Cyberscope

# Audit Report

# The Worldwide Token

July 2023

SHA256 d7af0533a835778e9212b58f43fcb985a985bbdd1cdd712e055483d1c273b0e1

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 |
|---|---|---|---|
| ● | MWV | Multisig Wallet Vulnerability | Unresolved |
| ● | MWR | Multisig Wallet Requirements | Unresolved |
| ● | HLI | Holders List Inconsistency | Unresolved |
| ● | CR | Code Repetition | Unresolved |
| ● | RCS | Redundant Conditional Statement | Unresolved |
| ● | RSK | Redundant Storage Keyword | Unresolved |
| ● | IDI | Immutable Declaration Improvement | Unresolved |
| ● | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ● | L13 | Divide before Multiply Operation | Unresolved |
| ● | L16 | Validate Variable Setters | Unresolved |

# Table of Contents

# Review

| | |
|---|---|
| **Contract Name** | BEP20Token |
| **Testing Deploy** | https://testnet.bscscan.com/address/0xf75f08893b64f05e2aae5a5b1d2208b3dc7fe1f4 |
| **Symbol** | WORLD |
| **Decimals** | 4 |

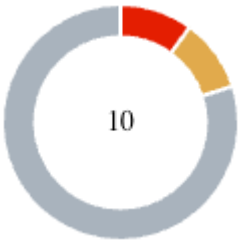
## Audit Updates

| | |
|---|---|
| **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 |

## Source Files

| | |
|---|---|
| **Filename** | SHA256 |
| **contracts/WorldToken.sol** | d7af0533a835778e9212b58f43fcb985a985bbdd1cdd712e055483d1c273b0e1 |

# Findings Breakdown




| | |
|---|---|
| Critical | 1 |
| Medium | 1 |
| Minor / Informative | 8 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 1 | 0 | 0 | 0 |
| Medium | 1 | 0 | 0 | 0 |
| Minor / Informative | 8 | 0 | 0 | 0 |

## 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. 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#L261,298 |
| **Status** | Unresolved |

## Description

The implementation of the multisig wallet functionality in the contract has a severe flaw that compromises its intended security. Currently, any authorized owner has unrestricted authority to modify the `numConfirmationsRequired` variable, which sets the minimum number of confirmations needed for executing admin functions. This means that an authorized owner could lower the required confirmations to just 1, effectively bypassing the whole purpose of having a multisig mechanism.

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 changeRequirement(uint256 _required) public onlyOwner {
    require(_required > 0 && _required <= owners.length, "MultisigWallet: invalid
requirement");
    numConfirmationsRequired = _required;

    emit RequirementChange(_required);
}

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 to 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.

## HLI - Holders List Inconsistency

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/WorldToken.sol#L724 |
| **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.

## CR - Code Repetition

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/WorldToken.sol#L364,381,397,580,605 |
| **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 functionality between the three tax functions can be reused.

```
if (tax > 15) {
    tax = 15;
}
if (tax < 5) {
    tax = 5;
}
```

The `claim` and `getClaimRewardsCount` functions 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#L591 |
| **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.

## RSK - Redundant Storage Keyword

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/WorldToken.sol#L349 |
| **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#L197 |
| **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
```

### 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.

## L04 - Conformance to Solidity Naming Conventions

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/WorldToken.sol#L161,163,164,176,177,261,269,291,304,318,337,839,850 |
| **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.

```
uint8 constant private _decimals = 4
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
address[] calldata _walletAddresses
uint256[] calldata _amounts
```

## 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.

## L13 - Divide before Multiply Operation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/WorldToken.sol#L585,586,610,611,624,625,629,630,634,635,639,640,644,645,650,651,652,656,657,658,662,663,664,668,669,670,676,677,678,682,683,684,688,689,690,694,695,696,700,701,702,708,709,710,714,715,716 |
| **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 percent = 7250 * (tax * 100) / 10000;
uint256 convertWorldAmount = percent * amount / 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.

## L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | contracts/WorldToken.sol#L196 |
| **Status** | Unresolved |

### Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
_owner = msgSender
```

### 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

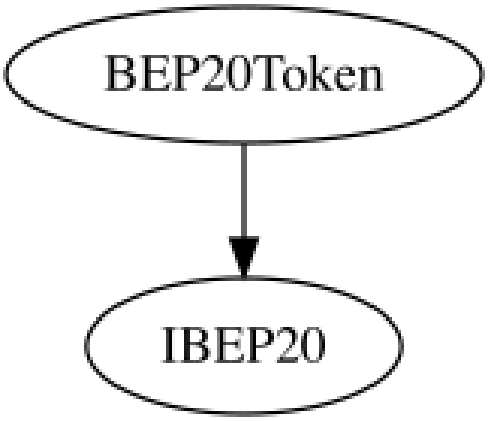
| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **IBEP20** | Interface | | | |
| | totalSupply | External | | - |
| | decimals | External | | - |
| | symbol | External | | - |
| | name | External | | - |
| | getOwner | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **BEP20Token** | Implementation | IBEP20 | | |
| | | Public | ✓ | - |
| | | External | Payable | - |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | isOwner | Public | | - |

| | | | | |
|---|---|---|---|---|
| | getOwner | External | | - |
| | addOwner | Public | ✓ | onlyOwner |
| | removeOwner | Public | ✓ | onlyOwner |
| | changeRequirement | Public | ✓ | onlyOwner |
| | submitTransaction | Public | ✓ | onlyOwner |
| | confirmTransaction | Public | ✓ | onlyOwner<br>txExists<br>notExecuted<br>notConfirmed |
| | executeTransaction | Internal | ✓ | onlyOwner<br>txExists<br>notExecuted |
| | revokeConfirmation | Public | ✓ | onlyOwner<br>txExists<br>notExecuted |
| | getOwners | Public | | - |
| | getTransactionCount | Public | | - |
| | getTransaction | Public | | - |
| | 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 |

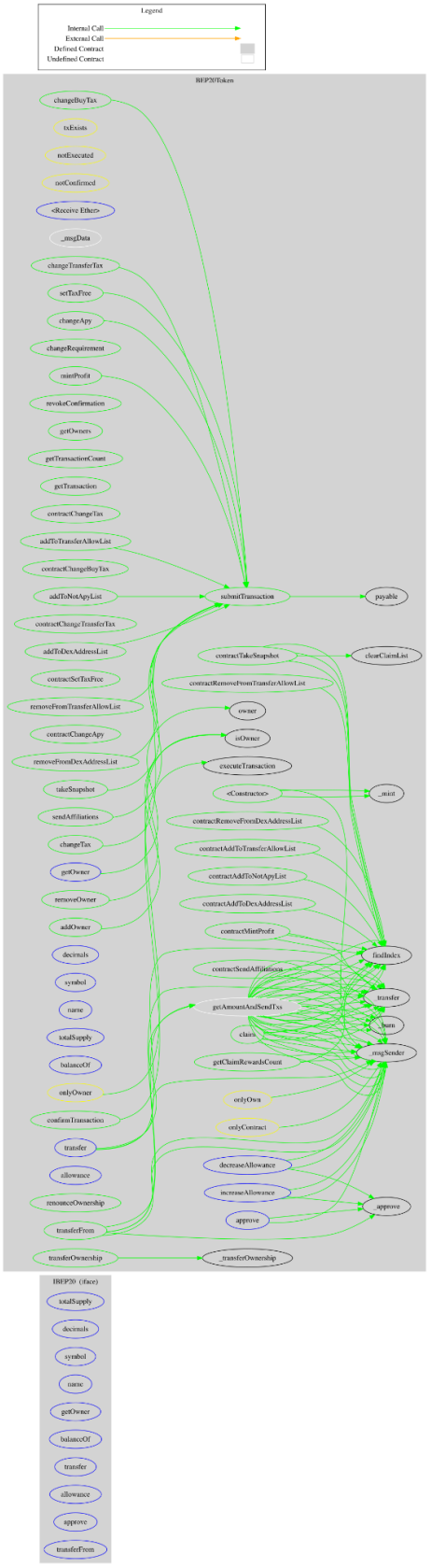
| | | | | |
|---|---|---|---|---|
| | contractMintProfit | Public | ✓ | onlyContract |
| | mintProfit | Public | ✓ | onlyOwner |
| | contractAddToTransferAllowList | Public | ✓ | onlyContract |
| | addToTransferAllowList | Public | ✓ | onlyOwner |
| | contractAddToNotApyList | Public | ✓ | onlyContract |
| | addToNotApyList | Public | ✓ | onlyOwner |
| | contractAddToDexAddressList | Public | ✓ | onlyContract |
| | addToDexAddressList | Public | ✓ | onlyOwner |
| | findIndex | Internal | | |
| | contractRemoveFromTransferAllowList | Public | ✓ | onlyContract |
| | removeFromTransferAllowList | Public | ✓ | onlyOwner |
| | contractRemoveFromDexAddressList | Public | ✓ | onlyContract |
| | removeFromDexAddressList | 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 | Public | ✓ | onlyOwner |
| | allowance | External | | - |
| | increaseAllowance | External | ✓ | - |
| | decreaseAllowance | External | ✓ | - |
| | _transfer | 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 |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BEP20Token
changeBuyTax
txExists
notExecuted
notConfirmed
<Receive Ether>
_msgData
changeTransferTax
setTaxFree
changeApy
changeRequirement
mintProfit
revokeConfirmation
getOwners
getTransactionCount
getTransaction
contractChangeTax
addToTransferAllowList
contractChangeBuyTax
addToNotApyList
submitTransaction
payable
contractChangeTransferTax
addToDexAddressList
contractSetTaxFree
removeFromTransferAllowList
contractChangeApy
removeFromDexAddressList
takeSnapshot
sendAffiliations
changeTax
getOwner
removeOwner
addOwner
decimals
symbol
name
totalSupply
balanceOf
onlyOwner
confirmTransaction
transfer
allowance
renounceOwnership
transferFrom
transferOwnership
contractTakeSnapshot
clearClaimList
contractRemoveFromTransferAllowList
owner
isOwner
executeTransaction
<Constructor>
_init
contractRemoveFromDexAddressList
contractAddToTransferAllowList
contractAddToNotApyList
contractAddToDexAddressList
contractMintProfit
findIndex
contractSendAffiliations
transfer
getAmountAndSendTxs
_burn
claim
_msgSender
getClaimRewardsCount
onlyOwn
onlyContract
decreaseAllowance
increaseAllowance
approve
_transferOwnership
IBEP20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom

# Summary

The Worldwide Token contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The Worldwide Token is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 15% 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