Cyberscope

Audit Report

Rising Coin

June 2023

Network	BSC
Address	0x48d278C2fe7D72c8BfD62c2f0d3415aAec4b7718
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TUU	Time Units Usage	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RC	Repetitive Calculations	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	RisingCoin
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0x48d278c2fe7d72c8bfd62c2f0d3415aaec4b7718
Address	0x48d278c2fe7d72c8bfd62c2f0d3415aaec4b7718
Network	BSC
Symbol	RSC
Decimals	18
Total Supply	100,012,240

Audit Updates

Initial Audit	15 Jun 2023

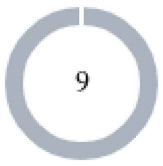
Source Files

Filename	SHA256
RisingCoin.sol	2b7e6e7b413662c477a79c9277a94ead6ed6aaa5f1339ee8d2f0f5d2e28dec3f

Overview

The RisingCoin contract is an ERC20 token contract that enables users to mine tokens, transfer them, and manage approvals. It includes features such as token information storage, ownership management, and a designated wallet for fund withdrawals. The contract allows users to mine tokens by sending a minimum amount of Ether, with daily and total mining limits in place. It implements a halving mechanism every four years to adjust the mining limits. Time management functions ensure users can only mine once per day. Additionally, the contract supports token transfers, approval mechanisms, and tracks token balances for each address. In summary, the RisingCoin contract provides a comprehensive token ecosystem with mining capabilities, ownership control, and standard ERC20 features.

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	9

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

MT - Mints Tokens

Criticality	Minor / Informative
Location	RisingCoin.sol#L149
Status	Unresolved

Description

The contract enables users to acquire $RSC tokens by making a minimum payment of 0.00064 ether. Upon successful payment, the contract mints 10 tokens and adds them to the user's balance, as well as, increasing the `totalSupply`. However, since the `mine` function is public and any user can call it, it arises the possibility of a significant inflation of the contract's token supply.

```
function mine()  external payable returns (bool)  {
    require(msg.value >= 0.00064 ether);
    require(block.timestamp >= lastMineTime[msg.sender] + mingingInterval,
"Cannot mine again today.");


    require( totalSupply_ <= maxSupply, "No more tokens can be mined.");

    lastMineTime[msg.sender] = block.timestamp;
    updateDailyTotal();

    require(dailyMineTotal <= dailyMineLimit);

    updateMintLimit();


    balances[msg.sender] += reward;
    totalSupply_ += reward;
    dailyMineTotal += reward;
    wallet.transfer(msg.value);
    // emit Transfer(address(0), msg.sender, amount);
    return true ;
}
```

Recommendation

The team is advised to enhance the logic of the `mine` function to make the minted token amount proportional to the price of the token at the time of calling the function. This adjustment aims to ensure that the number of tokens minted for the user aligns with the token's current price. If the token price is high, minting only a small amount of tokens may not have a significant impact on the token's price if the user decides to sell them. Conversely, if the minted tokens are trivial compared to the token's price, the mining feature becomes less attractive as users might prefer to purchase tokens from a decentralized exchange (DEX) instead. By aligning the minted token amount with the token's price, the team can create a more balanced and valuable mining experience for users.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	RisingCoin.sol#L23,30,84,85,177,186
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 public fourYears = 4 * 365 * 24 * 60 * 60;
uint256 public mingingInterval =  24 * 60* 60;
...
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	RisingCoin.sol#L164
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.

```
balances[msg.sender] += reward;
totalSupply_ += reward;
dailyMineTotal += reward;
wallet.transfer(msg.value);
```

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.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	RisingCoin.sol#L84,85,177,186
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

The contract executes the operations `24 * 60* 60` and `4 * 365 * 24 * 60 * 60` more than once. The result of these calculations is already stored in the `mingingInterval` and `fourYears` variables respectively. As a result, the contract will require additional resources when executing the functions thatinclude these operations.

```
timeSinceLastMined = block.timestamp + 24 * 60* 60;
timeSinceLastHalving = block.timestamp + 4 * 365 * 24 * 60 * 60;
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	RisingCoin.sol#L77,78,79,80,81,83
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
totalSupply_
decimals
name
symbol
owner
wallet
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	RisingCoin.sol#L13,18,23,25,30,31
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.

```
uint public maxSupply = 1000000000 * 10 ** 18
uint256 public reward = 10 * 10 ** 18
uint256 public fourYears = 4 * 365 * 24 * 60 * 60
uint256 public lastHalving
uint256 public mingingInterval =  24 * 60* 60
uint256 public lastMining
```

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	RisingCoin.sol#L114
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.

```
address _who
```

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/v0.8.17/style-guide.html#naming-convention.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	RisingCoin.sol#L83,101
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.

```
wallet = _wallet
destAddr.transfer(amount)
```

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	RisingCoin.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;
```

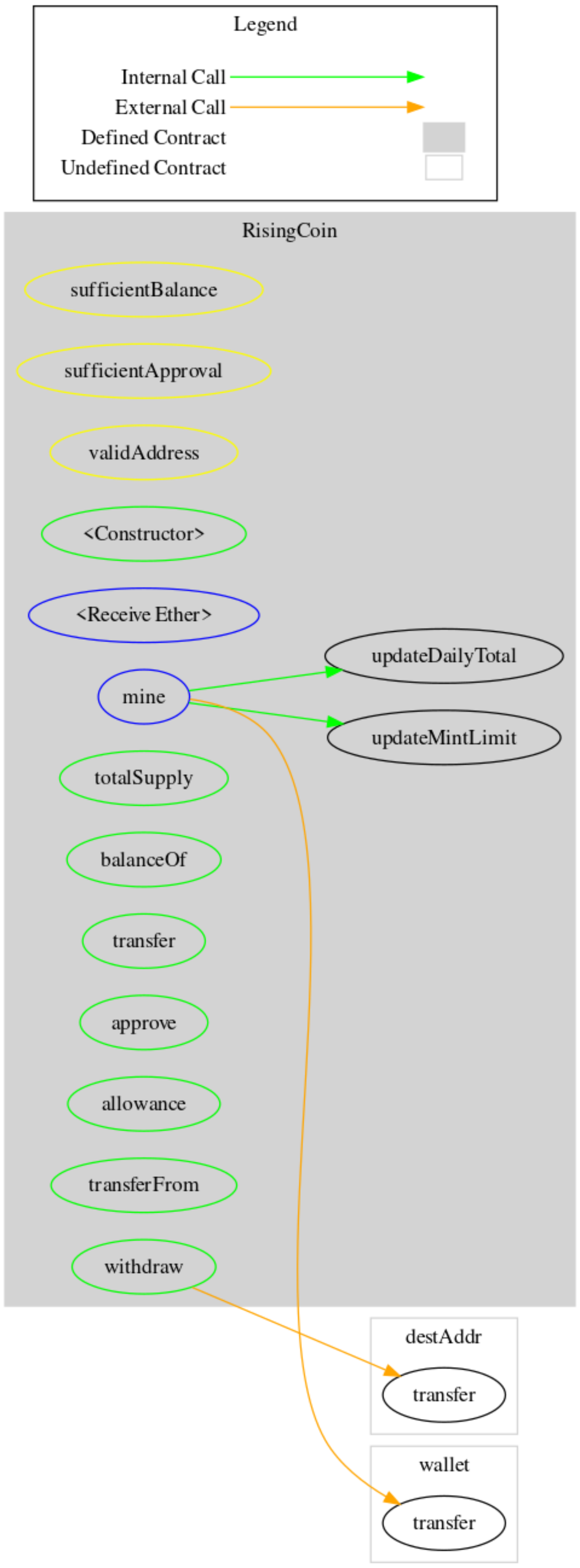
Recommendation

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
RisingCoin	Implementation			
		Public	✓	-
		External	Payable	-
	withdraw	Public	✓	-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	sufficientBalance validAddress
	approve	Public	✓	validAddress
	allowance	Public		-
	transferFrom	Public	✓	sufficientBalance sufficientApproval validAddress
	mine	External	Payable	-
	updateMintLimit	Internal	✓	
	updateDailyTotal	Internal	✓	

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
RisingCoin
sufficientBalance
sufficientApproval
validAddress
<Constructor>
<Receive Ether>
mine
updateDailyTotal
updateMintLimit
totalSupply
balanceOf
transfer
approve
allowance
transferFrom
withdraw
destAddr
transfer
wallet
transfer

Summary

Rising Coin contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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

https://www.cyberscope.io