Cyberscope

Audit Report

BenefitMine

September 2023

Network BSC

Address 0x4e1c1bd35397042319fe252d2e324ad439b19f1e

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
●	RTC	Redundant Type Casting	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	CoinToken
Compiler Version	v0.8.5+commit.a4f2e591
Optimization	200 runs
Explorer	https://bscscan.com/address/0x4e1c1bd35397042319fe252d2e324ad439b19f1e
Address	0x4e1c1bd35397042319fe252d2e324ad439b19f1e
Network	BSC
Symbol	BFM
Decimals	8
Total Supply	1,000,000,000

Audit Updates

Initial Audit	13 Sep 2023

Source Files

Filename	SHA256
CoinToken.sol	985b42cb858b3ff274cea6febfd64e40bbfed7de8a710cc40be9ef5689a3d19d

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	7

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

RTC - Redundant Type Casting

Criticality	Minor / Informative
Location	CoinToken.sol#L1247
Status	Unresolved

Description

The contract initializes the `decimals_` as `uint8` and `initialBalance_` as `uint256` types in the constructor. However, within the `_mint` function, the variable `decimals_` is explicitly cast again to `uint256`. This is redundant because `decimals_` is already type of `uint8`, not `uint256`, making the type casting unnecessary and potentially confusing.

```
    constructor(
        ...
        uint8 decimals_,
        uint256 initialBalance_,
        ...
    ) payable ERC20(name_, symbol_) ERC20Decimals(decimals_)  {
        ...
        _mint(tokenOwner,
initialBalance_*10**uint256(decimals_));
    }
```

Recommendation

It is recommended to consider removing the redundant type casting of `decimals_` in the `_mint` function. The code `initialBalance_*10**uint256(decimals_)` can be simplified to `_initialBalance_*10**decimals_`. This will improve code readability and eliminate any confusion arising from unnecessary type casting.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CoinToken.sol#L1067
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 _owner
```

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	CoinToken.sol#L537,563,573,588,598,613,623,637,647,655
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls,
avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value,
recipient may have reverted");
...
function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level
call failed");
    }

function functionCall(address target, bytes memory data, string
memory errorMessage) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0,
errorMessage);
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CoinToken.sol#L1245,1246
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.

```
payable(feeReceiver_).transfer(msg.value)
_owner   = tokenOwner
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	CoinToken.sol#L517,664
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { size := extcodesize(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	CoinToken.sol#L3,83,112,139,445,487,679,706,736,829,863,895,1052,1114,1137,1164,1230
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	CoinToken.sol#L1129
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.

```
IERC20(tokenAddress).transfer(owner(), tokenAmount)
```

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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		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	✓	
	_beforeTokenTransfer	Internal	✓	
ERC20Burnable	Implementation	Context, ERC20		
	burn	Public	✓	-
	burnFrom	Public	✓	-

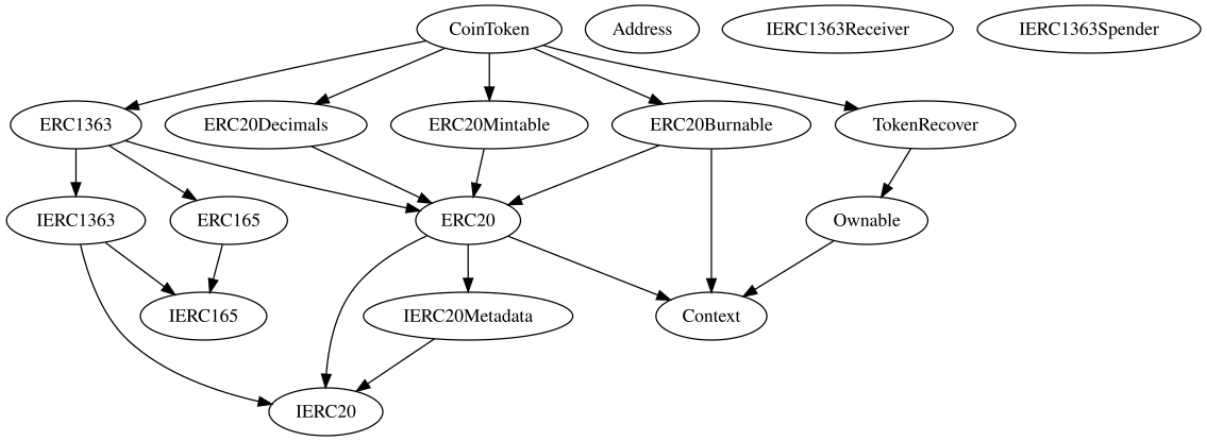
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
IERC165	Interface			
	supportsInterface	External		-
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
IERC1363	Interface	IERC20, IERC165		
	transferAndCall	External	✓	-
	transferAndCall	External	✓	-

	transferFromAndCall	External	✓	-
	transferFromAndCall	External	✓	-
	approveAndCall	External	✓	-
	approveAndCall	External	✓	-
IERC1363Receiver	Interface			
	onTransferReceived	External	✓	-
IERC1363Spender	Interface			
	onApprovalReceived	External	✓	-
ERC1363	Implementation	ERC20, IERC1363, ERC165		
	supportsInterface	Public		-
	transferAndCall	Public	✓	-
	transferAndCall	Public	✓	-
	transferFromAndCall	Public	✓	-
	transferFromAndCall	Public	✓	-
	approveAndCall	Public	✓	-
	approveAndCall	Public	✓	-
	_checkAndCallTransfer	Internal	✓	
	_checkAndCallApprove	Internal	✓	

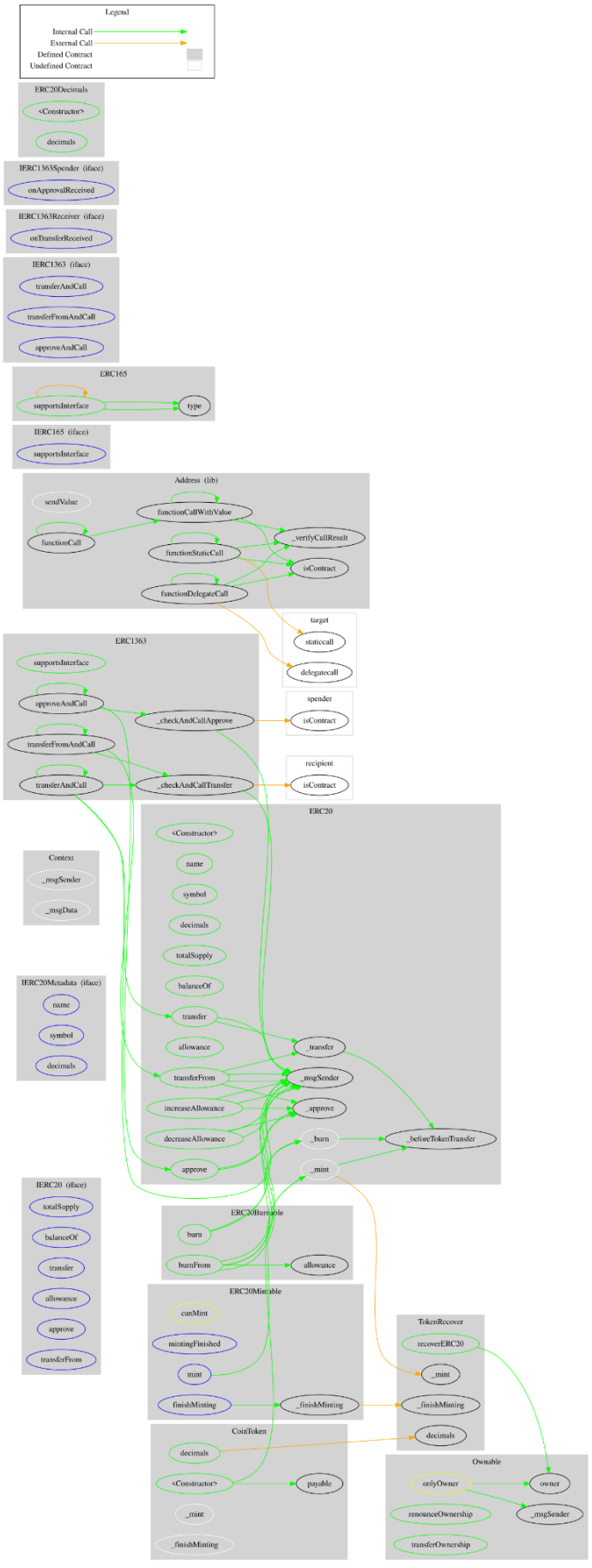
Ownable	Implementation	Context		
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
TokenRecover	Implementation	Ownable		
	recoverERC20	Public	✓	onlyOwner
ERC20Decimals	Implementation	ERC20		
		Public	✓	-
	decimals	Public		-
ERC20Mintable	Implementation	ERC20		
	mintingFinished	External		-
	mint	External	✓	canMint
	finishMinting	External	✓	canMint
	_finishMinting	Internal	✓	
CoinToken	Implementation	ERC20Decimals, ERC20Mintable, ERC20Burnable, ERC1363, TokenRecover		
		Public	Payable	ERC20 ERC20Decimals

	decimals	Public		-
	_mint	Internal	✓	onlyOwner
	_finishMinting	Internal	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20Decimals
<Constructor>
decimals
IERC1363Spender (iface)
onApprovalReceived
IERC1363Receiver (iface)
onTransferReceived
IERC1363 (iface)
transferAndCall
transferFromAndCall
approveAndCall
ERC165
supportsInterface
type
IERC165 (iface)
supportsInterface
Address (lib)
sendValue
functionCallWithValue
functionCall
functionStaticCall
_verifyCallResult
isContract
functionDelegateCall
target
staticcall
delegatecall
ERC1363
supportsInterface
approveAndCall
_checkAndCallApprove
spender
isContract
transferFromAndCall
transferAndCall
_checkAndCallTransfer
recipient
isContract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
increaseAllowance
decreaseAllowance
approve
_transfer
_msgSender
_approve
_burn
_mint
_beforeTokenTransfer
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ERC20Burnable
burn
burnFrom
allowance
ERC20Mintable
canMint
mintingFinished
mint
finishMinting
_finishMinting
TokenRecover
recoverERC20
_mint
_finishMinting
decimals
CoinToken
decimals
<Constructor>
payable
_mint
_finishMinting
Ownable
onlyOwner
owner
renounceOwnership
_msgSender
transferOwnership

Summary

BenefitMine contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. BenefitMine is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error 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.

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