Cyberscope

Audit Report

Oracle AI

February 2024

SHA256 ba8976ebadc3f9f8a5bf4fe9cef00c318cb59a536b3b3c79cdc1ebb0f3498587

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	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	PAE	Permanent Address Exclusion	Unresolved
Medium	TMUC	Token Metadata Unrestricted Change	Unresolved
Minor / Informative	MEM	Misleading Error Messages	Unresolved
Minor / Informative	MC	Missing Check	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
Minor / Informative	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor / Informative	PVC	Price Volatility Concern	Unresolved
Minor / Informative	UFP	Unused Function Parameter	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L05	Unused State Variable	Unresolved
Minor / Informative	L07	Missing Events Arithmetic	Unresolved
Minor / Informative	L08	Tautology or Contradiction	Unresolved

●	L11	Unnecessary Boolean equality	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Review

Contract Name	CoreToken
Testing Deploy	https://testnet.bscscan.com/address/0xad47d0fcd9abb0a1f0b7ba26bc653c62d1913fb7
Symbol	Anonnymous
Decimals	18
Total Supply	10,000,000,000
Badge Eligibility	Must Fix Criticals

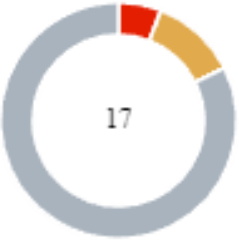
Audit Updates

Initial Audit	07 Feb 2024 https://github.com/cyberscope-io/audits/blob/main/2-oracle/v1/audit.pdf
Corrected Phase 2	13 Feb 2024

Source Files

Filename	SHA256
contracts/token_revised_.sol	ba8976ebadc3f9f8a5bf4fe9cef00c318cb59a536b3b3c79cdc1ebb0f3498587
contracts/lib/IV2Pair.sol	72e4d1f173754ea270e3fbb80e375440e50a4bd75a1654b83d66959b0a2a2bc6
contracts/lib/IRouter02.sol	f377cfd9244dfa9d707118bd71451b5edf8586bbcff343da59fcb034035a0fc5
contracts/lib/IRouter01.sol	13d90aa270f4305a1f70a2eac357b709caacff55d99022138f99f97bcb38cd02
contracts/lib/IFactoryV2.sol	295e59f29cb4b0374666ce6e900db1cb917c7931a5041d8c038b2a240849ef53
contracts/lib/IERC20.sol	11e301dcd4a99fd3fa03e54e6985b4e0b9310465c65e195ad25ec1a48ea2c138

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	14	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/token_revised_.sol#L554
Status	Unresolved

Description

The trading initially is disabled and the contract owner has to enable it. The contract owner has to call the `confirmLP` function.

```
if (!tradingEnabled) {
    revert("Trading not yet enabled!");
}
```

Furthermore, the contract owner has the authority to stop the sales, as described in detail in section PTRP. As a result, the contract might operate as a honeypot.

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.

Furthermore, the team is strongly encouraged to adhere to the recommendations outlined in the respective sections. By doing so, the contract can eliminate any potential of operating as a honeypot.

PAE - Permanent Address Exclusion

Criticality	Medium
Location	contracts/token_revised_.sol#L326
Status	Unresolved

Description

The `multiSendTokens` function implements a mechanism intended to prevent addresses from receiving tokens more than once in a single execution. This is achieved through a mapping that tracks whether an address has already been sent tokens, effectively preventing duplicate transfers within the same transaction. However, this implementation has the consequence of permanently marking addresses as having received tokens, thereby excluding them from participating in any future token distributions via this method. For example, a malicious user could deliberately send minimal amount of tokens to a wide range of addresses, thereby utilizing this mechanism to permanently exclude those addresses from future distributions.

```
function multiSendTokens(
    address[] memory accounts,
    uint256[] memory amounts
) external {
    require(accounts.length == amounts.length, "Lengths do not
match");
    mapping(address => bool) storage sent = _sent;
    for (uint8 i = 0; i < accounts.length; i++) {
        require(balanceOf(msg.sender) >= amounts[i]* 10 **
_decimals);
        require(!sent[accounts[i]],"Duplicate entry");

        _finalizeTransfer(
            msg.sender,
            accounts[i],
            amounts[i] * 10 ** _decimals,
            false,
            false,
            false,
            true
        );
        sent[accounts[i]] = true;
    }
}
```

Recommendation

It is recommended to revise the logic of `multiSendTokens` function, whether this function's intention is temporary or permanent exclusions of addresses from receiving tokens.

TMUC - Token Metadata Unrestricted Change

Criticality	Medium
Location	contracts/token_revised_.sol#L349
Status	Unresolved

Description

The `reveal` function possesses a functionality that allows the contract owner to update the token's metadata, specifically its name and symbol. This function is designed to restrict the update process to a single occurrence by employing a boolean check `require(!revealed,'Already revealed');` . However, the `revealed` boolean, which is intended to enforce this restriction, is not set to true within the `reveal` function after the token metadata is updated. Consequently, this permits the contract owner to modify the token's name and symbol multiple times, despite the presence of a control meant to prevent such repeated changes. This capability to alter the token's metadata at will could lead to potential misuse.

```
function reveal(
    string memory newName,
    string memory newSymbol
) public onlyOwner(){
    require(!revealed,'Already revealed');
    _name = newName;
    _symbol = newSymbol;
}
```

Recommendation

It is recommended that the contract be updated to ensure the `revealed` boolean is set to true as part of the `reveal` function's execution, immediately after the token's metadata is updated. This change would effectively enforce the intended single-use restriction of the `reveal` function, safeguarding the token's metadata against changes post-initial setup.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L143,320,321,333,378
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(!liquidityPoolInitialized)
require(threshold > 0)
require(thresholdDivisor%10 == 0 && thresholdDivisor > 0)
require(balanceOf(msg.sender) >= amounts[i]* 10 ** _decimals)
require(balanceOf(msg.sender) >= amounts[i] * 10 ** _decimals)
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L243,261
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically the `` `newRouter `` and `pair` can be set to zero address.

```
function setNewRouter(address newRouter) public onlyOwner {
    IRouter02 _newRouter = IRouter02(newRouter);
    ...
    }

function setLiquidityPoolPair(
    address pair,
    bool enabled
) public onlyOwner {
    if (enabled == false) {
        allLiquidityPoolPairs[pair] = false;
        ...
        allLiquidityPoolPairs[pair] = true;
        timeSinceLastPairCreated = block.timestamp;
    }
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L279,295,303,315,386,418
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 setTaxes(
    uint16 buyFee,
    uint16 sellFee,
    uint16 transferFee
) external onlyOwner {
    ...
    _taxRates.buyFee = buyFee;
    _taxRates.sellFee = sellFee;
    _taxRates.transferFee = transferFee;
}

function setTaxPercentages(
    uint16 marketing
) external onlyOwner {
    require(marketing>=0 && marketing<=100,'Total should be
100');
    _taxPercentages.marketing = marketing;
    _taxPercentages.dev = 100-marketing;
}

function setMaxTxPercent(
    uint256 percent,
    uint256 divisor
) external onlyOwner {
    ...
    _maxTxAmount = (_tSupply * percent) / divisor;
}

function setSwapSettings(
    uint256 threshold,
    uint256 thresholdDivisor,
    uint256 time
) external onlyOwner {
    require(threshold > 0);
    require(thresholdDivisor%10 == 0 && thresholdDivisor > 0);
    swapThreshold = (_tSupply * threshold) / thresholdDivisor;
    contractSwapTimer = time;
}
```

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	contracts/token_revised_.sol#L442
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 contractSwap(uint256 contractTokenBalance) internal
swapLock {

    TaxPercentages memory taxPercentages = _taxPercentages;

    if (
        _allowances[address(this)][address(dexRouter)] !=
type(uint256).max
    ) {
        _allowances[address(this)][address(dexRouter)] =
type(uint256).max;
    }

    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = dexRouter.WETH();


dexRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
        contractTokenBalance,
        0,
        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.

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L469,472
Status	Unresolved

Description

The contract sends funds to a `dev` and `marketing` address as part of the transfer flow. These addresses can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
if (taxPercentages.dev > 0) {
    _taxWallets.dev.transfer(devBalance);
}
if (taxPercentages.marketing > 0) {
    _taxWallets.marketing.transfer(marketingBalance);
}
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L315,586,600
Status	Unresolved

Description

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

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

```
function setSwapSettings(
    uint256 threshold,
    uint256 thresholdDivisor,
    uint256 time
) external onlyOwner {
    require(threshold > 0);
    require(thresholdDivisor%10 == 0 && thresholdDivisor > 0);
    swapThreshold = (_tSupply * threshold) / thresholdDivisor;
    contractSwapTimer = time;
}

if (contractTokenBalance >= swapThreshold) {
    contractTokenBalance = swapThreshold;
    contractSwap(contractTokenBalance);
    lastSwap = block.timestamp;
}
```

Recommendation

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

UFP - Unused Function Parameter

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L499
Status	Unresolved

Description

The function `_finalizeTransfer` has an unused function parameter `bool other`. This parameter is declared but not utilized anywhere within the function's body. Unused parameters can lead to confusion about the function's intended behavior and may suggest that there is incomplete implementation or redundant code.

```
function _finalizeTransfer(
    address from,
    address to,
    uint256 amount,
    bool takeFee,
    bool buy,
    bool sell,
    bool other
) internal returns (bool) {

    _tokenOwned[from] -= amount;
    uint256 amountReceived = (takeFee)
        ? takeTax(from, buy, sell, amount)
        : amount;
    _tokenOwned[to] += amountReceived;

    emit Transfer(from, to, amountReceived);
    return true;
}
```

Recommendation

It is recommended to review the purpose of the `bool other` parameter to determine if it was intended for use or if it is redundant. If the parameter is not needed, removing this parameter is advisable to enhance the code's clarity.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L17,93
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.

```
uint256 internal _tSupply = 100000000000000000000000000000
uint8 internal _decimals = 18
```

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/token_revised_.sol#L16,31,34,41,48
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.

```
uint256 constant taxDivisor = 10000

Fees public _taxRates =
        Fees({buyFee: 500, sellFee: 1000, transferFee: 0})

TaxPercentages public _taxPercentages =
        TaxPercentages({marketing: 70, dev: 30})
TaxWallets public _taxWallets
bool public _hasLiquidityBeenAdded = false
```

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	contracts/token_revised_.sol#L25
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.

```
mapping(address => bool) internal _isExcluded
```

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	contracts/token_revised_.sol#L311,322,426
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.

```
_maxTxAmount = (_tSupply * percent) / divisor
swapThreshold = (_tSupply * threshold) / thresholdDivisor
_maxWalletSize = (_tSupply * percent) / divisor
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L298
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(marketing>=0 && marketing<=100,'Total should be 100')
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L265
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.

```
enabled == false
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/token_revised_.sol#L134
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.

```
lpPair = pair
```

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.

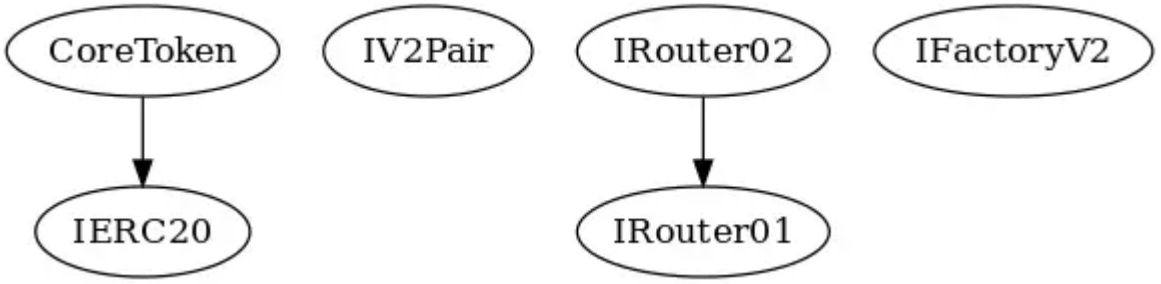
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CoreToken	Implementation	IERC20		
		Public	Payable	-
	balanceOf	Public		-
	confirmLP	Public	✓	onlyOwner
	setPairAddress	Public	✓	onlyOwner
	preInitializeTransfer	Public	✓	onlyOwner
	transferOwner	External	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	transfer	Public	✓	-
	approve	Public	✓	-
	approveContractContingency	Public	✓	onlyOwner
	transferFrom	External	✓	-
	setNewRouter	Public	✓	onlyOwner

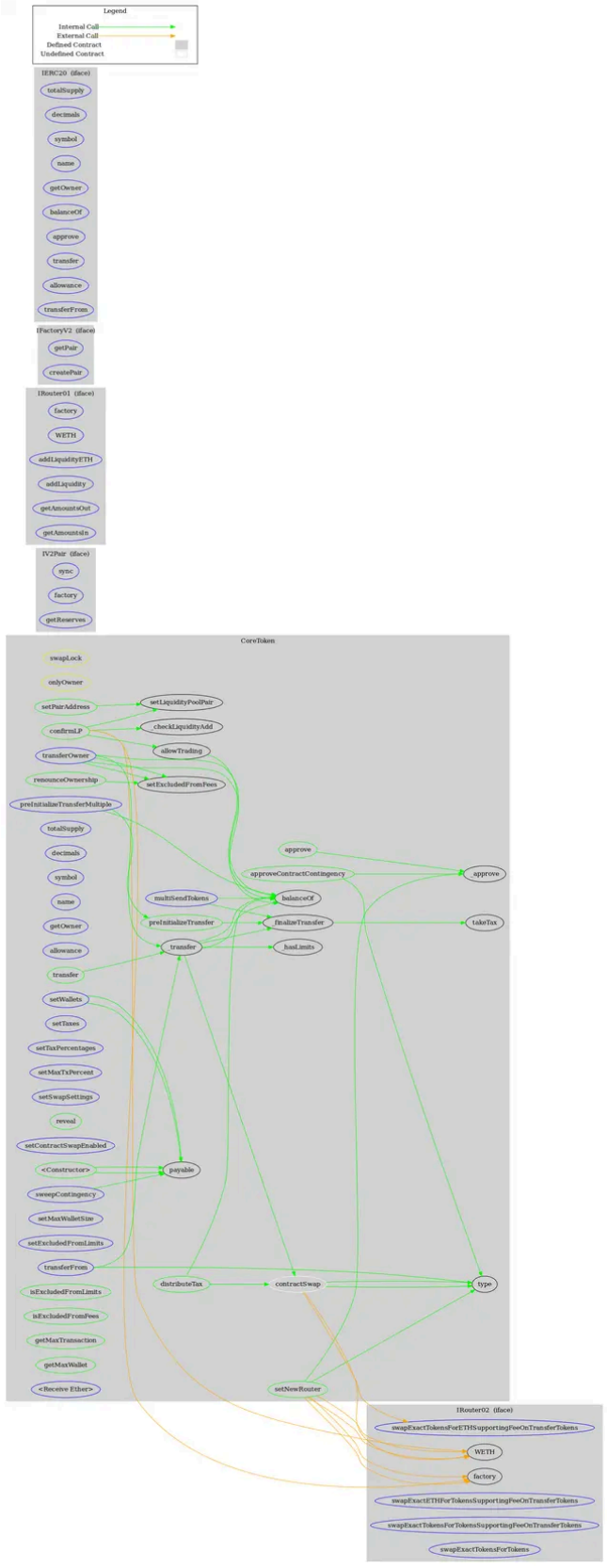
	setLiquidityPoolPair	Public	✓	onlyOwner
	setTaxes	External	✓	onlyOwner
	setTaxPercentages	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	setSwapSettings	External	✓	onlyOwner
	multiSendTokens	External	✓	-
	reveal	Public	✓	onlyOwner
	setContractSwapEnabled	External	✓	onlyOwner
	setWallets	External	✓	onlyOwner
	preInitializeTransferMultiple	External	✓	onlyOwner
	allowTrading	Internal	✓	
	takeTax	Internal	✓	
	setMaxWalletSize	External	✓	onlyOwner
	setExcludedFromLimits	External	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	contractSwap	Internal	✓	swapLock
	isExcludedFromLimits	Public		-
	isExcludedFromFees	Public		-
	setExcludedFromFees	Public	✓	onlyOwner
	getMaxTransaction	Public		-
	getMaxWallet	Public		-
	_finalizeTransfer	Internal	✓	
	_hasLimits	Internal		

	_transfer	Internal	✓	
	distributeTax	Public	✓	onlyOwner
	_approve	Internal	✓	
	_checkLiquidityAdd	Internal	✓	
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
approve
transfer
allowance
transferFrom
IFactoryV2 (iface)
getPair
createPair
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
getAmountsOut
getAmountsIn
IV2Pair (iface)
sync
factory
getReserves
CoreToken
swapLock
onlyOwner
setPairAddress
setLiquidityPoolPair
confirmLP
checkLiquidityAdd
transferOwner
allowTrading
renounceOwnership
setExcludedFromFees
preInitializeTransferMultiple
totalSupply
decimals
symbol
name
getOwner
allowance
transfer
approve
approveContractContingency
approve
multiSendTokens
balanceOf
preInitializeTransfer
finalizeTransfer
takeTax
transfer
hasLimits
setWallets
setTaxes
setTaxPercentages
setMaxTxPercent
setSwapSettings
reveal
setContractSwapEnabled
<Constructor>
payable
sweepContingency
setMaxWalletSize
setExcludedFromLimits
transferFrom
distributeTax
contractSwap
type
isExcludedFromLimits
isExcludedFromFees
getMaxTransaction
getMaxWallet
<Receive Ether>
setNewRouter
IRouter02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens

Summary

Oracle AI 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. A multi-wallet signing pattern will provide security against potential hacks. There is also a limit of max 25% 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