Cyberscope

Audit Report

Floyx

March 2023

GitHub	https://github.com/Floyxofficial/FloyxCoin
Commit	8d70bfb6b1dd01eb4987c9dd4fb063fd0270c891
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Floyx
Repository	https://github.com/Floyxofficial/FloyxCoin
Commit	8d70bfb6b1dd01eb4987c9dd4fb063fd0270c891
Testing Deploy	https://testnet.bscscan.com/address/0x52852e9e186315e7a3921bc0b8681d39e2d78cf6 https://testnet.bscscan.com/address/0xca14E80D5A8f6D75f8cA486FE44512FF1D5cf5C4
Symbol	FLOYX
Decimals	18

Audit Updates

Initial Audit	10 Mar 2023

Source Files

Filename	SHA256
access/AccessProtected.sol	c3bddd900e7a1491891603baf5466bf24837aa4d90d110647280a6c8709801f7
Floyx.sol	aea96472362c4faa474e9fc85270ec62eedcf7c39be5f5776d25a5ad584d3cd7
IFloyx.sol	43f6772830e65141b0187763d385e0a6e14e21cedaa85f591cc4ac1e9523eed9
SaleAndVest.sol	e9c145b12ef23582d94249b6fc127b0005721a6d3122805da6106f596a3ff429

Introduction

The Floyx ecosystem consists of two contracts. The Floyx contract, which is an ERC20 token contract, and the SaleAndVest contract, which implements an exchange and vesting mechanism, where a user can buy Floyx tokens with either the native currency or by exchanging USDT/USDC tokens. Users can also earn up to 24% more tokens when claiming their tokens.

Roles

Owner

The owner has authority over the following functions:

- `function setWallet(address payable wallet_)`
- `function updateWeiRate(uint256 rate_)`
- `function updateUsdRate(uint256 rate_)`
- `function updateLockPeriod(uint256 lockPeriod_)`
- `function updateTokenLimit(uint256 tokenLimit_)`
- `function allocateFloyxadmin(address beneficiary_, uint256 floyxAmount_)`
- `function updateCrowdsaleStatus(bool status_)`
- `function adminMaticWithdrawal(uint256 amount_)`
- `function adminFloyxWithdrawal(uint256 _amount)`
- `function adminUsdcWithdrawal(uint256 _amount)`
- `function adminUsdtWithdrawal(uint256 _amount)`

Admin

The admin has authority over the following functions:

- `function mint(address to, uint256 amount)`

User

The user can interact with the following functions:

- `function buyTokens()`
- `function buyTokenswithUsd(uint256 usdAmount_,bool usdc, bool usdt)`
- `function claimTokens()`

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	CO	Code Optimization	Unresolved
●	SC	Stops Claims	Unresolved
●	RA	Redundant Argument	Unresolved
🔴	MT	Mints Tokens	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CO - Code Optimization

Criticality	Minor / Informative
Location	SaleAndVest.sol#L138
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 can call the `mint` function of the `Floyx` contract up to 13 times at once. It would be more efficient to calculate the total amount to be minted and then call `mint` function once at the end. As a result, the `claimTokens` function can consume more gas.

```
function claimTokens()public{
    ...
    if (claimCount[msg.sender] == 0){
        _token.mint(msg.sender, totalTokensPurchased[msg.sender].mul(10).div(100) );
        claimCount[msg.sender] = 1;
    }

    for(uint i = claimCount[msg.sender]; i < monthDiff; i ++){
        claimCount[msg.sender] += 1;
        uint256 tokenAmount = (totalTokensPurchased[msg.sender].mul(75).div(1000));
// release 7.5% of the tokens
        tokenAmount = tokenAmount.add(tokenAmount.mul(2).div(100));
// extra 2% reward on every claim
        _token.mint(msg.sender, tokenAmount);
    }
}
```

Recommendation

The team is advised to take into consideration these segments 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.

SC - Stops Claims

Criticality	Minor / Informative
Location	SaleAndVest.sol#L139 Floyx.sol#L16
Status	Unresolved

Description

The contract owner has the authority to stop users from claiming their tokens. The owner may take advantage of it by setting the `lockPeriod` to a very big number. Additionally, the function mints tokens to the user. If the amount to be minted plus the tokens's total supply is greater than the `maxSupply` then the transaction will revert. This can happen if the owner sets the `maxSupply` to a value very close or equal to the total supply. As a result, the user will not be able to claim his tokens.

```
require(block.timestamp > lockPeriod, "Crowdsale: Can not claim during lock
period");
...
require(maxSupply>= amount+totalSupply(), "Floyx: Can not exceed max supply" );
```

Recommendation

The contract could embody a check for not allowing setting the `lockPeriod` more than a reasonable amount. Regarding the `maxSupply`, the contract could embody a check for not allowing users to buy floyx tokens if the total supply plus the amount exceeds the `maxSupply`. 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 time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

RA - Redundant Argument

Criticality	Minor / Informative
Location	SaleAndVest.sol#L118
Status	Unresolved

Description

The contract function `buyTokenswithUsd` accepts three arguments. The amount of USDC/USDT tokens used to buy FLOYX tokens and two boolean values, which indicate the token the user wants to use. Since there are only two options, the contract can determine the flow of the transaction with only one boolean value. As a result, the second boolean value passed to the function is redundant.

```
function buyTokenswithUsd(uint256 usdAmount_,bool usdc, bool usdt) public
nonReentrant { ... }
```

Recommendation

The team is advised to remove one of the two boolean values from the function's implementation.

MT - Mints Tokens

Criticality	Critical
Location	Floyx.sol#L15
Status	Unresolved

Description

The contract admin has the authority to mint tokens. The admin may take advantage of it by calling the `mint` function. As a result, the contract tokens will be highly inflated.

```
function mint(address to, uint256 amount) external onlyAdmin {
    require(maxSupply>= amount+totalSupply(), "Floyx: Can not exceed max supply" );
    _mint(to, amount);
}
```

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 time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Floyx.sol#L12 SaleAndVest.sol#L43,44,45,47
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
maxSupply
_token
_usdc
_usdt
_icoCap
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	SaleAndVest.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 on 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 unnecessarily the gas consumption.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. 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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	SaleAndVest.sol#L15,16,17,35,197,202,207
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.

```
IFloyx internal _token
IERC20 internal _usdc
IERC20 internal _usdt
event paymentProccessed(address receiver, uint256 amount, bytes info);
uint256 _amount
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	SaleAndVest.sol#L77,81,86,90
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
weiRate = rate_
usdRate = rate_
lockPeriod = lockPeriod_
userTokenLimit = tokenLimit_
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	SaleAndVest.sol#L154,155
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 tokenAmount = (totalTokensPurchased[msg.sender].mul(75).div(1000))
tokenAmount = tokenAmount.add(tokenAmount.mul(2).div(100))
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	SaleAndVest.sol#L42
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 = payable(adminWallet)
```

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	SaleAndVest.sol#L126,199,204,209
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.

```
usdc ? _usdc.transferFrom(msg.sender, address(this), usdAmount_) :
_usdt.transferFrom(msg.sender, address(this), usdAmount_)
_token.transfer(msg.sender, _amount)
_usdc.transfer(msg.sender, _amount)
_usdt.transfer(msg.sender, _amount)
```

Recommendation

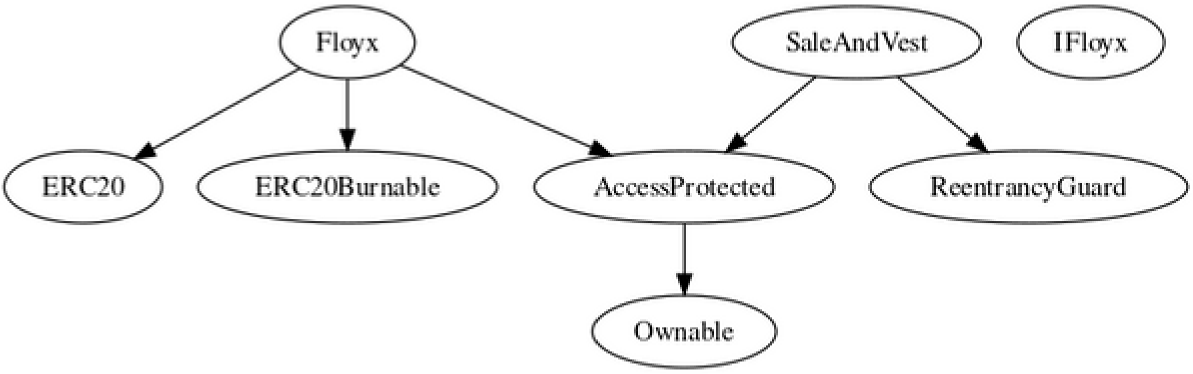
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

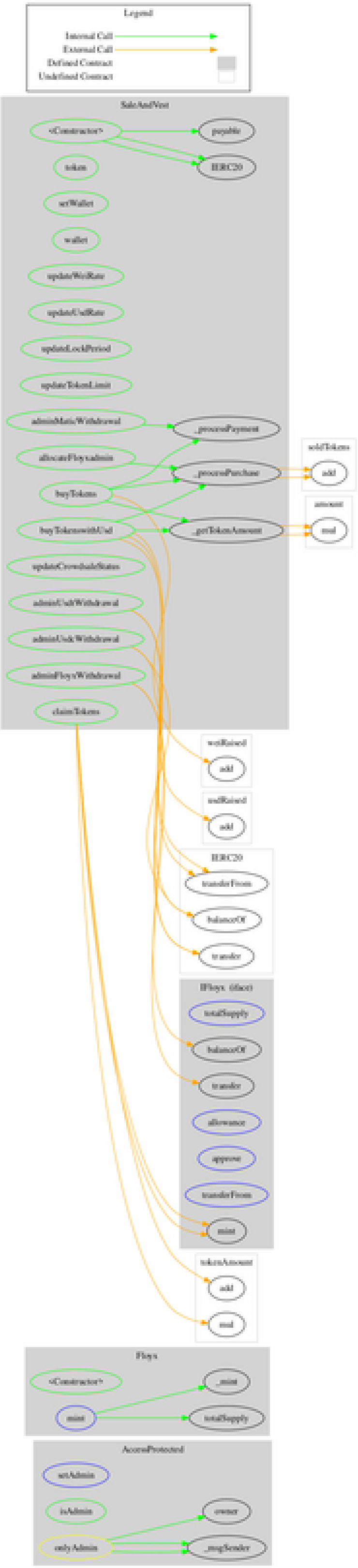
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Floyx	Implementation	ERC20, ERC20Burnable, AccessProtected		
		Public	✓	ERC20
	mint	External	✓	onlyAdmin
IFloyx	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
	mint	External	✓	-
SaleAndVest	Implementation	ReentrancyGuard, AccessProtected		
		Public	✓	Ownable
	token	Public		-
	setWallet	Public	✓	onlyOwner
	wallet	Public		-
	updateWeiRate	Public	✓	onlyOwner

	updateUsdRate	Public	✓	onlyOwner
	updateLockPeriod	Public	✓	onlyOwner
	updateTokenLimit	Public	✓	onlyOwner
	buyTokens	Public	Payable	nonReentrant
	buyTokenswithUsd	Public	✓	nonReentrant
	allocateFloyxadmin	Public	✓	onlyOwner
	claimTokens	Public	✓	-
	updateCrowdsaleStatus	Public	✓	onlyOwner
	_processPurchase	Internal	✓	
	_getTokenAmount	Internal		
	adminMaticWithdrawal	Public	✓	onlyOwner
	adminFloyxWithdrawal	Public	✓	onlyOwner
	adminUsdcWithdrawal	Public	✓	onlyOwner
	adminUsdtWithdrawal	Public	✓	onlyOwner
	_processPayment	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SaleAndVest
<Constructor>
payable
IERC20
token
setWallet
wallet
updateWeiRate
updateUsdRate
updateLockPeriod
updateTokenLimit
adminMaticWithdrawal
_processPayment
allocateFloyxadmin
_processPurchase
buyTokens
buyTokenswithUsd
_getTokenAmount
soldTokens
add
amount
mul
updateCrowdsaleStates
adminUsdtWithdrawal
adminUsdcWithdrawal
adminFloyxWithdrawal
claimTokens
weiRaised
add
usdRaised
add
IERC20
transferFrom
balanceOf
transfer
IFloyx (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
mint
tokenAmount
add
mul
Floyx
<Constructor>
_mint
mint
totalSupply
AccessProtected
setAdmin
isAdmin
owner
onlyAdmin
_msgSender

Summary

Floyx contracts implement a token, vesting, and rewards mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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