Cyberscope

Audit Report

Fruitcoins

October 2023

Network BSC

Address 0xa64553008D48cAc0B85aBBb05A50Ad0baAbed1A4

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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	TSD	Total Supply Diversion	Unresolved
Minor / Informative	DDP	Decimal Division Precision	Unresolved
Minor / Informative	RED	Redundant Event Declaration	Unresolved
Minor / Informative	RCS	Redundant Conditional Statement	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	FruitCoins
Compiler Version	v0.8.10+commit.fc410830
Optimization	200 runs
Explorer	https://bscscan.com/address/0xa64553008d48cac0b85abbb05a50ad0baabed1a4
Address	0xa64553008d48cac0b85abbb05a50ad0baabed1a4
Network	BSC
Symbol	FCS
Decimals	18
Total Supply	100,000,000

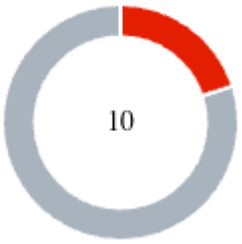
Audit Updates

Initial Audit	13 Oct 2023

Source Files

Filename	SHA256
contracts/fruitcoin.sol	cbeba921e03fcc77f2a3697a655fa8d841564b32077aa0ccffddbdea5c016d09

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/fruitcoin.sol#L25
Status	Unresolved

Description

The contract currently imposes high default sell fees, with a total of 56%, which is significantly above the recommended limit of 25%. These fees can deter users from trading the token, potentially impacting its liquidity and attractiveness to investors. While the contract includes a function (`decreaseTaxPercentage`) to reduce the sell fees after a specific time period, it relies on `addressA` to trigger the reduction. There is no guarantee that `addressA` will execute this function, which could leave the token with high sell fees indefinitely.

```
uint256 public liquidityTaxPercentage = 3; // 3% liquidity tax
uint256 public constant sellTaxCPercentage = 5;
sellTaxAPercentage = 27;
sellTaxBPercentage = 20;

uint256 transferAmount = _value -
    taxAmount -
    liquidityTaxAmount -
    burnAmount;
```

Recommendation

To address this issue and make the contract more attractive to users and investors, the team should consider reducing the default sell fees to a reasonable level, preferably within the 25% limit. This can help maintain liquidity and encourage trading. Additionally, while the contract includes a mechanism to decrease fees over time, it's important to ensure that the reduction process is not solely reliant on a specific address. Implement a governance or time-based automatic reduction system that gradually lowers the fees according to the specified schedule, regardless of manual intervention. This will provide greater transparency and predictability for token holders.

TSD - Total Supply Diversion

Criticality	Critical
Location	contracts/fruitcoin.sol#L151
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the contract, the amount that is added to the total supply does not equal the amount that is added to the balances. As a result, the sum of balances is diverse from the total supply.

Specifically, the `liquidityTaxAmount` is subtracted from the sender's balance, but it's not added to another's address balance.

```
uint256 transferAmount = _value -
    taxAmount -
    liquidityTaxAmount -
    burnAmount;

balanceOf[_from] -= _value;
balanceOf[_to] += transferAmount;

// Apply tax only if it's a sell transaction
if (taxAmount > 0) {
    balanceOf[addressA] += taxAmount / 3; // Distribute taxAmount equally
among addressA, addressB, and addressC
    balanceOf[addressB] += taxAmount / 3;
    balanceOf[addressC] += taxAmount / 3;
    emit Transfer(_from, addressA, taxAmount / 3);
    emit Transfer(_from, addressB, taxAmount / 3);
    emit Transfer(_from, addressC, taxAmount / 3);
}
if (burnAmount > 0) {
    totalSupply -= burnAmount;
    emit Transfer(_from, address(0), burnAmount); // Burn event
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/fruitcoin.sol#L163
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
balanceOf[addressA] += taxAmount / 3; // Distribute taxAmount equally among
addressA, addressB, and addressC
balanceOf[addressB] += taxAmount / 3;
balanceOf[addressC] += taxAmount / 3;
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	contracts/fruitcoin.sol#L32
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.

The contract declares certain events in its code. However, these events are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event Approval(
    address indexed owner,
    address indexed spender,
    uint256 value
);
```

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.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	contracts/fruitcoin.sol#L87
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.

The expression `block.timestamp >= (deploymentTime + 3 days)` is evaluated twice. Since this expression is evaluated inside the require function, the transaction will revert if the condition is false. Hence, the if-statement that follows is redundant.

```
require(
    block.timestamp > deploymentTime + 3 days,
    "ERC20: Can not Change Tax before 3days"
);

// Check if 3 days have passed since the contract deployment and start the
first decraise of tax
if (block.timestamp >= (deploymentTime + 3 days)) {
    // Decrease tax percentage from 27% to 10%
    sellTaxAPercentage = 10;
    sellTaxBPercentage = 7;
}
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/fruitcoin.sol#L59,60,61,62,63,64,65,69
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.

```
name
symbol
decimals
totalSupply
addressA
addressB
addressC
deploymentTime
```

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	contracts/fruitcoin.sol#L22
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 public liquidityTaxPercentage = 3
```

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	contracts/fruitcoin.sol#L176,185,186,187
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 _to
uint256 _value
address _from
```

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	contracts/fruitcoin.sol#L63,64,65
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.

```
addressA = _addressA
addressB = _addressB
addressC = _addressC
```

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	contracts/fruitcoin.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 ^0.8.10;
```

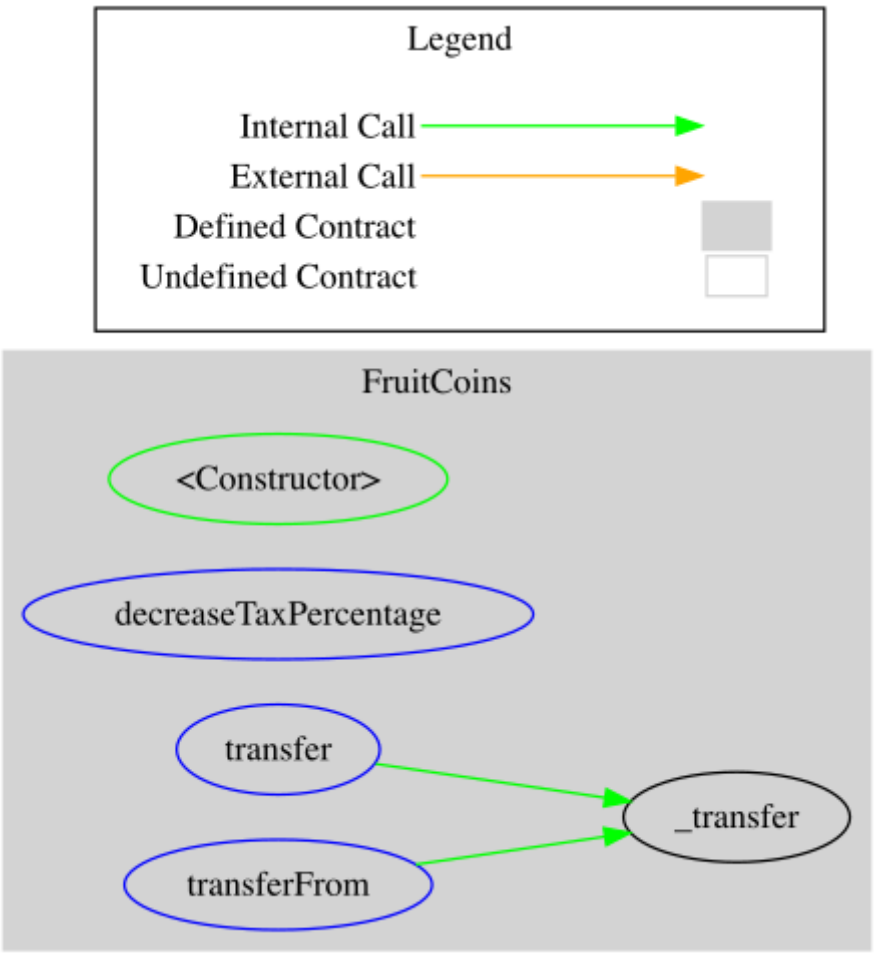
Recommendation

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**
FruitCoins	Implementation			
		Public	✓	-
	decreaseTaxPercentage	External	✓	-
	_transfer	Internal	✓	
	transfer	External	✓	-
	transferFrom	External	✓	-

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
FruitCoins
<Constructor>
decreaseTaxPercentage
transfer
_transfer
transferFrom

Summary

Fruitcoins contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There is a limit of max 15% buy fees and 56% sell fees. The contract implements a mechanism to reduce the sell fees to 26% once 3 days have passed after the contract's deployment, and to 16% once 23 days have passed after the contract's deployment. However, it should be noted that this mechanism can only be activated manually (the `addressA` has to call the `decreaseTaxPercentage` function).

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