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
●	MWV	Multisig Wallet Vulnerability	Unresolved
●	PIL	Potential Infinite Loop	Unresolved
●	MM	Multisig Misimplementation	Unresolved
●	TFD	Transfer Functions Distinction	Unresolved
●	TSO	Transaction Submission Optimization	Unresolved
●	MDR	Misleading Decoding Result	Unresolved
●	MTH	Misleading Token Holders	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	CR	Code Repetition	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	RSML	Redundant SafeMath Library	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
●	L23	ERC20 Interface Misuse	Unresolved

Table of Contents

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

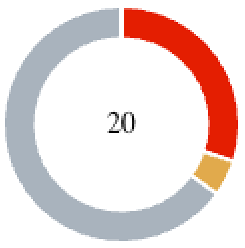
Source Files

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

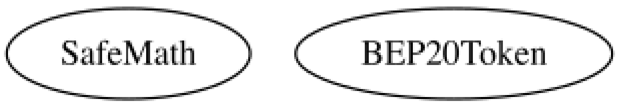
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
BEP20Token	Implementation			
		Public	✓	-
		External	Payable	-
	_msgSender	Internal		
	_msgData	Internal		
	isOwner	Public		-
	addOwner	Public	✓	onlyOwn
	removeOwner	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

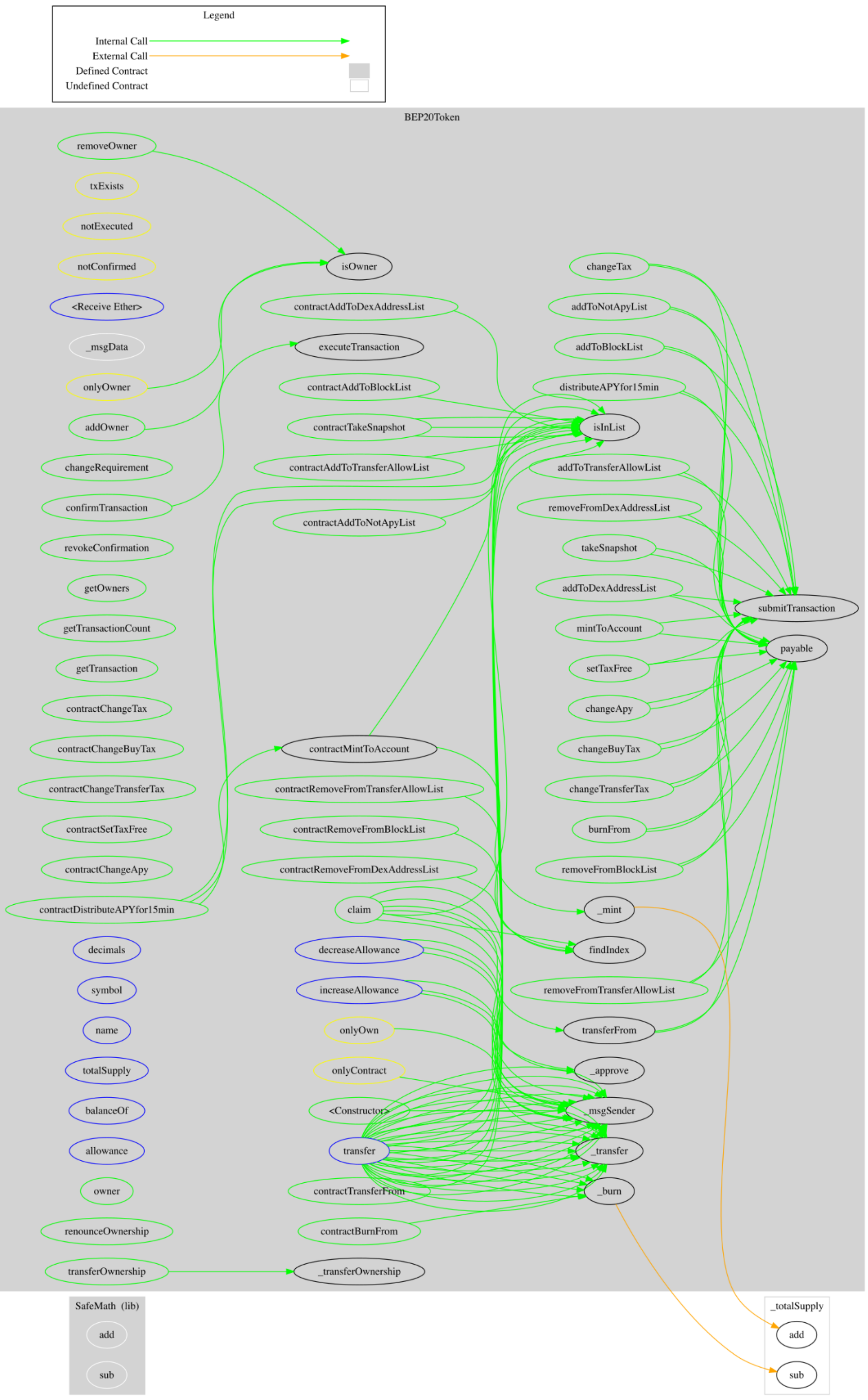
	addToDexAddressList	Public	✓	onlyOwner
	findIndex	Internal		
	contractRemoveFromTransferAllowList	Public	✓	onlyContract
	removeFromTransferAllowList	Public	✓	onlyOwner
	contractRemoveFromBlockList	Public	✓	onlyContract
	removeFromBlockList	Public	✓	onlyOwner
	contractRemoveFromDexAddressList	Public	✓	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
submitTransaction
mintToAccount
payable
setTaxFree
changeApy
changeBuyTax
changeTransferTax
burnFrom
removeFromBlockList
_mint
findIndex
removeFromTransferAllowList
transferFrom
_approve
_msgSender
_transfer
_burn
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

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