Cyberscope

Audit Report

BabyFarm

November 2023

Network	BSC
Address	0x8d25a02248b31227735cd4d4524a1bd54a951759
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	Unresolved

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	UPA	Unexcluded Pinksale Address	Unresolved
●	DFF	Duplicate Fee Functionality	Unresolved
●	CO	Code Optimization	Unresolved
●	RFF	Redundant Fee Function	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	CR	Code Repetition	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	BabyFarm
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x8d25a02248b31227735cd4d4524a1bd54a951759
Address	0x8d25a02248b31227735cd4d4524a1bd54a951759
Network	BSC
Symbol	BabyFarm
Decimals	18
Total Supply	100,000,000,000

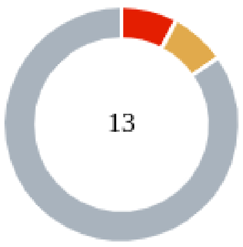
Audit Updates

Initial Audit	26 Nov 2023

Source Files

Filename	SHA256
contracts/BabyToken.sol	4ac3288030155ccea8752138b30137009e6f23725c80c0a31439e2336c1835ce

Findings Breakdown



●	Critical	1
●	Medium	1
●	Minor / Informative	11

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	11	0	0	0

BC - Blacklists Addresses

Criticality	Critical
Location	contracts/BabyToken.sol#L663
Status	Unresolved

Description

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

```
    function addInBlackList(address account, bool) public 
onlyOwner {
        blackListed[account] = true;
    }
```

Recommendation

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

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

Permanent Solution:

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

UPA - Unexcluded Pinksale Address

Criticality	Medium
Location	contracts/BabyToken.sol#L480,573
Status	Unresolved

Description

The contract is designed with a fee mechanism that applies to each transaction. This design, poses a significant obstacle for integration with platforms like Pinksale. Specifically, for the creation of a launchpad on Pinksale, the Pinksale factory address must be exempted from these transaction fees. Without this exemption, the creation of the pool on Pinksale will be prevented.

Additionally, the contract currently confuses the exclusion from fees with the buy/sell fees. The mechanism for excluding certain addresses from fees is not adequately distinguished from the buy/sell fee calculations. This confusion could lead to operational issues, especially when the contract interacts with decentralized exchange platforms where the pair address is involved. In such cases, the exclusion from fees should be clearly defined and should not merely rely on a specific receipt address.

```
   isExcludedFromFee[taxReceiver] = true;
    isExcludedFromFee[address(this)] = true;
...
    if (isExcludedFromFee[sender] &&
isExcludedFromFee[recipient]) {
        transferAmount = amount;
    }
    if (!isExcludedFromFee[sender] &&
!isExcludedFromFee[recipient]) {
        transferAmount = betweenFee(sender, amount);
    }
    if (isExcludedFromFee[sender] &&
!isExcludedFromFee[recipient]) {
        transferAmount = BuyFee(sender, amount);
    }
    if (!isExcludedFromFee[sender] &&
isExcludedFromFee[recipient]) {
        transferAmount = SellFee(sender, amount);
    }
```

Recommendation

It is recommended to modify the contract to exclude the Pinksale factory address from the fee mechanism. This will ensure compatibility with Pinksale and facilitate the smooth creation of pools on the platform.

DFF - Duplicate Fee Functionality

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L595,617
Status	Unresolved

Description

The contract is currently equipped with two separate functions, `BuyFee` and `SellFee`, each designed to calculate and deduct a fee for buy and sell transactions, respectively. These functions operate by applying a fee percentage, represented by the `_buy` and `_sell` variables. However, the values of `_buy` and `_sell` are equal and, these variables are not configured to be updated. This situation leads to functional redundancy since both `BuyFee` and `SellFee` essentially perform the same operation using different variables that hold the same value. This redundancy not only complicates the contract's structure but also introduces unnecessary complexity and potential for confusion, as two separate functions are maintained for what is effectively a single fee calculation process.

```
   function BuyFee(
        address account,
        uint256 amount /*, uint256 rate*/
    ) private returns (uint256) {
        uint256 transferAmount = amount;
        uint256 buyFee = amount.mul(_buy).div(10000);

        if (buyFee > 0) {
            transferAmount = transferAmount.sub(buyFee);
            _balances[address(taxReceiver)] =
_balances[address(taxReceiver)]
                .add(buyFee);
            _buyTotal = _buyTotal.add(buyFee);
            emit Transfer(account, address(taxReceiver),
buyFee);
        }
        return transferAmount;
    }


    function SellFee(
        address account,
        uint256 amount
    ) private returns (uint256) {
        uint256 transferAmount = amount;
        uint256 sellFee = amount.mul(_sell).div(10000);

        if (sellFee > 0) {
            transferAmount = transferAmount.sub(sellFee);
            _balances[address(taxReceiver)] =
_balances[address(taxReceiver)]
                .add(sellFee);
            _sellTotal = _sellTotal.add(sellFee);
            emit Transfer(account, address(taxReceiver),
sellFee);
        }

        return transferAmount;
    }
```

`

Recommendation

It is recommended to refactor the contract by consolidating the `BuyFee` and `SellFee` functions into a single function. Since the `_buy` and `_sell` fee variables are equal and immutable, there is no practical need to maintain separate functions for buy and sell fee calculations. A unified function, could be implemented to handle the fee calculation for both buying and selling scenarios. This approach would simplify the contract's logic, reduce redundancy, and improve the overall clarity and maintainability of the code. Additionally, this consolidation could lead to minor gas optimizations by reducing the number of function calls and simplifying the contract's execution flow.

CO - Code Optimization

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L677
Status	Unresolved

Description

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

Specifically the contract includes the `isBlackListed` function, which is intended to determine whether a given address is on a blacklist. The function's current implementation involves an if-else statement to check the condition `blackListed[_address] == true`. If this condition is true, the function `returns true;` otherwise, it `returns false`. The if-else structure adds additional lines of code without providing any functional benefit, as the same result can be achieved with a more direct approach. This redundancy not only impacts the readability of the code but also leads to slight inefficiencies in terms of gas usage.

```
    function isBlackListed(
            address _address
        ) public view returns (bool _blacklisted) {
            if (blackListed[_address] == true) {
                return true;
            } else {
                return false;
            }
        }
```

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. It is recommended to refactor the function for improved efficiency and readability. The function can be simplified by directly returning the result of the condition check, eliminating the need for the if-else statement. This streamlined version directly evaluates and returns the boolean value of `blackListed[_address]` , making the function more concise and reducing the gas cost associated with the execution of conditional statements. This change will enhance the overall clarity and performance of the contract.

RFF - Redundant Fee Function

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L474,640
Status	Unresolved

Description

The contract contains the `betweenFee` function, which ostensibly deducts a fee, termed `_inbetweenFee_`, from the transfer amount. This fee is intended to be a percentage of the transaction amount. However, the `_inbetweenFee_` variable is initialized with a value of 0 and lacks any mechanism for updating this value in the future. Consequently, the calculation within the `betweenFee` function (`amount.mul(_inbetweenFee_).div(10000)`) will always result in zero, rendering the fee deduction non-functional. This situation leads to the `betweenFee` function performing no actual fee deduction, as the condition `_inbetweenFee > 0` will never be met. Therefore, the function, in its current state, does not fulfill its intended purpose of fee reduction and merely adds unnecessary complexity to the contract.

```
    uint256 public _inbetweenFee_ = 0; // 0%
...
    function betweenFee(
        address account,
        uint256 amount
    ) private returns (uint256) {
        uint256 transferAmount = amount;

        uint256 _inbetweenFee =
amount.mul(_inbetweenFee_).div(10000);

        if (_inbetweenFee > 0) {
            transferAmount = transferAmount.sub(_inbetweenFee);
            _balances[address(taxReceiver)] =
_balances[address(taxReceiver)]
                .add(_inbetweenFee);
            _inbetweenFeeTotal =
_inbetweenFeeTotal.add(_inbetweenFee);
            emit Transfer(account, address(taxReceiver),
_inbetweenFee);
        }

        return transferAmount;
    }
```

Recommendation

It is recommended to streamline the contract by removing the `betweenFee` function. Since the `_inbetweenFee_` variable is permanently set to zero and lacks the capability to be updated, the function does not execute any real fee deduction. Eliminating this redundant function will simplify the contract's logic, reduce gas costs associated with unnecessary computations, and enhance overall contract efficiency. This change will ensure that the contract's functionality aligns with its actual behavior, thereby improving clarity and maintainability.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L663
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 addInBlackList(address account, bool) public onlyOwner
{
        blackListed[account] = true;
    }

    function removeFromBlackList(address account, bool) public
onlyOwner {
        blackListed[account] = false;
    }
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L560
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
require(
    !blackListed[sender],
    "You are blacklisted so you can not Transfer Gen tokens."
);
require(
    !blackListed[recipient],
    "blacklisted address canot be able to recieve Gen tokens."
);
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places. Specifically the contract could use the OR operator (||) instead of using two separate require statements.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/BabyToken.sol
Status	Unresolved

Description

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

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

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

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

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

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

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L458,459,460,465,470,471,472,474
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.

```
string private _name = "Baby Farm"
string private _symbol = "BabyFarm"
uint8 private _decimals = 18
uint256 internal _totalSupply = 100000000000 * 10 ** 18
uint256 public _buy = 500
uint256 public _sell = 500
address public taxReceiver =
0xA775917e13b9aeA6d2e98eA794Da0662864Cc627
uint256 public _inbetweenFee_ = 0
```

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	contracts/BabyToken.sol#L462,463,465,470,471,474,475,476,477,595,617,678
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.

```
mapping(address => uint256) internal _balances
mapping(address => mapping(address => uint256)) internal
_allowances
uint256 internal _totalSupply = 100000000000 * 10 ** 18
uint256 public _buy = 500
uint256 public _sell = 500
uint256 public _inbetweenFee_ = 0
uint256 public _buyTotal
uint256 public _sellTotal
uint256 public _inbetweenFeeTotal

...
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L680
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
blackListed[_address] == true
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L9,35,124,214,453
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.19;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/BabyToken.sol#L9,35,124,214,453
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;
pragma solidity ^0.8.19;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

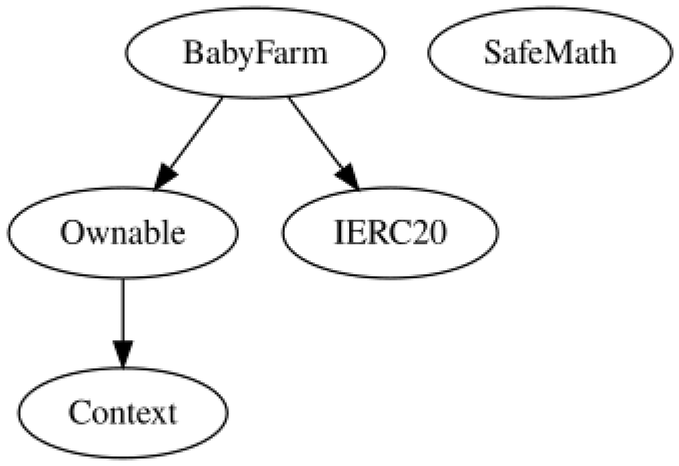
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

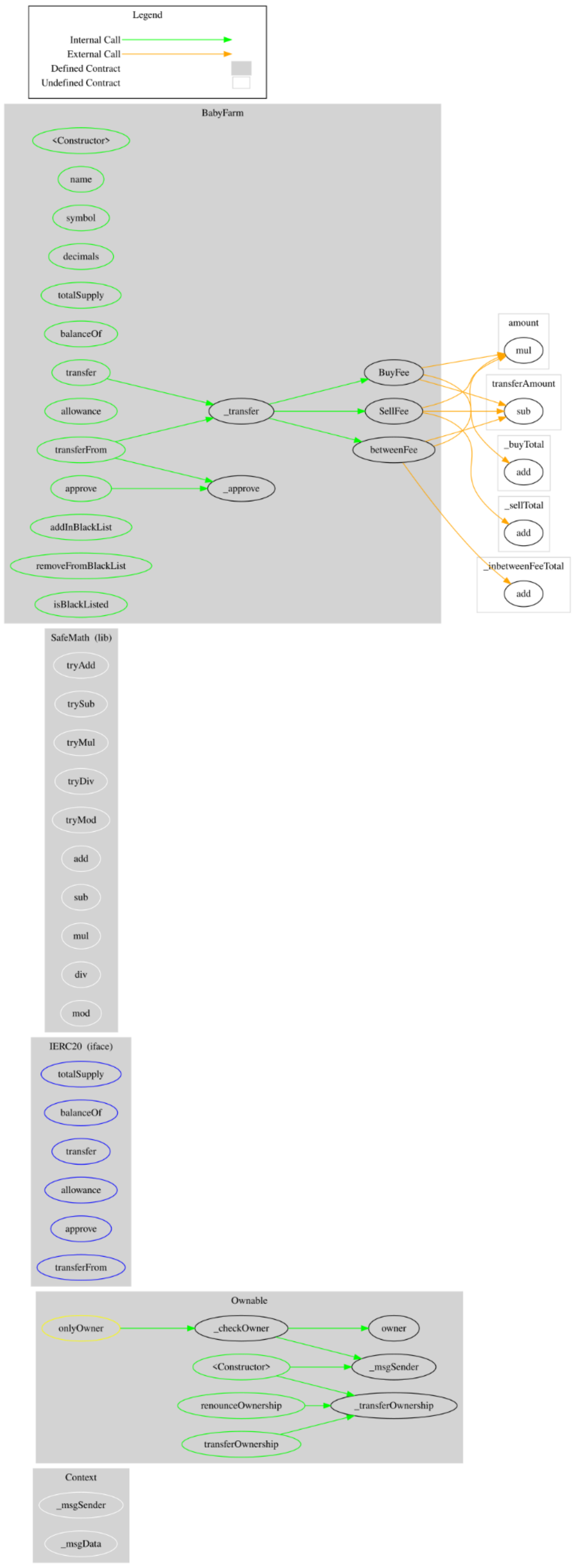
	transferFrom	External	✓	-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
BabyFarm	Implementation	IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-

	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	BuyFee	Private	✓	
	SellFee	Private	✓	
	betweenFee	Private	✓	
	addInBlackList	Public	✓	onlyOwner
	removeFromBlackList	Public	✓	onlyOwner
	isBlackListed	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BabyFarm
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
approve
addInBlackList
removeFromBlackList
isBlackListed
_transfer
_approve
BuyFee
SellFee
betweenFee
amount
mul
transferAmount
sub
_buyTotal
add
_sellTotal
add
_inbetweenFeeTotal
add
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
_checkOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
Context
_msgSender
_msgData

Summary

BabyFarm 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 massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of a 5% fee on buy and sell transactions.

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