Cyberscope

Audit Report

Rizz Token

October 2024

Network	ETH
Address	0x582dD5e7C8AF79d45a96dE4af5D1152a061aBb50
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	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	IDI	Immutable Declaration Improvement	Unresolved
	MEM	Missing Error Messages	Unresolved
	MEE	Missing Events Emission	Unresolved
	NCE	Non Compliant ERC20	Unresolved
	RRA	Redundant Repeated Approvals	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	TTR	Token Transfers Reversion Due To Multiple Swaps	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L05	Unused State Variable	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	Rizz
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://etherscan.io/address/0x582dd5e7c8af79d45a96de4af5d1152a061abb50
Address	0x582dd5e7c8af79d45a96de4af5d1152a061abb50
Network	ETH
Symbol	RIZZ
Decimals	9
Total Supply	100,000,000
Badge Eligibility	Yes

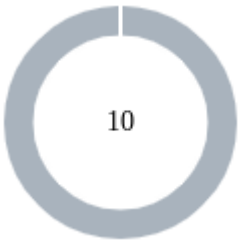
Audit Updates

Initial Audit	10 Oct 2024

Source Files

Filename	SHA256
Rizz.sol	8f967e1feccbe844f3c2834ed5f91cc8d1710fc45acaa3ec12f02e3aa7d7352a

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Rizz.sol#L164
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.

```
_taxWallet
```

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.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	Rizz.sol#L226,330,331,339
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(!bots[from] && !bots[to])
require(_msgSender()==_taxWallet)
require(_newFee<=_finalBuyTax && _newFee<=_finalSellTax)
```

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	Rizz.sol#L303,309,332,333
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.

```
bots[bots_[i]] = true;
...
bots[notbot[i]] = false;
...
_finalBuyTax=_newFee;
_finalSellTax=_newFee;
```

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.

NCE - Non Compliant ERC20

Criticality	Minor / Informative
Location	Rizz.sol#L223
Status	Unresolved

Description

The contract implements an ERC20 token, but it does not fully comply with the ERC20 standard due to the restriction on zero-amount transfers. Specifically, the `_transfer` function includes the following check: `solidity require(amount > 0, "Transfer amount must be greater than zero");` This restriction contradicts the ERC20 standard, which allows transfers of zero tokens. According to the ERC20 specification, a token contract **must** allow for zero-value transfers.

Recommendation

Remove the check to allow for zero-amount transfers, ensuring full compliance with the ERC20 standard.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	Rizz.sol#L280
Status	Unresolved

Description

The contract is designed to approve token transfers during the contract's operation by calling the `_approve` function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
_approve(address(this), address(uniswapV2Router), tokenAmount);
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
    tokenAmount,
    0,
    path,
    address(this),
    block.timestamp
);
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Rizz.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/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

TTR - Token Transfers Reversion Due To Multiple Swaps

Criticality	Minor / Informative
Location	Rizz.sol#L252
Status	Unresolved

Description

The `_transfer` function includes a mechanism that triggers token swapping when certain conditions are met, specifically when the transfer is directed to the Uniswap pair (`to == uniswapV2Pair`), and swap-related flags and thresholds are satisfied. However, the current implementation restricts token swapping to occur only once per block by enforcing the condition:

```
require(block.number > lastExecutedBlockNumber, "Exceeds the
maxWalletSize.");
```

This check ensures that the function will revert any subsequent attempt to trigger a swap within the same block, preventing multiple swaps from occurring. As a result, if two or more token transfers that meet the swap criteria are processed in the same block, any transfer after the first will fail and revert. This could disrupt user experience, especially in scenarios with high-frequency transactions, as multiple valid transfers may be reverted within a single block.

Recommendation

To resolve this issue, it is recommended to remove the `block.number` check that limits swaps to once per block, as it unnecessarily causes valid token transfers to revert when multiple swaps are triggered within the same block. By removing this restriction, token transfers can proceed smoothly without being affected by swap execution frequency.

Additionally, the current error message `Exceeds the maxWalletSize` does not accurately reflect the condition being checked. This message should be revised to provide clarity and align with the actual check being performed. A more appropriate error message should describe the block-based restriction or the intended condition that is failing.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Rizz.sol#L129,130,133,134,135,144,145,154
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.

```
uint256 private _initialBuyTax=20
uint256 private _initialSellTax=20
uint256 private _reduceBuyTaxAt=20
uint256 private _reduceSellTaxAt=20
uint256 private _preventSwapBefore=20
uint256 public _taxSwapThreshold= 1000000 * 10**_decimals
uint256 public _maxTaxSwap= 1000000 * 10**_decimals
uint8 public cooldownTimerInterval = 1
```

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	Rizz.sol#L108,138,139,140,141,142,143,144,145,329
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);
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 100000000 * 10**_decimals
string private constant _name = unicode"Rizz"
string private constant _symbol = unicode"RIZZ"
uint256 public _maxTxAmount = 2000000 * 10**_decimals
uint256 public _maxWalletSize = 2000000 * 10**_decimals
uint256 public _taxSwapThreshold= 1000000 * 10**_decimals
uint256 public _maxTaxSwap= 1000000 * 10**_decimals
uint256 _newFee
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Rizz.sol#L153
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
mapping(address => uint256) private cooldownTimer
```

Recommendation

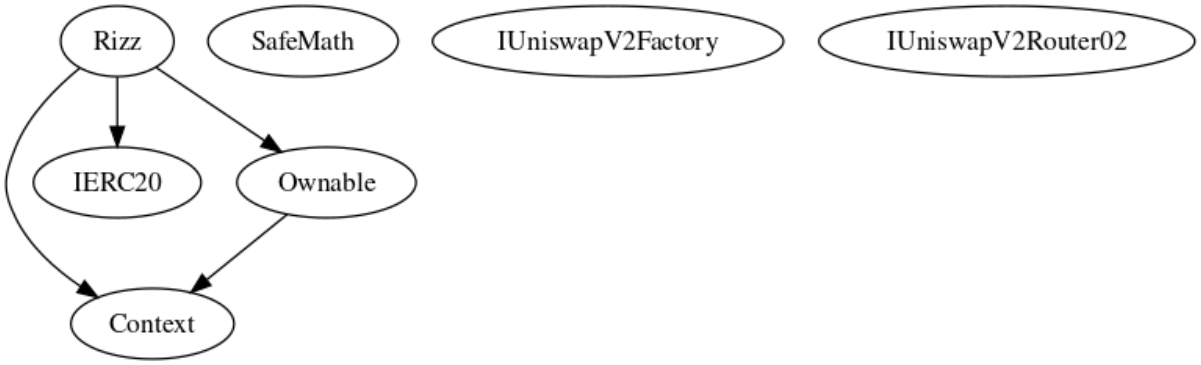
To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

Functions Analysis

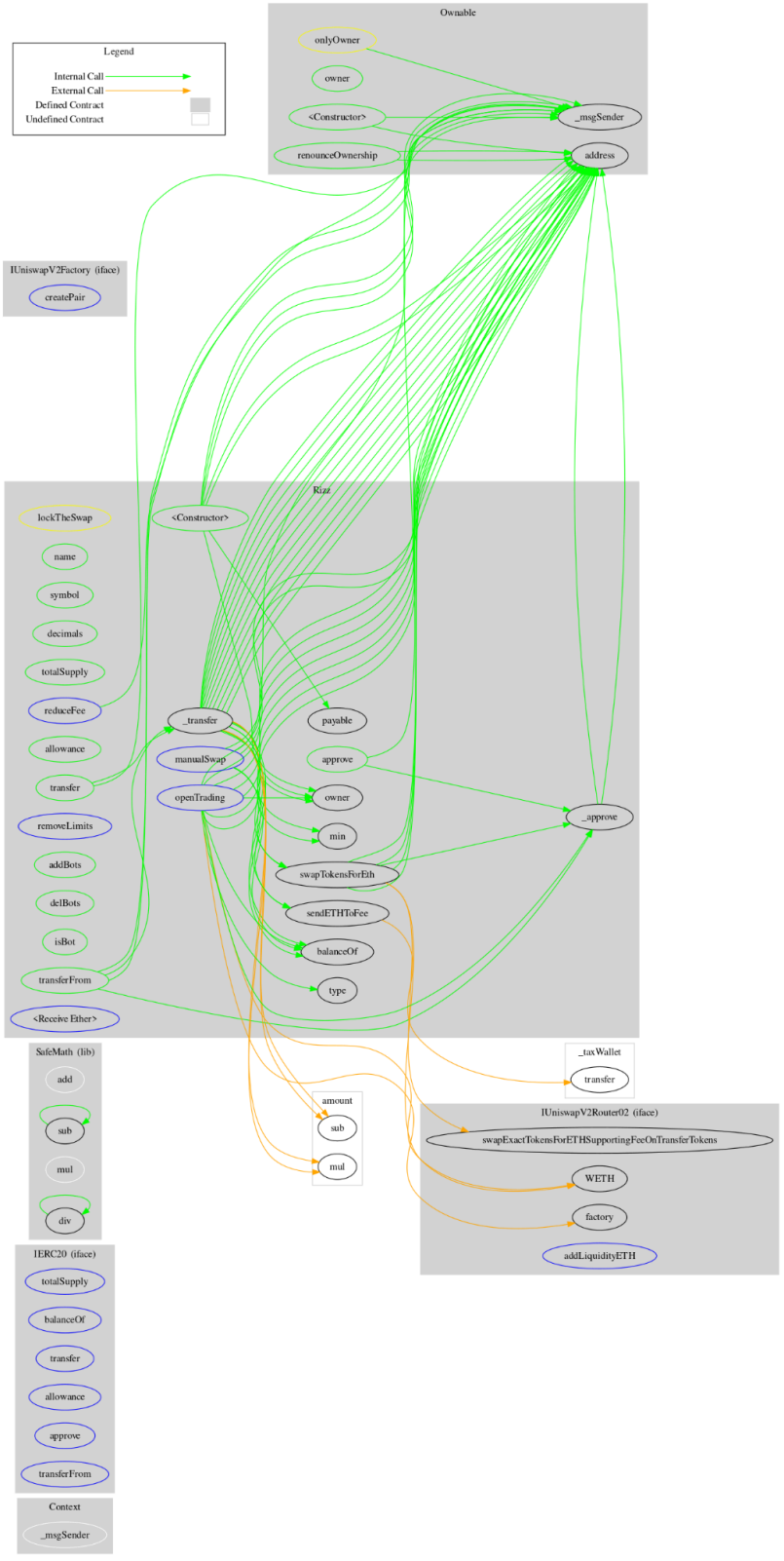
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
Rizz	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	

	min	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	removeLimits	External	✓	onlyOwner
	sendETHToFee	Private	✓	
	addBots	Public	✓	onlyOwner
	delBots	Public	✓	onlyOwner
	isBot	Public		-
	openTrading	External	✓	onlyOwner
	reduceFee	External	✓	-
		External	Payable	-
	manualSwap	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
renounceOwnership
_msgSender
address
IUniswapV2Factory (iface)
createPair
Rizz
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
reduceFee
allowance
transfer
removeLimits
addBots
delBots
isBot
transferFrom
<Receive Ether>
_transfer
manualSwap
openTrading
payable
approve
owner
min
swapTokensForEth
sendETHToFee
balanceOf
type
_approve
SafeMath (lib)
add
sub
mul
div
amount
sub
mul
_taxWallet
transfer
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

Rizz 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 blacklist addresses. 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

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