Cyberscope

Audit Report

BICHON

April 2024

Network BSC

Address 0x9F9A849DC8fbF43b40e7DEb80D125fF0ca00eF4e

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

everity	Code	Description	Status
	IDI	Immutable Declaration Improvement	Unresolved
	MEM	Misleading Error Messages	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	RRS	Redundant Require Statement	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	RZC	Redundant Zero Check	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L16	Validate Variable Setters	Unresolved
	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **3**
Review **4**
Audit Updates 4
Source Files 4
Findings Breakdown **5**
ST - Stops Transactions 6
Description 6
Recommendation 6
IDI - Immutable Declaration Improvement 7
Description 7
Recommendation 7
MEM - Misleading Error Messages 8
Description 8
Recommendation 8
PLPI - Potential Liquidity Provision Inadequacy 9
Description 9
Recommendation 9
PTRP - Potential Transfer Revert Propagation 11
Description 11
Recommendation 11
RRS - Redundant Require Statement 12
Description 12
Recommendation 12
RSML - Redundant SafeMath Library 13
Description 13
Recommendation 13
RSW - Redundant Storage Writes 14
Description 14
Recommendation 14
RZC - Redundant Zero Check 15
Description 15
Recommendation 16
L02 - State Variables could be Declared Constant 17
Description 17
Recommendation 17
L04 - Conformance to Solidity Naming Conventions 18
Description 18

Recommendation 19
L07 - Missing Events Arithmetic 20
Description 20
Recommendation 20
L09 - Dead Code Elimination 21
Description 21
Recommendation 21
L16 - Validate Variable Setters 22
Description 22
Recommendation 22
L19 - Stable Compiler Version 23
Description 23
Recommendation 23
Functions Analysis **24**
Inheritance Graph **27**
Flow Graph **28**
Summary **29**
Disclaimer **30**
About Cyberscope **31**

Review

Contract Name	Bichon
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://bscscan.com/address/0x9f9a849dc8fbf43b40e7deb80d125ff0ca00ef4e
Address	0x9f9a849dc8fbf43b40e7deb80d125ff0ca00ef4e
Network	BSC
Symbol	BCF
Decimals	18
Total Supply	99,999,999,999
Badge Eligibility	Yes

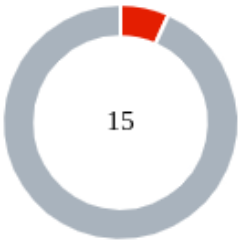
Audit Updates

Initial Audit	01 Apr 2024

Source Files

Filename	SHA256
Bichon.sol	0dc5684ca8ab0c418c6df7517a64985a3a0875c6bb2b5a11b277e35172047f5f

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	14	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Bichon.sol#L300
Status	Unresolved

Description

The contract owner has the authority to stop transactions, as described in detail in section `PTRP` . As a result, the contract might operate as a honeypot.

Recommendation

It is recommended to follow the `PTRP` finding's recommendation to mitigate the finding.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Bichon.sol#L182
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.

```
uniswapV2Pair
```

Recommendation

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

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	Bichon.sol#L308
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_msgSender()==_taxWallet)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Bichon.sol#L286
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = uniswapV2Router.WETH();
_approve(address(this), address(uniswapV2Router), tokenAmount);
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
    tokenAmount,
    0,
    path,
    address(this),
    block.timestamp
);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Bichon.sol#L300
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
(bool callSuccess, ) = payable(_taxWallet).call{value:
amount}("");
require(callSuccess, "Call failed");
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	Bichon.sol#L24
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Bichon.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Bichon.sol#L326,334
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
    function whiteListFromFee(address account) public onlyOwner
{
        _isExcludedFromFee[account] = true;
    }

    function changeTaxWallet(address payable newWallet)
external onlyOwner {
        _taxWallet = newWallet;
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RZC - Redundant Zero Check

Criticality	Minor / Informative
Location	Bichon.sol#L260,285,310
Status	Unresolved

Description

The contract contains the `swapTokensForEth` function that is intended to swap tokens for Ethereum. Within this function, there is an if statement that checks if the `tokenAmount` parameter is equal to zero and returns without executing any further logic if this condition is true. This check is meant to prevent unnecessary execution when there is no token amount to swap. However, the `swapTokensForEth` function is invoked in scenarios where the token amount has already been verified to be greater than zero. Specifically, it is called during transfer transactions after confirming that the `contractTokenBalance` exceeds a certain threshold. Additionally, the `manualSwap` function, which also calls `swapTokensForEth`, includes a preliminary check to ensure that the `tokenBalance` is greater than zero. Given these preceding validations, the zero-amount check within the `swapTokensForEth` function becomes redundant. This redundancy not only adds unnecessary lines of code but could also lead to confusion and inefficiencies in contract execution.

```
function swapTokensForEth(uint256 tokenAmount) private
lockTheSwap {
        if(tokenAmount==0){return;}
...

uint256 contractTokenBalance = balanceOf(address(this));
if (!inSwap && to == uniswapV2Pair && swapEnabled &&
contractTokenBalance>_taxSwapThreshold) {
    swapTokensForEth(contractTokenBalance);

...

function manualSwap() external {
    require(_msgSender()==_taxWallet);
    uint256 tokenBalance=balanceOf(address(this));
    if(tokenBalance>0){
      swapTokensForEth(tokenBalance);
}
```

Recommendation

It is recommended to remove the `if` statement that checks for `tokenAmount == 0` within the `swapTokensForEth` function. Eliminating this redundant check will streamline the function, reducing the contract's complexity and potentially saving gas for transactions that interact with this function. Simplifying the code in this manner also enhances readability and maintainability, making it easier for developers and auditors to understand the contract's logic and intentions.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Bichon.sol#L165,170
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 _taxSwapThreshold= 1000000 * 10**_decimals
bool private swapEnabled = true
```

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	Bichon.sol#L137,153,154,157,158,161,162,163,164,165
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 => bool) public _isExcludedFromFee
address payable public _taxWallet =
payable(0xf132E28953eD3b098dB377131D8623701f9bF8D4)
uint256 public _buyTax = 5
uint256 public _sellTax = 5
uint8 private constant _decimals = 18
uint256 private constant _tTotal = 99999999999 * 10**_decimals
string private constant _name = unicode"BICHON"
string private constant _symbol = unicode"BCF"
uint256 public _taxSwapThreshold= 1000000 * 10**_decimals
```

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	Bichon.sol#L322,323,327,335
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.

```
_buyTax = buyFee
_sellTax = sellFee;
_isExcludedFromFee[account] = true;
_taxWallet = newWallet;
```

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	Bichon.sol#L280
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 min(uint256 a, uint256 b) private pure returns
(uint256){
      return (a>b)?b:a;
    }
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Bichon.sol#L335
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.

```
_taxWallet = newWallet
```

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	Bichon.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 ^0.8.9;
```

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.

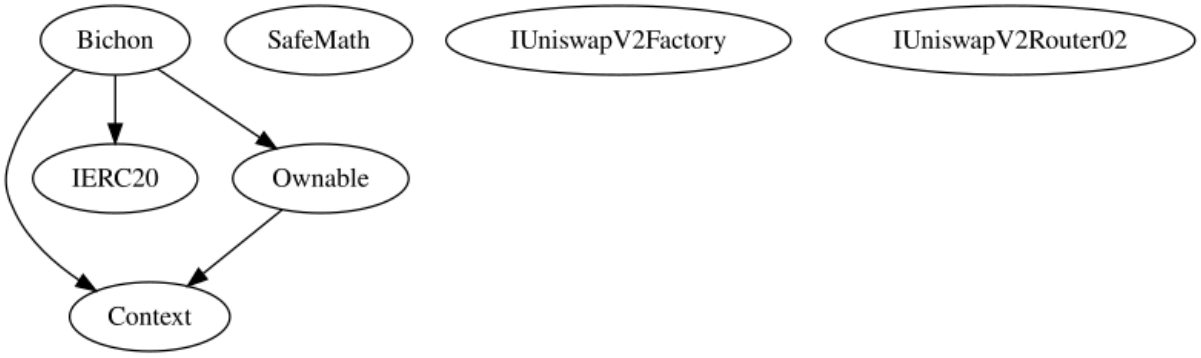
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		

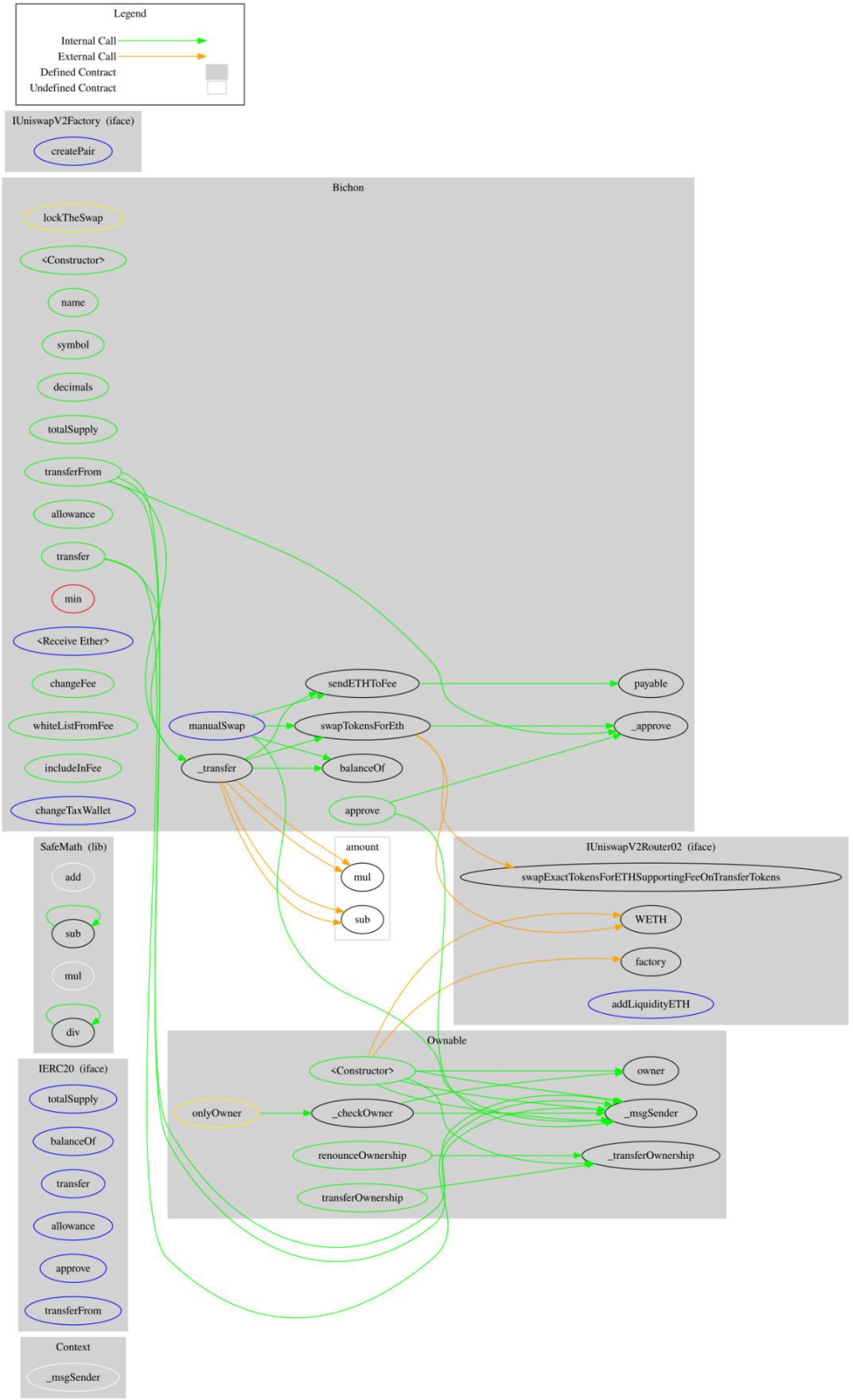
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
Bichon	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-

	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	min	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	sendETHToFee	Private	✓	
		External	Payable	-
	manualSwap	External	✓	-
	changeFee	Public	✓	onlyOwner
	whiteListFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	changeTaxWallet	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
createPair
Bichon
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
transferFrom
allowance
transfer
min
<Receive Ether>
changeFee
whiteListFromFee
includeInFee
changeTaxWallet
sendETHToFee
payable
manualSwap
swapTokensForEth
_approve
_transfer
balanceOf
approve
amount
mul
sub
SafeMath (lib)
add
sub
mul
div
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH
Ownable
<Constructor>
owner
onlyOwner
_checkOwner
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

BICHON contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 25% 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