Cyberscope

Audit Report

DOGES FARM

August 2023

Network BSC

Address 0xf0ac6f0a4a2f4c32463d947e20742839a94efc1b

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
	EPC	Existing Pair Creation	Unresolved
	CO	Code Optimization	Unresolved
	PVC	Price Volatility Concern	Unresolved
	MEM	Misleading Error Messages	Unresolved
	MEE	Missing Events Emission	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L09	Dead Code Elimination	Unresolved
	L16	Validate Variable Setters	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	DOGES_FARM_Token
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Explorer	https://bscscan.com/address/0xf0ac6f0a4a2f4c32463d947e20742839a94efc1b
Address	0xf0ac6f0a4a2f4c32463d947e20742839a94efc1b
Network	BSC
Symbol	DOGES
Decimals	18
Total Supply	100,000,000,000,000

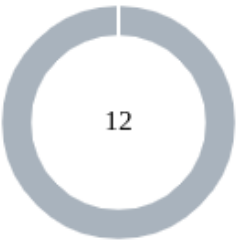
Audit Updates

Initial Audit	17 Aug 2023

Source Files

Filename	SHA256
DOGES_FARM_Token.sol	f51236bf8623960f2a2e467e836dadd16cbce7d426acd0cfc7ea91b4ba1ba945

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	12	0	0	0

EPC - Existing Pair Creation

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L798
Status	Unresolved

Description

The contract is using the `updateUniswapV2Router` function to update Uniswap V2 Router and create a new pair on the exchange. However, the function does not handle the scenario where a pair already exists prior to its execution. If a pair for the given tokens has already been established, the `createPair` function will revert and not proceed with the creation of a new pair. As a result, if a pair has been previously set up before the `updateUniswapV2Router` function is invoked, the contract will encounter an error when trying to call the `createPair` function within the `updateUniswapV2Router` function. This will prevent the successful execution of the `updateUniswapV2Router` function, effectively locking the contract to the old pair and preventing it from updating to the new pair address.

```
    function updateUniswapV2Router(address newAddress)
        external
        onlyOperator
    {
        require(
            newAddress != address(uniswapV2Router),
            "The router has that address already"
        );
        emit UpdateUniswapV2Router(newAddress,
address(uniswapV2Router));
        uniswapV2Router = IUniswapV2Router02(newAddress);
        address _uniswapV2Pair =
IUniswapV2Factory(uniswapV2Router.factory())
            .createPair(address(this), uniswapV2Router.WETH());
        uniswapV2Pair = _uniswapV2Pair;
    }
```

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 Ethereum blockchain or utilizing external libraries that provide contract verification services.

CO - Code Optimization

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L934
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 is currently using two separate if statements to determine whether the `takeFee` variable should be set to `false`. Both conditions, when met, produce the exact same result, which is setting the `takeFee` variable to `false`. Specifically:

1. The first if statement checks if either the from or to address is excluded from fees.
2. The second if statement checks if a wallet-to-wallet transfer should occur without a fee and ensures that neither the from nor the to address is the uniswapV2Pair.

Given that the outcome of both conditions is identical, there's an opportunity to simplify and optimize the code by combining these conditions using the logical OR (||) operator.

```
        if(_isExcludedFromFees[from] || _isExcludedFromFees[to]) {
            takeFee = false;
        }

        if(walletToWalletTransferWithoutFee && from != uniswapV2Pair &&
to != uniswapV2Pair) {
            takeFee = false;
        }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. It is recommended to consolidate the two if statements into a single statement using the `||` operator. This will not only make the code more

concise but also enhance its readability and maintainability. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L898,969
Status	Unresolved

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.

```
        uint256 contractTokenBalance = balanceOf(address(this));

        bool canSwap = contractTokenBalance >=
swapTokensAtAmount;

        if (swapEnabled &&
            canSwap &&
            !swapping &&
            !automatedMarketMakerPairs[from] &&
            _totalFeesOnBuy + _totalFeesOnSell > 0
        ) {
            ...
            if (swapWithLimit) {
                contractTokenBalance = swapTokensAtAmount;
            }
            ...
            if(contractTokenBalance > 0 && totalFee > 0) {
                address[] memory path = new address[](2);
                path[0] = address(this);
                path[1] = uniswapV2Router.WETH();


uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
                    contractTokenBalance,
                    0,
                    path,
                    marketingWallet,
                    block.timestamp);
            }

    function setSwapTokensAtAmount(uint256 newAmount) external
onlyOwner {
        require(
            newAmount > totalSupply() / 1_000_000,
            "New Amount must more than 0.0001% of total supply"
        );
        swapTokensAtAmount = newAmount;
    }
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L858,867
Status	Unresolved

Description

The contract is using misleading error messages.The `updateBuyFees` and `updateSellFees` functions that are designed to update the fee values of `_totalFeesOnBuy` and `_totalFeesOnSell` varriables respectively. Both functions allow the owner to set the respective fees up to a maximum value of 10%. This is evident from the condition `_marketingFeeOnBuy <= 10` and `_marketingFeeOnSell <= 10`. However, the error message associated with this condition states, "Fees must be less than 3%". 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.

```
    function updateBuyFees(uint256 _marketingFeeOnBuy) external
onlyOwner {
        require(
            _marketingFeeOnBuy <= 10,
            "Fees must be less than 3%"
        );
        _totalFeesOnBuy = _marketingFeeOnBuy;
        emit UpdateBuyFees(_totalFeesOnBuy);
    }

    function updateSellFees(uint256 _marketingFeeOnSell)
external onlyOwner {
        require(
            _marketingFeeOnSell <= 10,
            "Fees must be less than 3%"
        );
        _totalFeesOnSell = _marketingFeeOnSell;
        emit UpdateSellFees(_totalFeesOnSell);
    }
```

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.

Specifically, the team could consider updating the error message in both `updateBuyFees` and `updateSellFees` functions to accurately reflect the actual fee percentage allowed by the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L781
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 changeOperator(address newOperator) onlyOperator
public {
        require(newOperator != operator, "New operator is the
same as the old one");
        operator = newOperator;
    }
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L719
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 private DEAD =
0x000000000000000000000000000000000000dEaD
```

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	DOGES_FARM_Token.sol#L315,316,333,353,706,709,710,719,851,860,874,957,973
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);

...
```

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	DOGES_FARM_Token.sol#L970
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 = newAmount
```

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	DOGES_FARM_Token.sol#L77,101,126,136,155,169,186,196,211,221,236,260,272,664
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) {
        // This method relies on
extcodesize/address.code.length, which returns 0
        // for contracts in construction, since the code is
only stored at the end
        // of the constructor execution.

        return account.code.length > 0;
    }

function sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

...
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L810
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 = _uniswapV2Pair
```

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	DOGES_FARM_Token.sol#L277
Status	Unresolved

Description

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

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

```
assembly {
                let 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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	DOGES_FARM_Token.sol#L790
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.

```
ERC20token.transfer(msg.sender, balance)
```

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**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	

	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
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	✓	-

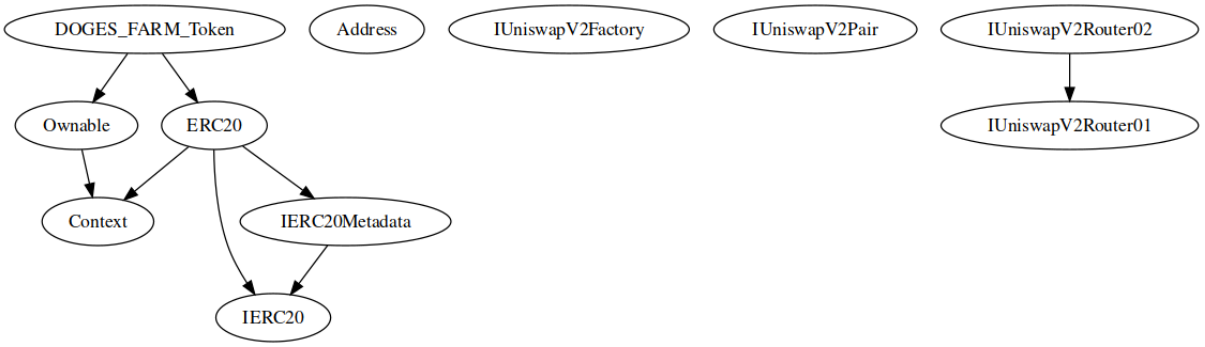
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	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	IUniswapV2 Router01		

	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	transfer	External	✓	-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-

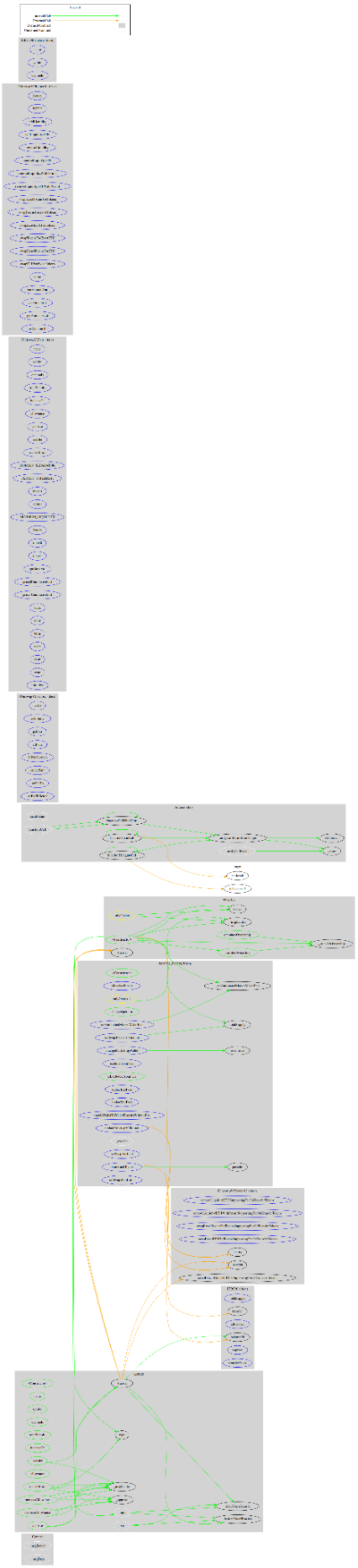
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
DOGES_FARM_Token	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	changeOperator	Public	✓	onlyOperator
	claimStuckTokens	External	✓	onlyOwner

	isContract	Internal		
	updateUniswapV2Router	External	✓	onlyOperator
	setAutomatedMarketMakerPair	External	✓	onlyOperator
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromFees	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	enableWalletToWalletTransferWithoutFee	External	✓	onlyOwner
	changeMarketingWallet	External	✓	onlyOwner
	_transfer	Internal	✓	
	setSwapEnabled	External	✓	onlyOwner
	setSwapTokensAtAmount	External	✓	onlyOwner
	setSwapWithLimit	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

DOGES FARM contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. DOGES FARM is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 10% 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