Audit Report

0xElonDogeBaseOptimismArbitrumLineaPolygonZKInu

Aug 2023

SHA256 84039afc1dde7daf02f7bb1a9d16d3de0ff9f95dcda4c6c8eb0d6b48f3e77659

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed
●	MPR	Manipulates Pair Reserves	Unresolved

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PUA	Potential Unsynchronised Allowances	Unresolved
●	ZD	Zero Division	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	NOO	Numeric Operation Optimization	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	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
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved

●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	TEST
Testing Deploy	https://testnet.bscscan.com/address/0xa4c7caa850b3a0e7876814e20f0ee54fd7ca3b6e
Symbol	ETST
Decimals	9
Total Supply	1,000,000,000,000,000,000

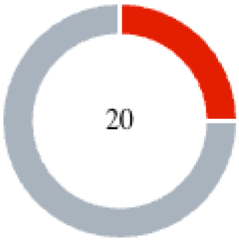
Audit Updates

Initial Audit	09 Aug 2023

Source Files

Filename	SHA256
contracts/newrebase.sol	84039afc1dde7daf02f7bb1a9d16d3de0ff9f95dcda4c6c8eb0d6b48f3e77659

Findings Breakdown



- Critical 5
- Medium 0
- Minor / Informative 15

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	5	0	0	0
Medium	0	0	0	0
Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/newrebase.sol#L1316,1331
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enabled the owner will not be able to disable them again.

```
if (!authorizations[sender] && !authorizations[recipient]) {
    require(tradingOpen, "Trading not open yet");
}
```

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

```
SellFee = _Fee;

uint256 amountReceived = shouldTakeFee(sender)
    ? takeFee(sender, amount)
    : amount;
_balances[recipient] = _balances[recipient].add(
    amountReceived.mul(BASE)
);
```

Recommendation

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

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

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/newrebase.sol#L1402,1411
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setSellFee` and `setBuyFee` function with a high percentage value, or by setting the `feeDenominator` less than the fee percentage.

```
//setting sell fees
function setSellFee(
    uint256 _Fee,
    uint256 _feeDenominator
) external authorized {
    SellFee = _Fee;
    feeDenominator = _feeDenominator;
}

//setting buy fees
function setBuyFee(
    uint256 _Fee,
    uint256 _feeDenominator
) external authorized {
    BuyFee = _Fee;
    feeDenominator = _feeDenominator;
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

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

MPR - Manipulates Pair Reserves

Criticality	Critical
Location	contracts/newrebase.sol#L1267,127
Status	Unresolved

Description

The `setBaseratioAndSync` function, when invoked by the owner with a high value, enforces a modification in the price ratio of the pair address. For instance, it can alter the ratio from `1000 TEST -> 10 BNB` to `0.001 TEST -> 10 BNB`.

Subsequently, if the owner promptly follows up by utilizing the `setBaseratio` function, which does not trigger a synchronization of the pair, to establish a lower ratio, and subsequently executes a sales transaction, the owner effectively seizes the entire pool. In other terms, the pair ratio will remain unchanged, while the owner's balance will increase.

```
function setBaseratio(uint256 _ratio) external onlyOwner {
    BASE = _ratio;
}

function setBaseratioAndSync(uint256 _ratio) external onlyOwner {
    BASE = _ratio;
    IUniswapV2Pair(uniswapV2PairAddress).sync();
}
```

Recommendation

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

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

PUA - Potential Unsynchronised Allowances

Criticality	Critical
Location	contracts/newrebase.sol#L1252
Status	Unresolved

Description

The contract employs a token normalization mechanism that is controlled by the `BASE` variable. This mechanism involves the multiplication and division of token quantities as needed. However, a significant drawback arises from this approach, as adjustments to the `BASE` value by the owner can lead in a lack of synchronization in the allowances.

```
function allowance(
    address holder,
    address spender
) public view override returns (uint256) {
    return _allowances[holder][spender];
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that the `allowances` are always in sync with any modification in the `BASE` variable.

ZD - Zero Division

Criticality	Critical
Location	contracts/newrebase.sol#L1229,1249
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
_totalSupply.div(BASE)
_balances[account].div(BASE)
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L1212,1214
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 variables `isFeeExempt` and `isNoFee` both serve the same purpose from our understanding. Both are used to determine if a given address should be excluded from fees or not. As a result, declaring both variables is redundant.

```
mapping(address => bool) isFeeExempt;
mapping(address => bool) isNoFee;
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L1371,1393,1398,1421
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 modifies the state of certain variables even when their current state is equal to the provided argument. As a result, the contract performs redundant storage writes.

```
isNoFee[_NoFee] = yes
tradingOpen = true
isFeeExempt[holder]
FeeReceiver = _receiver
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L1268,1366,1367,1371,1393,1398,1421
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.

```
BASE = _ratio
uniswapV2PairAddress = _pair
isPair[_pair] = yes
isNoFee[_NoFee] = yes
tradingOpen = true
isFeeExempt[holder] = exempt
FeeReceiver = _receiver
```

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.

NOO - Numeric Operation Optimization

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L1206
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 `1 * 10 ** 18` operation can be simplified further: `1e18`.

```
uint256 _totalSupply = 1 * 10 ** 18 * (10 ** _decimals)
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/newrebase.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.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L385,397,409,421
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
AddressSlot storage r
BooleanSlot storage r
Bytes32Slot storage r
Uint256Slot storage r
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L1200,1201,1206
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 DEAD = 0x000000000000000000000000000000000000dEaD;
address ZERO = 0x0000000000000000000000000000000000000000;
uint256 _totalSupply = 1 * 10 ** 18 * (10 ** _decimals);
```

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/newrebase.sol#L1101,1128,1130,1200,1201,1203,1204,1205,1206,1208,1210,1218,1219,1220,1221,1223,1267,1271,1365,1370,1392,1403,1404,1412,1413,1420,1431
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 (uint256)
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
...
```

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	contracts/newrebase.sol#L1268,1272,1406,1415
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.

```
BASE = _ratio
SellFee = _Fee
BuyFee = _Fee
```

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	contracts/newrebase.sol#L15,22,30,41,51,136,154,190,206,248,264,302,320,350,383,395,407,419,793,829,855,906,957,1259
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 max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a > b ? a : b;
    }

function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
...
        return (a & b) + (a ^ b) / 2;
    }

function ceilDiv(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b - 1) / b can overflow on addition, so we distribute.
        return a == 0 ? 0 : (a - 1) / b + 1;
    }

...
```

Recommendation

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

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L98,101,113,117,118,119,120,121,122,128,1384,1385
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
denominator := div(denominator, twos)
inverse *= 2 - denominator * inverse
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L1024,1366,1421
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.

```
owner = adr
uniswapV2PairAddress = _pair
FeeReceiver = _receiver
```

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	contracts/newrebase.sol#L62,387,399,411,423
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 mm := mulmod(x, y, not(0))
                prod0 := mul(x, y)
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
            r.slot := slot
        }
...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L3
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

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

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/newrebase.sol#L1432
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, ERC20Token.balanceOf(address(this)));
```

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**
Math	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
StorageSlot	Library			
	getAddressSlot	Internal		

	getBooleanSlot	Internal		
	getBytes32Slot	Internal		
	getUint256Slot	Internal		
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
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	✓	

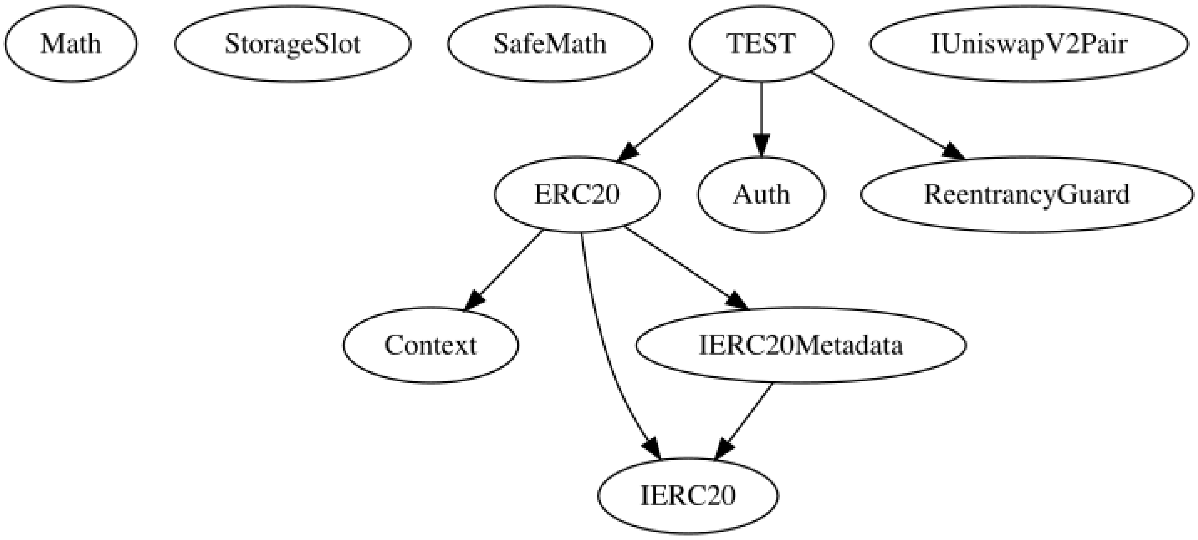
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Auth	Implementation			
		Public	✓	-
	authorize	Public	✓	onlyOwner
	unauthorize	Public	✓	onlyOwner
	isOwner	Public		-
	isAuthorized	Public		-
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Pair	Interface			
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
	permit	External	✓	-
	totalSupply	External		-

	balanceOf	External		-
	allowance	External		-
	DOMAIN_SEPARATOR	External		-
	nonces	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	name	External		-
	symbol	External		-
	decimals	External		-
	PERMIT_TYPEHASH	External		-
	MINIMUM_LIQUIDITY	External		-
ReentrancyGuard	Implementation			
		Public	✓	-
TEST	Implementation	ERC20, Auth, ReentrancyGuard		
		Public	✓	Auth

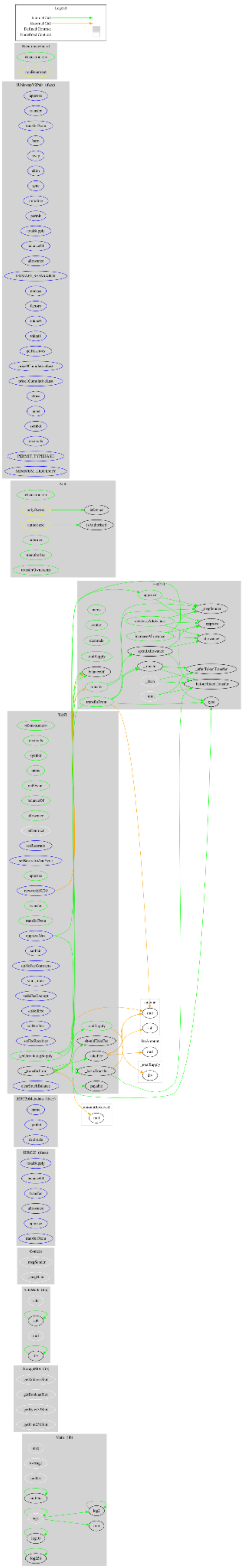
	totalSupply	Public		-
	decimals	Public		-
	symbol	Public		-
	name	Public		-
	getOwner	Public		-
	balanceOf	Public		-
	allowance	Public		-
	isContract	Internal		
	setBaseratio	External	✓	onlyOwner
	setBaseratioAndSync	External	✓	onlyOwner
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	shouldTakeFee	Internal		
	setPair	External	✓	authorized
	setNoFeeContracts	External	✓	authorized
	takeFee	Internal	✓	
	start_trade	External	✓	onlyOwner
	setIsFeeExempt	External	✓	authorized
	setSellFee	External	✓	authorized

	setBuyFee	External	✓	authorized
	setFeeReceiver	External	✓	authorized
	clearStuckBalance	External	✓	authorized
	recoverERC20	External	✓	authorized
	getCirculatingSupply	Public		-

Inheritance Graph



Flow Graph



Summary

0xElonDogeBaseOptimismArbitrumLineaPolygonZKInu contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and manipulate the fees. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. A multi-wallet signing pattern will provide security against potential hacks. Additionally, certain admin functions are accessible from authorized users. Temporarily locking the contract or removing all authorized users and renouncing ownership will eliminate all the contract threats.

Disclaimer

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

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

About Cyberscope

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

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

The Cyberscope team

https://www.cyberscope.io