Cyberscope

Audit Report

HAPPYFARM

July 2024

Network BSC

Address 0x0862067fb63ef96c5fa7b4a53146b90156ad8888

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
	AVU	Ambiguous Variable Usage	Unresolved
	IIFI	Ineffective Inviter Fee Implementation	Unresolved
	IFIU	Internal Flag Inconsistent Usage	Unresolved
	MEM	Misleading Error Messages	Unresolved
	MEE	Missing Events Emission	Unresolved
	PAV	Pair Address Validation	Renounced
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PVC	Price Volatility Concern	Renounced
	RSML	Redundant SafeMath Library	Unresolved
	RC	Repetitive Calculations	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
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved
●	L22	Potential Locked Ether	Unresolved

Table of Contents

Review

Contract Name	GGGTOKEN
Compiler Version	v0.8.6+commit.11564f7e
Optimization	200 runs
Explorer	https://bscscan.com/address/0x0862067fb63ef96c5fa7b4a53146b90156ad8888
Address	0x0862067fb63ef96c5fa7b4a53146b90156ad8888
Network	BSC
Symbol	HPA
Decimals	18
Total Supply	1,000,000
Badge Eligibility	Yes

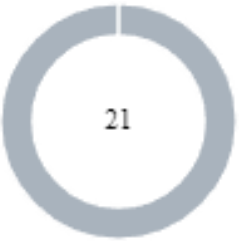
Audit Updates

Initial Audit	18 Jul 2024

Source Files

Filename	SHA256
GGGTOKEN.sol	d0697e51b50851de08a8595ebb133a98026c9b1b7c2473ad12f94c280150a249

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	21

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	19	0	2	0

AVU - Ambiguous Variable Usage

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L3012,3258
Status	Unresolved

Description

The `_tranFee` variable in the `_transfer` function is used in a manner that suggests it functions as an indicator with multiple states, but it is implemented as a uint. The setTranFee function allows the owner to set the value of `_tranFee` to any unsigned integer value. In the `_transfer` function, the logic checks if `_tranFee` is non-zero and then further checks if `_tranFee` is equal to 1. If `_tranFee` is neither 0 nor 1, the transfer amount is adjusted using the `takeSell` function by default.

```
}else if(_tranFee!=0) { //transfer
    if(_tranFee==1)
        amount =    takeBuy(from,amount);
    else
        amount = takeSell(from,amount);
}

function setTranFee(uint value) external onlyOwner {
        _tranFee = value;
    }
```

Recommendation

To ensure the correct and safe usage of `_tranFee`, it is recommended to refactor the implementation to clearly define and handle the different states represented by `_tranFee`. Instead of using a uint, which can have any value, a more robust approach should be adopted that limits `_tranFee` to the intended states. This approach will make the code more readable and maintainable, while also preventing invalid values from being assigned to `_tranFee`.

IIFI - Ineffective Inviter Fee Implementation

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L3266
Status	Unresolved

Description

The takeInviterFee function is designed to distribute a portion of the transaction amount as inviter fees. This function determines whether the transaction involves a buy or sell on Uniswap, and then iterates through the _inviters array to calculate and distribute the fees to inviter addresses. However, the _inviters array, which is supposed to hold the fee distribution percentages for each inviter, is always empty. Consequently, the loop for (uint256 i = 0; i < _inviters.length; i++) will never execute, making the inviter fee logic ineffective. This results in the inviter fee calculation and distribution logic is rendered useless since the array meant to hold inviter percentages is never populated. Furthermore, the inviter fee is intended to be divided among the inviters, but since the loop doesn't run, the inviter fee is instead transferred in its entirety to the market address _marketAddr. Lastly, the current implementation fails to achieve the intended incentivization of inviters, undermining any marketing or community growth strategy relying on inviter rewards.

```
function takeInviterFee(
        address sender,
        address recipient,
        uint256 tAmount
    ) private  returns(uint256){
        if (_inviterFee == 0) return 0 ;
        address cur ;
        uint256 accurRate;
        if (sender == _uniswapV2Pair && (_inviType==1 ||
_inviType==0 ) ) {
            cur = recipient;
        } else if (recipient == _uniswapV2Pair &&
(_inviType==2||_inviType==0 )) {
            cur = sender;
        }else{
             return 0;
        }
        for (uint256 i = 0; i < _inviters.length; i++) {
            cur = getPar(cur);
            if (cur == address(0)) {
                break;
            }
            accurRate = accurRate.add(_inviters[i]);
            uint256 curTAmount =
tAmount.mul(_inviters[i]).div(10000);
            super._transfer(sender, cur, curTAmount);
        }
        if(_inviterFee.sub(accurRate)!=0){
            super._transfer(sender, _marketAddr,
tAmount.mul(_inviterFee.sub(accurRate)).div(10000) ) ;
        }
        return tAmount.mul(_inviterFee).div(10000);
    }
```

Recommendation

To resolve this issue, the _inviters array must be properly initialized and managed. Ensure that the _inviters array is populated with valid inviter fee percentages before any transactions occur. Additionally, thoroughly review the inviter fee logic to ensure that it correctly calculates and distributes fees as intended. Comprehensive documentation and transparency about how inviters are managed and rewarded will also enhance trust and engagement within the community. Implementing these changes will make the inviter fee mechanism functional and effective, supporting the token's growth and incentivization strategy.

IFIU - Internal Flag Inconsistent Usage

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L2974
Status	Unresolved

Description

The `_transfer` function in the smart contract under audit deviates from the more commonly implemented version by using `from != address(this)` as a condition instead of using an internal swapping flag. This difference can lead to potential issues such as reentrancy risks and logic flaws. The typical implementation uses an internal flag, such as `swapping`, to prevent reentrancy and ensure that the contract's state is appropriately managed during token swaps and transfers.

```
if (
            canSwap &&
            from != address(this) &&
            !automatedMarketMakerPairs[from] &&
            from != owner() &&
            to != owner()&&
            getFeeDeno()>0&&
            _startTimeForSwap>0
        ) {
            if (getMarketFee() > 0) {
            ...
}
```

Recommendation

It is recommended to implement an internal swapping flag to prevent reentrancy and ensure the contract's state integrity during token swaps and transfers. This flag should be set to true at the beginning of the critical operations and reset to false at the end. This approach will help mitigate the risks of reentrancy attacks and unintended contract behavior.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L2181,2289,2391,2724
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(totalSupply() > 0)
require(false)
require(!excludedFromDividends[account])
require(!_init)
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L2793,2803,3222,3258,3342,3346
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 setSwapTokensAtAmount(uint256 amount) external onlyOwner {
        _swapTokensAtAmount = amount;
    }

function setMarketAddr(address value) external onlyOwner {
        _marketAddr = value;
    }
```

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.

PAV - Pair Address Validation

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L3346
Status	Renounced

Description

The contract is missing address validation in the pair address argument. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair address provided as an argument. The pair address is a parameter used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in the pair address argument can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

```
function setUniswapV2Pair(address value) external onlyOwner {
    _uniswapV2Pair = value;
    _setAutomatedMarketMakerPair(_uniswapV2Pair,true);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair address argument, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair existence in the decentralized application. Prior to interacting with the pair address contract, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L3101
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.

```
function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = _token;
        // make the swap

_uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            _router,
            block.timestamp
        );
        try IERC20(_token).transferFrom(_router,address(this)
,IERC20(_token).balanceOf(_router)) {} catch {}
    }
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	GGGTOKEN.sol#2793,2970
Status	Renounced

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `_swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function setSwapTokensAtAmount(uint256 amount) external onlyOwner {
    _swapTokensAtAmount = amount;
}

bool canSwap = contractTokenBalance >= _swapTokensAtAmount;
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	GGGTOKEN.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.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L2972
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive. For instance in the follwoing example there are multiple sections that are recalculated:

- `getFeeDeno()`
- `getMarketFee()`
- `getLiqFee()`

```
if (
    canSwap &&
    from != address(this) &&
    !automatedMarketMakerPairs[from] &&
    from != owner() &&
    to != owner()&&
    getFeeDeno()>0&&
    _startTimeForSwap>0
) {
    if (getMarketFee() > 0) {
        uint256 marketingTokens = contractTokenBalance
            .mul(getMarketFee())
            .div(getFeeDeno());
        swapAndSendToFee(marketingTokens);
    }

    if (getLiqFee() > 0) {
        uint256 swapTokens =
contractTokenBalance.mul(getLiqFee()).div(
            getFeeDeno()
        );
        swapAndLiquify(swapTokens);
    }

    uint256 sellTokens = balanceOf(address(this));
    if (sellTokens > 0 && dividendTracker.totalSupply()>0 ) {
        swapAndSendDividends(sellTokens);
    }
}
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and

gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L198,2652,2653,2654,2718,2719,2720,2721,2722
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.

```
uint8 public _inviType;
uint256 public _maxHave;
uint256 public _maxBuyTax;
uint256 public _maxSellTax;

...
```

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	GGGTOKEN.sol#L140,194,195,196,198,1170,1383,1387,1396,1454,1459,1761,1793,1798,1842,1865,1866,1883,2154,2174,2175,2176,2177,2236,2243,2255,2269,2440,2653,2654,2658,2665,2666,2667,2668,2669,2670,2671,2672,2674,2675,2710,2711,2712,2714,2715,2716,2717,2718,2719,2720,2721,2722,2727,3335,3336,3337
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.

```
address public _owner
uint256 public _startTimeForSwap
uint256 public _intervalSecondsForSwap
address public _uniswapV2Pair
uint8 public _enabOwnerAddLiq

...
```

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	GGGTOKEN.sol#L191
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.

```
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	GGGTOKEN.sol#L2794,3259,3263,3343
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;

_tranFee = value;

...
```

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	GGGTOKEN.sol#L470,868,882,903,938,966,991,1001,1020,1034,1053,1063,1080,1090,1107,1383,1626,1957,2288
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 _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
        }
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L3002
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.

```
inFee;
```

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	GGGTOKEN.sol#L2176,2177,2236,2243,2255,2269
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

...
```

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	GGGTOKEN.sol#L316,2804,3347,3364
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.

```
_uniswapV2Pair = value;

...
```

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	GGGTOKEN.sol#L851,869,887,944,1119,3318
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 {
            size := extcodesize(account)
        }

...
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L3114
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.

```
try IERC20(_token).transferFrom(_router,address(this)
,IERC20(_token).balanceOf(_router)) {} catch {}
```

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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	GGGTOKEN.sol#L2791
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {}
```

Recommendation

The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
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		

	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

	_setOwner	Private	✓	
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		
Clones	Library			
	clone	Internal	✓	
	cloneDeterministic	Internal	✓	
	predictDeterministicAddress	Internal		
	predictDeterministicAddress	Internal		

Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
IERC20Upgradeable	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20MetadataUpgradeable	Interface	IERC20Upgradeable		
	name	External		-
	symbol	External		-
	decimals	External		-
Initializable	Implementation			

ContextUpgradeable	Implementation	Initializable		
	__Context_init	Internal	✓	initializer
	__Context_init_unchained	Internal	✓	initializer
	_msgSender	Internal		
	_msgData	Internal		
ERC20Upgradeable	Implementation	Initializable, ContextUpgradeable, IERC20Upgradeable, IERC20MetadataUpgradeable		
	__ERC20_init	Internal	✓	initializer
	__ERC20_init_unchained	Internal	✓	initializer
	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	✓	
OwnableUpgradeable	Implementation	Initializable, ContextUpgradeable		
	__Ownable_init	Internal	✓	initializer
	__Ownable_init_unchained	Internal	✓	initializer
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
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	✓	-
SafeMathInt	Library			
	mul	Internal		

	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-
	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-
DividendPayingTokenOptionalInterface	Interface			

	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingToken	Implementation	ERC20Upgradeable, OwnableUpgradeable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
	__DividendPayingToken_init	Internal	✓	initializer
	distributeCAKEDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
BABYTOKENDividendTracker	Implementation	OwnableUpgradeable, DividendPayingToken		

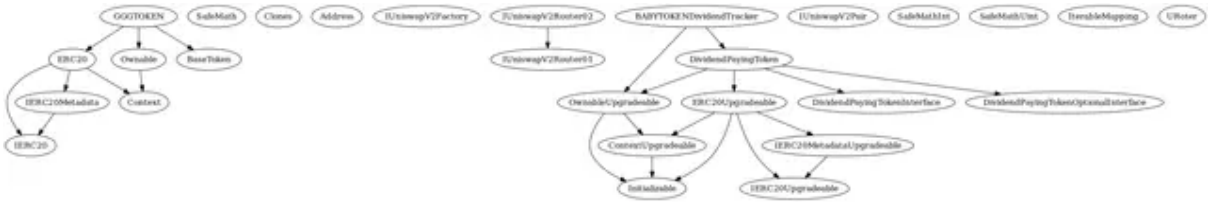
	initialize	External	✓	initializer
	_transfer	Internal		
	withdrawDividend	Public		-
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	Public		-
	updateClaimWait	External	✓	onlyOwner
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
BaseToken	Implementation			
GGGTOKEN	Implementation	ERC20, Ownable, BaseToken		
		Public	✓	-
	initialize	Public	✓	-
		External	Payable	-
	setSwapTokensAtAmount	External	✓	onlyOwner

	excludeFromFees	External	✓	onlyOwner
	setMarketAddr	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateGasForProcessing	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	getMinimumTokenBalanceForDividends	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		-
	_transfer	Internal	✓	
	takeBuy	Private	✓	
	takeSell	Private	✓	

	swapAndSendToFee	Private	✓	
	swapAndLiquify	Private	✓	
	swapTokensForEth	Private	✓	
	swapTokensForCake	Private	✓	
	addLiquidity	Private	✓	
	swapAndSendDividends	Private	✓	
	updateDividendTracker	Public	✓	onlyOwner
	_takeInviter	Private	✓	
	setBatchBot	Public	✓	onlyOwner
	getBot	Public		-
	addBot	Private	✓	
	getFeeDeno	Public		-
	getMarketFee	Public		-
	getLiqFee	Public		-
	getRewardFee	Public		-
	getSellFee	Public		-
	getBuyFee	Public		-
	setTranFee	External	✓	onlyOwner
	setDropNum	External	✓	onlyOwner
	takeInviterFee	Private	✓	
	getInvitersDetail	Public		-
	excludeFromBatchFee	External	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner

	isContract	Internal		
	bind	Private	✓	
	getPar	Public		-
	setInviKillBlock	Public	✓	onlyOwner
	setUniswapV2Pair	External	✓	onlyOwner
	takeBot	Private	✓	
URoter	Implementation			
		Public	✓	-

Inheritance Graph



Flow Graph

Summary

HAPPYFARM contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. HAPPYFARM is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract ownership has been renounced.

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