Cyberscope

A TAC Security Company

Audit Report

Ruvi AI

June 2025

Network	ETH
Address	0xb26FaC9E2ca768A2294e74fF11Fa9c79A698F307
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	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	MEE	Missing Events Emission	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PIF	Potentially Inaccessible Funds	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSRS	Redundant SafeMath Require Statement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	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

Contract Name	RuviToken
Compiler Version	v0.8.29+commit.ab55807c
Optimization	200 runs
Explorer	https://etherscan.io/address/0xb26fac9e2ca768a2294e74ff11fa9c79a698f307
Address	0xb26fac9e2ca768a2294e74ff11fa9c79a698f307
Network	ETH
Symbol	RUVI
Decimals	18
Total Supply	5,000,000,000
Badge Eligibility	Must Fix Criticals

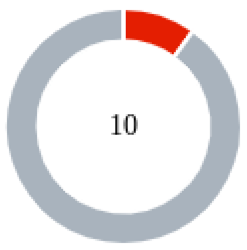
Audit Updates

Initial Audit	09 Jun 2025

Source Files

Filename	SHA256
RuviToken.sol	70ae3398bb1e4c72ee563b28497fd46dde53173b6e56d8113bbfd8d3970bd5da

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	9

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	RuviToken.sol#L254
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 enabled the owner will not be able to disable them again.

```
if (!whitelist[from] && !whitelist[to]) {
require(trading,"Trading is not enabled yet");
}
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	RuviToken.sol#L229,233,237,242
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 removeStuckEth(address _receiver) public onlyOwner {
payable(_receiver).transfer(address(this).balance);
}

function removeStuckToken(address _token, address _receiver, uint256 _amount)
public onlyOwner {
IERC20(_token).transfer(_receiver, _amount);
}
```

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.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	RuviToken.sol#L257
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
require(amount > 0, "Amount must be greater than zero");
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PIF - Potentially Inaccessible Funds

Criticality	Minor / Informative
Location	RuviToken.sol#L233,237
Status	Unresolved

Description

The contract implements methods that handle receiving and sending native and ERC20 tokens. These operations are facilitated through the `removeStuckEth` , `removeStuckToken` , and `receive` methods. If ownership of the contract is renounced and it subsequently receives or holds funds, it will be unable to manage or transfer those funds. As a result, the funds may become locked and permanently inaccessible.

```
function removeStuckEth(address _receiver) public onlyOwner {
payable(_receiver).transfer(address(this).balance);
}

function removeStuckToken(address _token, address _receiver, uint256 _amount)
public onlyOwner {
IERC20(_token).transfer(_receiver, _amount);
}
```

Recommendation

The team should consider implementing safeguards to prevent the contract from receiving funds once the ownership has been renounced. Alternatively, the contract could implement checks to prevent or restrict ownership renouncement while it holds funds.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	RuviToken.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	RuviToken.sol#L275
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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	RuviToken.sol#L125,126,127,128
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.

```
string private _name = "Ruvi AI"
string private _symbol = "RUVI"
uint8 private _decimals = 18
uint256 private _totalSupply = 5_000_000_000 * 1e18
```

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	RuviToken.sol#L47,229,233,242
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
address _receiver
uint256 _amount
address _token
bool _exmpt
address _user
```

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	RuviToken.sol#L230,242
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.

```
payable(_receiver).transfer(address(this).balance)
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	RuviToken.sol#L234
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(_token).transfer(_receiver, _amount)
```

Recommendation

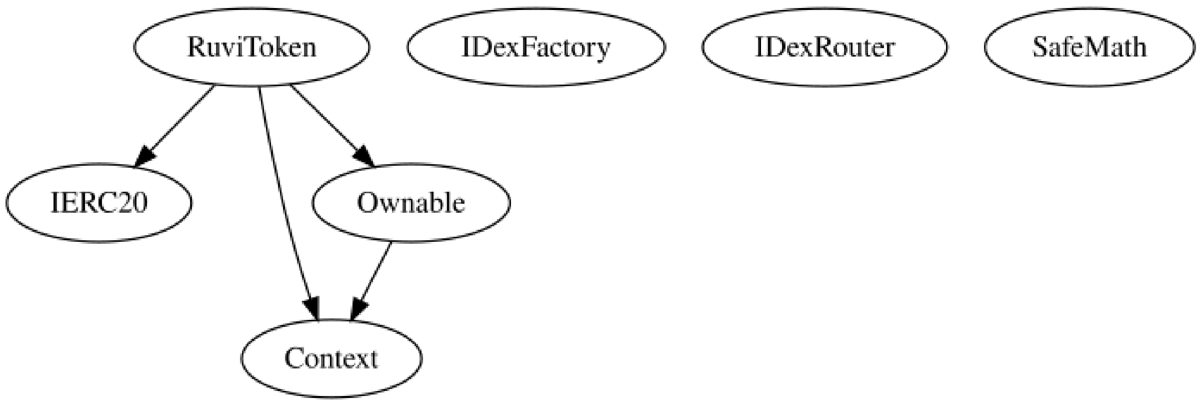
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

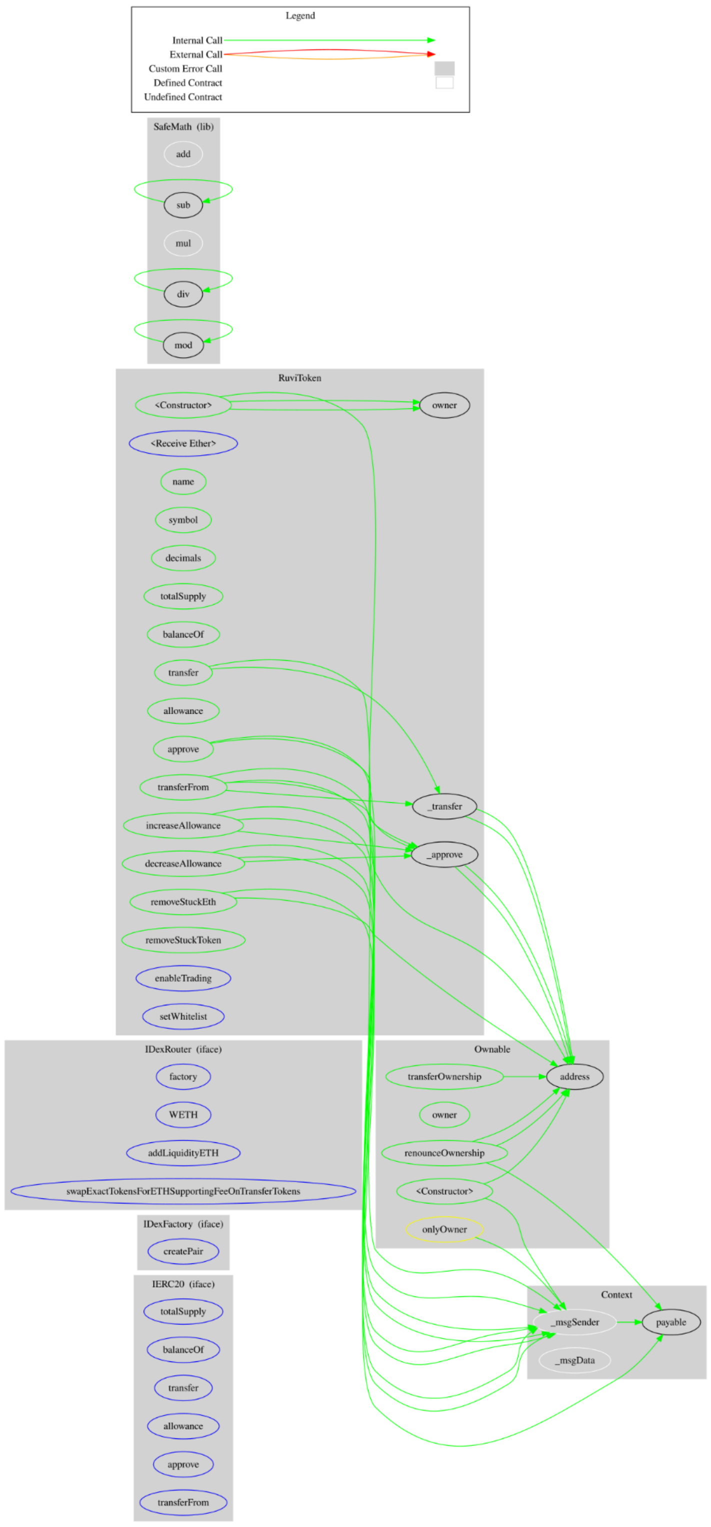
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
RuviToken	Implementation	Context, IERC20, Ownable		
		Public	✓	-
		External	Payable	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	removeStuckEth	Public	✓	onlyOwner
	removeStuckToken	Public	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	setWhitelist	External	✓	onlyOwner
	_approve	Private	✓	

	_transfer	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
mul
div
mod
RuviToken
<Constructor>
owner
<Receive Ether>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
_transfer
increaseAllowance
_approve
decreaseAllowance
removeStuckEth
removeStuckToken
enableTrading
setWhitelist
IDexRouter (iface)
factory
WETH
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens
Ownable
transferOwnership
address
owner
renounceOwnership
<Constructor>
onlyOwner
IDexFactory (iface)
createPair
Context
_msgSender
payable
_msgData
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Ruvi AI 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. 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

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.

A *TAC Security* Company

The Cyberscope team

cyberscope.io