Cyberscope

Audit Report

ChipToken

June 2023

Network	BSC Testnet
Address	0x46B83e8B29c3b0382128b160AC47F34d3A4A6b80
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

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	MLF	Misleading Liquidity Feature	Unresolved
●	CSO	Conditional Statement Optimization	Unresolved
●	RC	Redundant Calculations	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	FSA	Fixed Swap Address	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved

●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	ChipToken
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0x46b83e8b29c3b0382128b160ac47f34d3a4a6b80
Address	0x46b83e8b29c3b0382128b160ac47f34d3a4a6b80
Network	BSC_TESTNET
Symbol	CHIPT
Decimals	18
Total Supply	1,000,000

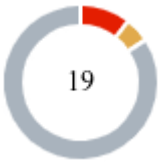
Audit Updates

Initial Audit	01 Jun 2023

Source Files

Filename	SHA256
ChipToken.sol	13a930cec41b4896868e6fd4d8121290c653c94acec6bb5048331905b8fa7407

Findings Breakdown



Critical	2
Medium	1
Minor / Informative	16

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	1	0	0	0
Minor / Informative	16	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	ChipToken.sol#L456
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting either of the following wallets to the zero address:

- `teamWallet` .
- `marketingWallet` .
- `treasuryWallet` .

It should be noted that this is possible only when the fee amount is greater than zero.

```
if (fee > 0) {
    super._transfer(sender, teamWallet,
fee.mul(sellFeeTeam).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).add(
sellFeeTreasury)));
    super._transfer(sender, marketingWallet,
fee.mul(sellFeeMarketing).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity)
.add(sellFeeTreasury)));
    super._transfer(sender, treasuryWallet,
fee.mul(sellFeeTreasury).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).
add(sellFeeTreasury)));
}
```

Recommendation

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

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

TSD - Total Supply Diversion

Criticality	Critical
Location	ChipToken.sol#L456
Status	Unresolved

Description

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

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

The contract splits the fee amount into 3 portions and transfers each portion to its corresponding wallet. However, since the divisor is composed of 4 fee percentages, a piece of the fee amount will be left out. In other terms, the `fee.mul(sellFeeLiquidity).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).add(sellFeeTreasury))` amount is not calculated.

```
if (fee > 0) {
    super._transfer(sender, teamWallet,
fee.mul(sellFeeTeam).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).add(
sellFeeTreasury)));
    super._transfer(sender, marketingWallet,
fee.mul(sellFeeMarketing).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity)
.add(sellFeeTreasury)));
    super._transfer(sender, treasuryWallet,
fee.mul(sellFeeTreasury).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).
add(sellFeeTreasury)));
}
```

Recommendation

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

MLF - Misleading Liquidity Feature

Criticality	Medium
Location	ChipToken.sol#L475,488
Status	Unresolved

Description

The contract lacks the capability to generate ETH and none of the fees are directed towards the contract's address, rendering the execution of the `_addLiquidity` function impossible. Consequently, the contract repetitively checks the condition `tokenAmount > 0 && ethAmount > 0` in every transaction, even though it will always be false. If a user sends tokens and ETH to the contract, it will activate the liquidity mechanism. However, it would be more appropriate for this feature to be available as an external function. As a result, the liquidity mechanism is misleading.

```
function _afterTokenTransfer(
    address from,
    address to,
    uint256 amount
) internal virtual override {
    super._afterTokenTransfer(from, to, amount);

    // Auto-add liquidity on transfers to liquidity pool
    if (from != liquidityPool && to == liquidityPool) {
        _addLiquidity();
    }
}

function _addLiquidity() private {
    uint256 tokenAmount = balanceOf(address(this));
    uint256 ethAmount = address(this).balance;

    if (tokenAmount > 0 && ethAmount > 0) {
        IPancakeRouter pancakeRouter =
IPancakeRouter(0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3);
        address pancakePair =
IPancakeFactory(pancakeRouter.factory()).getPair(address(this),
pancakeRouter.WETH());
        if (pancakePair == address(0)) {
            pancakePair =
IPancakeFactory(pancakeRouter.factory()).createPair(address(this),
pancakeRouter.WETH());
        }

        pancakeRouter.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0,
            0,
            address(this),
            block.timestamp
        );
    }
}
```

Recommendation

The team is advised to either remove the misleading liquidity mechanism or to use it as an external function.

CSO - Conditional Statement Optimization

Criticality	Minor / Informative
Location	ChipToken.sol#L431
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract checks certain conditions to calculate the fee amount that will be subtracted from the initial amount. Some of these conditions set the `fee` to the same value. Hence, these conditions could be merged into one.

```
if (sender == liquidityPool) {
    // Transfers from liquidity pool are not subject to fees
    fee = 0;
} else if (recipient == liquidityPool) {
    // Transfers to liquidity pool are subject to liquidity fee
    fee = amount.mul(sellFeeLiquidity).div(10000);
} else if (sender == burnAddress) {
    // Transfers from burn address are not subject to fees
    fee = 0;
} else if (sender == teamWallet) {
    // Transfers from team wallet are subject to team fee
    fee = amount.mul(sellFeeTeam).div(10000);
} else if (sender == marketingWallet) {
    // Transfers from marketing wallet are subject to marketing fee
    fee = amount.mul(sellFeeMarketing).div(10000);
} else {
    // Regular transfers are subject to treasury fee
    fee = amount.mul(sellFeeTreasury).div(10000);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

A suggested implementation would be the following:

```
if (sender == liquidityPool || sender == burnAddress) {
    fee = 0;
}
```

RC - Redundant Calculations

Criticality	Minor / Informative
Location	ChipToken.sol#L457,458,459
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract calculates the same value more than once. The arithmetic operation `sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).add(sellFeeTreasury)` is the accumulation of the contract's fee percentages and is used as the denominator for each transfer. As a result, the contract performs redundant calculations.

```
super._transfer(sender, teamWallet,
fee.mul(sellFeeTeam).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).add(
sellFeeTreasury)));
super._transfer(sender, marketingWallet,
fee.mul(sellFeeMarketing).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity)
.add(sellFeeTreasury)));
super._transfer(sender, treasuryWallet,
fee.mul(sellFeeTreasury).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).
add(sellFeeTreasury)));
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	ChipToken.sol#L457,458,459
Status	Unresolved

Description

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

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

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

```
super._transfer(sender, teamWallet,
fee.mul(sellFeeTeam).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).add(
sellFeeTreasury)));
super._transfer(sender, marketingWallet,
fee.mul(sellFeeMarketing).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity)
.add(sellFeeTreasury)));
super._transfer(sender, treasuryWallet,
fee.mul(sellFeeTreasury).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).
add(sellFeeTreasury)));
```

Recommendation

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

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	ChipToken.sol#L333,334,335,336,337.339,340,341,342,343
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
uint256 public buyFeeTeam = 25; // 0.25%
uint256 public buyFeeMarketing = 100; // 1%
uint256 public buyFeeLiquidity = 125; // 1.25%
uint256 public buyFeeBurn = 250; // 2.5%
uint256 public buyFeeTreasury = 300; // 3%
...
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	ChipToken.sol#L345,346,347,348,349,351,352,353,354,356
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The contract implements a fee mechanism. Additionally, the fee variables are declared with the `sell` prefix, indicating that they are only applied on sales. However, these fees are applied on buys and simple transfers as well. As a result, these variable names are misleading.

```
uint256 public sellFeeTeam = 50; // 0.5%
uint256 public sellFeeMarketing = 100; // 1%
uint256 public sellFeeLiquidity = 200; // 2%
uint256 public sellFeeBurn = 250; // 2.5%
uint256 public sellFeeTreasury = 400; // 4%
...
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	ChipToken.sol#L361,365,369,373,377,,397
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function setTeamWallet(address wallet) external onlyOwner {
    teamWallet = wallet;
}
function setMarketingWallet(address wallet) external onlyOwner {
    marketingWallet = wallet;
}
function setTreasuryWallet(address wallet) external onlyOwner {
    treasuryWallet = wallet;
}
function setLiquidityPool(address pool) external onlyOwner {
    liquidityPool = pool;
}
...
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	ChipToken.sol#L493
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.

```
IPancakeRouter pancakeRouter =
IPancakeRouter(0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3);
address pancakePair =
IPancakeFactory(pancakeRouter.factory()).getPair(address(this),
pancakeRouter.WETH());
if (pancakePair == address(0)) {
    pancakePair = IPancakeFactory(pancakeRouter.factory()).createPair(address(this),
pancakeRouter.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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	ChipToken.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, and 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 at https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ChipToken.sol#L331,339,340,341,342,343,351,352,353,354,355
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 burnAddress = 0x000000000000000000000000000000000000dEaD
uint256 public MaxbuyFeeTeam = 100
uint256 public MaxbuyFeeMarketing = 100
uint256 public MaxbuyFeeLiquidity = 500
uint256 public MaxbuyFeeBurn = 500
uint256 public MaxbuyFeeTreasury = 500
uint256 public MaxsellFeeTeam = 100
uint256 public MaxsellFeeMarketing = 100
uint256 public MaxsellFeeLiquidity = 500
uint256 public MaxsellFeeBurn = 500
uint256 public MaxsellFeeTreasury = 500
```

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	ChipToken.sol#L127,339,340,341,342,343,351,352,353,354,355
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);
uint256 public MaxbuyFeeTeam = 100
uint256 public MaxbuyFeeMarketing = 100
uint256 public MaxbuyFeeLiquidity = 500
uint256 public MaxbuyFeeBurn = 500
uint256 public MaxbuyFeeTreasury = 500
uint256 public MaxsellFeeTeam = 100
uint256 public MaxsellFeeMarketing = 100
uint256 public MaxsellFeeLiquidity = 500
uint256 public MaxsellFeeBurn = 500
uint256 public MaxsellFeeTreasury = 500
```

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	ChipToken.sol#L6
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	ChipToken.sol#L410
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.

```
sellFeeTeam = teamFee
```

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	ChipToken.sol#L34,263
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;
    }

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	ChipToken.sol#L445,457,458,459
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.

```
fee = amount.mul(sellFeeMarketing).div(10000)
super._transfer(sender, teamWallet,
fee.mul(sellFeeTeam).div(sellFeeTeam.add(sellFeeMarketing).add(sellFeeLiquidity).add(
sellFeeTreasury)))
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ChipToken.sol#L362,366,370,374
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.

```
teamWallet = wallet
marketingWallet = wallet
treasuryWallet = wallet
liquidityPool = pool
```

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	ChipToken.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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		

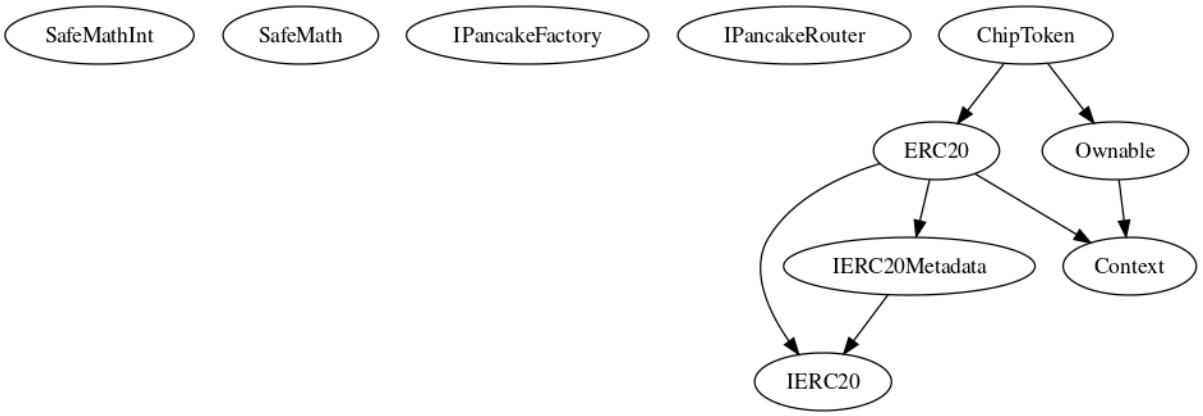
	sub	Internal		
	div	Internal		
	mod	Internal		
IPancakeFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IPancakeRouter	Interface			
	WETH	External		-
	factory	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	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	✓	-

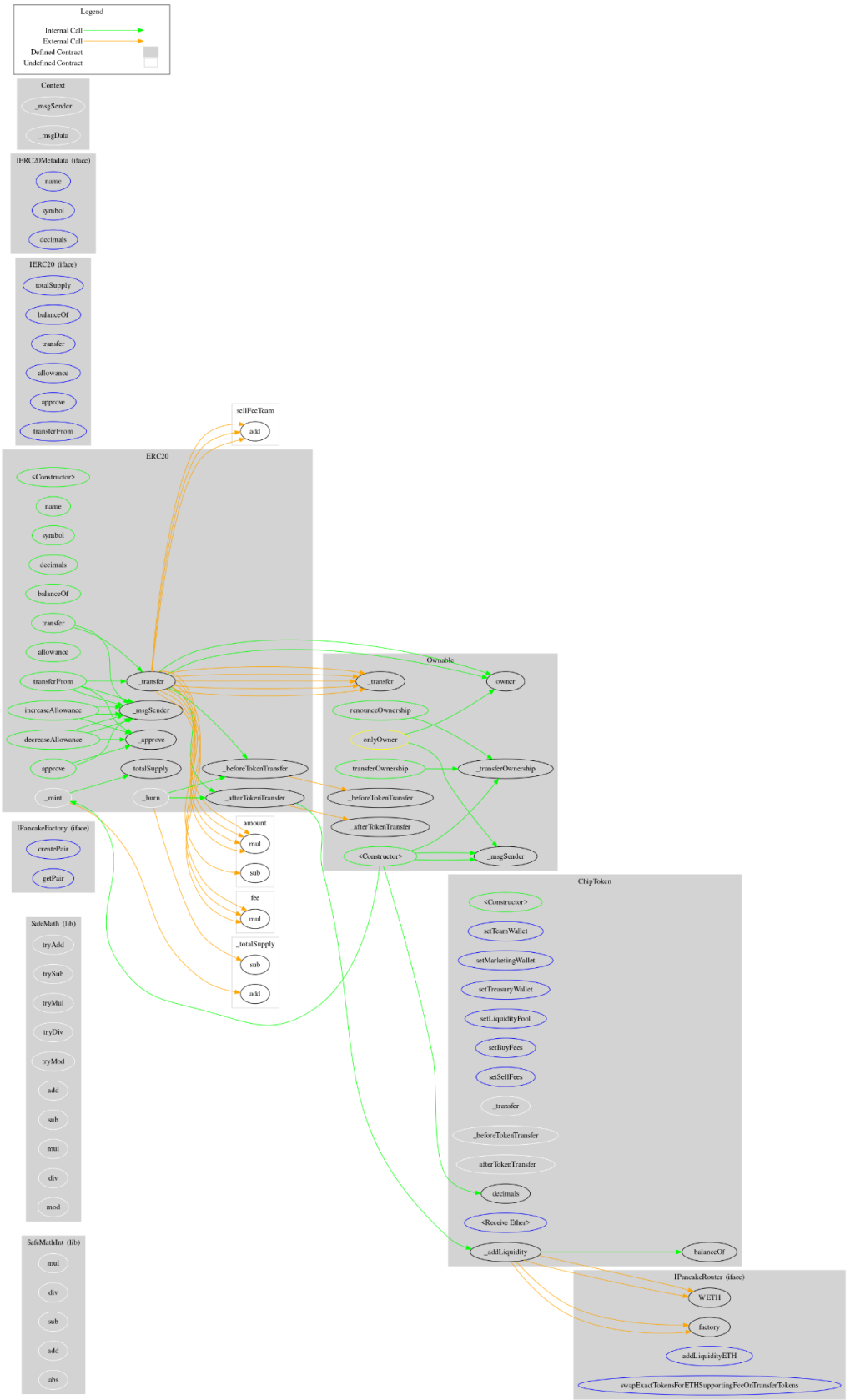
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ChipToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	setTeamWallet	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setTreasuryWallet	External	✓	onlyOwner
	setLiquidityPool	External	✓	onlyOwner
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	_transfer	Internal	✓	

	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	_addLiquidity	Private	✓	
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
sellFeeTeam
add
ERC20
<Constructor>
name
symbol
decimals
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_mint
_transfer
_msgSender
_approve
totalSupply
_burn
_beforeTokenTransfer
_afterTokenTransfer
Ownable
_transfer
owner
renounceOwnership
onlyOwner
transferOwnership
_transferOwnership
_beforeTokenTransfer
_afterTokenTransfer
<Constructor>
_msgSender
amount
mul
sub
fee
mul
_totalSupply
sub
add
IPancakeFactory (iface)
createPair
getPair
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
SafeMathInt (lib)
mul
div
sub
add
abs
ChipToken
<Constructor>
setTeamWallet
setMarketingWallet
setTreasuryWallet
setLiquidityPool
setBuyFees
setSellFees
_transfer
_beforeTokenTransfer
_afterTokenTransfer
decimals
<Receive Ether>
_addLiquidity
balanceOf
IPancakeRouter (iface)
WETH
factory
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens

Summary

ChipToken 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 stopping 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 5% 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