Cyberscope

Audit Report

BITCHIP

January 2024

Network	BSC
Address	0x11b204caa3c0ce4c7f99f5beafe73e3c2560c9b6
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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	FRV	Fee Restoration Vulnerability	Unresolved
	FSA	Fixed Swap Address	Unresolved
	MEM	Misleading Error Message	Unresolved
	RES	Redundant Event Statement	Unresolved
	RRS	Redundant Require Statement	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	OCTD	Transfers Contract's Tokens	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
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	BITCHIP
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0x11b204caa3c0ce4c7f99f5beafe73e3c2560c9b6
Address	0x11b204caa3c0ce4c7f99f5beafe73e3c2560c9b6
Network	BSC
Symbol	BTP
Decimals	18
Total Supply	10,000,000,000

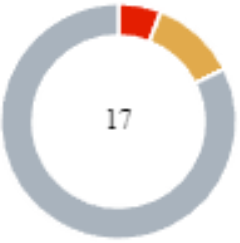
Audit Updates

Initial Audit	14 Jan 2024

Source Files

Filename	SHA256
BITCHIP.sol	677effcb1f619462ec929cf9069e5c269f928f670fc6a40ea905c300add2506c

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	14	0	0	0

ST - Stops Transactions

Criticality	Medium
Location	BITCHIP.sol#L627
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` to zero. As a result, the contract may operate as a honeypot.

```
function setMaxTxPercent(uint256 maxTxPercent) external
onlyOwner() {
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**2
    );
}
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	BITCHIP.sol#L619,623,657,662,667,673,679
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setTaxFeePercent` and `setLiquidityFeePercent` functions with a high percentage value.

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner()
{
    _taxFee = taxFee;
}

function setLiquidityFeePercent(uint256 liquidityFee) external
onlyOwner() {
    _liquidityFee = liquidityFee;
}

...
```

Recommendation

The contract could embody a check for the maximum acceptable value. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Medium
Location	BITCHIP.sol#L633
Status	Unresolved

Description

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

```
function addBotToBlackList(address account) external
onlyOwner() {
    require(account !=
0x3Cd700decF29ce7c0aCbC895A5352Ae44602bD49, 'We can not
blacklist Uniswap router.');
    require(!_isBlackListedBot[account], "Account is already
blacklisted");
    _isBlackListedBot[account] = true;
    _blackListedBots.push(account);
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	BITCHIP.sol#L931
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
function removeAllFee() private {
    if(_taxFee == 0 && _liquidityFee == 0) return;

    _previousTaxFee = _taxFee;
    _previousLiquidityFee = _liquidityFee;

    _taxFee = 0;
    _liquidityFee = 0;
}

function restoreAllFee() private {
    _taxFee = _previousTaxFee;
    _liquidityFee = _previousLiquidityFee;
}

function _tokenTransfer(address sender, address recipient,
uint256 amount,bool takeFee) private {
    if(!takeFee)
        removeAllFee();
    ...
    if(!takeFee)
        restoreAllFee();
}
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	BITCHIP.sol#L452
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
// Create a uniswap pair for this new token
uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
    .createPair(address(this), _uniswapV2Router.WETH());
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

MEM - Misleading Error Message

Criticality	Minor / Informative
Location	BITCHIP.sol#L633
Status	Unresolved

Description

The `addBotToBlackList` function is designed to prevent certain addresses from interacting with the contract's functionalities by blacklisting them. The function contains a `require` statement with the intention to safeguard the Uniswap Router from being blacklisted. However, the address provided in this `require` statement is actually assigned as the `ownerWallet` within the contract, not the Uniswap Router. The actual Uniswap Router is instantiated at a different address. This discrepancy creates a misleading situation, where the error message suggests protection for the Uniswap Router, but the address being protected is, in fact, the owner's wallet. Such misleading information can cause confusion and potentially lead to incorrect assumptions about the contract's security and functionality.

```
address public ownerWallet =
0x3Cd700decF29ce7c0aCbC895A5352Ae44602bD49;

function addBotToBlackList(address account) external
onlyOwner() {
    require(account !=
0x3Cd700decF29ce7c0aCbC895A5352Ae44602bD49, 'We can not
blacklist Uniswap router.');
    require(!_isBlackListedBot[account], "Account is already
blacklisted");
    _isBlackListedBot[account] = true;
    _blackListedBots.push(account);
}
```

Recommendation

it is recommended to revise the `require` statement in the addBotToBlackList function to accurately reflect the address of the Uniswap Router. The correct address of the Uniswap Router should be used in the `require` check to ensure that the contract's intention of

protecting the router from blacklisting is effectively implemented. Furthermore, reviewing and updating the error messages to accurately describe the logic and purpose of the `require` statements can enhance the clarity and reliability of the contract. Such changes will not only correct the current inconsistency but also aid in maintaining the transparency and integrity of the contract, which is crucial for user trust and contract auditability.

RES - Redundant Event Statement

Criticality	Minor / Informative
Location	BITCHIP.sol#L435
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. Specifically, the event `MinTokensBeforeSwapUpdated` is not emitted in the contract's implementation. Hence, it is redundant.

```
event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
```

Recommendation

The team is advised to take this segment into consideration and rewrite them so that 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. It is recommended to remove redundant events.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	BITCHIP.sol#L20
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	BITCHIP.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.

```
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.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	BITCHIP.sol#L691
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueBEP20` function.

```
function rescueBEP20(address tokenAdd, uint256 amount) external
onlyOwner{
    IERC20(tokenAdd).transfer(ownerWallet, 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BITCHIP.sol#L386,389,393,394,395,429
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.

```
address public ownerWallet =
0x3Cd700decF29ce7c0aCbC895A5352Ae44602bD49
uint256 private _tTotal = 10000 * 10**6 * 10**18
string private _name = "BITCHIP"
string private _symbol = "BTP"
uint8 private _decimals = 18
uint256 private numTokensSellToAddToLiquidity = 5000 * 10**6 *
10**18
```

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	BITCHIP.sol#L198,199,216,236,407,410,413,414,416,417,426,428,648,663,669,675,745,751
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint256 public _taxFee = 1
uint256 public _liquidityFee = 8
uint256 public _buyTaxFee = 2
uint256 public _buyLiquidityFee = 8
uint256 public _sellTaxFee = 2
uint256 public _sellLiquidityFee = 8
mapping (address => AddressFee) public _addressFees
uint256 public _maxTxAmount = 10000 * 10**6 * 10**18
bool _enabled
uint256 _addressLiquidityFee

...
```

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	BITCHIP.sol#L616,620,624,627,659,741
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.

```
function includeInFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = false;
}
function setMaxTxPercent(uint256 maxTxPercent) external
onlyOwner() {
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**2
    );
}
function setBuyFee(uint256 buyTaxFee, uint256 buyLiquidityFee)
external onlyOwner {
    _buyTaxFee = buyTaxFee;
    _buyLiquidityFee = buyLiquidityFee;
}

function _takeLiquidity(uint256 tLiquidity) private {
    ...
    }
```

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	BITCHIP.sol#L75,85,93,97,101,105,110
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 isContract(address account) internal view returns
(bool) {
        // According to EIP-1052, 0x0 is the value returned for
not-yet created accounts
        // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

...
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BITCHIP.sol#L82,119
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 { codehash := extcodehash(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

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	BITCHIP.sol#L1
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 ^0.8.7;
```

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	BITCHIP.sol#L688
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(tokenAdd).transfer(ownerWallet, 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**
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		
	mod	Internal		
	mod	Internal		
Context	Implementation			

	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
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		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
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	✓	-
BITCHIP	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-

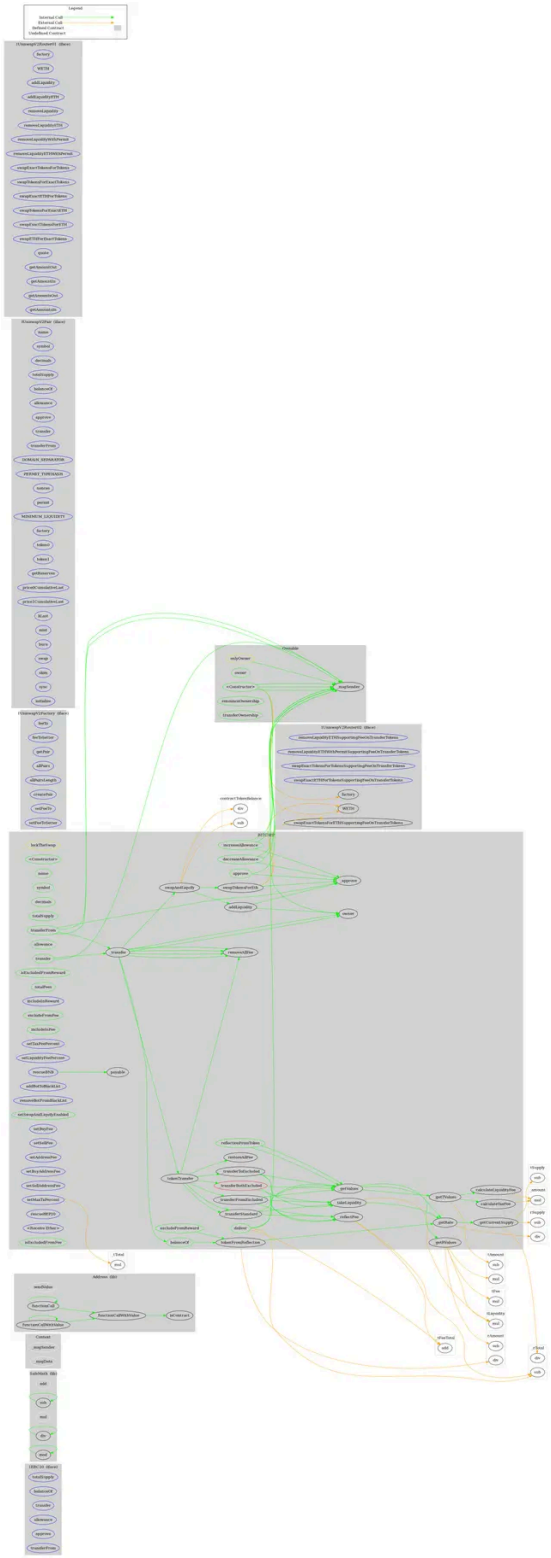
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setTaxFeePercent	External	✓	onlyOwner
	setLiquidityFeePercent	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	addBotToBlackList	External	✓	onlyOwner
	removeBotFromBlackList	External	✓	onlyOwner

	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setBuyFee	External	✓	onlyOwner
	setSellFee	External	✓	onlyOwner
	setAddressFee	External	✓	onlyOwner
	setBuyAddressFee	External	✓	onlyOwner
	setSellAddressFee	External	✓	onlyOwner
	rescueBNB	External	✓	onlyOwner
	rescueBEP20	External	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	calculateTaxFee	Private		
	calculateLiquidityFee	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	

	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	

Inheritance Graph

Flow Graph



Summary

BITCHIP 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, manipulate the fees and blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io