Cyberscope

Audit Report

Kendu Inu

June 2024

Network	ETH
Address	0xaa95f26e30001251fb905d264aa7b00ee9df6c18
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
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RCI	Redundant Check Inefficiency	Unresolved
●	RED	Redundant Event Declaration	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Review

Contract Name	Kendu
Compiler Version	v0.8.13+commit.abaa5c0e
Optimization	200 runs
Explorer	https://etherscan.io/address/0xaa95f26e30001251fb905d264aa7b00ee9df6c18
Address	0xaa95f26e30001251fb905d264aa7b00ee9df6c18
Network	ETH
Symbol	Kendu
Decimals	18
Total Supply	1,000,000,000,000
Badge Eligibility	Yes

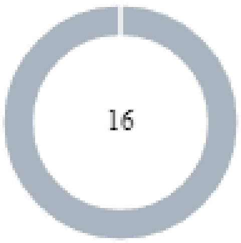
Audit Updates

Initial Audit	11 Jun 2024

Source Files

Filename	SHA256
Kendu.sol	24da439faadee2a24195d76af66cdcccef8b5bb4a11ecd764ee83650727357cd

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	16

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

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	Kendu.sol#L1252,1262
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
uint256 amountToSwapForETH = contractBalance.sub(liquidityTokens);

uint256 ethForLiquidity = ethBalance - ethForMarketing - ethForDev;
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Kendu.sol#L1180
Status	Unresolved

Description

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

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

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

```
if (automatedMarketMakerPairs[to] && sellTotalFees > 0){
    fees = amount.mul(sellTotalFees).div(100);
    tokensForLiquidity += fees * sellLiquidityFee /
sellTotalFees;
    tokensForDev += fees * sellDevFee / sellTotalFees;
    tokensForMarketing += fees * sellMarketingFee /
sellTotalFees;
}
// on buy
else if(automatedMarketMakerPairs[from] && buyTotalFees > 0) {
    fees = amount.mul(buyTotalFees).div(100);
    tokensForLiquidity += fees * buyLiquidityFee /
buyTotalFees;
    tokensForDev += fees * buyDevFee / buyTotalFees;
    tokensForMarketing += fees * buyMarketingFee /
buyTotalFees;
}
```

Recommendation

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

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Kendu.sol#L998,1005,1011,1016,1021,1028,1033,1038,1043...
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 updateMaxTxnAmount(uint256 newNum) external onlyOwner
{
    require(newNum >= (totalSupply() * 1 / 1000)/1e18, "Cannot
set maxTransactionAmount lower than 0.1%");
    maxTransactionAmount = newNum * (10**18);
}

function updateMaxWalletAmount(uint256 newNum) external
onlyOwner {
    require(newNum >= (totalSupply() * 5 / 1000)/1e18, "Cannot
set maxWallet lower than 0.5%");
    maxWallet = newNum * (10**18);
}/
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Kendu.sol#L1204
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] = uniswapV2Router.WETH();

    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // make the swap
    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );

}
```

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.

RCI - Redundant Check Inefficiency

Criticality	Minor / Informative
Location	Kendu.sol#L1114
Status	Unresolved

Description

The contract is structured with nested if statements that include checks to determine the status of various conditions before executing certain operations. Specifically, within the outer if statement, the condition checks whether both from and to are not equal to owner, among other conditions. Subsequently, within an inner if statement, there is a repeated check to ascertain if `to != owner()`. This repetition is redundant because the condition `to != owner()` has already been validated in the preceding outer if statement. As a result, the second check for `to != owner()` within the inner if statement is unnecessary and does not contribute any additional logic or security to the contract. This redundancy can lead to inefficiencies in the contract's execution and may cause confusion or misinterpretation of the contract's intended logic.

```
if(limitsInEffect){
        if (
            from != owner() &&
            to != owner() &&
            to != address(0) &&
            to != address(0xdead) &&
            !swapping
        ){
            if(!tradingActive){
                require(_isExcludedFromFees[from] ||
_isExcludedFromFees[to], "Trading is not active.");
            }

            // at launch if the transfer delay is enabled,
ensure the block timestamps for purchasers is set -- during
launch.
            if (transferDelayEnabled){
                if (to != owner() && to !=
address(uniswapV2Router) && to != address(uniswapV2Pair)){

require(_holderLastTransferTimestamp[tx.origin] <=
block.number, "_transfer:: Transfer Delay enabled.  Only one
purchase per block allowed.");
                    _holderLastTransferTimestamp[tx.origin] =
block.number;
                }
            }
```

Recommendation

It is recommended to remove the second if statement that checks `to != owner()` within the inner if block. Since this condition is already evaluated in the outer if statement, its presence in the inner block is superfluous and does not alter the outcome of the conditional checks. Eliminating this redundant check will streamline the contract's logic, making it more efficient and easier to understand. This simplification will not only enhance the readability of the contract but also reduce the potential for errors or misunderstandings related to the contract's intended behavior.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	Kendu.sol#L919,935,937,1099
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event UpdateUniswapV2Router(address indexed newAddress, address
indexed oldAddress);

event AutoNukeLP();

event ManualNukeLP();

event BoughtEarly(address indexed sniper);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	Kendu.sol#L475
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Kendu.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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Kendu.sol#L998,1005,1011,1016,1021,1028,1033,1038,1043...
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function updateMaxTxnAmount(uint256 newNum) external onlyOwner
{
    require(newNum >= (totalSupply() * 1 / 1000)/1e18, "Cannot
set maxTransactionAmount lower than 0.1%");
    maxTransactionAmount = newNum * (10**18);
}

function updateMaxWalletAmount(uint256 newNum) external
onlyOwner {
    require(newNum >= (totalSupply() * 5 / 1000)/1e18, "Cannot
set maxWallet lower than 0.5%");
    maxWallet = newNum * (10**18);
}/
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Kendu.sol#L39,40,56,725,909,921,923,1043,1051,1275
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);
mapping (address => bool) public
_isExcludedMaxTransactionAmount
event marketingWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event devWalletUpdated(address indexed newWallet, address
indexed oldWallet);
uint256 _liquidityFee
uint256 _marketingFee
uint256 _DevFee

...
```

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	Kendu.sol#L657,882
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
mapping (address => uint256) private _holderFirstBuyTimestamp
```

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	Kendu.sol#L1020,1026,1031,1044,1052
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
maxTransactionAmount = newNum * (10**18)
maxWallet = newNum * (10**18)
buyMarketingFee = _marketingFee
sellMarketingFee = _marketingFee
```

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	Kendu.sol#L404,703,709,716
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);

        _balances[account] = _balances[account].sub(amount,
"ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

Recommendation

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

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Kendu.sol#L1177,1178,1179,1180,1184,1185,1186,1187
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.

```
fees = amount.mul(buyTotalFees).div(100)
tokensForLiquidity += fees * buyLiquidityFee / buyTotalFees
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	Kendu.sol#L954
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.

```
uint256 totalSupply = 1 * 10 ** 12 * 10 ** decimals()
```

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	Kendu.sol#L1082,1087
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.

```
marketingWallet = newMarketingWallet
devWallet = newWallet
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
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	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

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		
		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	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-

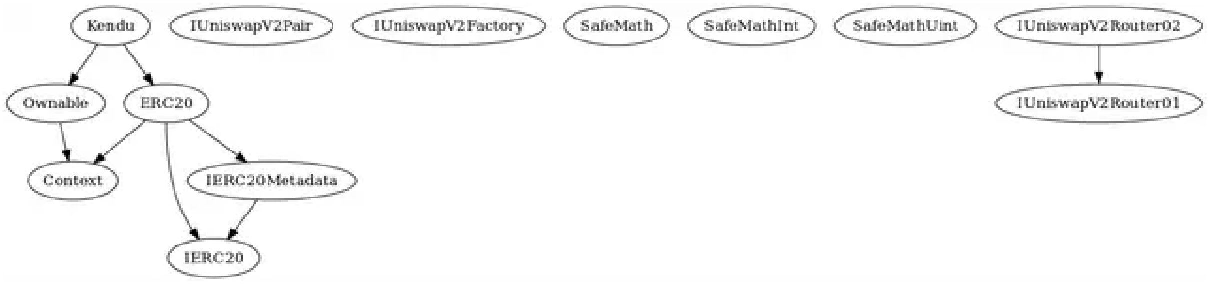
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	Interface			
	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	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Kendu	Implementation	ERC20, Ownable		
		Public	✓	ERC20

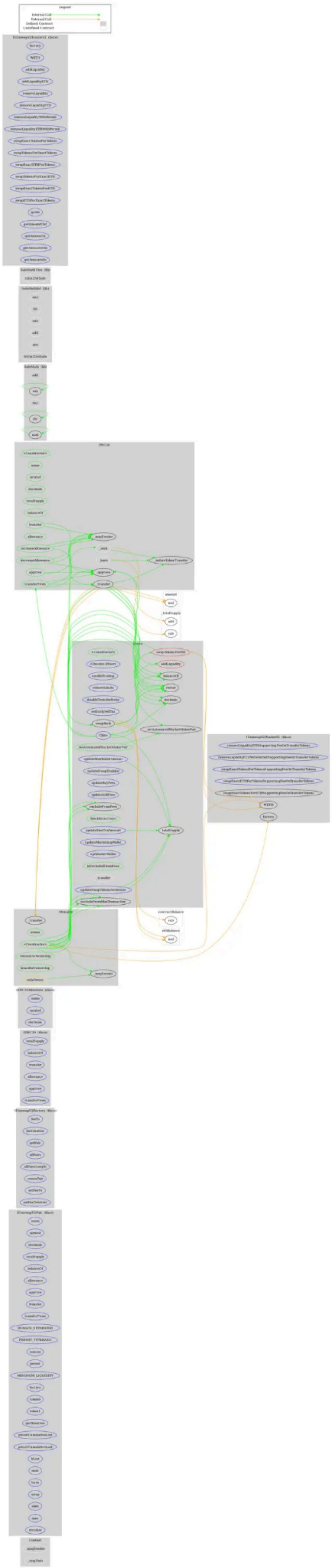
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	setEarlySellTax	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	blacklistAccount	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updatedevWallet	External	✓	onlyOwner
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	

	Chire	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
removeLiquidityWithPermit
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
swapExactETHForTokens
swapTokensForExactETH
swapExactTokensForETH
swapETHForExactTokens
quote
getAmountOut
getAmountIn
getAmountsOut
getAmountsIn
add
sub
mul
div
mod

Summary

Kendu Inu contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Kendu Inu 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