Cyberscope

Audit Report

Meme Buddha

July 2023

Commit c286019a3fca0681dd94e736e78dd4af70a04d06

Repository https://github.com/mebuvip/mebu/blob/main/contract.sol

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DDP	Decimal Division Precision	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Repository	https://github.com/mebuvip/mebu/blob/main/contract.sol
Commit	c286019a3fca0681dd94e736e78dd4af70a04d06
Testing Deploy	https://testnet.bscscan.com/address/0xc6fca300f6b3a3855fce851fa41442a381f8cb47

Audit Updates

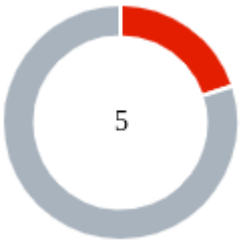
Initial Audit	25 Jul 2023

Source Files

Filename	SHA256
contracts/contract.sol	9d12202fe86018dbd9bc1a3571645eb51507b801b53cdf4c3601c3db5fd593b7
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/token/ERC20/ERC20.sol	d20d52b4be98738b8aa52b5bb0f88943f62128969b33d654fbca731539a7fe0a
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	0344809a1044e11ece2401b4f7288f414ea41fa9d1dad24143c84b737c9fc02e

@openzeppelin/contracts/security/Pausable.sol	2072248d2f79e661c149fd6a6593a8a3f03 8466557c9b75e50e0b001bcb5cf97
@openzeppelin/contracts/access/Ownable.sol	a8e4e1ae19d9bd3e8b0a6d46577eec098c 01fbaffd3ec1252fd20d799e73393b

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	4

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

ST - Stops Transactions

Criticality	Critical
Location	contracts/contract.sol#L39,51
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by calling the `pause` function. As a result, the contract may operate as a honeypot.

```
   function pause() public onlyOwner {
        _pause();
    }

    function _beforeTokenTransfer(address from, address to,
uint256 amount)
        internal
        override
    {
        if (from != owner()) {
            require(!paused(), "ERC20Pausable: token transfer
while paused");
        }
      ...
    }
```

Recommendation

It is recommended to review the code and assess the actual necessity of the `pause` functionality within the context of the contract's intended use and operations. If it's determined that the `pause` function does not serve a clear and justified purpose, it would be prudent to consider removing this functionality from the contract. Also 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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	contracts/contract.sol#L57
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
uint256 distributeAmount = fee / 3;

 ...
 _burn(from, distributeAmount);
 _transfer(from, charityWallet, distributeAmount);
 _transfer(from, teamWallet, distributeAmount);
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/contract.sol#L19,24,29,34
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 updates the `charityWallet`, `teamWallet`, and `liquidityPools` of an address even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
    function setCharityWallet(address _charityWallet) public
onlyOwner {
        require(_charityWallet != address(0), "Invalid charity
wallet address");
        charityWallet = _charityWallet;
    }

    function setTeamWallet(address _teamWallet) public
onlyOwner {
        require(_teamWallet != address(0), "Invalid team wallet
address");
        teamWallet = _teamWallet;
    }

    function addLiquidityPool(address _liquidityPool) public
onlyOwner {
        require(_liquidityPool != address(0), "Invalid
liquidity pool address");
        liquidityPools[_liquidityPool] = true;
    }

    function removeLiquidityPool(address _liquidityPool) public
onlyOwner {
        require(_liquidityPool != address(0), "Invalid
liquidity pool address");
        liquidityPools[_liquidityPool] = false;
    }
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/contract.sol#L19,24,29,34
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 _charityWallet
address _teamWallet
address _liquidityPool
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/contract.sol#L2
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 ^0.8.9;
```

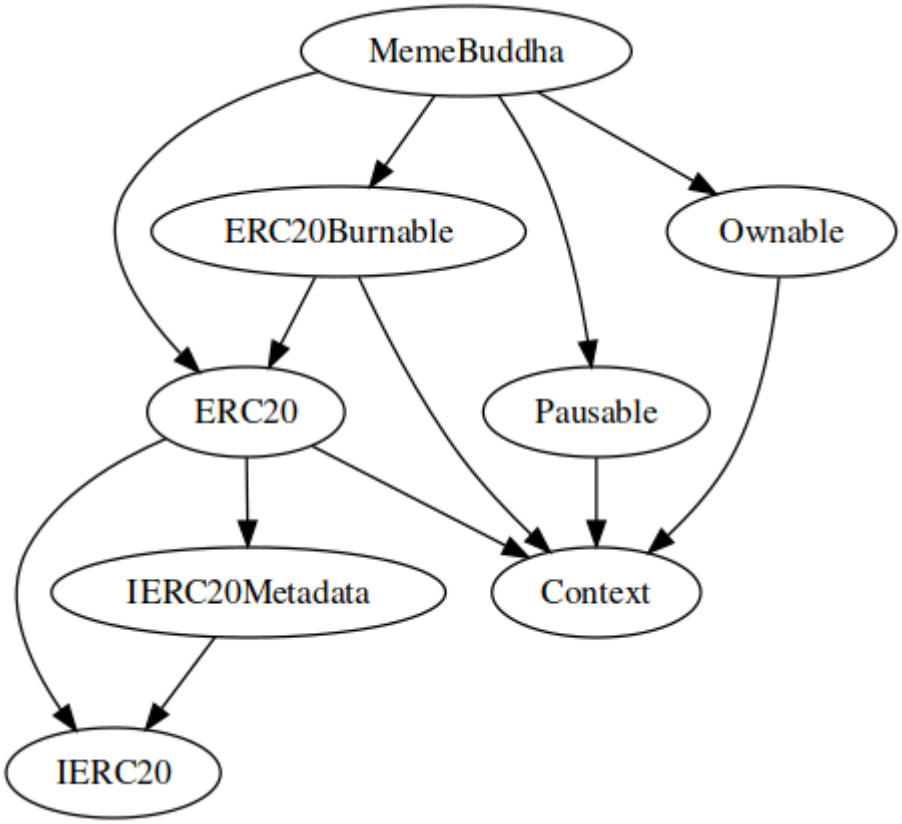
Recommendation

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.

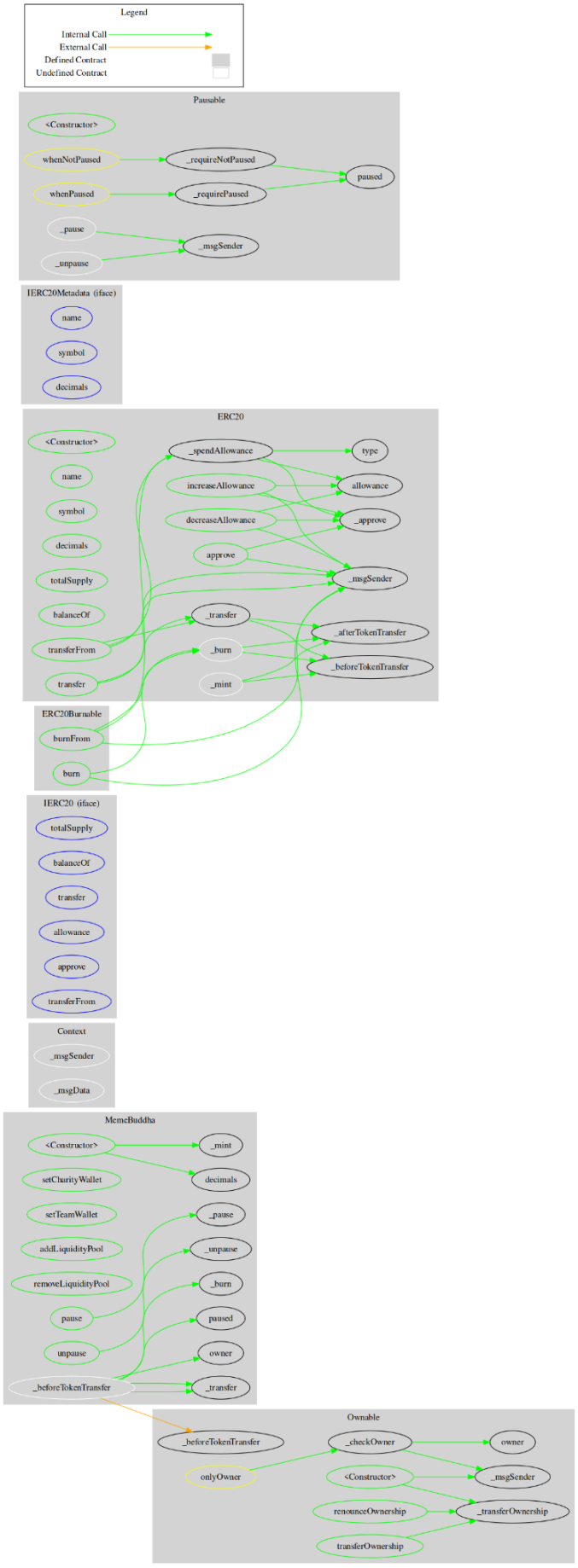
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
MemeBuddha	Implementation	ERC20, ERC20Burnable, Pausable, Ownable		
		Public	✓	ERC20
	setCharityWallet	Public	✓	onlyOwner
	setTeamWallet	Public	✓	onlyOwner
	addLiquidityPool	Public	✓	onlyOwner
	removeLiquidityPool	Public	✓	onlyOwner
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Pausable
<Constructor>
whenNotPaused
_requireNotPaused
paused
whenPaused
_requirePaused
_pause
_msgSender
_unpause
IERC20Metadata (iface)
name
symbol
decimals
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
increaseAllowance
decreaseAllowance
approve
_transfer
_burn
_mint
type
allowance
_approve
_msgSender
_afterTokenTransfer
_beforeTokenTransfer
ERC20Burnable
burnFrom
burn
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
MemeBuddha
<Constructor>
setCharityWallet
setTeamWallet
addLiquidityPool
removeLiquidityPool
pause
unpause
_beforeTokenTransfer
_mint
decimals
_pause
_unpause
_burn
paused
owner
_transfer
Ownable
_beforeTokenTransfer
onlyOwner
_checkOwner
<Constructor>
renounceOwnership
transferOwnership
owner
_msgSender
_transferOwnership

Summary

Meme Buddha 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. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. The fees are fixed to 6% for the sales.

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