Cyberscope

A TAC Security Company

Audit Report

OWO

August 2025

Network	ETH
Address	0x806a72273B961145cf5c5f040Ad1FCd112b3f11A
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
●	CO	Code Optimization	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	MTEE	Missing Transfer Event Emission	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PGA	Potential Griefing Attack	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RF	Redundant Functionality	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	UAR	Unexcluded Address Restrictions	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope’s smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	OWO
Compiler Version	v0.8.23+commit.f704f362
Optimization	200 runs
Explorer	https://etherscan.io/address/0x806a72273b961145cf5c5f040ad1fcd112b3f11a
Address	0x806a72273b961145cf5c5f040ad1fcd112b3f11a
Network	ETH
Symbol	OWO
Decimals	9
Total Supply	420,690,000,000,000

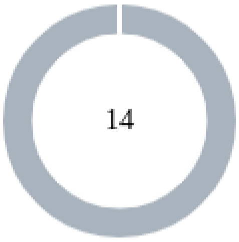
Audit Updates

Initial Audit	15 Aug 2025

Source Files

Filename	SHA256
OWO.sol	e3432ae29f7d8b033ae2484850889daad3969015fc97028ab96e5c5c901d0e28

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	14

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

CO - Code Optimization

Criticality	Minor / Informative
Location	OWO.sol#L227
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations. Specifically, when `_buyCount` is `0`, the condition `_buyCount > _reduceBuyTaxAt` will always evaluate to `false`, resulting in the use of `_initialBuyTax`. This makes the ternary operation redundant in this context.

```
if(_buyCount==0){
taxAmount =
amount.mul((_buyCount>_reduceBuyTaxAt)?_finalBuyTax:_initialBuyTax).div(100);
}
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	OWO.sol#L163
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_taxWallet
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	OWO.sol#L225,333,334,342
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to 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(!bots[from] && !bots[to])
require(_msgSender()==_taxWallet)
require(_newFee<=_finalBuyTax && _newFee<=_finalSellTax)
```

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.

MTEE - Missing Transfer Event Emission

Criticality	Minor / Informative
Location	OWO.sol#L59
Status	Unresolved

Description

The contract does not emit an event when portions of the main amount are transferred during the transfer process. This lack of event emission results in decreased transparency and traceability regarding the flow of tokens, and hinders the ability of decentralized applications (dApps), such as blockchain explorers, to accurately track and analyze these transactions.

```
_balances[_msgSender()] = _tTotal;
```

Recommendation

It is advisable to incorporate the emission of detailed event logs following each asset transfer. These logs should encapsulate key transaction details, including the identities of the sender and receiver, and the quantity of assets transferred. Implementing this practice will enhance the reliability and transparency of transaction tracking systems, ensuring accurate data availability for ecosystem participants.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	OWO.sol#L226
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
function _transfer(address from, address to, uint256 amount) private
{
  ...
  require(amount > 0, "Transfer amount must be greater than zero");
  ...
  }
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

PGA - Potential Griefing Attack

Criticality	Minor / Informative
Location	OWO.sol#L254
Status	Unresolved

Description

The contract includes functionality designed to enforce specific conditions on transactions. However, this design is vulnerable to griefing attacks, where malicious actors can exploit the contract's logic to interfere with legitimate user operations.

In this case, the contract enforces transactional limits based on on-chain activity. A third party could strategically interact with the contract's state to disrupt normal user operations, resulting in failed transactions or unintended behavior.

Such griefing attacks could undermine the contract's usability and obstruct legitimate user operations.

```
require(sellCount < 3, "Only 3 sells per block!");
```

Recommendation

The team is advised to review the transfer mechanism to ensure that all legitimate operations are processed as intended. This will help maintain the integrity of user activities and strengthen trust in the system.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	OWO.sol#L279
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 lockTheSwap {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        _approve(address(this), address(uniswapV2Router), tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            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.

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	OWO.sol#L258,304
Status	Unresolved

Description

The contract sends funds to a `_taxWallet` as part of the transfer flow. This address 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.

```
function sendETHToFee(uint256 amount) private {
_taxWallet.transfer(amount);
}
```

Recommendation

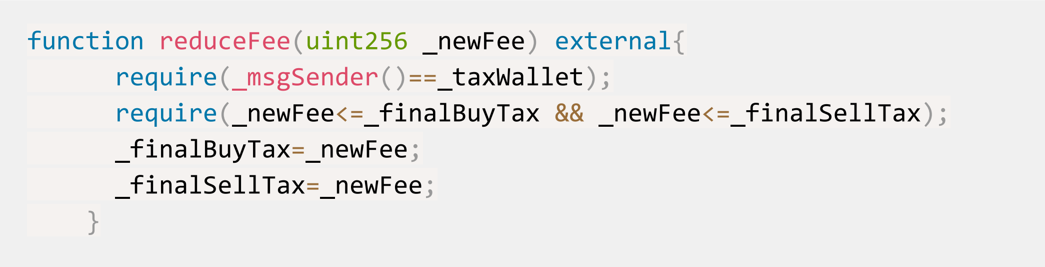
The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

RF - Redundant Functionality

Criticality	Minor / Informative
Location	OWO.sol#L336
Status	Unresolved

Description

The contract includes a `reduceFee` function intended to lower the values of `_finalBuyTax` and `_finalSellTax` . However, both of these variables are initially set to zero. As a result, any attempt to reduce them further is ineffective, rendering the function redundant.



```
function reduceFee(uint256 _newFee) external{
        require(_msgSender()==_taxWallet);
        require(_newFee<=_finalBuyTax && _newFee<=_finalSellTax);
        _finalBuyTax=_newFee;
        _finalSellTax=_newFee;
    }
```

Recommendation

Removing redundant functions improves code readability and reduces complexity, making the codebase easier to maintain and understand. This also helps optimize performance by eliminating unnecessary operations.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	OWO.sol#L279
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        _approve(address(this), address(uniswapV2Router), tokenAmount);

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	OWO.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/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	OWO.sol#L19,60
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```
_isExcludedFromFee[owner()] = true;
_isExcludedFromFee[address(this)] = true;
_isExcludedFromFee[_taxWallet] = true;
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	OWO.sol#L128,129,132,133,134,144,145
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 private _initialBuyTax=25
uint256 private _initialSellTax=25
uint256 private _reduceBuyTaxAt=25
uint256 private _reduceSellTaxAt=25
uint256 private _preventSwapBefore=26
uint256 public _taxSwapThreshold= 4206900000000 * 10**_decimals
uint256 public _maxTaxSwap= 4206900000000 * 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	OWO.sol#L109,138,139,140,141,142,143,144,145,332
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 WETH() external pure returns (address);
uint8 private constant _decimals = 9
uint256 private constant _tTotal = 420690000000000 * 10**_decimals
string private constant _name = unicode"Owo"
string private constant _symbol = unicode"OWO"
uint256 public _maxTxAmount = 8206900000000 * 10**_decimals
uint256 public _maxWalletSize = 8206900000000 * 10**_decimals
uint256 public _taxSwapThreshold= 4206900000000 * 10**_decimals
uint256 public _maxTaxSwap= 4206900000000 * 10**_decimals
uint256 _newFee
```

Recommendation

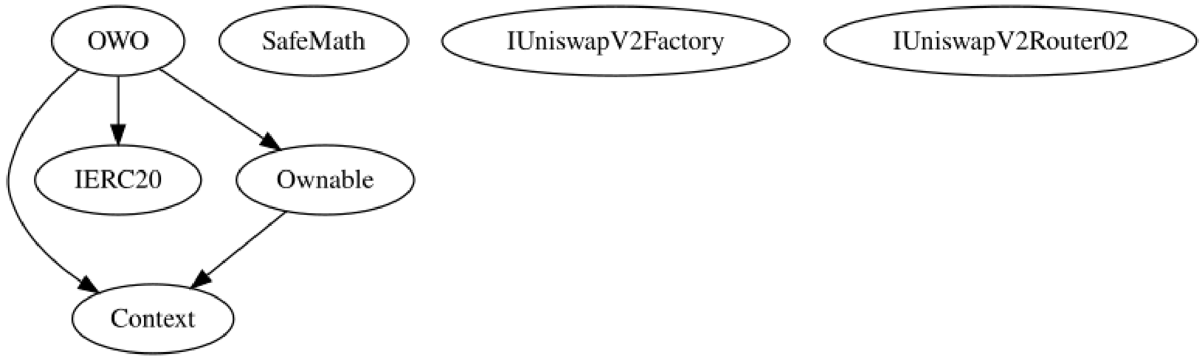
By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

Functions Analysis

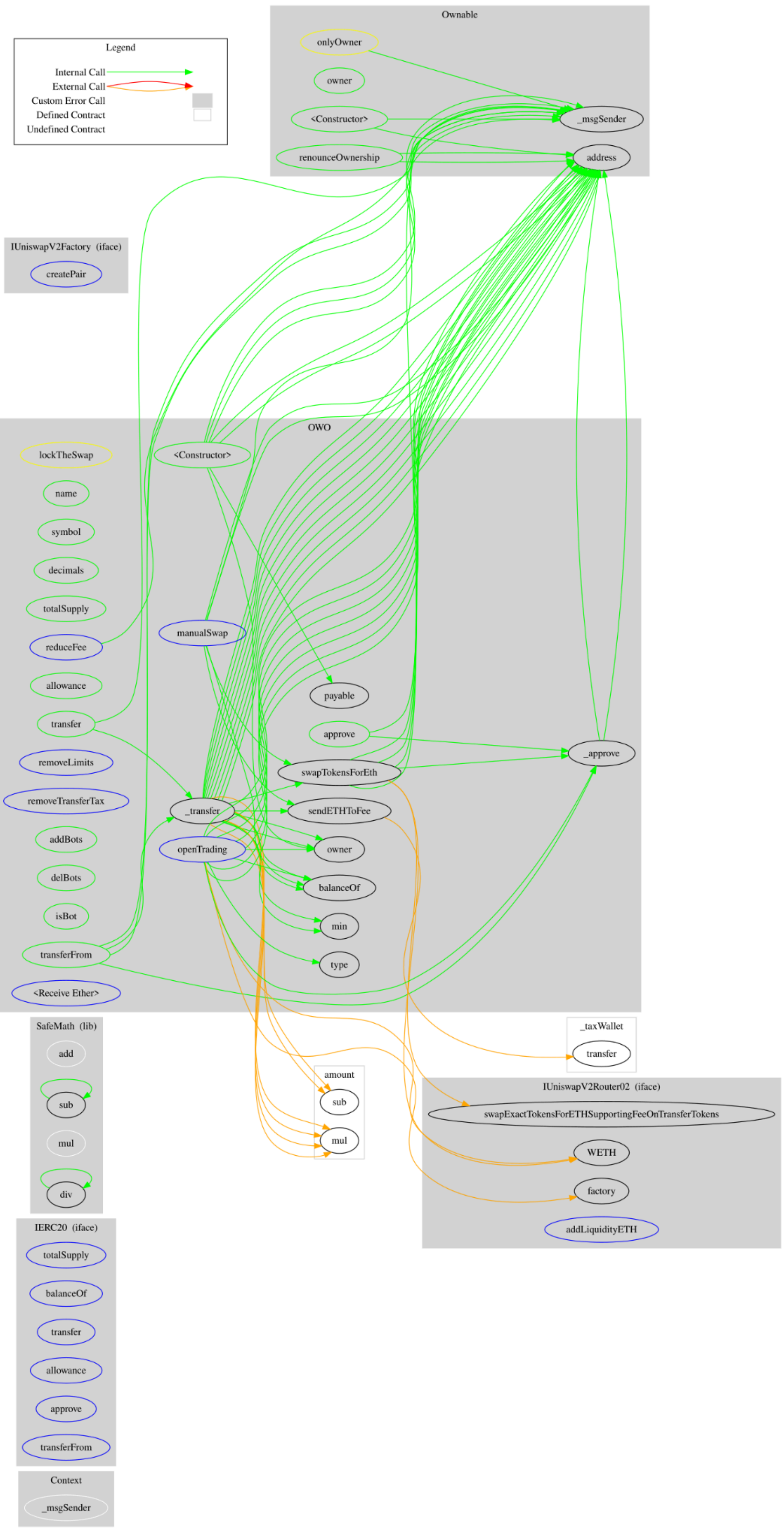
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
OWO	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	min	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	removeLimits	External	✓	onlyOwner
	removeTransferTax	External	✓	onlyOwner
	sendETHToFee	Private	✓	
	addBots	Public	✓	onlyOwner

	delBots	Public	✓	onlyOwner
	isBot	Public		-
	openTrading	External	✓	onlyOwner
	reduceFee	External	✓	-
		External	Payable	-
	manualSwap	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
address
IUniswapV2Factory (iface)
createPair
OWO
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
reduceFee
manualSwap
allowance
payable
transfer
approve
_approve
removeLimits
swapTokensForEth
removeTransferTax
_transfer
sendETHToFee
addBots
openTrading
owner
delBots
balanceOf
isBot
min
transferFrom
type
<Receive Ether>
SafeMath (lib)
add
sub
mul
div
_taxWallet
transfer
amount
sub
mul
IUniswapV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
addLiquidityETH
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

OWO contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. OWO is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

https://etherscan.io/tx/0xc736c0dbf4802e076aeb1c1f6f31e1e9e50d5ff7c9bca8015f15e40c48200c66

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 TAC 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.

A ***TAC Security*** *Company*

The Cyberscope team

cyberscope.io