPrice	External	✓	onlyOwner
	addToRaiDAOFund	External	✓	onlyOwner
	addVestingEntry	External	✓	onlyAuthorized
	releaseVesting	External	✓	nonReentrant
	getUserVestingEntries	External		-
	setContract	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_checkOwner
_msgSender
owner
<Constructor>
Ownable.OwnableInvalidOwner
transferOwnership
address
Ownable.OwnableUnauthorizedAccount
renounceOwnership
_transferOwnership
Context
_msgSender
_msgData
_contextSuffixLength
ReentrancyGuard
<Constructor>
_nonReentrantAfter
nonReentrant
_nonReentrantBefore
ReentrancyGuard.ReentrancyGuardReentrantCall
_reentrancyGuardEntered
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
YieldVesting
<Constructor>
onlyAuthorized
onlyPartnerBurnContract
addAllowedToken
updateTokenPrice
addToRaiDAOFund
addVestingEntry
releaseVesting
getUserVestingEntries
setContract
raiDAOFund
add
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
balance
add

Summary

RAIFI contract implements a vesting 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