Cyberscope

Audit Report

FLOSHIDO INU

February 2023

Type	BEP20
Network	BSC
Address	0x87e04a05499cb8d352c2E367870D4cf0Ead460F0
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 4
Analysis 5
Diagnostics 6
L04 - Conformance to Solidity Naming Conventions 7
Description 7
Recommendation 8
L07 - Missing Events Arithmetic 9
Description 9
Recommendation 9
L09 - Dead Code Elimination 10
Description 10
Recommendation 10
L12 - Using Variables before Declaration 11
Description 11
Recommendation 11
L13 - Divide before Multiply Operation 12
Description 12
Recommendation 12
L14 - Uninitialized Variables in Local Scope 13
Description 13
Recommendation 13
L20 - Succeeded Transfer Check 14
Description 14

Recommendation 14
Contract Functions 15
Inheritance Graph 20
Flow Graph 21
Summary 22
Disclaimer 23
About Cyberscope 24

Review

Contract Name	FLOSHIDOINU
Compiler Version	v0.8.18+commit.87f61d96
Optimization	500 runs
Explorer	https://bscscan.com/address/0x87e04a05499cb8d352c2e367870d4cf0ead460f0
Address	0x87e04a05499cb8d352c2e367870d4cf0ead460f0
Network	BSC
Symbol	Floshido
Decimals	9
Total Supply	100,000,000,000

Audit Updates

Initial Audit	06 Feb 2023 https://github.com/cyberscope-io/audits/blob/main/floshido/v1/audit.pdf
Corrected Phase 2	17 Feb 2023 https://github.com/cyberscope-io/audits/blob/main/floshido/v2/audit.pdf
Corrected Phase 3	20 Feb 2023

Source Files

Filename	SHA256
FLOSHIDOINU.sol	0a98718c2e044d31bf84c2f381523a3f892078dd89d53449b1dc774d6bf99d3e

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L12	Using Variables before Declaration	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L20	Succeeded Transfer Check	Unresolved

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	FLOSHIDOINU.sol#L33,115,118,127,128,129,130,131,146,152,159,160,161,162,163,187,412,481
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);
mapping (address => uint256) _tOwned
mapping (address => mapping (address => uint256)) _allowances
uint256 constant private startingSupply = 100_000_000_000
string constant private _name = "FLOSHIDO INU"
string constant private _symbol = "Floshido"
uint8 constant private _decimals = 9
uint256 constant private _tTotal = startingSupply * (10 ** _decimals)

Fees public _taxRates = Fees({
        buyFee: 300,
        sellFee: 500,
        transferFee: 0
    })

...
```

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	FLOSHIDOINU.sol#L449,461,471,488
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.

```
_maxWalletSize = (_tTotal * percent) / divisor
swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor
piSwapPercent = priceImpactSwapPercent
cashierGas = gas
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	FLOSHIDOINU.sol#L527
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _basicTransfer(address from, address to, uint256 amount) internal returns
(bool) {
        _tOwned[from] -= amount;
        _tOwned[to] += amount;
        emit Transfer(from, to, amount);
        return true;
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	FLOSHIDOINU.sol#L640,652
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint256 initThreshold
uint256 initSwapAmount
bool check
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	FLOSHIDOINU.sol#L715,716
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 a loss of prediction.

```
uint256 feeAmount = amount * currentFee / masterTaxDivisor
uint256 burnAmount = feeAmount * ratios.burn / total
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	FLOSHIDOINU.sol#L640,651,652
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 initSwapAmount
uint256 initThreshold
bool checked
bool check
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	FLOSHIDOINU.sol#L752
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.

```
TOKEN.transfer(_owner, TOKEN.balanceOf(address(this)))
```

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.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IFactoryV2	Interface			
	getPair	External		-
	createPair	External	✓	-
IV2Pair	Interface			
	factory	External		-
	getReserves	External		-
	sync	External	✓	-
IRouter01	Interface			
	factory	External		-
	WETH	External		-

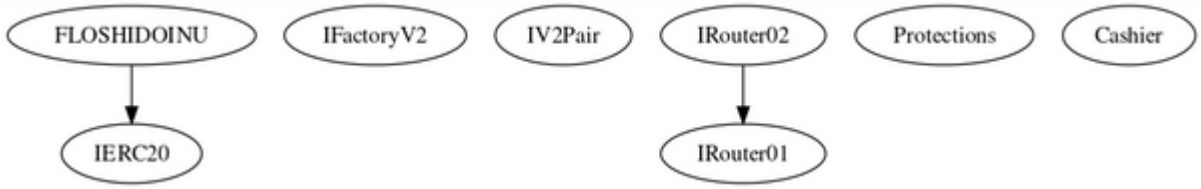
	addLiquidityETH	External	Payable	-
	addLiquidity	External	✓	-
	swapExactETHForTokens	External	Payable	-
	getAmountsOut	External		-
	getAmountsIn	External		-
IRouter02	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
Protections	Interface			
	checkUser	External	✓	-
	setLaunch	External	✓	-
	getInits	External	✓	-
	setLpPair	External	✓	-
	setProtections	External	✓	-
	removeSniper	External	✓	-
Cashier	Interface			
	setRewardsProperties	External	✓	-
	tally	External	✓	-
	load	External	Payable	-
	cashout	External	✓	-
	giveMeWelfarePlease	External	✓	-
	getTotalDistributed	External		-
	getUserInfo	External		-

	getUserRealizedRewards	External		-
	getPendingRewards	External		-
	initialize	External	✓	-
	getCurrentReward	External		-
FLOSHIDOINU	Implementation	IERC20		
		Public	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	Public	✓	onlyOwner
	transfer	External	✓	-
	transferFrom	External	✓	-
	setNewRouter	External	✓	onlyOwner
	setLpPair	External	✓	onlyOwner
	setInitializers	External	✓	onlyOwner
	isExcludedFromFees	External		-
	isExcludedFromDividends	External		-
	isExcludedFromProtection	External		-
	isExcludedFromLimits	External		-

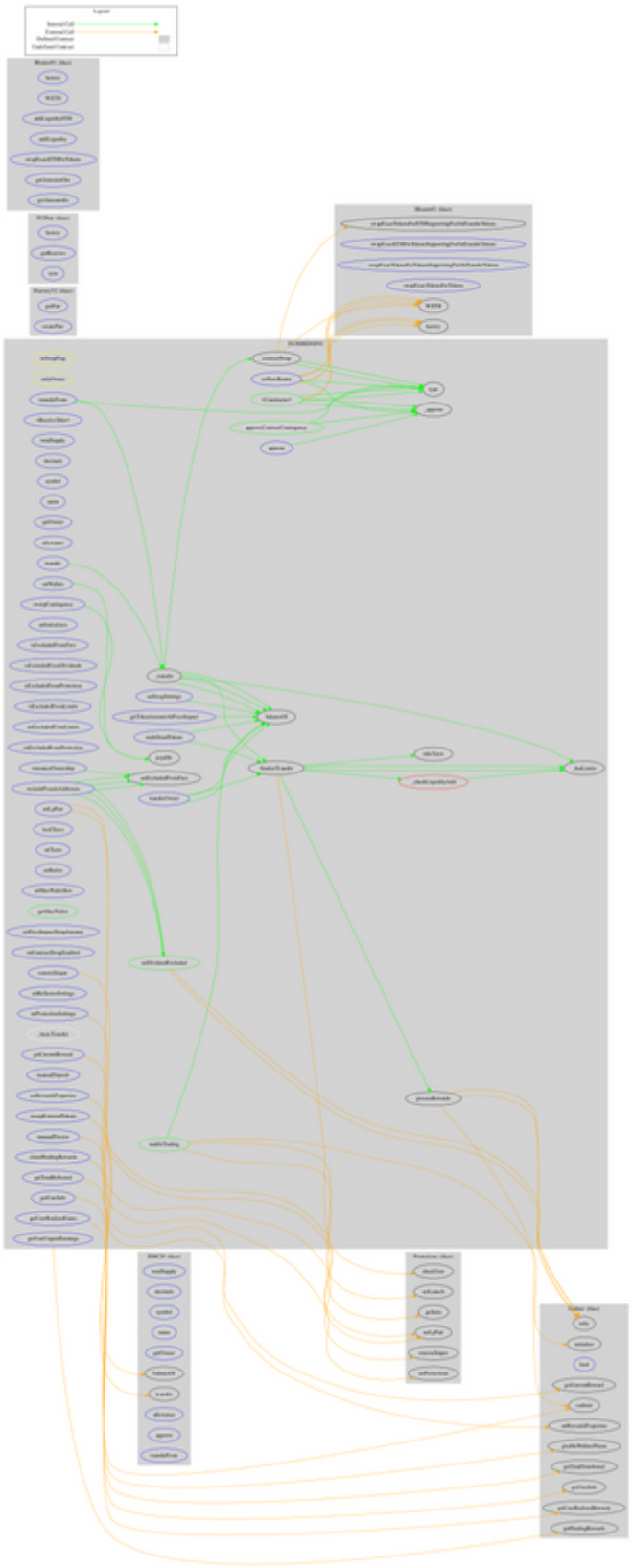
	setExcludedFromLimits	External	✓	onlyOwner
	setDividendExcluded	Public	✓	onlyOwner
	setExcludedFromFees	Public	✓	onlyOwner
	setExcludedFromProtection	External	✓	onlyOwner
	removeSniper	External	✓	onlyOwner
	setProtectionSettings	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	lockTaxes	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setRatios	External	✓	onlyOwner
	setMaxWalletSize	External	✓	onlyOwner
	getMaxWallet	Public		-
	getTokenAmountAtPriceImpact	External		-
	setSwapSettings	External	✓	onlyOwner
	setPriceImpactSwapAmount	External	✓	onlyOwner
	setContractSwapEnabled	External	✓	onlyOwner
	setRewardsProperties	External	✓	onlyOwner
	setReflectorSettings	External	✓	onlyOwner
	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_basicTransfer	Internal	✓	
	_transfer	Internal	✓	
	contractSwap	Internal	✓	inSwapFlag
	_checkLiquidityAdd	Private	✓	
	enableTrading	Public	✓	onlyOwner
	finalizeTransfer	Internal	✓	
	processRewards	Internal	✓	
	manualProcess	External	✓	-
	takeTaxes	Internal	✓	

	multiSendTokens	External	✓	onlyOwner
	manualDeposit	External	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	sweepExternalTokens	External	✓	onlyOwner
	claimPendingRewards	External	✓	-
	getTotalReflected	External		-
	getUserInfo	External		-
	getUserRealizedGains	External		-
	getUserUnpaidEarnings	External		-
	getCurrentReward	External		-

Inheritance Graph



Flow Graph



Summary

FLOSHIDO INU 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. There is also a limit of max 20% fees.

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

Cyberscope

Audit Report

FLOSHIDO INU

February 2023

Type	BEP20
Network	BSC
Address	0x84E70f388AAD5b4Df0Ee5935fdA76C27C3Bb63aD
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Analysis 4
BC - Blacklists Addresses 5
Description 5
Recommendation 5
Diagnostics 6
RSML - Redundant SafeMath Library 8
Description 8
Recommendation 8
ISP - Incorrect Swap Path 9
Description 9
Recommendation 10
L04 - Conformance to Solidity Naming Conventions 11
Description 11
Recommendation 12
L05 - Unused State Variable 13
Description 13
Recommendation 13
L07 - Missing Events Arithmetic 14
Description 14
Recommendation 14
L09 - Dead Code Elimination 15
Description 15

Recommendation 16
L12 - Using Variables before Declaration 17
Description 17
Recommendation 17
L14 - Uninitialized Variables in Local Scope 18
Description 18
Recommendation 18
L15 - Local Scope Variable Shadowing 19
Description 19
Recommendation 19
L16 - Validate Variable Setters 20
Description 20
Recommendation 20
L17 - Usage of Solidity Assembly 21
Description 21
Recommendation 22
L19 - Stable Compiler Version 23
Description 23
Recommendation 23
L20 - Succeeded Transfer Check 24
Description 24
Recommendation 24
Contract Functions 25
Inheritance Graph 34
Flow Graph 35
Summary 36
Disclaimer 37
About Cyberscope 38

Review

Contract Name	CoinToken
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x84e70f388aad5b4df0ee5935fda76c27c3bb63ad
Address	0x84e70f388aad5b4df0ee5935fda76c27c3bb63ad
Network	BSC
Symbol	Floshido
Decimals	18
Total Supply	99,999,999,999

Audit Updates

Initial Audit	06 Feb 2023

Source Files

Filename	SHA256
CoinToken.sol	94c91e8981795b7064a0e604c7674bda3ed826f2d48ca49dc08eb8d18986dea2

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

BC - Blacklists Addresses

Criticality	Medium
Location	CoinToken.sol#L1377
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `EnemyAddress` function.

```
function EnemyAddress(address account, bool value) external onlyOwner{
    _isEnemy[account] = value;
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	RSML	Redundant SafeMath Library	Unresolved
●	ISP	Incorrect Swap Path	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

●	L20	Succeeded Transfer Check	Unresolved

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	CoinToken.sol#L81
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.

ISP - Incorrect Swap Path

Criticality	Critical
Location	CoinToken.sol#L1643
Status	Unresolved

Description

The `swapTokensForCake()` function declares a path array with a length of 3. The reward address should always be fixed to the wBNB address since the dividend tracker reward token cannot be changed. Since the reward address is always the wBNB address, then the last index of the path is not initialized. The pair wBNB - zero address causes the router swap mechanism to revert. This revert propagates to the contract and causes the entire transfer to revert.

```
function swapTokensForCake(uint256 tokenAmount) private {
    address[] memory path = new address[](3);
    path[0] = address(this);
    if(rewardToken == uniswapV2Router.WETH()){
        path[1] = rewardToken;
    }else{
        path[1] = uniswapV2Router.WETH();
        path[2] = rewardToken;
    }
    _approve(address(this), address(uniswapV2Router), tokenAmount);
    // make the swap
    uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
        tokenAmount,
        0,
        path,
        address(this),
        block.timestamp
    );
}
```

Recommendation

To resolve these issue, the team is advised to set the `swapTokensAtAmount` variable a high value so that the contract will not trigger the `swapTokensForCake()` function. The contract fees should also be set to zero, so that the users are not charged without a fee that it is unabled to be claimed.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CoinToken.sol#L539,703,704,721,791,796,864,871,878,888,1013,1161,1164,1170,1174,1178,1189,1232,1233,1234,1236,1240,1377
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);
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
address public REWARD_TOKEN
uint256 constant internal magnitude = 2**128
address _owner
address _account

function MAPGet(address key) public view returns (uint) {
        return tokenHoldersMap.values[key];
    }

...
```

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	CoinToken.sol#L138
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.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	CoinToken.sol#L1486,1495,1504
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.

```
swapTokensAtAmount = amount
buyTokenRewardsFee = rewardsFee
sellTokenRewardsFee = rewardsFee
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	CoinToken.sol#L184,210,228,242,262,898
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function clone(address implementation) internal returns (address instance) {
...
            mstore(ptr,
0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000000)
            mstore(add(ptr, 0x14), shl(0x60, implementation))
            mstore(add(ptr, 0x28),
0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000000)
            instance := create(0, ptr, 0x37)
        }
        require(instance != address(0), "ERC1167: create failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	CoinToken.sol#L1588
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint256 lastProcessedIndex
uint256 iterations
uint256 claims
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	CoinToken.sol#L1550,1554,1588
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 fees
uint256 DFee
uint256 lastProcessedIndex
uint256 iterations
uint256 claims
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	CoinToken.sol#L816,864,871,878,888,1301
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
address _owner
uint256 totalSupply = totalSupply_ * (10**18)
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CoinToken.sol#L817,1316,1350,1369,1490
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.

```
REWARD_TOKEN = _rewardTokenAddress
uniswapV2Pair = _uniswapV2Pair
_marketingWalletAddress = wallet
deadWallet = addr
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	CoinToken.sol#L211,229,247
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
            let ptr := mload(0x40)
            mstore(ptr,
0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000)
            mstore(add(ptr, 0x14), shl(0x60, implementation))
            mstore(add(ptr, 0x28),
0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000)
            instance := create(0, ptr, 0x37)
...
assembly {
            let ptr := mload(0x40)
            mstore(ptr,
0x3d602d80600a3d3981f3363d3d373d3d3d363d730000000000000000000000000)
            mstore(add(ptr, 0x14), shl(0x60, implementation))
            mstore(add(ptr, 0x28),
0x5af43d82803e903d91602b57fd5bf3000000000000000000000000000000000)
            instance := create2(0, ptr, 0x37, salt)
        }

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	CoinToken.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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	CoinToken.sol#L1601
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(rewardToken).transfer(_marketingWalletAddress, newBalance)
```

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.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
Clones	Library			
	clone	Internal	✓	
	cloneDeterministic	Internal	✓	
	predictDeterministicAddress	Internal		
	predictDeterministicAddress	Internal		
ERC20	Implementation	Context, IERC20, IERC20Metadata		

		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_cast	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-

	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
DividendPayingTokenInterfa	Interface			

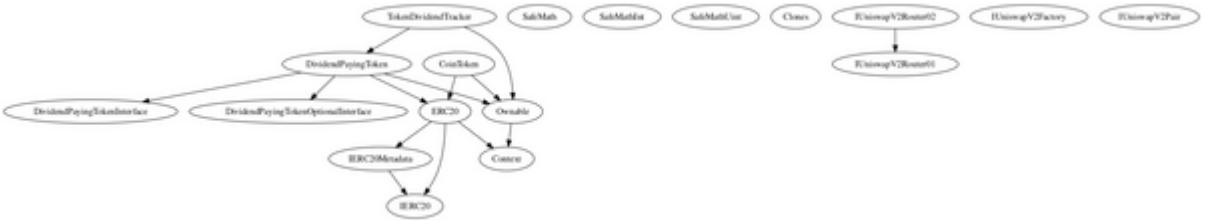
ce				
	dividendOf	External		-
	withdrawDividend	External	✓	-
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
	distributeCAKEDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_cast	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
TokenDividendTracker	Implementation	Ownable, DividendPayingToken		

		Public	✓	DividendPayingToken
	_transfer	Internal		
	withdrawDividend	Public		-
	setMinimumTokenBalanceForDividends	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	isExcludedFromDividends	Public		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
	MAPGet	Public		-
	MAPGetIndexOfKey	Public		-
	MAPGetKeyAtIndex	Public		-
	MAPSize	Public		-
	MAPSet	Public	✓	-
	MAPRemove	Public	✓	-
CoinToken	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
		External	Payable	-
	updateMinimumTokenBalanceForDividends	Public	✓	onlyOwner
	updateUniswapV2Router	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner

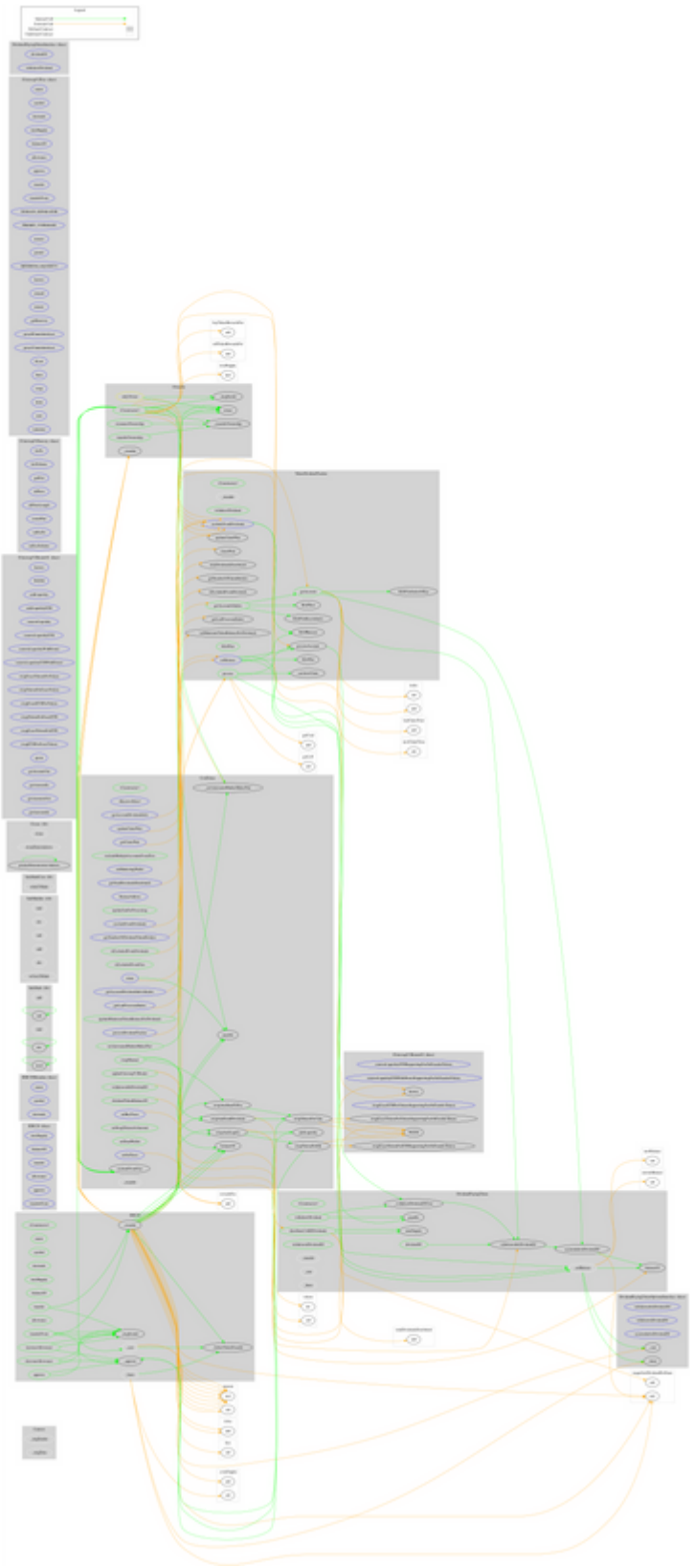
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	EnemyAddress	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	swapManual	Public	✓	onlyOwner
	setSwapTokensAtAmount	Public	✓	onlyOwner
	setDeadWallet	Public	✓	onlyOwner
	setBuyTaxes	External	✓	onlyOwner
	setSelTaxes	External	✓	onlyOwner
	_transfer	Internal	✓	
	swapAndSendToFee	Private	✓	
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	

	swapTokensForCake	Private	✓	
	addLiquidity	Private	✓	
	swapAndSendDividends	Private	✓	

Inheritance Graph



Flow Graph



Summary

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. The contract contains a major issue in the swap mechanism, there are more information in the ISP - Incorrect Swap Path finding. There is also a limit of max 25% buy/sell fees.

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

Cyberscope

Audit Report

FLOSHIDO INU

February 2023

SHA256 24cb5596707239103212c659d4463291888e2e1059128b07c2bb663d71cde99f

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Analysis 4
Diagnostics 5
L04 - Conformance to Solidity Naming Conventions 6
Description 6
Recommendation 7
L07 - Missing Events Arithmetic 8
Description 8
Recommendation 8
L09 - Dead Code Elimination 9
Description 9
Recommendation 9
L12 - Using Variables before Declaration 10
Description 10
Recommendation 10
L13 - Divide before Multiply Operation 11
Description 11
Recommendation 11
L14 - Uninitialized Variables in Local Scope 12
Description 12
Recommendation 12
L20 - Succeeded Transfer Check 13
Description 13

Recommendation 13
Contract Functions 14
Inheritance Graph 19
Flow Graph 20
Summary 21
Disclaimer 22
About Cyberscope 23

Review

Contract Name	FLOSHIDOINU
Testing Deploy	https://testnet.bscscan.com/address/0xea0ea4c86af8a3fdbd5a5b6ae1374e307fd59281
Symbol	FLOSHIDO
Decimals	9
Total Supply	100,000,000,000

Audit Updates

Initial Audit	06 Feb 2023 https://github.com/cyberscope-io/audits/blob/main/floshido/v1/audit.pdf
Corrected Phase 2	17 Feb 2023

Source Files

Filename	SHA256
contracts/2-15-2023_FLOSHIDO_INU.sol	24cb5596707239103212c659d4463291888e2e1059128b07c2bb663d71cde99f

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/2-15-2023_FLOSHIDO_INU.sol#L33,115,118,127,128,129,130,131,146,152,159,160,161,162,163,187,412,481
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);
mapping (address => uint256) _tOwned
mapping (address => mapping (address => uint256)) _allowances
uint256 constant private startingSupply = 100_000_000_000
string constant private _name = "FLOSHIDO INU"
string constant private _symbol = "FLOSHIDO"
uint8 constant private _decimals = 9
uint256 constant private _tTotal = startingSupply * (10 ** _decimals)

Fees public _taxRates = Fees({
        buyFee: 1000,
        sellFee: 1000,
        transferFee: 1000
    })

...
```

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	contracts/2-15-2023_FLOSHIDO_INU.sol#L449,461,471,488
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.

```
_maxWalletSize = (_tTotal * percent) / divisor
swapThreshold = (_tTotal * thresholdPercent) / thresholdDivisor
piSwapPercent = priceImpactSwapPercent
cashierGas = gas
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/2-15-2023_FLOSHIDO_INU.sol#L527
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _basicTransfer(address from, address to, uint256 amount)
internal returns (bool) {
        _tOwned[from] -= amount;
        _tOwned[to] += amount;
        emit Transfer(from, to, amount);
        return true;
    }
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	contracts/2-15-2023_FLOSHIDO_INU.sol#L663,675
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint256 initThreshold
uint256 initSwapAmount
bool check
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/2-15-2023_FLOSHIDO_INU.sol#L592,610
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 toLiquify = ((contractTokenBalance * ratios.liquidity) /
(ratios.total)) / 2
uint256 liquidityBalance = (amtBalance * toLiquify) / swapAmt
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/2-15-2023_FLOSHIDO_INU.sol#L663,674,675
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 initSwapAmount
uint256 initThreshold
bool checked
bool check
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/2-15-2023_FLOSHIDO_INU.sol#L767
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.

```
TOKEN.transfer(_owner, TOKEN.balanceOf(address(this)))
```

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.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IFactoryV2	Interface			
	getPair	External		-
	createPair	External	✓	-
IV2Pair	Interface			
	factory	External		-
	getReserves	External		-
	sync	External	✓	-
IRouter01	Interface			
	factory	External		-
	WETH	External		-

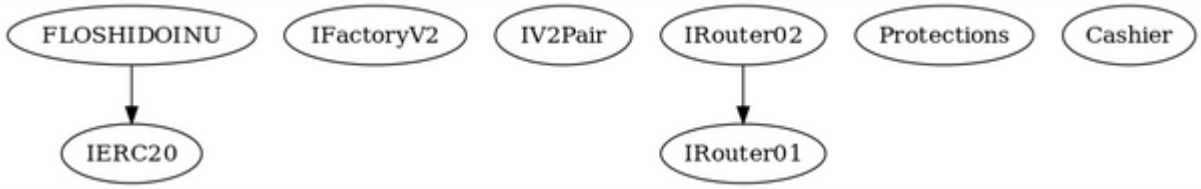
	addLiquidityETH	External	Payable	-
	addLiquidity	External	✓	-
	swapExactETHForTokens	External	Payable	-
	getAmountsOut	External		-
	getAmountsIn	External		-
IRouter02	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
Protections	Interface			
	checkUser	External	✓	-
	setLaunch	External	✓	-
	getInits	External	✓	-
	setLpPair	External	✓	-
	setProtections	External	✓	-
	removeSniper	External	✓	-
Cashier	Interface			
	setRewardsProperties	External	✓	-
	tally	External	✓	-
	load	External	Payable	-
	cashout	External	✓	-
	giveMeWelfarePlease	External	✓	-
	getTotalDistributed	External		-
	getUserInfo	External		-

	getUserRealizedRewards	External		-
	getPendingRewards	External		-
	initialize	External	✓	-
	getCurrentReward	External		-
FLOSHIDOINU	Implementation	IERC20		
		Public	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	Public	✓	onlyOwner
	transfer	External	✓	-
	transferFrom	External	✓	-
	setNewRouter	External	✓	onlyOwner
	setLpPair	External	✓	onlyOwner
	setInitializers	External	✓	onlyOwner
	isExcludedFromFees	External		-
	isExcludedFromDividends	External		-
	isExcludedFromProtection	External		-
	isExcludedFromLimits	External		-

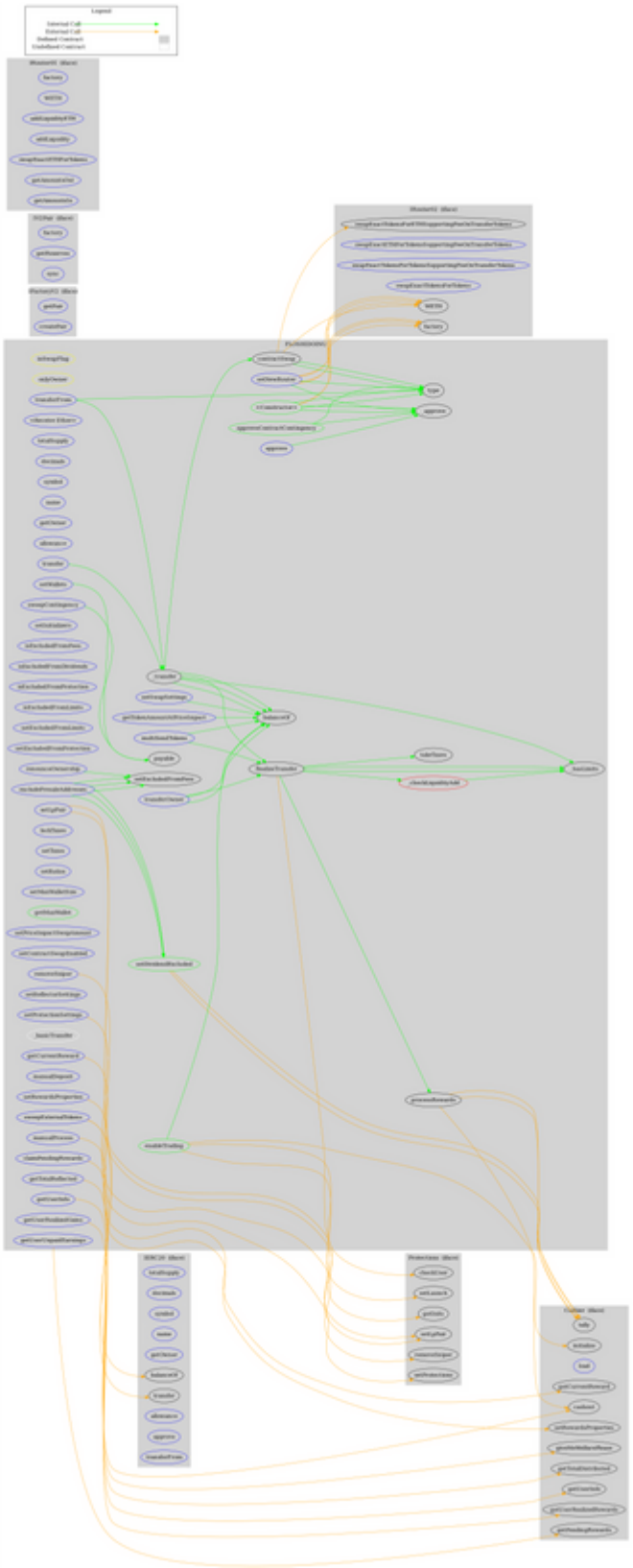
	setExcludedFromLimits	External	✓	onlyOwner
	setDividendExcluded	Public	✓	onlyOwner
	setExcludedFromFees	Public	✓	onlyOwner
	setExcludedFromProtection	External	✓	onlyOwner
	removeSniper	External	✓	onlyOwner
	setProtectionSettings	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	lockTaxes	External	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setRatios	External	✓	onlyOwner
	setMaxWalletSize	External	✓	onlyOwner
	getMaxWallet	Public		-
	getTokenAmountAtPriceImpact	External		-
	setSwapSettings	External	✓	onlyOwner
	setPriceImpactSwapAmount	External	✓	onlyOwner
	setContractSwapEnabled	External	✓	onlyOwner
	setRewardsProperties	External	✓	onlyOwner
	setReflectorSettings	External	✓	onlyOwner
	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_basicTransfer	Internal	✓	
	_transfer	Internal	✓	
	contractSwap	Internal	✓	inSwapFlag
	_checkLiquidityAdd	Private	✓	
	enableTrading	Public	✓	onlyOwner
	finalizeTransfer	Internal	✓	
	processRewards	Internal	✓	
	manualProcess	External	✓	-
	takeTaxes	Internal	✓	

	multiSendTokens	External	✓	onlyOwner
	manualDeposit	External	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	sweepExternalTokens	External	✓	onlyOwner
	claimPendingRewards	External	✓	-
	getTotalReflected	External		-
	getUserInfo	External		-
	getUserRealizedGains	External		-
	getUserUnpaidEarnings	External		-
	getCurrentReward	External		-

Inheritance Graph



Flow Graph



Summary

FLOSHIDO INU 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. There is also a limit of max 20% fees.

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