Cyberscope

Audit Report

Araracoin

October 2024

Repository	https://github.com/araracoin/AraraCoin
Commit	b194752d3248ce4bd3a73acae98ce6a0b132d75d
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	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	MC	Missing Check	Unresolved
●	PBV	Percentage Boundaries Validation	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	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

Repository	https://github.com/araracoin/AraraCoin
Commit	b194752d3248ce4bd3a73acae98ce6a0b132d75d
Badge Eligibility	Must Fix Criticals

Audit Updates

Initial Audit	12 Oct 2024

Source Files

Filename	SHA256
MyVestingWallet.sol	7d4d137f8066c6f6784c82ea0043c58ee70ff93ada9cf511dc51e44db06fd1ef
AraraCoin.sol	0c9eba81bcbd7e436c9d9bac16bfb1ec5e35ba702a1656ae474da586b167a368
tax/StaticTaxHandler.sol	0b00d646e53ee820b4a0d2548c4be243a3b19403c1db055825c6f21a5119a358
tax/ITaxHandler.sol	aca00b85b472d011f913cb304b1a62701f74ce2e7bde70c3d93a104559d79131
abstract/VestingWalletCliff.sol	b75c452ec322a7747397e4860f64bb34db69dbb44c9236d8b3a3f15bcb2a83e4

Overview

AraraCoin Contract

The AraraCoin contract is an ERC20 token implementation with additional functionality for tax management, trading control, and vesting. It integrates OpenZeppelin libraries such as `ERC20`, `ERC20Permit`, and `Ownable` to provide standard token functionality, permit-based approvals, and ownership controls. The contract allows the distribution of its total supply (100 billion tokens) to various wallets, including the vesting contract, during deployment. It also incorporates a tax handler to calculate and apply taxes on token transfers, with trading restrictions in place before trading is fully enabled. The contract owner has control over tax handling and trading permissions, making it flexible for governance and future updates.

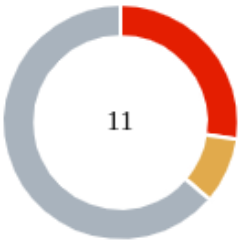
StaticTaxHandler Contract

The StaticTaxHandler contract manages tax collection for specific transactions, including those involving liquidity additions and sales. The contract allows the owner to set a tax percentage and manage a list of exempted addresses that are not subject to taxation. It provides flexibility for the protocol by enabling updates to tax rates (which can only be lowered) and controlling tax-exempt addresses. The owner can add or remove addresses from the exemption list, and it is recommended that ownership of this contract be transferred to a DAO-controlled timelock or a multisig wallet for security and decentralization.

MyVestingWallet Contract

The MyVestingWallet contract is a customized token vesting solution built on top of OpenZeppelin's `VestingWallet` and `VestingWalletCliff` contracts. It allows for the gradual release of tokens over a specified duration to a beneficiary while enforcing a cliff period before any tokens can be claimed. The contract requires the specification of a beneficiary address, a start timestamp, a vesting duration, and a cliff period at the time of deployment. This ensures that tokens are securely locked and gradually released according to the predefined schedule, supporting structured and secure token vesting.

Findings Breakdown



Critical	3
Medium	1
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	1	0	0	0
Minor / Informative	7	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	AraraCoin.sol#L129
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
if (!tradingEnabled) {
    require(_canTrade.contains(from), "AraraCoin trade is
disabled");
}
```

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. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	tax/StaticTaxHandler.sol#L55
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 `setTaxPercentage` function with a high percentage value.

```
   function setTaxPercentage(uint256 newTaxPercentage) external
onlyOwner {
        require(
            newTaxPercentage >= 0,
            "Invalid Value."
        );

        uint256 oldTaxPercentage = taxPercentage;
        taxPercentage = newTaxPercentage;

        emit TaxPercentageUpdated(oldTaxPercentage,
newTaxPercentage);
    }
```

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.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Critical
Location	AraraCoin.sol#L73
Status	Unresolved

Description

The `_transfer` function is 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

Specifically, the contract is using the `StaticTaxHandler` contract to apply tax amounts. However, during the constructor phase, the contract attempts to distribute the `TOTAL_SUPPLY` without considering the fees applied to the token. As a result, when the token distribution is executed, the transaction reverts because the available tokens are insufficient to cover both the distribution and the fees. This can lead to failures in the initial token setup, preventing the intended distribution from occurring.

```
// Distribute the total supply according to the pre-defined percentages
_transfer(msg.sender, marketingWallet, 3_000_000_000 * 10 ** 18);  // 3%
to marketing services
_transfer(msg.sender, consultingWallet, 1_250_000_000 * 10 ** 18);  //
1.25% to consulting services
_transfer(msg.sender, auditWallet, 750_000_000 * 10 ** 18);  // 0.75% to
audit service
_transfer(msg.sender, preSaleWallet, TOTAL_SUPPLY * 10 / 100); // 10% to
pre-sale
_transfer(msg.sender, launchWallet, TOTAL_SUPPLY * 20 / 100); // 20% to
launch
_transfer(msg.sender, investorsWallet, 6_250_000_000 * 10 ** 18);  //
6.25% tokens for investors
_transfer(msg.sender, investorsYearOneVestingWallet, 6_250_000_000 * 10
** 18);  // 6.25% tokens for investors' year-one vesting
_transfer(msg.sender, investorsYearTwoVestingWallet, 12_500_000_000 * 10
** 18);  // 12.5% tokens for investors' year-two vesting
_transfer(msg.sender, companyVestingWallet, 9_600_000_000 * 10 ** 18);
// 9.6% to comapny vesting
_transfer(msg.sender, foundersVestingWallet, 5_600_000_000 * 10 ** 18);
// 5.6% to founders vesting
_transfer(msg.sender, teamVestingWallet, 4_800_000_000 * 10 ** 18);  //
4.8% to team vesting
_transfer(msg.sender, preservationProjectsVestingContract, TOTAL_SUPPLY
* 20 / 100); // 20% to preservation projects
```

Recommendation

It is recommended to account for the applied fees when distributing the total supply of tokens. The contract should either allocate additional tokens to cover the fees or reduce the distribution amounts accordingly. Proper calculations must be implemented to ensure the total supply is sufficient to cover both the distributed tokens and any associated fees, preventing transaction reverts. Additionally, thorough testing should be performed to confirm that the distribution functions properly under all fee conditions.

Additionally, 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
```

UTPD - Unverified Third Party Dependencies

Criticality	Medium
Location	AraraCoin.sol#L104,134
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.

Specifically, the contract contains the `taxHandler`, which plays a central role in determining and applying transaction taxes. The owner of the contract holds the exclusive authority to modify the taxHandler's address. While this is common in many contracts, the taxHandler's correct and secure operation is essential for the proper functioning of the contract. If a malicious or incorrectly implemented `taxHandler` be set by the owner, then the contract's entire tax-related logic could be compromised. This could result in unexpected or excessive fees being applied to transactions, negatively impacting users and the contract’s ecosystem. Without safeguards, a malicious `taxHandler` could be used to siphon funds or disrupt the contract's intended operations.

```
   function setTaxWallet(address taxWalletAddress) external onlyOwner {
        address oldTaxWalletAddress = taxWallet; // Store the current
tax wallet address
        taxWallet = taxWalletAddress; // Set the new tax wallet address

        emit TaxWalletChanged(oldTaxWalletAddress, taxWallet); // Emit
event with old and new tax wallet addresses
    }

    function _update(address from, address to, uint256 value)
        internal virtual override(ERC20)
    {
       ...

        // Calculate the tax based on the sender, receiver, and amount
        uint256 tax = taxHandler.getTax(from, to, value);
        ...
        }
```

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.

It is recommended to perform the external call to the `taxHandler` within a try-catch block to handle any potential failures or malicious actions gracefully. Additionally, it is important that the try-catch block includes checks to ensure that the fees applied are within reasonable limits, not exceeding 25%, as noted in the `ELFM` finding. This measure helps maintain the integrity of the contract and prevents abuse of the `taxHandler` mechanism.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	AraraCoin.sol#L95,103,111 tax/StaticTaxHandler.sol#L72,82
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Specifically, the owner has the authority to update the tax handler contract, update the tax wallet address, add addresses that are allowed to trade before trading is enabled, and manage tax exemptions by adding or removing addresses from the set of tax-exempted addresses.

```
   function setTaxHandler(address taxHandlerAddress) external onlyOwner {
        address oldTaxHandlerAddress = address(taxHandler); // Store the
current tax handler address
        taxHandler = ITaxHandler(taxHandlerAddress); // Set the new tax
handler address

        emit TaxHandlerChanged(oldTaxHandlerAddress, taxHandlerAddress);
// Emit event with old and new tax handler addresses
    }

    function setTaxWallet(address taxWalletAddress) external onlyOwner {
        address oldTaxWalletAddress = taxWallet; // Store the current tax
wallet address
        taxWallet = taxWalletAddress; // Set the new tax wallet address

        emit TaxWalletChanged(oldTaxWalletAddress, taxWallet); // Emit
event with old and new tax wallet addresses
    }

    function addCanTrade(
        address[] calldata allowedAddresses
    ) external onlyOwner {
        require(!tradingEnabled, "AraraCoin trading already enabled"); //
Ensure trading isn't enabled yet
        require(allowedAddresses.length != 0, "AraraCoin invalid
parameters"); // Ensure there are addresses to add

        // Add each address in the provided list to the set of addresses
allowed to trade
        for (uint256 i = 0; i < allowedAddresses.length; i++) {
            _canTrade.add(allowedAddresses[i]);
        }
    }
```

```
    function addExemption(address exemption) external onlyOwner {
        if (_exempted.add(exemption)) {
            emit TaxExemptionUpdated(exemption, true);
        }
    }

    function removeExemption(address exemption) external onlyOwner {
        if (_exempted.remove(exemption)) {
            emit TaxExemptionUpdated(exemption, false);
        }
    }
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MC - Missing Check

Criticality	Minor / Informative
Location	MyVestingWallet.sol#L8
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically the contract is missing a check to verify that the `startTimestamp` is greater than the current block timestamp.

```
contract MyVestingWallet is VestingWalletCliff {
    constructor(
        address beneficiary,
        uint64 startTimestamp,
        uint64 durationSeconds,
        uint64 cliffSeconds
    )
        VestingWallet(beneficiary, startTimestamp,
durationSeconds)
        VestingWalletCliff(cliffSeconds)
    {}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	tax/StaticTaxHandler.sol#L33,55
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.

Specifically, the `taxPercentage` varaible should not exceed the value of `100_000_000` .

```
   constructor(address initialOwner, uint256
initialTaxPercentage)
      Ownable(initialOwner)
    {
        taxPercentage = initialTaxPercentage;
    }

    function setTaxPercentage(uint256 newTaxPercentage)
external onlyOwner {
        require(
            newTaxPercentage >= 0,
            "Invalid Value."
        );

        uint256 oldTaxPercentage = taxPercentage;
        taxPercentage = newTaxPercentage;

        emit TaxPercentageUpdated(oldTaxPercentage,
newTaxPercentage);
    }
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	AraraCoin.sol#L18,19,20,21,22,23,24,25,26,27,28,29
Status	Unresolved

Description

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

```
address public marketingWallet =
0x23618e81E3f5cdF7f54C3d65f7FBc0aBf5B21E8f
address public consultingWallet =
0xa0Ee7A142d267C1f36714E4a8F75612F20a79720
address public auditWallet =
0x70997970C51812dc3A010C7d01b50e0d17dc79C8
address public preSaleWallet =
0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC
address public launchWallet =
0x90F79bf6EB2c4f870365E785982E1f101E93b906
address public investorsWallet =
0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65
address public investorsYearOneVestingWallet =
0x71bE63f3384f5fb98995898A86B02Fb2426c5788
address public investorsYearTwoVestingWallet =
0xFABB0ac9d68B0B445fB7357272Ff202C5651694a
address public teamVestingWallet =
0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc
address public foundersVestingWallet =
0x1CBd3b2770909D4e10f157cABC84C7264073C9Ec
address public companyVestingWallet =
0xdF3e18d64BC6A983f673Ab319CCaE4f1a57C7097
address public preservationProjectsVestingWallet =
0x976EA74026E726554dB657fA54763abd0C3a0aa9
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	StaticTaxHandler.sol#L56
Status	Unresolved

Description

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

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

```
require(
            newTaxPercentage >= 0,
            "Invalid Value."
        )
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	AraraCoin.sol#L106
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.

```
taxWallet = taxWalletAddress
```

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	VestingWalletCliff.sol#L3 StaticTaxHandler.sol#L2 MyVestingWallet.sol#L2 AraraCoin.sol#L3 abstract/VestingWalletCliff.sol#L3
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;
pragma solidity ^0.8.24;
```

Recommendation

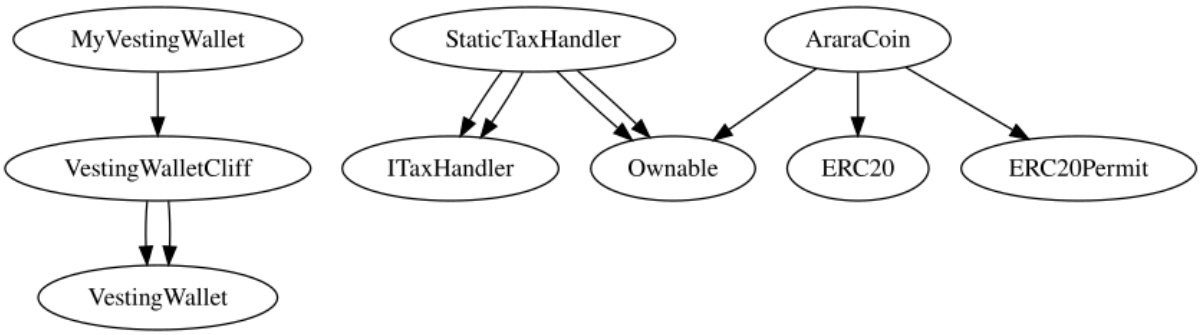
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

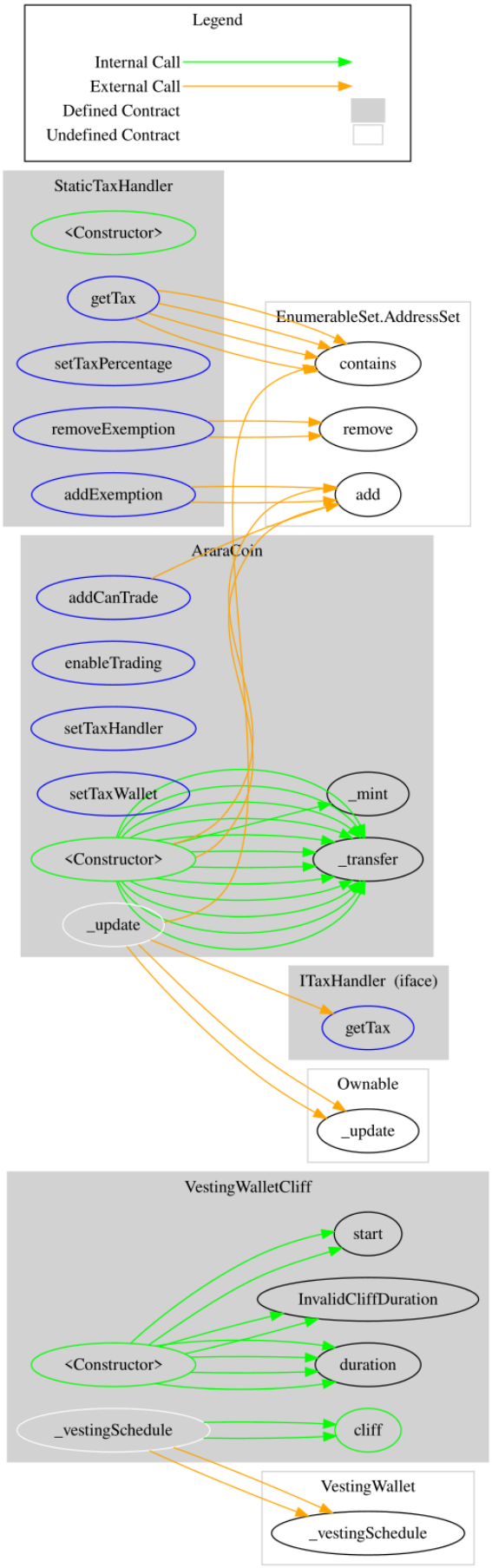
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
VestingWalletCliff	Implementation	VestingWallet		
		Public	✓	-
	cliff	Public		-
	_vestingSchedule	Internal		
StaticTaxHandler	Implementation	ITaxHandler, Ownable		
		Public	✓	Ownable
	getTax	External		-
	setTaxPercentage	External	✓	onlyOwner
	addExemption	External	✓	onlyOwner
	removeExemption	External	✓	onlyOwner
MyVestingWallet	Implementation	VestingWalletCliff		
		Public	✓	VestingWallet VestingWalletCliff
AraraCoin	Implementation	ERC20, ERC20Permit, Ownable		
		Public	✓	ERC20 ERC20Permit Ownable

	enableTrading	External	✓	onlyOwner
	setTaxHandler	External	✓	onlyOwner
	setTaxWallet	External	✓	onlyOwner
	addCanTrade	External	✓	onlyOwner
	_update	Internal	✓	
StaticTaxHandler	Implementation	ITaxHandler, Ownable		
		Public	✓	Ownable
	getTax	External		-
	setTaxPercentage	External	✓	onlyOwner
	addExemption	External	✓	onlyOwner
	removeExemption	External	✓	onlyOwner
VestingWalletCliff	Implementation	VestingWallet		
		Public	✓	-
	cliff	Public		-
	_vestingSchedule	Internal		

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
StaticTaxHandler
<Constructor>
getTax
setTaxPercentage
removeExemption
addExemption
EnumerableSet.AddressSet
contains
remove
add
AraraCoin
addCanTrade
enableTrading
setTaxHandler
setTaxWallet
<Constructor>
_update
_mint
_transfer
ITaxHandler (iface)
getTax
Ownable
_update
VestingWalletCliff
<Constructor>
_vestingSchedule
start
InvalidCliffDuration
duration
cliff
VestingWallet
_vestingSchedule

Summary

The AraraCoin contract implements a token with vesting and tax management features. This audit investigates security vulnerabilities, business logic flaws, and potential improvements. Certain functions, such as the ability to manipulate fees and control trading, could be abused by the owner. Implementing a multi-signature wallet for key actions will enhance security, while renouncing ownership or temporarily locking the contract can mitigate risks associated with owner privilege abuse.

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