Cyberscope

Audit Report

The Worldwide Token

Aug 2023

Network	BSC
Address	0xe0813549BB6817305Dfd31Ae24b09d3a1c041e9e
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
●	MWV	Multisig Wallet Vulnerability	Unresolved
●	MWR	Multisig Wallet Requirements	Unresolved
●	MFI	Misleading Function Implementation	Unresolved
●	CR	Code Repetition	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	HLI	Holders List Inconsistency	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
Audit Updates 6
Source Files 6
Findings Breakdown 7
ELFM - Exceeds Fees Limit 8
Description 8
Recommendation 8
MWV - Multisig Wallet Vulnerability 9
Description 9
Recommendation 9
MWR - Multisig Wallet Requirements 10
Description 10
Recommendation 11
MFI - Misleading Function Implementation 12
Description 12
Recommendation 12
CR - Code Repetition 13
Description 13
Recommendation 13
RCS - Redundant Conditional Statement 14
Description 14
Recommendation 14
HLI - Holders List Inconsistency 15
Description 15
Recommendation 15
RSK - Redundant Storage Keyword 16
Description 16
Recommendation 16
IDI - Immutable Declaration Improvement 17
Description 17
Recommendation 17
L02 - State Variables could be Declared Constant 18
Description 18
Recommendation 18
L04 - Conformance to Solidity Naming Conventions 19
Description 19

Recommendation 20
L05 - Unused State Variable 21
Description 21
Recommendation 21
L09 - Dead Code Elimination 22
Description 22
Recommendation 22
L13 - Divide before Multiply Operation 23
Description 23
Recommendation 23
Functions Analysis **24**
Inheritance Graph **30**
Flow Graph **31**
Summary **32**
Team Update 32
Disclaimer **33**
About Cyberscope **34**

Review

Contract Name	WorldToken
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xe0813549bb6817305dfd31ae24b09d3a1c041e9e
Address	0xe0813549bb6817305dfd31ae24b09d3a1c041e9e
Network	BSC
Symbol	WORLD
Decimals	18
Total Supply	10,000,000

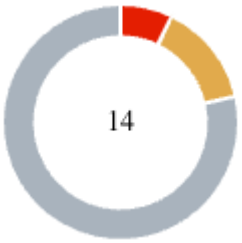
Audit Updates

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 https://github.com/cyberscope-io/audits/blob/main/1-world/v5/audit.pdf
Corrected Phase 6	11 Aug 2023

Source Files

Filename	SHA256
contracts/WorldToken.sol	3b7ed5630001fd26f5ae6483ad5922443ddf3f48af7b6a407916d9d5074377f5

Findings Breakdown



Critical	1
Medium	2
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	2	0	0	0
Minor / Informative	11	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L510
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 <= 33, "no more than 33%");
    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 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#L397,432
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#L1160
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.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L789,818
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#L804
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#L937
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	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	WorldToken.sol#L273
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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	WorldToken.sol#L149,152,153
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 = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
IDEXRouter public router
address public pair
```

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	WorldToken.sol#L101,149,222,224,225,238,239,386,393,399,421,434,448,467,1064,1137,1151
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 = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	WorldToken.sol#L149
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.

```
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	WorldToken.sol#L975
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	WorldToken.sol#L790,791,793,819,820,834,835,840,841,846,847,852,853,857,858,859,864,868,869,870,874,875,876,882,883,884,888,889,890,894,895,896,900,901,902,909,910,911,917,918,919,923,924,925
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		-

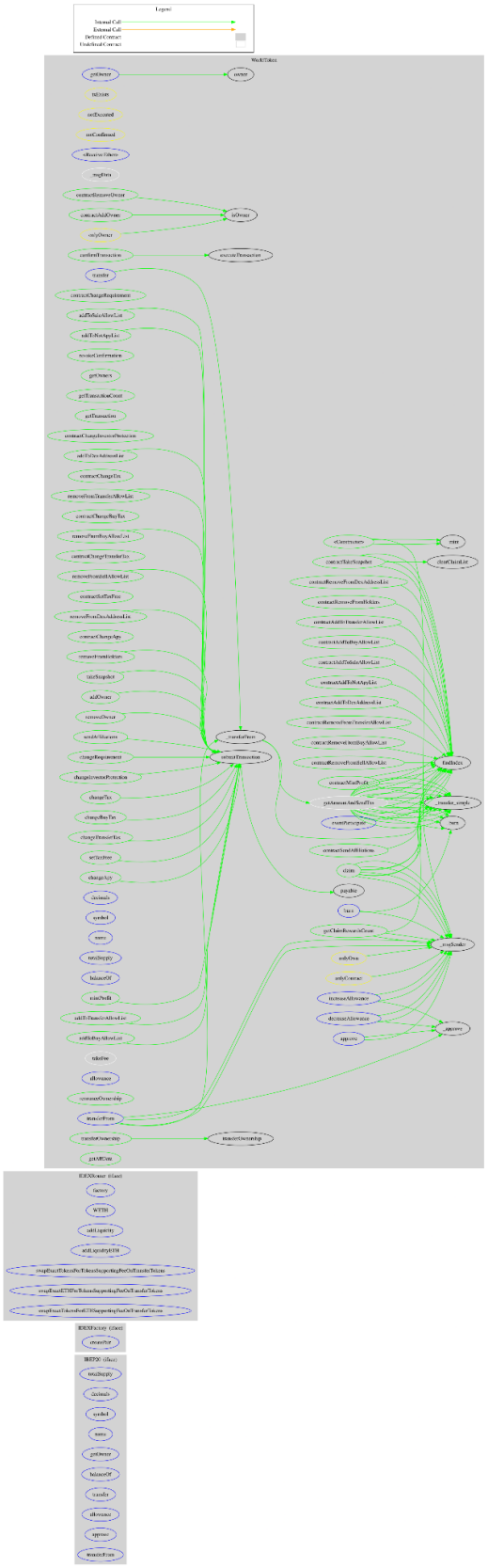
	clearClaimList	Internal	✓	
	contractTakeSnapshot	Public	✓	onlyContract
	takeSnapshot	Public	✓	onlyOwner
	claim	Public	✓	-
	getClaimRewardsCount	Public		-
	getAmountAndSendTxs	Internal	✓	
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	takeFee	Internal	✓	
	allowance	External		-
	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



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. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 33% 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

Cyberscope

Audit Report

The Worldwide Token

July 2023

SHA256 a7fdf03b661cfba9a3ec137d737b43a37a66ecb996b98da80f9df2bb070b0e90

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	Unresolved
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	TFD	Transfer Functions Distinction	Unresolved
Minor	MAF	Misleading APY Functionality	Unresolved
Minor	MEE	Missing Events Emission	Unresolved
Minor	L02	State Variables could be Declared Constant	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	L09	Dead Code Elimination	Unresolved
Minor	L11	Unnecessary Boolean equality	Unresolved
Minor	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
Audit Updates 5
Source Files 5
Findings Breakdown 6
ELFM - Exceeds Fees Limit 7
Description 7
Recommendation 7
MT - Mints Tokens 8
Description 8
Recommendation 9
BT - Burns Tokens 10
Description 10
Recommendation 10
BC - Blacklists Addresses 11
Description 11
Recommendation 11
TFD - Transfer Functions Distinction 12
Description 12
Recommendation 13
MAF - Misleading APY Functionality 14
Description 14
Recommendation 14
MEE - Missing Events Emission 15
Description 15
Recommendation 15
L02 - State Variables could be Declared Constant 16
Description 16
Recommendation 16
L04 - Conformance to Solidity Naming Conventions 17
Description 17
Recommendation 17
L09 - Dead Code Elimination 18
Description 18
Recommendation 18
L11 - Unnecessary Boolean equality 19
Description 19

Recommendation 19
L13 - Divide before Multiply Operation 20
Description 20
Recommendation 20
Functions Analysis **21**
Inheritance Graph **25**
Flow Graph **26**
Summary **27**
Disclaimer **28**
About Cyberscope **29**

Review

Contract Name	BEP20Token
Testing Deploy	https://testnet.bscscan.com/address/0xc3f8b61d7b11ca1d2dd9eceb1fd066bce071ad66
Symbol	WORLD
Decimals	4

Audit Updates

Initial Audit	07 Jul 2023

Source Files

Filename	SHA256
contracts/WorldToken.sol	a7fdf03b661cfba9a3ec137d737b43a37a66ecb996b98da80f9df2bb070b0e90

Findings Breakdown



Critical	4
Medium	1
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	4	0	0	0
Medium	1	0	0	0
Minor / Informative	7	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/WorldToken.sol#L392
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 changeTax(uint256 newTax) public onlyOwner returns (bool) {
    tax = newTax;
}
```

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.

MT - Mints Tokens

Criticality	Critical
Location	contracts/WorldToken.sol#L709,717
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` or `mintToAccount` function. As a result, the contract tokens will be highly inflated.

```
function mint(uint256 amount) public onlyOwner returns (bool) {
  _mint(_msgSender(), amount);
  if (!isHolder(_msgSender())) {
      holders.push(_msgSender());
  }
  return true;
}

function mintToAccount(address addr, uint256 amount) public onlyOwner
returns (bool) {
  _mint(address(addr), amount);
  if (!isHolder(addr)) {
      holders.push(addr);
  }
  return true;
}
```

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. 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.

BT - Burns Tokens

Criticality	Critical
Location	contracts/WorldToken.sol#L660
Status	Unresolved

Description

The contract owner has the authority to burn tokens from a specific address. The owner may take advantage of it by calling the `burnFrom` function. As a result, the targeted address will lose the corresponding tokens.

```
function burnFrom(address addr, uint256 amount) public onlyOwner returns
(bool) {
  _burn(addr, amount);
  return true;
}
```

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. 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.

BC - Blacklists Addresses

Criticality	Critical
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `addToBlockList` function.

```
function addToBlockList(address wallet) public onlyOwner returns (bool) {
  if(isInBlockList(wallet) == false){
     BlockList.push(wallet);
  }
}
```

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. 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.

TFD - Transfer Functions Distinction

Criticality	Medium
Location	contracts/WorldToken.sol#L601,655
Status	Unresolved

Description

The `transfer` and `transferFrom` functions of an ERC20 token are used to transfer tokens from one user to another.The contract implements both functions. However, there is a distinction between the implementation of each function. For instance, the `transferFrom` function only transfers the given amount from the sender to the recipient, while the `transfer` function has additional functionality, like a fee mechanism and allows transaction only if the `msg.sender` is not blacklisted. As a result, the functions implementation is not consistent.

```
function transfer(address recipient, uint256 amount) external returns
(bool) {
  if(isInBlockList(msg.sender) == false){
    uint256 percentage = tax/4;
    uint256 amountTax = (amount * percentage / 100);
    uint256 transferAmount = amount - (amountTax*4);
    if(isInTransferAllowList(msg.sender) == false){
      _burn(msg.sender, amountTax);
      _transfer(_msgSender(), liquidCenterWallet, amountTax);
      _transfer(_msgSender(), worldwideTreasuryWallet, amountTax);
      _transfer(_msgSender(), teamWallet, amountTax);
    } else {
      transferAmount = amount;
    }

    _transfer(_msgSender(), recipient, transferAmount);

    if (!isHolder(recipient)) {
        holders.push(recipient);
    }
  }
  return true;
}

function transferFrom(address sender, address recipient, uint256 amount)
public onlyOwner returns (bool) {
  _transfer(sender, recipient, amount);
  return true;
}
```

Recommendation

The team is advised to ensure that the implementation of the `transfer` and `transferFrom` functions is consistent.

MAF - Misleading APY Functionality

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L396,404
Status	Unresolved

Description

The functions `calculateAPY` and `distributeAPY` are misleadingly named as they do not actually calculate the Annual Percentage Yield (APY) as expected. Instead, they utilize a fixed apyPercentage value to mint tokens exponentially based on the holders' balances.

```
function calculateAPY(address holder) external view returns (uint256) { ... }
function distributeAPY() public onlyOwner returns (bool)  { ... }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them to accurately reflect their purpose.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L425,431,437
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.

```
allowListTransferTax.push(wallet);
BlockList.push(wallet);
dexAddressList.push(wallet);
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L348,364,366
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 public totalAPY
uint256 public maxTotalSupply
address public magmaVaporizerWallet
```

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#L358
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.

```
address[] public BlockList
```

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#L813
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.

```
internal {
    _burn(account, amount);
    _approve(account, _msgSender(),
_allowances[account][_msgSender()].sub(amount, "BEP20: burn amount exceeds
allowance"));
  }
}
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L424,430,436,602,606
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.

```
lowListTransferTax.push(wallet);


push(wallet);
    }
  }
...
  }

 percentage = tax/4;
      uint25

   _burn(msg.sender, amountTax);
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L603,604,605
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 transferAmount = amount - (amountT
    if(isInTransferAllowList(msg.sender) == fa
```

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

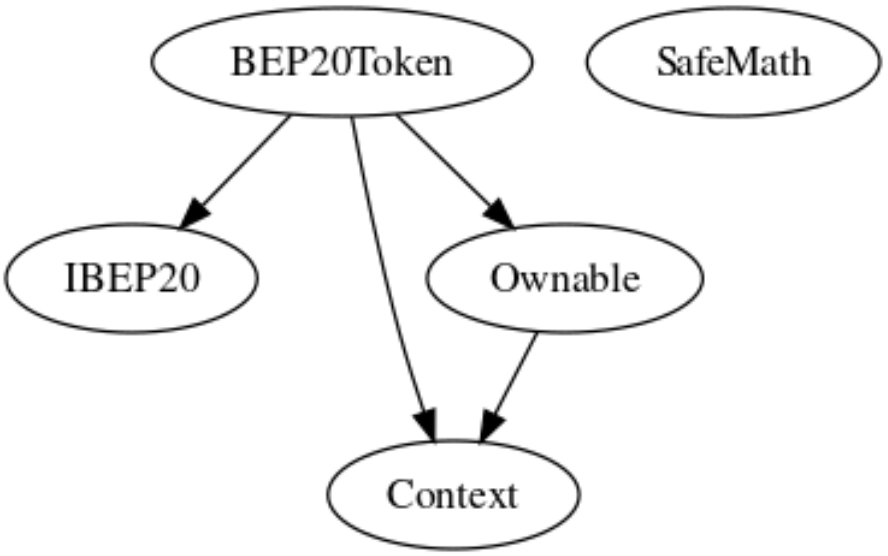
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
		Internal	✓	
	_msgSender	Internal		
	_msgData	Internal		
SafeMath	Library			

	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
BEP20Token	Implementation	Context, IBEP20, Ownable		
		Public	✓	-
	changeTax	Public	✓	onlyOwner
	calculateAPY	External		-
	distributeAPY	Public	✓	onlyOwner
	addToTransferAllowList	Public	✓	onlyOwner
	addToBlockList	Public	✓	onlyOwner

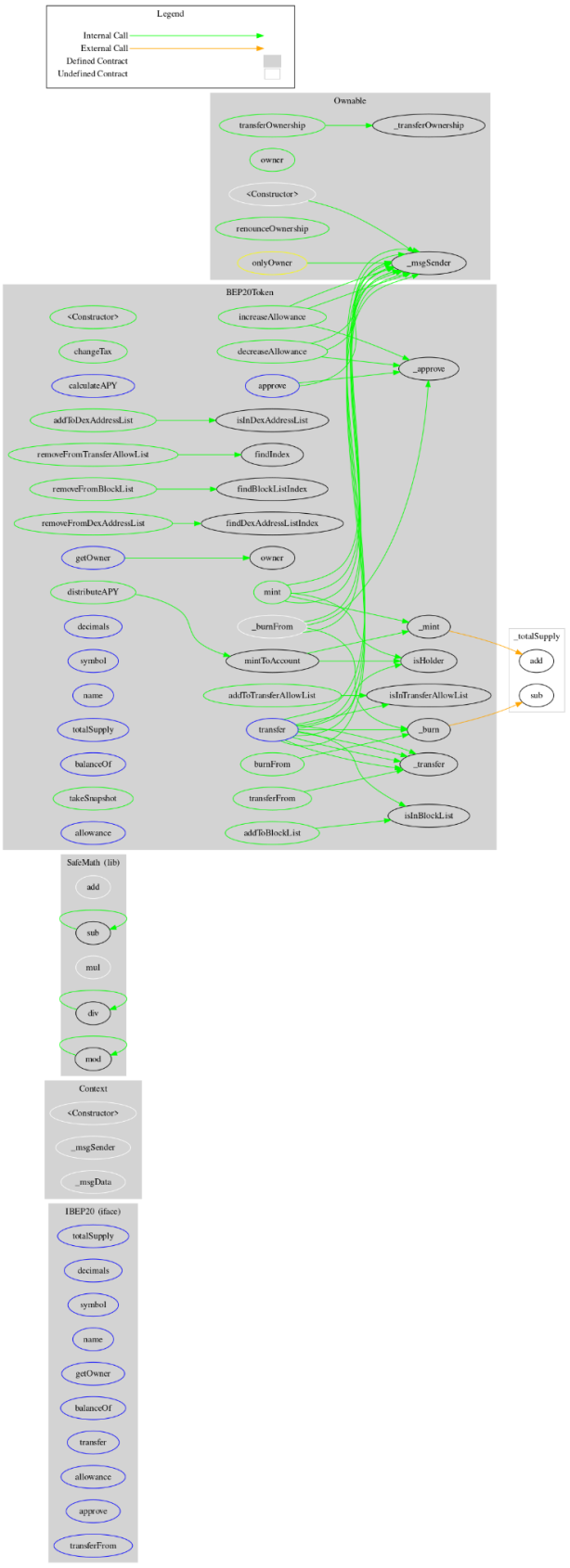
	addToDexAddressList	Public	✓	onlyOwner
	findIndex	Public		-
	findBlockListIndex	Public		-
	findDexAddressListIndex	Public		-
	removeFromTransferAllowList	Public	✓	onlyOwner
	removeFromBlockList	Public	✓	onlyOwner
	removeFromDexAddressList	Public	✓	onlyOwner
	isInTransferAllowList	Internal		
	isInDexAddressList	Internal		
	isHolder	Internal		
	isInBlockList	Internal		
	getOwner	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	totalSupply	External		-
	balanceOf	External		-
	takeSnapshot	Public	✓	onlyOwner
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	Public	✓	onlyOwner
	burnFrom	Public	✓	onlyOwner

	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	mint	Public	✓	onlyOwner
	mintToAccount	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_burnFrom	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
transferOwnership
_transferOwnership
owner
<Constructor>
renounceOwnership
onlyOwner
_msgSender
BEP20Token
<Constructor>
changeTax
calculateAPY
addToDexAddressList
removeFromTransferAllowList
removeFromBlockList
removeFromDexAddressList
getOwner
distributeAPY
decimals
symbol
name
totalSupply
balanceOf
takeSnapshot
allowance
increaseAllowance
decreaseAllowance
approve
isInDexAddressList
findIndex
findBlockListIndex
findDexAddressListIndex
owner
mint
_burnFrom
mintToAccount
addToTransferAllowList
transfer
burnFrom
transferFrom
addToBlockList
_approve
_mint
isHolder
isInTransferAllowList
_burn
_transfer
isInBlockList
_totalSupply
add
sub
SafeMath (lib)
add
sub
mul
div
mod
Context
<Constructor>
_msgSender
_msgData
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. There are some functions that can be abused by the owner like manipulate the fees, mint tokens, burn tokens from any address and massively blacklist addresses. if the contract owner abuses the mint functionality, then the contract will be highly inflated. if the contract owner abuses the burn functionality, then the users could lost their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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

Cyberscope

Audit Report

The Worldwide Token

July 2023

SHA256 20de4b252d5de581fc93471b58eec114c09f4ddb81dc29b4ae4715119e6e670e

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	ZD	Zero Division	Unresolved
Medium	TFD	Transfer Functions Distinction	Unresolved
Minor / Informative	PSU	Potential Subtraction Underflow	Unresolved
Minor / Informative	MMN	Misleading Method Naming	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L11	Unnecessary Boolean equality	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
Audit Updates 5
Source Files 5
Overview 6
Findings Breakdown 7
ST - Stops Transactions 8
Description 8
Recommendation 9
ELFM - Exceeds Fees Limit 10
Description 10
Recommendation 11
MT - Mints Tokens 12
Description 12
Recommendation 13
BT - Burns Tokens 14
Description 14
Recommendation 14
BC - Blacklists Addresses 15
Description 15
Recommendation 15
ZD - Zero Division 16
Description 16
Recommendation 16
TFD - Transfer Functions Distinction 17
Description 17
Recommendation 18
PSU - Potential Subtraction Underflow 19
Description 19
Recommendation 19
MMN - Misleading Method Naming 20
Description 20
Recommendation 21
L04 - Conformance to Solidity Naming Conventions 22
Description 22
Recommendation 23
L09 - Dead Code Elimination 24

Description 24
Recommendation 24
L11 - Unnecessary Boolean equality 25
Description 25
Recommendation 25
L13 - Divide before Multiply Operation 26
Description 26
Recommendation 26
Functions Analysis **27**
Inheritance Graph **31**
Flow Graph **32**
Summary **33**
Disclaimer **34**
About Cyberscope **35**

Review

Contract Name	BEP20Token
Testing Deploy	https://testnet.bscscan.com/address/0xe7ecdcc3b925d7ab70d0b1c708498c03eb5d5ecd
Symbol	WORLD
Decimals	4

Audit Updates

Initial Audit	07 Jul 2023 https://github.com/cyberscope-io/audits/tree/main/1-world/v1/audit.pdf
Corrected Phase 2	13 Jul 2023

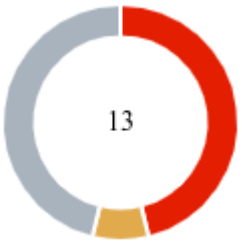
Source Files

Filename	SHA256
contracts/WorldToken_(V3).sol	20de4b252d5de581fc93471b58eec114c09f4ddb81dc29b4ae4715119e6e670e

Overview

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. According to the previously mentioned issues, the contract cannot be assumed that it is in a production-ready state. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's gas consumption and optimize it accordingly to minimize costs and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Findings Breakdown



Critical	6
Medium	1
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	6	0	0	0
Medium	1	0	0	0
Minor / Informative	6	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/WorldToken_(V3).sol#L664
Status	Unresolved

Description

The contract implements a fee mechanism which can be disabled. Once disabled, the owner cannot enable it again. The fee amount is splitted into several portions and transferred to their correnspondingrecipient. However, the contract executes certain operations to calculate the fee percentage, and each of these operations results in a percentage greater than the percentage denominator. As a result, the majority of the transactions will revert.

```
function transfer(address recipient, uint256 amount) external returns
(bool) {
  if(isInBlockList(msg.sender) == false){
    uint256 transferAmount = amount;

    if(taxFree == 0){
      ...
    }

    _transfer(_msgSender(), recipient, transferAmount);
    return true;
  }
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that users can interact with the contract without interruptions. Some possible solutions could be to:

- disable the fees so that transactions proceed as expected
- completely remove the fee functionality from the contract
- rewrite the fee functionality from the ground up.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	contracts/WorldToken_(V3).sol#L397,402,407
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 either the `changeTax` , `changeBuyTax` , or `changeTransferTax` function with a high percentage value.

```
function changeTax(uint256 newTax) public onlyOwner returns (bool) {
    tax = newTax;
    return true;
}

function changeBuyTax(uint256 newTax) public onlyOwner returns (bool) {
    buyTax = newTax;
    return true;
}

function changeTransferTax(uint256 newTax) public onlyOwner returns (bool)
{
    transferTax = 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.

MT - Mints Tokens

Criticality	Critical
Location	contracts/WorldToken_(V3).sol#L885,893
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` or `mintToAccount` function. As a result, the contract tokens will be highly inflated.

```
function mint(uint256 amount) public onlyOwner returns (bool) {
  _mint(_msgSender(), amount);
  if (!isHolder(_msgSender())) {
      holders.push(_msgSender());
  }
  return true;
}

function mintToAccount(address addr, uint256 amount) public onlyOwner
returns(bool) {
  _mint(address(addr), amount);
  if (!isHolder(addr)) {
      holders.push(addr);
  }
  return true;
}
```

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. 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.

BT - Burns Tokens

Criticality	Critical
Location	contracts/WorldToken_(V3).sol#L836
Status	Unresolved

Description

The contract owner has the authority to burn tokens from a specific address. The owner may take advantage of it by calling the `burnFrom` function. As a result, the targeted address will lose the corresponding tokens.

```
function burnFrom(address addr, uint256 amount) public onlyOwner returns
(bool) {
  _burn(addr, amount);
  return true;
}
```

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. 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.

BC - Blacklists Addresses

Criticality	Critical
Location	contracts/WorldToken_(V3).sol#L445
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `addToBlockList` function.

```
function addToBlockList(address wallet) public onlyOwner returns (bool) {
  if(isInBlockList(wallet) == false){
     BlockList.push(wallet);
     emit addToBlockListEvent(wallet);
  }
}
```

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. 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.

ZD - Zero Division

Criticality	Critical
Location	contracts/WorldToken_(V3).sol#L776
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

```
uint256 percent = 50*10000/(transferTax*100);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

TFD - Transfer Functions Distinction

Criticality	Medium
Location	contracts/WorldToken_(V3).sol#L664,831
Status	Unresolved

Description

The `transfer` and `transferFrom` functions of an ERC20 token are used to transfer tokens from one user to another.The contract implements both functions. However, there is a distinction between the implementation of each function. For instance, the `transferFrom` function only transfers the given amount from the sender to the recipient, while the `transfer` function has additional functionality, like a fee mechanism and allows transaction only if the `msg.sender` is not blacklisted. As a result, the functions implementation is not consistent.

```
function transfer(address recipient, uint256 amount) external returns
(bool) {
  if(isInBlockList(msg.sender) == false){
    uint256 transferAmount = amount;

    if(taxFree == 0){
      ...
    }

    _transfer(_msgSender(), recipient, transferAmount);
    return true;
  }
}

function transferFrom(address sender, address recipient, uint256 amount)
public onlyOwner returns (bool) {
  _transfer(sender, recipient, amount);
  return true;
}
```

Recommendation

The team is advised to ensure that the implementation of the `transfer` and `transferFrom` functions is consistent.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	contracts/WorldToken_(V3).sol#L674
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

As part of the transfer flow the contract calculated the convertWorldAmount percentage. However, if the `tax` is less than 10000, the `convertWorldAmount` will be greater than the `transferAmount` .

```
uint256 percent = 7250*10000/tax;
uint256 convertWorldAmount = percent*amount/10000;
transferAmount = transferAmount - convertWorldAmount;
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	contracts/WorldToken_(V3).sol#L424
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The function `distributeAPY` utilizes a fixed `ApyPerAnnum` value to mint tokens exponentially based on the holders' balances. This functionality has nothing to do with Annual Percentage Yield (APY). As a result, the method's name is misleading.

```
function distributeAPY() public onlyOwner returns (bool)  {
    uint256 apyAmount = 0;
    for (uint256 i = 0; i < holders.length; i++) {
        address holder = holders[i];
        uint256 holderBalance = _balances[holder];
        apyAmount += (holderBalance * ApyPerAnnum) / 1000000;
        if ((_totalSupply + apyAmount) <= maxTotalSupply){
          mintToAccount(holder, apyAmount);
          emit APYDistribution(holder, apyAmount);
        }
    }
    return true;
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/WorldToken_(V3).sol#L360,367,368,379,380,381
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.

```
address[] public BlockList
uint256 public constant maxTotalSupply = 100000000000
uint256 public ApyPerAnnum = 21
event allowListTransferTaxEvent(address addr);
event addToBlockListEvent(address addr);
event addToDexAddressListEvent(address addr);
```

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_(V3).sol#L989
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 _burnFrom(address account, uint256 amount) internal {
    _burn(account, amount);
  }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/WorldToken_(V3).sol#L439,446,453,665,771
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.

```
isInTransferAllowList(wallet) == false
isInBlockList(wallet) == false
isInDexAddressList(wallet) == false
isInBlockList(msg.sender) == false
isInTransferAllowList(msg.sender) == 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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/WorldToken_(V3).sol#L640,641,672,673,677,678,682,683,687,688,692,693,704,705,706,710,711,712,716,717,718,722,723,724,737,738,739,743,744,745,749,750,751,755,756,757,761,762,763,776,777,778,782,783,784
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.

```
percent = 125*10000/(tax*100)
percent = percent*10000/tax
```

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

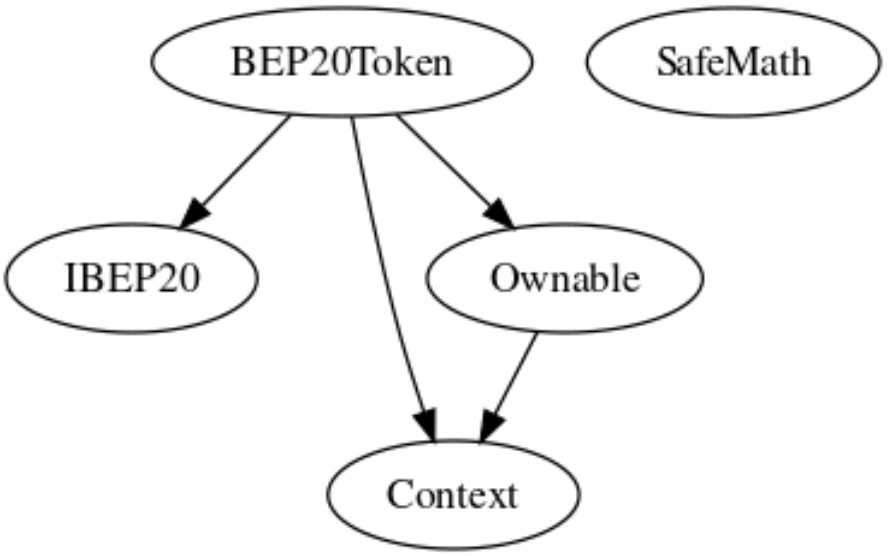
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
		Internal	✓	
	_msgSender	Internal		
	_msgData	Internal		
SafeMath	Library			

	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
BEP20Token	Implementation	Context, IBEP20, Ownable		
		Public	✓	-
	changeTax	Public	✓	onlyOwner
	changeBuyTax	Public	✓	onlyOwner
	changeTransferTax	Public	✓	onlyOwner
	enableTaxFree	Public	✓	onlyOwner
	changeApy	Public	✓	onlyOwner

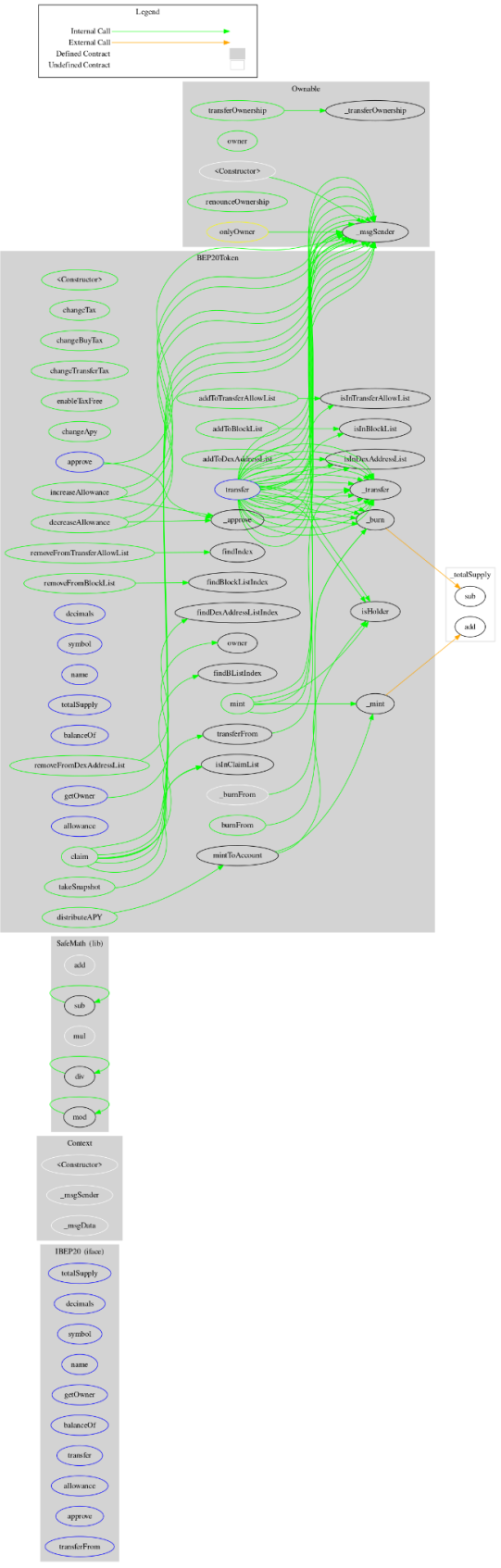
	distributeAPY	Public	✓	onlyOwner
	addToTransferAllowList	Public	✓	onlyOwner
	addToBlockList	Public	✓	onlyOwner
	addToDexAddressList	Public	✓	onlyOwner
	findIndex	Public		-
	findBlockListIndex	Public		-
	findDexAddressListIndex	Public		-
	findBListIndex	Public		-
	removeFromTransferAllowList	Public	✓	onlyOwner
	removeFromBlockList	Public	✓	onlyOwner
	removeFromDexAddressList	Public	✓	onlyOwner
	isInTransferAllowList	Internal		
	isInDexAddressList	Internal		
	isHolder	Internal		
	isInBlockList	Internal		
	isInClaimList	Internal		
	getOwner	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	totalSupply	External		-
	balanceOf	External		-
	takeSnapshot	Public	✓	onlyOwner

	claim	Public	✓	-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	Public	✓	onlyOwner
	burnFrom	Public	✓	onlyOwner
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	mint	Public	✓	onlyOwner
	mintToAccount	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_burnFrom	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
transferOwnership
_transferOwnership
owner
<Constructor>
renounceOwnership
onlyOwner
_msgSender
BEP20Token
<Constructor>
changeTax
changeBuyTax
changeTransferTax
enableTaxFree
changeApy
approve
increaseAllowance
decreaseAllowance
removeFromTransferAllowList
removeFromBlockList
decimals
symbol
name
totalSupply
balanceOf
removeFromDexAddressList
getOwner
allowance
claim
takeSnapshot
distributeAPY
addToTransferAllowList
addToBlockList
addToDexAddressList
transfer
_approve
findIndex
findBlockListIndex
findDexAddressListIndex
owner
findBListIndex
mint
transferFrom
isInClaimList
_burnFrom
burnFrom
mintToAccount
isInTransferAllowList
isInBlockList
isInDexAddressList
transfer
_burn
isHolder
_mint
_totalSupply
sub
add
SafeMath (lib)
add
sub
mul
div
mod
Context
<Constructor>
_msgSender
_msgData
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. There are some functions that can be abused by the owner like stop transactions, manipulating the fees, mint tokens, burning tokens from any address, and massively blacklist addresses. If the contract owner abuses the mint functionality, then the contract will be highly inflated. if the contract owner abuses the burn functionality, then the users could lose their tokens. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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

Cyberscope

Audit Report

The Worldwide Token

July 2023

SHA256 fec0469837eeaabc079cbb4158bfb40a81d18fd3bcf4584515eff126ef7c3037

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	Multisign
●	BT	Burns Tokens	Multisign
●	BC	Blacklists Addresses	Multisign

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	MWV	Multisig Wallet Vulnerability	Unresolved
Critical	PIL	Potential Infinite Loop	Unresolved
Critical	MM	Multisig Misimplementation	Unresolved
Medium	TFD	Transfer Functions Distinction	Unresolved
Minor / Informative	TSO	Transaction Submission Optimization	Unresolved
Minor / Informative	MDR	Misleading Decoding Result	Unresolved
Minor / Informative	MTH	Misleading Token Holders	Unresolved
Minor / Informative	RCS	Redundant Conditional Statement	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	MMN	Misleading Method Naming	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	RSK	Redundant Storage Keyword	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved

●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L23	ERC20 Interface Misuse	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 4
Review 6
Audit Updates 6
Source Files 6
Overview 7
Findings Breakdown 8
MT - Mints Tokens 9
Description 9
Recommendation 9
BT - Burns Tokens 10
Description 10
Recommendation 10
BC - Blacklists Addresses 11
Description 11
Recommendation 11
MWV - Multisig Wallet Vulnerability 12
Description 12
Recommendation 12
PIL - Potential Infinite Loop 13
Description 13
Recommendation 13
MM - Multisig Misimplementation 14
Description 14
Recommendation 14
TFD - Transfer Functions Distinction 16
Description 16
Recommendation 17
TSO - Transaction Submission Optimization 18
Description 18
Recommendation 18
MDR - Misleading Decoding Result 19
Description 19
Recommendation 19
MTH - Misleading Token Holders 20
Description 20
Recommendation 21
RCS - Redundant Conditional Statement 22

Description 22
Recommendation 23
CR - Code Repetition 24
Description 24
Recommendation 25
MMN - Misleading Method Naming 26
Description 26
Recommendation 26
RSML - Redundant SafeMath Library 27
Description 27
Recommendation 27
RSK - Redundant Storage Keyword 28
Description 28
Recommendation 28
IDI - Immutable Declaration Improvement 29
Description 29
Recommendation 29
L04 - Conformance to Solidity Naming Conventions 30
Description 30
Recommendation 31
L13 - Divide before Multiply Operation 32
Description 32
Recommendation 32
L16 - Validate Variable Setters 33
Description 33
Recommendation 33
L23 - ERC20 Interface Misuse 34
Description 34
Recommendation 34
Functions Analysis **35**
Inheritance Graph **39**
Flow Graph **40**
Summary **41**
Team Update 41
Disclaimer **42**
About Cyberscope **43**

Review

Contract Name	BEP20Token
Testing Deploy	https://testnet.bscscan.com/address/0xe7ecdcc3b925d7ab70d0b1c708498c03eb5d5ecd
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

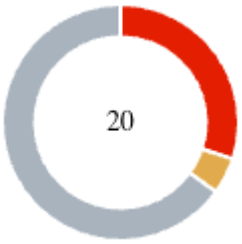
Source Files

Filename	SHA256
contracts/WorldToken.sol	fec0469837eeaabc079cbb4158bfb40a81d18fd3bcf4584515eff126ef7c3037

Overview

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. According to the previously mentioned issues, the contract cannot be assumed that it is in a production-ready state. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's gas consumption and optimize it accordingly to minimize costs and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Findings Breakdown



Critical	6
Medium	1
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	3
Medium	1	0	0	0
Minor / Informative	13	0	0	0

MT - Mints Tokens

Criticality	Critical
Location	contracts/WorldToken.sol#L740
Status	Multisign

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mintToAccount` function. As a result, the contract tokens will be highly inflated.

```
function mintToAccount(address addr, uint256 amount) public onlyOwner
returns (uint256) {
    bytes memory data =
abi.encodeWithSignature("contractMintToAccount(address,uint256)", addr,
amount);
    return submitTransaction(payable(address(this)), 0, data);
}
```

Recommendation

The team should carefully manage the private keys of the owner’s account. Renouncing the ownership will eliminate the threats but it is non-reversible.

BT - Burns Tokens

Criticality	Critical
Location	contracts/WorldToken.sol#L713
Status	Multisign

Description

The contract owner has the authority to burn tokens from a specific address. The owner may take advantage of it by calling the `burnFrom` function. As a result, the targeted address will lose the corresponding tokens.

```
function burnFrom(address addr, uint256 amount) public onlyOwner returns
(uint256) {
    bytes memory data =
abi.encodeWithSignature("burnFrom(address,uint256)", addr, amount);
    return submitTransaction(payable(address(this)), 0, data);
}
```

Recommendation

The team should carefully manage the private keys of the owner’s account. Renouncing the ownership will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Critical
Location	contracts/WorldToken.sol#L426
Status	Multisign

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `addToBlockList` function.

```
function addToBlockList(address wallet) public onlyOwner returns (uint256)
{
    bytes memory data =
abi.encodeWithSignature("contractAddToBlockList(address)", wallet);
    return submitTransaction(payable(address(this)), 0, data);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. 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 the original contract 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 the owner's 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 account. Renouncing the ownership will eliminate the threats but it is non-reversible.

PIL - Potential Infinite Loop

Criticality	Critical
Location	contracts/WorldToken.sol#L703
Status	Unresolved

Description

The `transferFrom` function is protected through a multisig wallet pattern. The contract encodes the functions's ABI and arguments and passes it to the `submitTransaction` function. However, the function's implementation is in the `contractTransferFrom` function, and the contract encodes the `transferFrom` ABI. As a result, the transtactions will enter an infinite loop and eventually revert.

```
function transferFrom(address sender, address recipient, uint256 amount) public
onlyOwner returns (uint256) {
    bytes memory data =
abi.encodeWithSignature("transferFrom(address,address,uint256)", sender, recipient,
amount);
    return submitTransaction(payable(address(this)), 0, data);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that the multisig functionality works as expected. The issue can be addressed by modifying the `transferFrom` function:

```
function transferFrom(address sender, address recipient, uint256 amount) public
onlyOwner returns (uint256) {
    bytes memory data =
abi.encodeWithSignature("contractTransferFrom(address,address,uint256)", sender,
recipient, amount);
    return submitTransaction(payable(address(this)), 0, data);
}
```

MM - Multisig Misimplementation

Criticality	Critical
Location	contracts/WorldToken.sol#L240
Status	Unresolved

Description

The contract incorporates a custom multisig wallet implementation. However, any admin function that requires a multisig pattern will not be able to proceed. The `executeTransaction` function includes the following condition, where a transaction's confirmations must be greater than or equal to the required confirmations. As a result, every invocation of the `confirmTransaction` function will revert.

```
require(
    transaction.numConfirmations >= numConfirmationsRequired,
    "cannot execute tx"
);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that the multisig functionality works as expected. A recommended approach would be to modify the `confirmTransaction` function so that it executes the transaction only when the confirmations have reached the required threshold.

```
function confirmTransaction(
    uint _txIndex
) public onlyOwner txExists(_txIndex) notExecuted(_txIndex) notConfirme(_txIndex) {
    Transaction storage transaction = transactions[_txIndex];
    transaction.numConfirmations += 1;
    isConfirmed[_txIndex][msg.sender] = true;

    emit ConfirmTransaction(msg.sender, _txIndex);

    if (transaction.numConfirmations >= numConfirmationsRequired) {
        executeTransaction(_txIndex);
    }
}
```

The `require(transaction.numConfirmations >= numConfirmationsRequired, "cannot execute tx");` statement should not be omitted from the `executeTransaction` function because the visibility it set to public.

TFD - Transfer Functions Distinction

Criticality	Medium
Location	contracts/WorldToken.sol#L573,703
Status	Unresolved

Description

The `transfer` and `transferFrom` functions of an ERC20 token are used to transfer tokens from one user to another.The contract implements both functions. However, there is a distinction between the implementation of each function. For instance, the `transferFrom` function only transfers the given amount from the sender to the recipient, while the `transfer` function has additional functionality, like a fee mechanism and allows transaction only if the `msg.sender` is not blacklisted. As a result, the functions implementation is not consistent.

```
function transfer(address recipient, uint256 amount) external returns
(bool) {
  if(!isInList(msg.sender, blockList)){
    uint256 transferAmount = amount;
    if(taxFree == 0){
      ...
    }

    _transfer(_msgSender(), recipient, transferAmount);
    return true;
  }
  return false;
}

function transferFrom(address sender, address recipient, uint256 amount)
public onlyOwner returns (uint256) {
    bytes memory data =
abi.encodeWithSignature("transferFrom(address,address,uint256)", sender,
recipient, amount);
    return submitTransaction(payable(address(this)), 0, data);
}
```

Recommendation

The team is advised to ensure that the implementation of the `transfer` and `transferFrom` functions is consistent.

TSO - Transaction Submission Optimization

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L320,336,349,359,370,390,403,415,428,441,466,484,502,547,705,715,742
Status	Unresolved

Description

The `submitTransaction` function serves the purpose of adding a new pending transaction to be executed once all required confirmations are received. This function requires three arguments: a contract address, an integer value, and the encoded data that includes the function signature and its corresponding arguments. However, it is apparent that the first two arguments remain constant for every function invocation, and only the `data` argument varies. As a result, passing the first two arguments is redundant.

```
submitTransaction(payable(address(this)), 0, data)
```

Recommendation

To improve the code efficiency and eliminate redundancy, it is recommended to modify the implementation of the `submitTransaction` function. Instead of requiring the `_to` and `_value` arguments to be passed repeatedly, these values can be stored as constants or predefined variables within the contract or the function itself. By doing so, it streamlines the function call, reduces duplication, and simplifies the code, as the first two arguments will no longer need to be supplied each time the function is invoked.

MDR - Misleading Decoding Result

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L250
Status	Unresolved

Description

The current implementation of the methods using the `onlyContract` modifier in the contract may produce misleading and useless information due to the inconsistent return type. The result of these methods is not always of type `uint256`, which can lead to confusion and provide inaccurate information.

```
(uint256 result) = abi.decode(data, (uint256));
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that these method calls consistently provide relevant and meaningful information.

MTH - Misleading Token Holders

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L732,758
Status	Unresolved

Description

The contract currently allows minting tokens to users as long as the `totalSupply` is less than or equal to the maximum supply. Additionally, it adds the user to the holders array if they are not already on the list. However, a flaw exists wherein a user can be added to the holders list even if the totalSupply has exceeded the maximum supply. Consequently, it is possible for a user with zero tokens to be included in the holders list.

```
function contractMintToAccount(address addr, uint256 amount) public onlyContract
returns (bool) {
    _mint(address(addr), amount);
    if (!isInList(addr, holders)) {
        holders.push(addr);
    }
    return true;
}

function _mint(address account, uint256 amount) internal {
  if (_totalSupply <= MAX_TOTAL_SUPPLY) {
    require(account != address(0), "BEP20: mint to the zero address");

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
  }
}
```

Recommendation

The team is advised to implement proper checks before adding a user to the holders list. The contract should verify that the totalSupply is within the defined limits and only add the user to the list if they possess a non-zero token balance. By incorporating these necessary checks, we can ensure the integrity of the holders list and prevent the inclusion of users with zero tokens.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L550
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.

As described in detail in the RC section, it is apparent that the `isInList` and `findIndex` functions share a similar purpose and yield the same result when used. Consequently, including both of these functions within the `claim` function is redundant.

```
function claim() public returns (bool) {
    if (isInList(_msgSender(), claimList)) {
        uint256 total = 0;
        for (uint256 i = 0; i < claimList.length; i++) {
            total += balances[claimList[i]];
        }
        uint256 myBalance = balances[_msgSender()];
        uint256 percent = (myBalance * 100) / total;
        uint256 amount = (_balances[worldPoolWallet] * percent) / 100;
        transferFrom(worldPoolWallet, _msgSender(), amount);

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

        // Mark the element as deleted by replacing it with the last element
        claimList[uint256(index)] = claimList[claimList.length - 1];

        // Decrease the array length by 1
        claimList.pop();
    }
    return true;
}
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L308,325,341,505
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 `isInList` function currently has the same functionality as the `findIndex` function, albeit with a different return type. To optimize the code and eliminate redundancy, it is advisable for the contract to reuse the existing `findIndex` function and use its return statement accordingly.

```
function isInList(address account, address[] memory list) internal pure returns
(bool) {
    for (uint256 i = 0; i < list.length; i++) {
        if (list[i] == account) {
            return true;
        }
    }
    return false;
}
```

The functionality between the three tax functions can be reused.

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

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.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L388
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The function `distributeAPY` utilizes a fixed `ApyPerAnnum` value to mint tokens exponentially based on the holders' balances. This functionality has nothing to do with Annual Percentage Yield (APY). As a result, the method's name is misleading.

```
function distributeAPYfor15min() public onlyOwner returns (uint256) {
    bytes memory data =
abi.encodeWithSignature("contractDistributeAPYfor15min()");
    return submitTransaction(payable(address(this)), 0, data);
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/WorldToken.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.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L293
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#L138,139,140,144,145,146,147,148
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.

```
_name
_symbol
_decimals
worldPoolWallet
convertWorldWalletBuy
convertWorldWalletTransfer
luquidityPoolWorld
affiliateWallet
```

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#L112,113,114,194,202,203,204,224,236,260,281
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.

```
event allowListTransferTaxEvent(address addr);
event addToBlockListEvent(address addr);
event addToDexAddressListEvent(address addr);

ic onlyOwn {

int _value,
ytes memory
 ) public onlyOwne
blic onlyOwne
       public
```

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#L557,558,580,581,585,586,590,591,595,596,600,601,606,607,608,612,613,614,618,619,620,624,625,626,632,633,634,638,639,640,644,645,646,650,651,652,656,657,658,668,669,670,674,675,676
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.

```
yTax*100) / 10000;
            uint256
 = percent * amount / 10000;
            transfer
```

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#L135
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.

L23 - ERC20 Interface Misuse

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L703
Status	Unresolved

Description

The ERC20 is a standard interface for tokens on the blockchain. It defines a set of functions and events that a contract must implement in order to be considered an ERC20 token. According to the ERC20 interface, the transfer function returns a bool value, which indicates the success or failure of the transfer. If the transfer is successful, the function returns true. If the transfer fails, the function returns false. The contract implements the transfer function without the return value.

```
ddress sender, address recipient, uint256 amount) public onlyOwner returns
(uint256) {
        bytes memory data =
abi.encodeWithSignature("transferFrom(address,address,uint256)", sender,
recipient, amount);
        return submitTransaction(payable(address(this)), 0, data);
    }

    function contra
```

Recommendation

The incorrect implementation of the ERC20 interface 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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
BEP20Token	Implementation			
		Public	✓	-
		External	Payable	-
	_msgSender	Internal		
	_msgData	Internal		
	isOwner	Public		-
	addOwner	Public	✓	onlyOwn
	removeOwner	Public	✓	onlyOwn
	changeRequirement	Public	✓	onlyOwn
	submitTransaction	Public	✓	onlyOwner
	confirmTransaction	Public	✓	onlyOwner txExists notExecuted notConfirmed
	executeTransaction	Public	✓	onlyOwner txExists notExecuted

	revokeConfirmation	Public	✓	onlyOwner txExists 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
	contractDistributeAPYfor15min	Public	✓	onlyContract
	distributeAPYfor15min	Public	✓	onlyOwner
	contractAddToTransferAllowList	Public	✓	onlyContract
	addToTransferAllowList	Public	✓	onlyOwner
	contractAddToNotApyList	Public	✓	onlyContract
	addToNotApyList	Public	✓	onlyOwner
	contractAddToBlockList	Public	✓	onlyContract
	addToBlockList	Public	✓	onlyOwner
	contractAddToDexAddressList	Public	✓	onlyContract

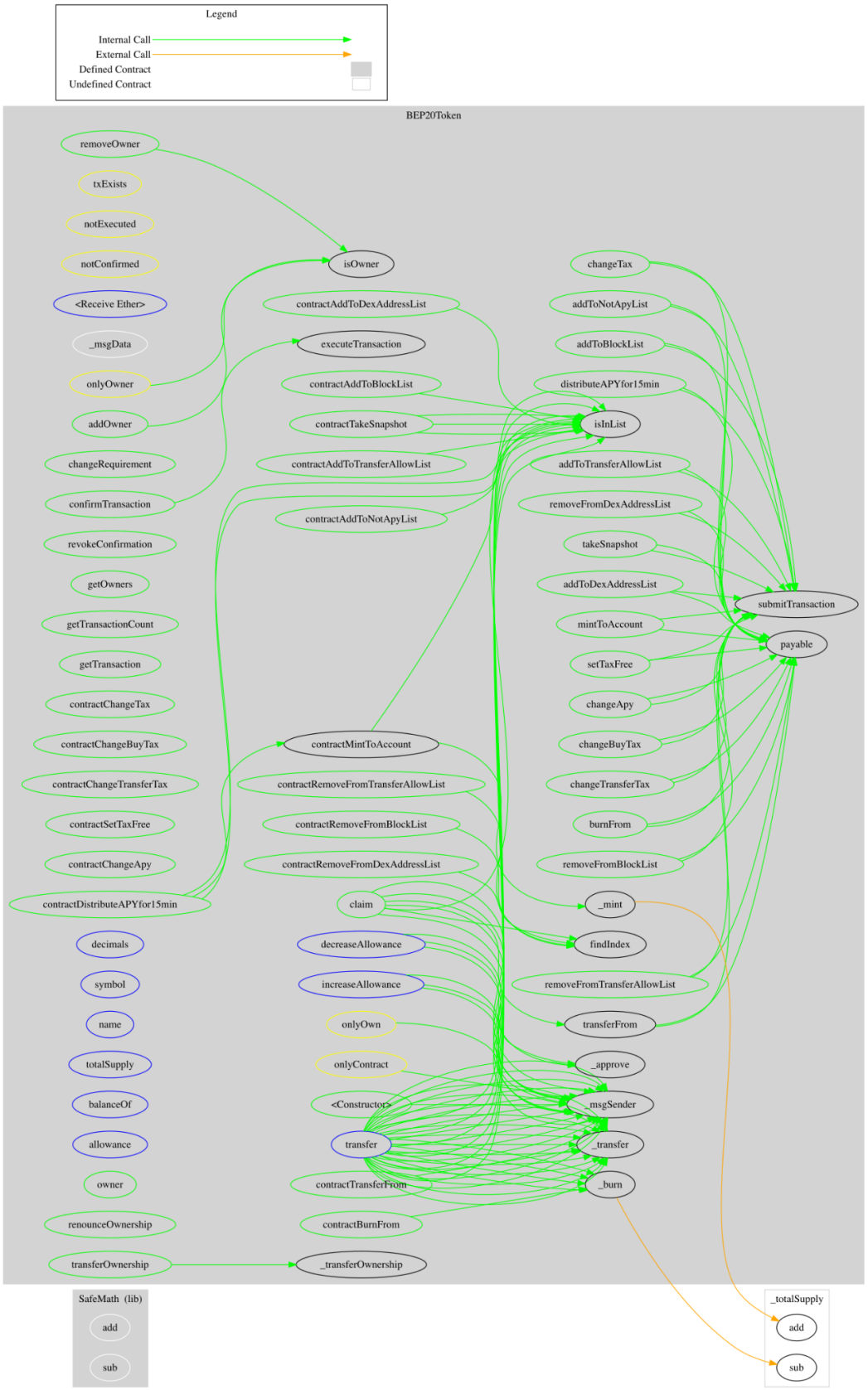
	addToDexAddressList	Public	✓	onlyOwner
	findIndex	Internal		
	contractRemoveFromTransferAllowList	Public	✓	onlyContract
	removeFromTransferAllowList	Public	✓	onlyOwner
	contractRemoveFromBlockList	Public	✓	onlyContract
	removeFromBlockList	Public	✓	onlyOwner
	contractRemoveFromDexAddressList	Public	✓	onlyContract
	removeFromDexAddressList	Public	✓	onlyOwner
	isInList	Internal		
	decimals	External		-
	symbol	External		-
	name	External		-
	totalSupply	External		-
	balanceOf	External		-
	contractTakeSnapshot	Public	✓	onlyContract
	takeSnapshot	Public	✓	onlyOwner
	claim	Public	✓	-
	transfer	External	✓	-
	allowance	External		-
	contractTransferFrom	Public	✓	onlyContract
	transferFrom	Public	✓	onlyOwner
	contractBurnFrom	Public	✓	onlyContract
	burnFrom	Public	✓	onlyOwner

	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	contractMintToAccount	Public	✓	onlyContract
	mintToAccount	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwn
	transferOwnership	Public	✓	onlyOwn
	_transferOwnership	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
BEP20Token
removeOwner
txExists
notExecuted
notConfirmed
<Receive Ether>
_msgData
onlyOwner
addOwner
changeRequirement
confirmTransaction
revokeConfirmation
getOwners
getTransactionCount
getTransaction
contractChangeTax
contractChangeBuyTax
contractChangeTransferTax
contractSetTaxFree
contractChangeApy
contractDistributeAPYfor15min
decimals
symbol
name
totalSupply
balanceOf
allowance
owner
renounceOwnership
transferOwnership
isOwner
contractAddToDexAddressList
executeTransaction
contractAddToBlockList
contractTakeSnapshot
contractAddToTransferAllowList
contractAddToNotApyList
contractMintToAccount
contractRemoveFromTransferAllowList
contractRemoveFromBlockList
contractRemoveFromDexAddressList
claim
decreaseAllowance
increaseAllowance
onlyOwn
onlyContract
<Constructor>
transfer
contractTransferFrom
contractBurnFrom
_transferOwnership
changeTax
addToNotApyList
addToBlockList
distributeAPYfor15min
isInList
addToTransferAllowList
removeFromDexAddressList
takeSnapshot
addToDexAddressList
mintToAccount
setTaxFree
changeApy
changeBuyTax
changeTransferTax
burnFrom
removeFromBlockList
_mint
findIndex
removeFromTransferAllowList
transferFrom
_approve
_msgSender
_transfer
_burn
submitTransaction
payable
SafeMath (lib)
add
sub
_totalSupply
add
sub

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 mint tokens, burn tokens from any address, and massively blacklist addresses. If the contract owner abuses the mint functionality, then the contract will be highly inflated. If the contract owner abuses the burning functionality, the users could lose their tokens. 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 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. However, as described in detail in the MM section, the implementation needs to be reviewed and rewritten to work as expected.

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

Cyberscope

Audit Report

The Worldwide Token

July 2023

SHA256 9871d0a93032725f67f0936d954ae92e088e1a48051a9311f2ff3df8aaad0c3c

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	Multisign
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Critical	MM	Multisig Misimplementation	Unresolved
Critical	MWV	Multisig Wallet Vulnerability	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	RCS	Redundant Conditional Statement	Unresolved
Minor / Informative	RSK	Redundant Storage Keyword	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L13	Divide before Multiply Operation	Unresolved
Minor / Informative	L16	Validate Variable Setters	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
Audit Updates 5
Source Files 5
Overview 6
Findings Breakdown 7
MT - Mints Tokens 8
Description 8
Recommendation 8
MM - Multisig Misimplementation 9
Description 9
Recommendation 9
MWV - Multisig Wallet Vulnerability 11
Description 11
Recommendation 11
CR - Code Repetition 12
Description 12
Recommendation 13
RCS - Redundant Conditional Statement 14
Description 14
Recommendation 14
RSK - Redundant Storage Keyword 15
Description 15
Recommendation 15
IDI - Immutable Declaration Improvement 16
Description 16
Recommendation 16
L04 - Conformance to Solidity Naming Conventions 17
Description 17
Recommendation 18
L13 - Divide before Multiply Operation 19
Description 19
Recommendation 19
L16 - Validate Variable Setters 20
Description 20
Recommendation 20
Functions Analysis 21

Inheritance Graph **25**
Flow Graph **26**
Summary **27**
Team Update 27
Disclaimer **28**
About Cyberscope **29**

Review

Contract Name	BEP20Token
Testing Deploy	https://testnet.bscscan.com/address/0x59fb0136341106fe82f8d3c1bc39a4f8ebfb1168
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

Source Files

Filename	SHA256
contracts/WorldToken.sol	9871d0a93032725f67f0936d954ae92e088e1a48051a9311f2ff3df8aaad0c3c

Overview

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. According to the previously mentioned issues, the contract cannot be assumed that it is in a production-ready state. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's gas consumption and optimize it accordingly to minimize costs and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	1
Medium	0	0	0	0
Minor / Informative	7	0	0	0

MT - Mints Tokens

Criticality	Critical
Location	contracts/WorldToken.sol#L447
Status	Multisign

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mintProfit` function. As a result, the contract tokens will be highly inflated.

```
function mintProfit() public onlyOwner returns (uint256) {
    bytes memory data = abi.encodeWithSignature("contractMintProfit()");
    uint256 tx_index = submitTransaction(data);
    emit SendDistributeTxToConfirm(tx_index);
    return tx_index;
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. Renouncing the ownership will eliminate the threats but it is non-reversible.

MM - Multisig Misimplementation

Criticality	Critical
Location	contracts/WorldToken.sol#L302
Status	Unresolved

Description

The contract incorporates a custom multisig wallet implementation. However, any admin function that requires a multisig pattern will not be able to proceed. The `executeTransaction` function includes the following condition, where a transaction's confirmations must be equal to the required confirmations. As a result, every invocation of the `confirmTransaction` function will revert.

```
require(
    transaction.numConfirmations == numConfirmationsRequired,
    "cannot execute tx"
);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so that the multisig functionality works as expected. A recommended approach would be to modify the the `confirmTransaction` function so that it executes the transaction only when the confirmations have reached the required threshold.

```
function confirmTransaction(
    uint _txIndex
) public onlyOwner txExists(_txIndex) notExecuted(_txIndex)
notConfirme(_txIndex) {
    Transaction storage transaction = transactions[_txIndex];
    transaction.numConfirmations += 1;
    isConfirmed[_txIndex][msg.sender] = true;

    emit ConfirmTransaction(msg.sender, _txIndex);

    if (transaction.numConfirmations >= numConfirmationsRequired) {
        executeTransaction(_txIndex);
    }
}
```

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 account.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/WorldToken.sol#L367,384,400,584,609
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#L239,245,595
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#L352
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#L200
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#L161,163,164,176,177,257,265,287,298,319,340,846,857
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
...
```

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#L589,590,614,615,628,629,633,634,638,639,643,644,648,649,654,655,656,660,661,662,666,667,668,672,673,674,680,681,682,686,687,688,692,693,694,698,699,700,704,705,706,716,717,718,722,723,724
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

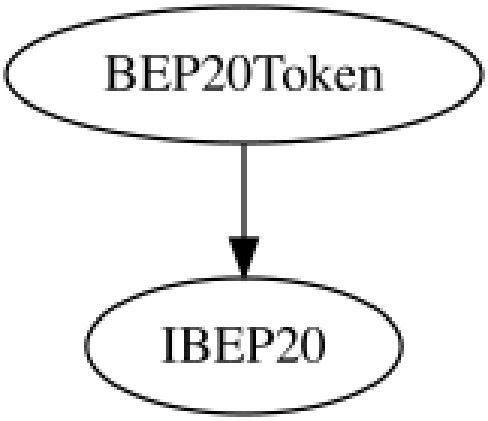
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	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 txExists notExecuted notConfirmed
	executeTransaction	Internal	✓	onlyOwner txExists notExecuted
	revokeConfirmation	Public	✓	onlyOwner txExists 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

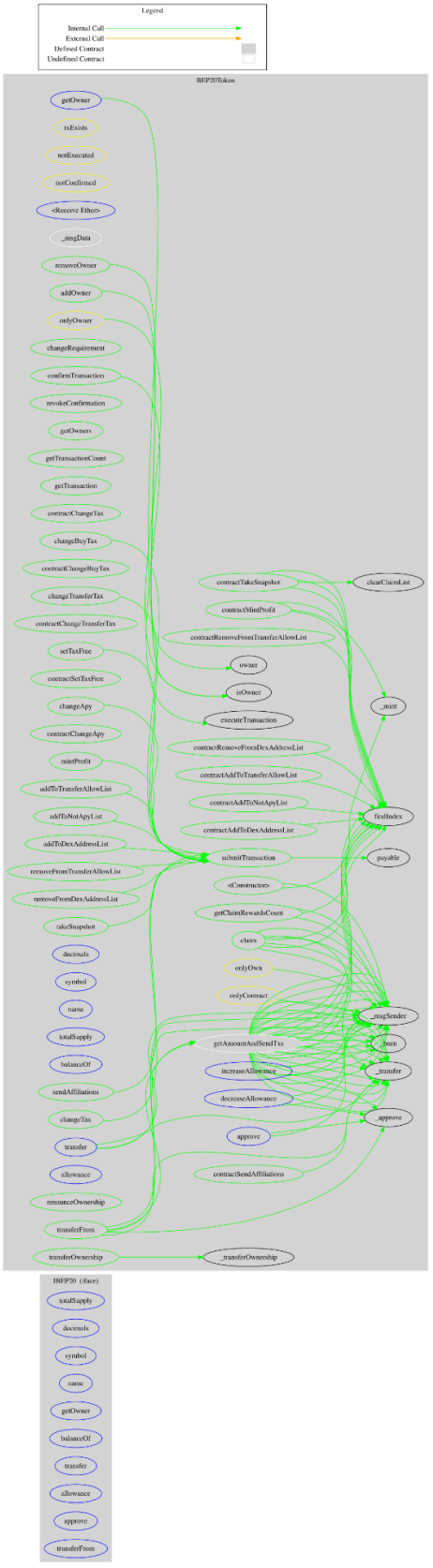
	contractMintProfit	Public	✓	onlyContract
	mintProfit	Public	✓	onlyOwner
	contractAddToTransferAllowList	Public	✓	onlyContract
	addToTransferAllowList	Public	✓	onlyOwner
	contractAddToNotApyList	Public	✓	onlyContract
	addToNotApyList	Public	✓	onlyOwner
	contractAddToDexAddressList	Public	✓	onlyContract
	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
getOwner
txExists
notExecuted
notConfirmed
<Receive Ether>
_msgData
removeOwner
addOwner
onlyOwner
changeRequirement
confirmTransaction
revokeConfirmation
getOwners
getTransactionCount
getTransaction
contractChangeTax
changeBuyTax
contractChangeBuyTax
changeTransferTax
contractChangeTransferTax
setTaxFree
contractSetTaxFree
changeApy
contractChangeApy
mintProfit
addToTransferAllowList
addToNotApyList
addToDexAddressList
removeFromTransferAllowList
removeFromDexAddressList
takeSnapshot
decimals
symbol
name
totalSupply
balanceOf
sendAffiliations
changeTax
transfer
allowance
renounceOwnership
transferFrom
transferOwnership
contractTakeSnapshot
contractMintProfit
contractRemoveFromTransferAllowList
owner
isOwner
executeTransaction
contractRemoveFromDexAddressList
contractAddToTransferAllowList
contractAddToNotApyList
contractAddToDexAddressList
submitTransaction
<Constructor>
getClaimRewardsCount
claim
onlyOwn
onlyContract
getAmountAndSendTxs
increaseAllowance
decreaseAllowance
approve
contractSendAffiliations
_transferOwnership
clearClaimList
_mint
findIndex
payable
_msgSender
_burn
_transfer
approve
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. There are some functions that can be abused by the owner like mint tokens. If the contract owner abuses the mint functionality, then the contract will be highly inflated. 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 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. However, as described in detail in the MM section, the implementation needs to be reviewed and rewritten to work as expected.

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

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

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
Audit Updates 5
Source Files 5
Findings Breakdown 6
MWV - Multisig Wallet Vulnerability 7
Description 7
Recommendation 7
MWR - Multisig Wallet Requirements 8
Description 8
Recommendation 9
HLI - Holders List Inconsistency 10
Description 10
Recommendation 10
CR - Code Repetition 11
Description 11
Recommendation 12
RCS - Redundant Conditional Statement 13
Description 13
Recommendation 13
RSK - Redundant Storage Keyword 14
Description 14
Recommendation 14
IDI - Immutable Declaration Improvement 15
Description 15
Recommendation 15
L04 - Conformance to Solidity Naming Conventions 16
Description 16
Recommendation 17
L13 - Divide before Multiply Operation 18
Description 18
Recommendation 18
L16 - Validate Variable Setters 19
Description 19
Recommendation 19
Functions Analysis 20
Inheritance Graph 24

Flow Graph **25**
Summary **26**
Team Update 26
Disclaimer **27**
About Cyberscope **28**

Review

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

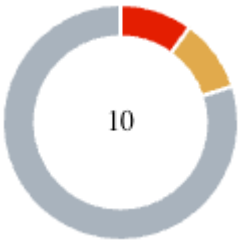
Audit Updates

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

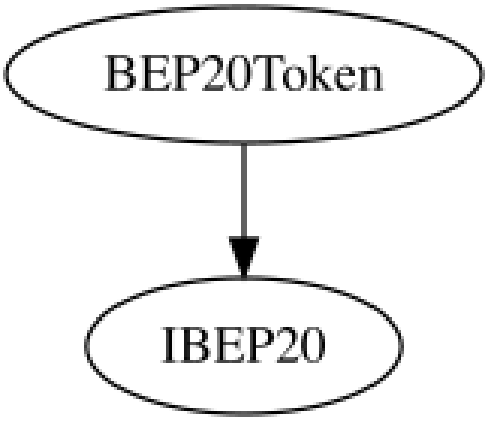
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	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 txExists notExecuted notConfirmed
	executeTransaction	Internal	✓	onlyOwner txExists notExecuted
	revokeConfirmation	Public	✓	onlyOwner txExists 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

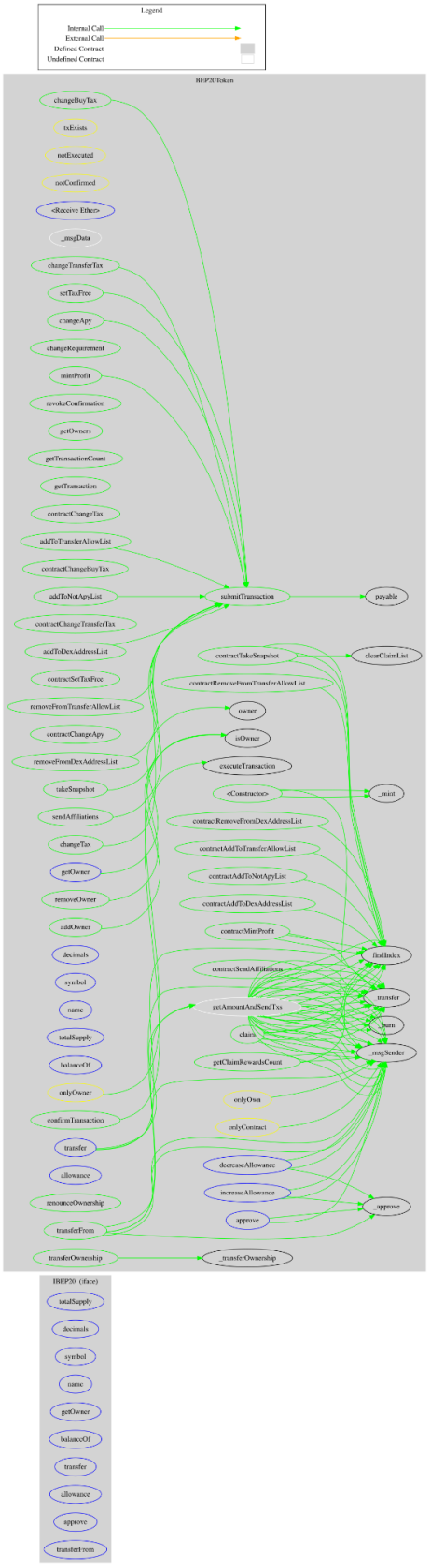
	contractMintProfit	Public	✓	onlyContract
	mintProfit	Public	✓	onlyOwner
	contractAddToTransferAllowList	Public	✓	onlyContract
	addToTransferAllowList	Public	✓	onlyOwner
	contractAddToNotApyList	Public	✓	onlyContract
	addToNotApyList	Public	✓	onlyOwner
	contractAddToDexAddressList	Public	✓	onlyContract
	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

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

Analysis 1
Diagnostics 2
Table of Contents 3
Review 5
Audit Updates 5
Source Files 6
Findings Breakdown 7
ELFM - Exceeds Fees Limit 8
Description 8
Recommendation 8
MWV - Multisig Wallet Vulnerability 9
Description 9
Recommendation 9
MWR - Multisig Wallet Requirements 10
Description 10
Recommendation 11
MFI - Misleading Function Implementation 12
Description 12
Recommendation 12
RRF - Redundant Router Functionality 13
Description 13
Recommendation 13
MEM - Misleading Error Messages 14
Description 14
Recommendation 14
CR - Code Repetition 15
Description 15
Recommendation 15
RCS - Redundant Conditional Statement 16
Description 16
Recommendation 16
HLI - Holders List Inconsistency 17
Description 17
Recommendation 17
RSK - Redundant Storage Keyword 18
Description 18
Recommendation 18
IDI - Immutable Declaration Improvement 19
Description 19

Recommendation 19
L02 - State Variables could be Declared Constant 20
Description 20
Recommendation 20
L04 - Conformance to Solidity Naming Conventions 21
Description 21
Recommendation 22
L09 - Dead Code Elimination 23
Description 23
Recommendation 23
L13 - Divide before Multiply Operation 24
Description 24
Recommendation 24
Functions Analysis **25**
Inheritance Graph **31**
Flow Graph **32**
Summary **33**
Team Update 33
Disclaimer **34**
About Cyberscope **35**

Review

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

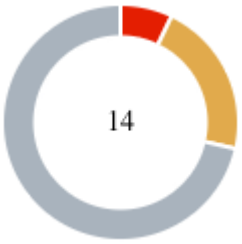
Audit Updates

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		-

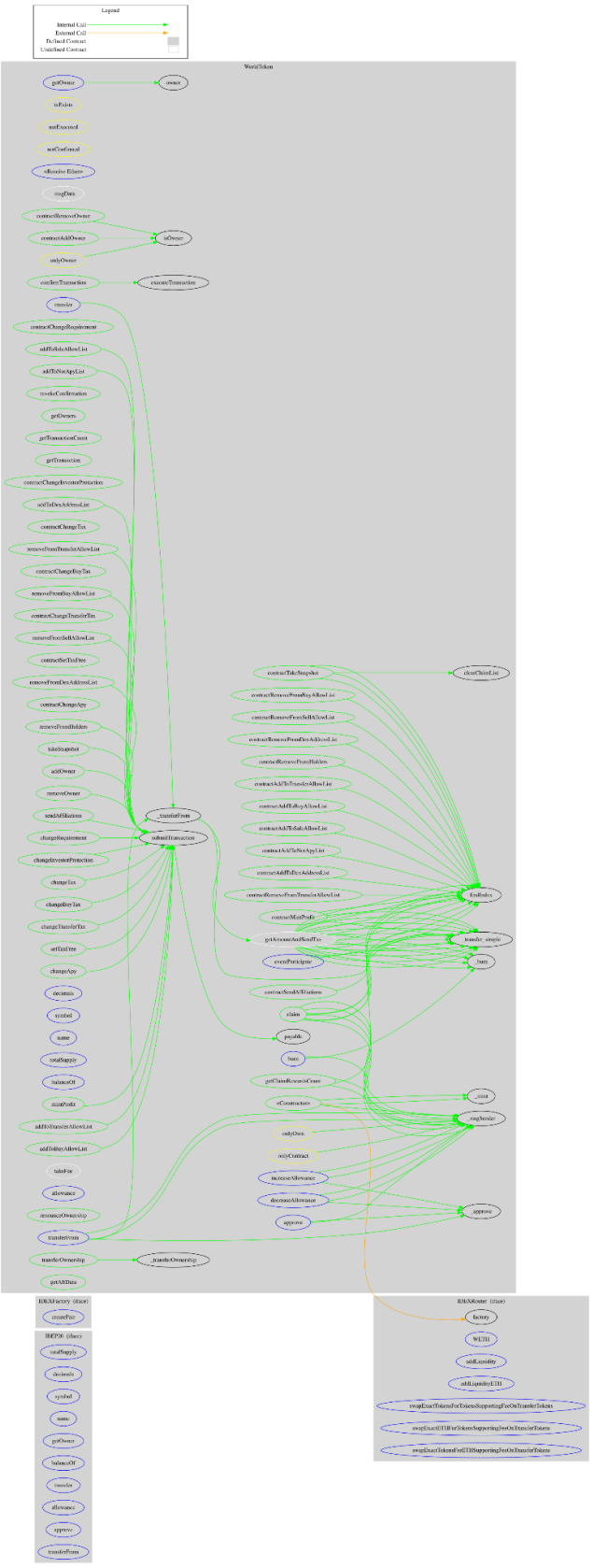
	clearClaimList	Internal	✓	
	contractTakeSnapshot	Public	✓	onlyContract
	takeSnapshot	Public	✓	onlyOwner
	claim	Public	✓	-
	getClaimRewardsCount	Public		-
	getAmountAndSendTxs	Internal	✓	
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	takeFee	Internal	✓	
	allowance	External		-
	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