Cyberscope

Audit Report

The Worldwide Token

Aug 2023

SHA256 77600b7e566a2b3a533a5cc7ab4428eee29c5ab440df7f673cb4395b5f4c11c1

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	MWV	Multisig Wallet Vulnerability	Unresolved
Medium	MWR	Multisig Wallet Requirements	Unresolved
Medium	MFI	Misleading Function Implementation	Unresolved
Minor / Informative	RRF	Redundant Router Functionality	Unresolved
Minor / Informative	MEM	Misleading Error Messages	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	RCS	Redundant Conditional Statement	Unresolved
Minor / Informative	HLI	Holders List Inconsistency	Unresolved
Minor / Informative	RSK	Redundant Storage Keyword	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L02	State Variables could be Declared Constant	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Review

Contract Name	WorldToken
Testing Deploy	https://testnet.bscscan.com/address/0xf9ce5e21acfecc5b099acf4299891f4daa2a67b2
Symbol	WORLD
Decimals	18

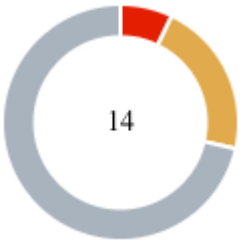
Audit Updates

Initial Audit	13 Jul 2023 https://github.com/cyberscope-io/audits/blob/main/1-world/v1/audit.pdf
Corrected Phase 2	17 Jul 2023 https://github.com/cyberscope-io/audits/blob/main/1-world/v2/audit.pdf
Corrected Phase 3	21 Jul 2023 https://github.com/cyberscope-io/audits/blob/main/1-world/v3/audit.pdf
Corrected Phase 4	24 Jul 2023 https://github.com/cyberscope-io/audits/blob/main/1-world/v4/audit.pdf
Corrected Phase 5	09 Aug 2023

Source Files

Filename	SHA256
contracts/WorldToken.sol	77600b7e566a2b3a533a5cc7ab4428eee29c5ab440df7f673cb4395b5f4c11c1

Findings Breakdown



Critical	1
Medium	3
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	3	0	0	0
Minor / Informative	10	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Medium
Location	contracts/WorldToken.sol#L501
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 `changeTax` function with a high percentage value.

```
function contractChangeTax(uint256 newTax) public onlyContract returns
(bool) {
    require (newTax <= 40, "no more than 30%");
    if (investorProtection == 0) {
        require (newTax >= 5 && newTax <= 15, "tax must be between 5% and
15%");
    }
    tax = newTax;
    return true;
}
```

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.

MWV - Multisig Wallet Vulnerability

Criticality	Critical
Location	contracts/WorldToken.sol
Status	Unresolved

Description

The implementation of the multisig wallet functionality in the contract grants any authorized owner the authority to add or remove authorized wallets by utilizing the `addOwner` and `removeOwner` methods, respectively. This is possible due to the mutability of `numConfirmationsRequired` variable, as described in detail at the MWR section. However, this approach poses a significant security vulnerability. If any of the authorized owners' wallet is compromised, an attacker could exploit this situation by removing all other authorized wallets and executing any proposal without legitimate authorization.

Recommendation

The team should carefully manage the private keys of the owner’s and authorized owners' accounts.

MWR - Multisig Wallet Requirements

Criticality	Medium
Location	contracts/WorldToken.sol#L393,428
Status	Unresolved

Description

The implementation of the multisig wallet functionality in the contract has a severe flaw that compromises its intended security. Currently, the `numConfirmationsRequired` variable can be modified through the multisig wallet functionality, which sets the minimum number of confirmations needed for executing admin functions. The required confirmations can be set to a minimum of 1, effectively bypassing the whole purpose of having a multisig mechanism. Additionally, the `numConfirmationsRequired` variable is initialized to 1.

As a result of this vulnerability, the multisig wallet fails to provide the expected security benefits. Any authorized owner could potentially call admin functions without needing sufficient approvals from other owners, rendering the concept of multiple authorizations meaningless.

```
function contractChangeRequirement(uint256 _required) public onlyContract
{
    require(_required > 0 && _required <= owners.length, "MultisigWallet:
invalid requirement");
    numConfirmationsRequired = _required;

    emit RequirementChange(_required);
}

function changeRequirement(uint256 _required) public onlyOwner returns
(uint) {
    bytes memory data =
abi.encodeWithSignature("contractChangeRequirement(uint256)", _required);
    return submitTransaction(data);
}

if(transactions[_txIndex].numConfirmations == numConfirmationsRequired){
    executeTransaction(_txIndex);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that the contract provides the expected security benefits. A recommended approach would be to either modify the `numConfirmationsRequired` to be a constant variable and greater or equal than 2, or modify the `require(_required > 0 && _required <= owners.length, "MultisigWallet: invalid requirement");` code segment so that the `_required` variable is at least 2.

This significantly enhances the security of the multisig wallet mechanism by enforcing a reasonable minimum number of confirmations for executing admin functions. It also prevents the multisig wallet from being compromised by a single authorized owner and ensures that critical actions require a collaborative effort from multiple authorized parties.

MFI - Misleading Function Implementation

Criticality	Medium
Location	contracts/WorldToken.sol#L1154
Status	Unresolved

Description

The `eventParticipate` function provides a fee-free transfer mechanism for tokens. Given the presence of both options within the contract, users might opt to utilize this function for their token transfers instead of the designated `transfer` or `transferFrom` functions. Consequently, the fee mechanism could potentially lose its relevance.

```
function eventParticipate(address to, uint256 amount) external {
    _transfer_simple(msg.sender, to, amount);
}
```

Recommendation

The team is advised to take these segments into consideration and either remove them if they serve no purpose or rewrite them so that they follow the team's business logic. For instance, if the function was not intended to be publicly accessible then the team should add proper guards to avoid non authorized calls.

RRF - Redundant Router Functionality

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L324
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 initializes a router object at the constructor. However, neither the `router` nor the `pair` variables are used in a meaningful way by the contract. As a result, their declaration is redundant.

```
router = IDEXRouter(0x9Ac64Cc6e4415144C455BD8E4837Fea55603e5c3);
pair = IDEXFactory(router.factory()).createPair(WBNB, address(this));
```

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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L502
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 (newTax <= 40, "no more than 30%");
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L783,812
Status	Unresolved

Description

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

The `claim` and `getClaimRewardsCount` function share some code segments which are identical.

```
uint256 total = 0;
for (uint256 i = 0; i < claimList.length; i++) {
    total += balances[claimList[i]];
}
uint256 myBalance = balances[_msgSender()];
uint256 percent = (myBalance * 10000) / total;
uint256 amount = (_balances[worldPoolWallet] * percent) / 10000;
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L798
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 `findIndex(_msgSender(), claimList)` is checked twice in the `claim` function. As a result, the following code segments are redundant.

```
int256 index = findIndex(_msgSender(), claimList);
require(index >= 0 && uint256(index) < claimList.length, "Invalid index");
```

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.

HLI - Holders List Inconsistency

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L931
Status	Unresolved

Description

The current implementation of the contract lacks consistency between the `holders` list and the `dexAddressList`, which can lead to confusion and unexpected behavior. The contract adds a user to the holders list only if the user is not already present in the dexAddressList. However, when the owner adds or removes a user from the dexAddressList using the `addToDexAddressList` and `removeFromDexAddressList` functions, respectively, the contract does not update the holders list accordingly.

This inconsistency creates a discrepancy between the two lists, as a user can exist in one list but not the other, depending on the order of operations or the actions taken by the contract owner.

```
if(findIndex(recipient, dexAddressList) == -1){
    if (findIndex(recipient, holders) == -1) {
        holders.push(recipient);
    }
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them to ensure a more coherent system, by updating the holders list whenever a user is added to or removed from the dexAddressList.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L479
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.

```
Transaction storage transaction
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L273,324,325
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.

```
_totalSupply
router
pair
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L149
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 WBNB = 0xae13d989daC2f0dEbFf460aC112a837C89BAa7cd
```

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/WorldToken.sol#L101,149,222,224,225,238,239,386,393,399,421,434,448,467,1062,1135,1149
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);
address WBNB = 0xae13d989daC2f0dEbFf460aC112a837C89BAa7cd
uint8 constant private _decimals = 18
string constant private _symbol = "WORLD"
string constant private _name = "WORLD"
event allowListTransferTaxEvent(address addr);
event addToDexAddressListEvent(address addr);
uint256 _required
bytes memory _data
uint _txIndex
...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L973
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 takeFee(address sender, uint256 amount) internal returns
(uint256) {
    uint256 feeAmount = amount/10;


    _balances[address(this)] += feeAmount;
    emit Transfer(sender, address(this), feeAmount);

    return amount - feeAmount;
}
```

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/WorldToken.sol#L788,789,791,817,818,832,833,838,839,844,845,850,851,855,856,857,862,866,867,868,872,873,874,880,881,882,886,887,888,892,893,894,898,899,900,907,908,909,915,916,917,921,922,923
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.

```
uint256 percent = (myBalance * 10000) / total;
uint256 amount = (_balances[worldPoolWallet] * percent) / 10000;
uint256 amountToTeam = amount * 300 / 10000;
...
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-

	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupporting FeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFee OnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFee OnTransferTokens	External	✓	-
WorldToken	Implementation	IBEP20		
		Public	✓	-
		External	Payable	-
	_msgSender	Internal		
	_msgData	Internal		
	isOwner	Public		-
	getOwner	External		-
	contractAddOwner	Public	✓	onlyContract
	addOwner	Public	✓	onlyOwner
	contractRemoveOwner	Public	✓	onlyContract
	removeOwner	Public	✓	onlyOwner
	contractChangeRequirement	Public	✓	onlyContract
	changeRequirement	Public	✓	onlyOwner
	submitTransaction	Public	✓	onlyOwner
	confirmTransaction	Public	✓	onlyOwner txExists notExecuted notConfirmed

	executeTransaction	Internal	✓	onlyOwner txExists notExecuted
	revokeConfirmation	Public	✓	onlyOwner txExists notExecuted
	getOwners	Public		-
	getTransactionCount	Public		-
	getTransaction	Public		-
	contractChangeInvestorProtection	Public	✓	onlyContract
	changeInvestorProtection	Public	✓	onlyOwner
	contractChangeTax	Public	✓	onlyContract
	changeTax	Public	✓	onlyOwner
	contractChangeBuyTax	Public	✓	onlyContract
	changeBuyTax	Public	✓	onlyOwner
	contractChangeTransferTax	Public	✓	onlyContract
	changeTransferTax	Public	✓	onlyOwner
	contractSetTaxFree	Public	✓	onlyContract
	setTaxFree	Public	✓	onlyOwner
	contractChangeApy	Public	✓	onlyContract
	changeApy	Public	✓	onlyOwner
	contractMintProfit	Public	✓	onlyContract
	mintProfit	Public	✓	onlyOwner
	contractAddToTransferAllowList	Public	✓	onlyContract
	addToTransferAllowList	Public	✓	onlyOwner
	contractAddToBuyAllowList	Public	✓	onlyContract

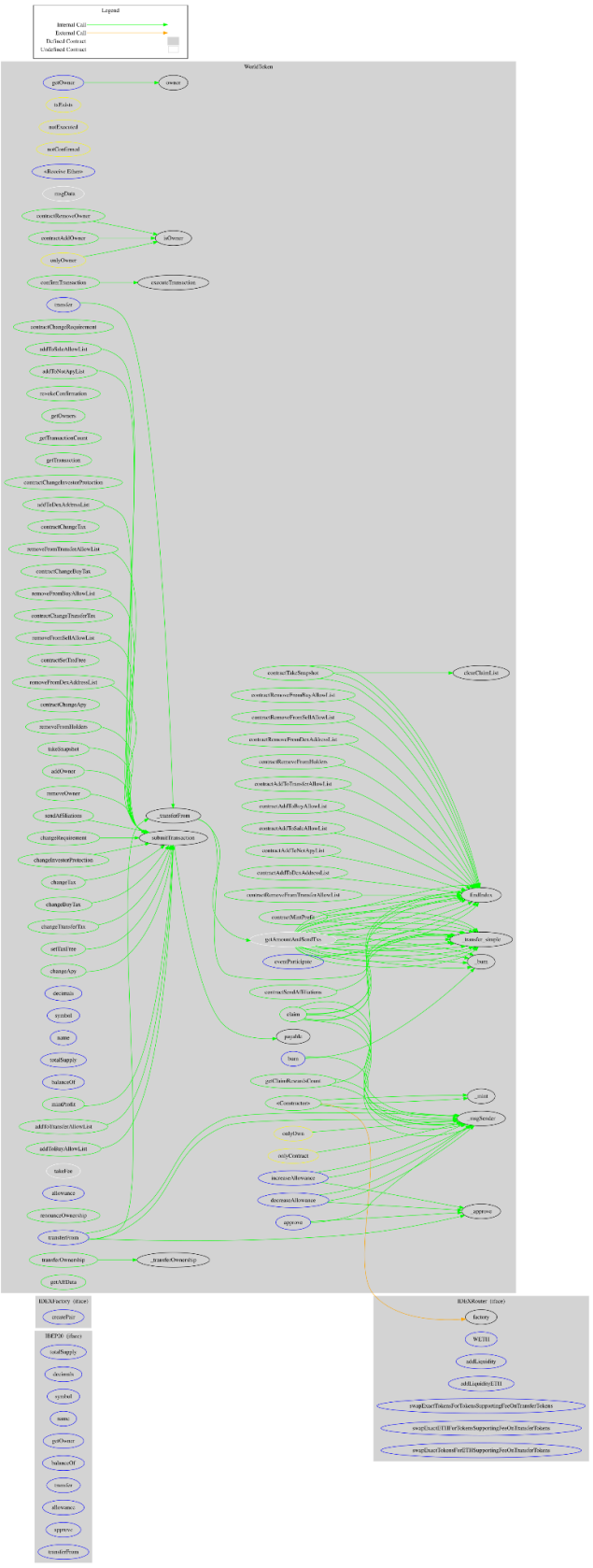
	addToBuyAllowList	Public	✓	onlyOwner
	contractAddToSaleAllowList	Public	✓	onlyContract
	addToSaleAllowList	Public	✓	onlyOwner
	contractAddToNotApyList	Public	✓	onlyContract
	addToNotApyList	Public	✓	onlyOwner
	contractAddToDexAddressList	Public	✓	onlyContract
	addToDexAddressList	Public	✓	onlyOwner
	findIndex	Internal		
	contractRemoveFromTransferAllowList	Public	✓	onlyContract
	removeFromTransferAllowList	Public	✓	onlyOwner
	contractRemoveFromBuyAllowList	Public	✓	onlyContract
	removeFromBuyAllowList	Public	✓	onlyOwner
	contractRemoveFromSellAllowList	Public	✓	onlyContract
	removeFromSellAllowList	Public	✓	onlyOwner
	contractRemoveFromDexAddressList	Public	✓	onlyContract
	removeFromDexAddressList	Public	✓	onlyOwner
	contractRemoveFromHolders	Public	✓	onlyContract
	removeFromHolders	Public	✓	onlyOwner
	decimals	External		-
	symbol	External		-
	name	External		-
	totalSupply	External		-
	balanceOf	External		-

	clearClaimList	Internal	✓	
	contractTakeSnapshot	Public	✓	onlyContract
	takeSnapshot	Public	✓	onlyOwner
	claim	Public	✓	-
	getClaimRewardsCount	Public		-
	getAmountAndSendTxs	Internal	✓	
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	takeFee	Internal	✓	
	allowance	External		-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	_transfer_simple	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	approve	External	✓	-
	_approve	Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwn
	transferOwnership	Public	✓	onlyOwn
	_transferOwnership	Internal	✓	
	contractSendAffiliations	Public	✓	onlyContract

	sendAffiliations	Public	✓	onlyOwner
	eventParticipate	External	✓	-
	getAffData	Public		-
	burn	External	✓	-

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
WorldToken
getOwner
owner
isFalse
notExecuted
notConfirmed
receive Ether
msgData
contractRemoveOwner
contractAddOwner
isOwner
onlyOwner
confirmTransaction
executeTransaction
transfer
contractChangeRequirement
addToSaleAllowList
addToNotApplist
revokeConfirmation
getOwners
getTransactionCount
getTransaction
contractChangeInvestorProtection
addToDexAddressList
contractChangeTax
removeFromTransferAllowList
contractChangeBuyTax
removeFromBuyAllowList
contractChangeTransferTax
removeFromSellAllowList
contractSetTxFree
removeFromDexAddressList
contractChangeApy
removeFromHolders
takeSnapshot
addOwner
removeOwner
setAffiliations
changeRequirement
changeInvestorProtection
changeTax
changeBuyTax
changeTransferTax
setTaxFree
changeApy
decimals
symbol
name
totalSupply
balanceOf
mintProfit
addToTransferAllowList
addToBuyAllowList
takeFee
allowance
renounceOwnership
transferFrom
transferOwnership
getAllData
_transferFrom
submitTransaction
contractTakeSnapshot
contractRemoveFromBuyAllowList
contractRemoveFromSellAllowList
contractRemoveFromDexAddressList
contractRemoveFromHolders
contractAddToTransferAllowList
contractAddToBuyAllowList
contractAddToSaleAllowList
contractAddToNotApplist
contractAddToDexAddressList
contractRemoveFromTransferAllowList
contractMintProfit
getAccountAndSendTax
eventParticipate
contractSendAllStations
claim
payable
burn
getClaimRewardsCount
clearChainList
_mint
msgSender
approve
IDEXFactory (iface)
createPair
IDEXPair (iface)
IDEXRouter (iface)
factory
WETH
addLiquidity
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens

Summary

The Worldwide Token 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 manipulating the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 40% sell, 10% buy, and 5% transfer fees.

Team Update

The team implemented a custom multi-sig wallet functionality, where the authorized addresses can confirm or revoke an admin transaction. At the time of this audit, the minimum required signatures for a transaction to be executed is one.

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