Cyberscope

Audit Report

MyBricks

March 2023

Type	BEP20
Network	BSC
Address	0xcf110164BF0e3Ed7859a6750b3326B861AEE028E
Audited by	© cyberscope

Table of Contents

Review

Contract Name	MyUSD
Compiler Version	v0.6.12+commit.27d51765
Optimization	200 runs
Explorer	https://bscscan.com/address/0xcf110164bf0e3ed7859a6750b3326b861aee028e
Address	0xcf110164bf0e3ed7859a6750b3326b861aee028e
Network	BSC
Symbol	MyUSD
Decimals	18
Total Supply	100

Audit Updates

Initial Audit	17 Mar 2023

Source Files

Filename	SHA256
MyUSD.sol	2e7b09b624528a048177925a6a35bfeace7c8af6c312aa6188a13a2fe1f00c4f

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Unresolved
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	MyUSD.sol#L1503
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 `setTaxFeePercent` function with a high percentage value.

```
function setSellFee(uint256 _value) external {
    require(msg.sender == taxCollectorAddress, "!fee");
    require(_value != 0, "zero");
    sellFee = _value;
}
```

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	MyUSD.sol#L1528,1599
Status	Unresolved

Description

The contract minters have the authority to mint tokens. The minters may take advantage of it by calling the `mint` and/or `distributeReward` functions. As a result, the contract tokens will be highly inflated.

```
function mint(address recipient_, uint256 amount_) public onlyMinter returns (bool)
{
    uint256 balanceBefore = balanceOf(recipient_);
    _mint(recipient_, amount_);
    uint256 balanceAfter = balanceOf(recipient_);

    return balanceAfter > balanceBefore;
}
...
function distributeReward(
    address _myUSDPool
) external onlyOperator {
    require(!rewardPoolDistributed, "only can distribute once");
    require(_myUSDPool != address(0), "!_myUSDPool");
    rewardPoolDistributed = true;
    _mint(_myUSDPool, INITIAL_MyUSD_POOL_DISTRIBUTION);
}
```

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.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	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
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	MyUSD.sol#L1290
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.

```
Role storage rol
```

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	MyUSD.sol#L1413,1414
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
uniswapV2Route
pai
```

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	MyUSD.sol#L1379
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 minSwapAmount = 100000000000000000000
```

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	MyUSD.sol#L974,975,992,1047,1352,1360,1383,1438,1442,1455,1462,1474,1486,1491,1497,1503,1510,1516,1600,1610,1611,1612
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint256 public constant INITIAL_MyUSD_POOL_DISTRIBUTION = 100000 ether
address public MyUSDOracle
IERC20 USDC = IERC20(0x8AC76a51cc950d9822D68b83fE1Ad97B32Cd580d)
address _address
uint8 _index
uint256 _value
uint256 _burnThreshold
bool _value
address _MyUSDOracle
address _taxOffice

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	MyUSD.sol#L1506
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
sellFee = _value
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	MyUSD.sol#L1443,1456
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(_index >= 0, "Index has to be higher than 0")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	MyUSD.sol#L599,671,678,686,1466
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 _setupDecimals(uint8 decimals_) internal virtual {
        _decimals = decimals_;
    }

function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
...
        return a < b ? a : b;
    }

function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }

...
```

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.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	MyUSD.sol#L1467
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint144 _price
```

Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MyUSD.sol#L1467,1560
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint144 _price
uint256 feeAmount
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	MyUSD.sol#L3,27,104,318,621,662,693,853,920,957,1010,1042,1251
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity >=0.6.0 <0.8.0;
pragma solidity 0.6.12;
pragma solidity >=0.5.0;
pragma solidity >=0.6.2;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	MyUSD.sol#L1614
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
_token.transfer(_to, _amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		

	div	Internal		
	mod	Internal		
ERC20	Implementation	Context, IERC20		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_setupDecimals	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-
Math	Library			
	max	Internal		

	min	Internal		
	average	Internal		
SafeMath8	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
Operator	Implementation	Context, Ownable		
		Internal	✓	
	operator	Public		-
	isOperator	Public		-
	transferOperator	Public	✓	onlyOwner
	_transferOperator	Internal	✓	
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-

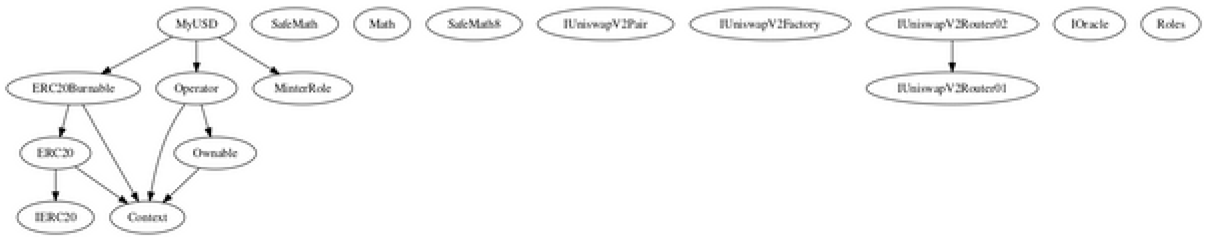
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-

	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		

	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IOracle	Interface			
	update	External	✓	-
	consult	External		-
	twap	External		-
Roles	Library			
	add	Internal	✓	
	remove	Internal	✓	
	has	Internal		
MinterRole	Implementation			
		Public	✓	-
	isMinter	Public		-
	addMinter	Public	✓	onlyMinter
	removeMinter	Public	✓	onlyMinter
	renounceMinter	Public	✓	-
	_addMinter	Internal	✓	
	_removeMinter	Internal	✓	
MyUSD	Implementation	ERC20Burnable, Operator, MinterRole		

		Public	✓	ERC20
	getTaxTiersTwapsCount	Public		-
	getTaxTiersRatesCount	Public		-
	isAddressExcluded	Public		-
	setTaxTiersTwap	Public	✓	onlyTaxOffice
	setTaxTiersRate	Public	✓	onlyTaxOffice
	setBurnThreshold	Public	✓	onlyTaxOffice
	_getMyUSDPrice	Internal		
	setTakeFee	Public	✓	onlyOperator
	enableAutoCalculateTax	Public	✓	onlyTaxOffice
	disableAutoCalculateTax	Public	✓	onlyTaxOffice
	setMyUSDOracle	Public	✓	onlyOperatorOrTaxOffice
	setTaxOffice	Public	✓	onlyOperatorOrTaxOffice
	setTaxCollectorAddress	Public	✓	onlyTaxOffice
	setSellFee	External	✓	-
	excludeAddress	Public	✓	onlyOperatorOrTaxOffice
	includeAddress	Public	✓	onlyOperatorOrTaxOffice
	mint	Public	✓	onlyMinter
	burn	Public	✓	-
	burnFrom	Public	✓	onlyOperator
	_transfer	Internal	✓	
	swapAndSendToFee	Private	✓	swapping
	isNotInSwap	External		-
	distributeReward	External	✓	onlyOperator
	governanceRecoverUnsupported	External	✓	onlyOperator

Inheritance Graph



Flow Graph

Summary

MyUSD 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, and mint tokens. If the contract owner abuses the mint functionality, 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.

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