



# Smart Contract Audit

FOR  
**MBC\$**  
DATED : 9 June 23'



# HIGH RISK FINDINGS

## Centralization – Trades must be enabled

**Severity:** High

**function:** enableTrading

**Status:** Open

### Overview:

The smart contract owner must enable trades for holders. If trading remain disabled, no one would be able to buy/sell/transfer tokens.

```
function enableTrading() external onlyOwner {  
    tradingActive = true;  
    swapEnabled = true;  
}
```

### Suggestion

To mitigate this centralization issue, we propose the following options:

1. Renounce Ownership: Consider relinquishing control of the smart contract by renouncing ownership. This would remove the ability for a single entity to manipulate the router, reducing centralization risks.
2. Multi-signature Wallet: Transfer ownership to a multi-signature wallet. This would require multiple approvals for any changes to the mainRouter, adding an additional layer of security and reducing the centralization risk.
3. Transfer ownership to a trusted and valid 3<sup>rd</sup> party in order to guarantee enabling of the trades



# HIGH RISK FINDINGS

## Logical – Setting tax wallet to

0x00  
000

Severity: **High**

**function:** updateTaxWallet

**Status:** Open

### Overview:

Setting the tax wallet to the null address (0x00000000000000000000000000000000) may lead to the disruption of sale transactions when the "rescueSwap" feature is active. Specifically, accumulated fees, which should be transferred to the tax wallet, would fail to be processed since the null address is not a valid recipient. Consequently, the entire sale transaction would be reverted due to this internal swap failure.

```
function updateTaxWallet(address newTaxWallet) external onlyOwner {  
    taxWallet = newTaxWallet;  
    emit TaxWalletUpdated(newTaxWallet, taxWallet);  
}  
  
function swapBack() private {  
    uint256 contractBalance = balanceOf(address(this));  
  
    if (rescueSwap) {  
        if (contractBalance > 0) {  
            super._transfer(address(this), taxWallet, contractBalance);  
        }  
        return;  
    }  
    //rest of the code  
}
```

### Suggestion

Prevent the tax wallet from being set to the null address by introducing a requirement check within the updateTaxWallet function. A modified version of the function would look like this:

```
function updateTaxWallet(address newTaxWallet) external onlyOwner {  
    require(newTaxWallet != address(0), "can not set tax wallet to address 0");  
    taxWallet = newTaxWallet;  
  
    emit TaxWalletUpdated(newTaxWallet, taxWallet);  
}
```

# HIGH RISK FINDINGS

## Logical – fees are not resetted

Severity: **High**

**function:** swapBack

**Status:** Open

### Overview:

Currently, the variables **tokensForLiquidity** and **tokensForTax** represent the amount of tokens accumulated for each type of tax. However, the swapBack function fails to reset these two variables to zero after sending the contract token balance to the tax wallet.

While the owner can manually reset these variables to zero using the **resetTaxAmount** function, a failure to perform this action could cause an internal swap to revert. This would occur when the **rescueSwap** flag is set to false and an attempt is made to swap more tokens to BNB than what the contract has.

```
function resetTaxAmount() public onlyOwner {  
    tokensForLiquidity = 0;  
    tokensForTax = 0;  
}  
  
function swapBack() private {  
    uint256 contractBalance = balanceOf(address(this));  
  
    if (rescueSwap) {  
        if (contractBalance > 0) {  
            super._transfer(address(this), taxWallet, contractBalance);  
        }  
        return;  
    }  
    //rest of the code  
}
```

### Suggestion

Ensure that **tokensForLiquidity** and **tokensForTax** are reset to zero in the **swapBack** function. A modified version of the function is as follows:

```
function swapBack() private {  
    uint256 contractBalance = balanceOf(address(this));  
  
    if (rescueSwap) {  
        if (contractBalance > 0) {  
            super._transfer(address(this), taxWallet, contractBalance);  
        }  
        tokensForLiquidity = 0;  
        tokensForTax = 0;  
        return;  
    }  
    //rest of the code  
}
```



# AUDIT SUMMARY

**Project name - MBC\$**

**Date:** 9 June, 2023

**Scope of Audit-** Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

**Audit Status: Passed**

## Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	3	1	0	0
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0



# USED TOOLS

---

## Tools:

### 1- Manual Review:

A line by line code review has been performed by audit ace team.

**2- BSC Test Network:** All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

### 3- Slither :

The code has undergone static analysis using Slither.

### Testnet version:

The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:

<https://testnet.bscscan.com/token/0x43A6eF91394C399F765aF5e2D1b9539C07CF7ede>

---



# Token Information

---

**Token Name :** MemeMillionaires\_\$BoysClub\$

**Token Symbol:** MBC\$

**Decimals:** 9

**Token Supply:** 1,000,000,000,000

**Token Address:**

0x3aa72d5522986eB327521637D45f30DD67eFedcd

**Checksum:**

b0534ddce337345588aa005e78ff6b65a1887a13

**Owner:**

0xFbAeEa4cD5D3aD2c93EFfa5C8eC76aa949cC042

**(at time of writing the audit)**

**Deployer:**

0xFbAeEa4cD5D3aD2c93EFfa5C8eC76aa949cC042

---



# TOKEN OVERVIEW

---

## Fees:

Buy Fees: 0-10%

Sell Fees: 0-10%

Transfer Fees: 0-10%

---

## Fees Privilege: Owner

---

Ownership: 0xFbAeEa4cD5D3aD2c93EFfa5C8eC76aa949cC042

---

## Minting: none

---

## Max Tx Amount/ Max Wallet Amount: No

---

## Blacklist: No

---

## Other Privileges: - initial distribution of tokens

- including or excluding from fees
  - changing swap threshold
  - changing fees
-



# AUDIT METHODOLOGY

---

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code is exercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

# VULNERABILITY CHECKLIST



Return values of low-level calls



**Gasless Send**



Private modifier



Using block.timestamp



Multiple Sends



Re-entrancy



Using Suicide



Tautology or contradiction



Gas Limit and Loops



Timestamp Dependence



Address hardcoded



Revert/require functions



Exception Disorder



Use of tx.origin



Using inline assembly



Integer overflow/underflow



Divide before multiply



Dangerous strict equalities



Missing Zero Address Validation



Using SHA3



Compiler version not fixed



Using throw



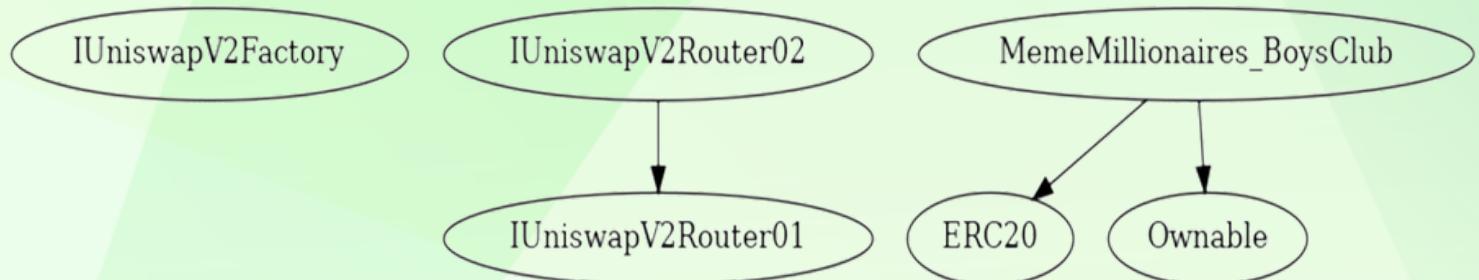
# CLASSIFICATION OF RISK

Severity	Description
◆ Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
◆ High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
◆ Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
◆ Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
◆ Gas Optimization / Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

## Findings

Severity	Found
◆ Critical	0
◆ High-Risk	3
◆ Medium-Risk	1
◆ Low-Risk	0
◆ Gas Optimization / Suggestions	0

# INHERITANCE TREE





## POINTS TO NOTE

---

- owner is able to change fees in range of 0-10% for buy/sell/transfer transactions.
- owner is not able to blacklist an arbitrary wallet
- owner is not able to set limit for buy/sell/transfer/holding amounts
- owner is not able to mint new tokens
- owner is not able to disable trades
- owner must enable trades manually



# CONTRACT ASSESSMENT

Contract	Type	Bases			
L	**Function Name**	**Visibility**	**Mutability**	**Modifiers**	
**IUniswapV2Factory**	Interface				
L   feeTo   External !     NO !					
L   feeToSetter   External !     NO !					
L   getPair   External !     NO !					
L   allPairs   External !     NO !					
L   allPairsLength   External !     NO !					
L   createPair   External !     NO !					
L   setFeeTo   External !     NO !					
L   setFeeToSetter   External !     NO !					
**IUniswapV2Router01**	Interface				
L   factory   External !     NO !					
L   WETH   External !     NO !					
L   addLiquidity   External !     NO !					
L   addLiquidityETH   External !     NO !					
L   removeLiquidity   External !     NO !					
L   removeLiquidityETH   External !     NO !					
L   removeLiquidityWithPermit   External !     NO !					
L   removeLiquidityETHWithPermit   External !     NO !					
L   swapExactTokensForTokens   External !     NO !					
L   swapTokensForExactTokens   External !     NO !					
L   swapExactETHForTokens   External !     NO !					
L   swapTokensForExactETH   External !     NO !					
L   swapExactTokensForETH   External !     NO !					
L   swapETHForExactTokens   External !     NO !					
L   quote   External !     NO !					
L   getAmountOut   External !     NO !					
L   getAmountIn   External !     NO !					
L   getAmountsOut   External !     NO !					
L   getAmountsIn   External !     NO !					
**IUniswapV2Router02**	Interface	IUniswapV2Router01			
L   removeLiquidityETHSupportingFeeOnTransferTokens   External !     NO !					
L   removeLiquidityETHWithPermitSupportingFeeOnTransferTokens   External !     NO !					
L   swapExactTokensForTokensSupportingFeeOnTransferTokens   External !     NO !					
L   swapExactETHForTokensSupportingFeeOnTransferTokens   External !     NO !					
L   swapExactTokensForETHSupportingFeeOnTransferTokens   External !     NO !					



# CONTRACT ASSESSMENT

```
| **MemeMillionaires_BoysClub** | Implementation | ERC20, Ownable |||
| L | <Constructor> | Public ! | (●) | ERC20 |
| L | <Receive Ether> | External ! | (Ξ) | NO !
| L | enableTrading | External ! | (●) | onlyOwner |
| L | airdropToWallets | External ! | (●) | onlyOwner |
| L | decimals | Public ! | | NO !
| L | updateSwapEnabled | External ! | (●) | onlyOwner |
| L | updateRescueSwap | External ! | (●) | onlyOwner |
| L | updateBuyFees | External ! | (●) | onlyOwner |
| L | updateSellFees | External ! | (●) | onlyOwner |
| L | updateTransferFees | External ! | (●) | onlyOwner |
| L | excludeFromFees | Public ! | (●) | onlyOwner |
| L | setAutomatedMarketMakerPair | External ! | (●) | onlyOwner |
| L | _setAutomatedMarketMakerPair | Private (🔒) | (●) ||
| L | updateTaxWallet | External ! | (●) | onlyOwner |
| L | isExcludedFromFees | External ! | | NO !
| L | _transfer | Internal (🔒) | (●) ||
| L | swapTokensForEth | Private (🔒) | (●) ||
| L | addLiquidity | Private (🔒) | (●) ||
| L | resetTaxAmount | Public ! | (●) | onlyOwner |
| L | swapBack | Private (🔒) | (●) ||
```

## ### Legend

Symbol	Meaning
(●)	Function can modify state
(Ξ)	Function is payable



# STATIC ANALYSIS

```
MemeMillionaires.BoysClub._transfer(address,address,uint256) (contracts/Token.sol#1094-1160) has costly operations inside a loop:  
- tokensForLiquidity += fees * buyLiquidityFee / buyTotalFees (contracts/Token.sol#1141)  
MemeMillionaires.BoysClub._transfer(address,address,uint256) (contracts/Token.sol#1094-1160) has costly operations inside a loop:  
- tokensForTax += fees * buyTaxFee / buyTotalFees (contracts/Token.sol#1142)  
MemeMillionaires.BoysClub._transfer(address,address,uint256) (contracts/Token.sol#1094-1160) has costly operations inside a loop:  
- tokensForLiquidity += fees * transferLiquidityFee / transferTotalFees (contracts/Token.sol#1147)  
MemeMillionaires.BoysClub.transfer(address,address,uint256) (contracts/Token.sol#1094-1160) has costly operations inside a loop:  
- tokensForTax += fees * transferTaxFee / transferTotalFees (contracts/Token.sol#1148)  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
Context.msgData() (contracts/Token.sol#20-212) is never used and should be removed  
ERC20.burn(address,uint256) (contracts/Token.sol#608-624) is never used and should be removed  
SafeMath.add(uint256,uint256) (contracts/Token.sol#81-83) is never used and should be removed  
SafeMath.div(uint256,uint256,string) (contracts/Token.sol#175-180) is never used and should be removed  
SafeMath.mod(uint256,uint256) (contracts/Token.sol#139-141) is never used and should be removed  
SafeMath.mod(uint256,uint256,string) (contracts/Token.sol#197-202) is never used and should be removed  
SafeMath.sub(uint256,uint256,string) (contracts/Token.sol#156-161) is never used and should be removed  
SafeMath.tryAdd(uint256,uint256) (contracts/Token.sol#10-16) is never used and should be removed  
SafeMath.tryDiv(uint256,uint256) (contracts/Token.sol#52-57) is never used and should be removed  
SafeMath.tryMod(uint256,uint256) (contracts/Token.sol#64-69) is never used and should be removed  
SafeMath.tryMul(uint256,uint256) (contracts/Token.sol#35-45) is never used and should be removed  
SafeMath.trySub(uint256,uint256) (contracts/Token.sol#23-28) is never used and should be removed  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.17 (contracts/Token.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.16  
solc-0.8.20 is not recommended for deployment  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Low level call in MemeMillionaires.BoysClub.swapBack() (contracts/Token.sol#1200-1239):  
- (success,None) = address(taxWallet).call(value: address(this).balance) () (contracts/Token.sol#1238)  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Function IUniswapV2Router01.WETH() (contracts/Token.sol#716) is not in mixedCase  
Contract MemeMillionaires.BoysClub (contracts/Token.sol#886-1240) is not in CapWords  
Parameter MemeMillionaires.BoysClub.updateBuyFees(uint256,uint256)._taxFee (contracts/Token.sol#1047) is not in mixedCase  
Parameter MemeMillionaires.BoysClub.updateBuyFees(uint256,uint256).liquidityFee (contracts/Token.sol#1047) is not in mixedCase  
Parameter MemeMillionaires.BoysClub.updateSellFees(uint256,uint256).taxFee (contracts/Token.sol#1054) is not in mixedCase  
Parameter MemeMillionaires.BoysClub.updateSellFees(uint256,uint256).liquidityFee (contracts/Token.sol#1054) is not in mixedCase  
Parameter MemeMillionaires.BoysClub.updateTransferFees(uint256,uint256)._taxFee (contracts/Token.sol#1061) is not in mixedCase  
Parameter MemeMillionaires.BoysClub.updateTransferFees(uint256,uint256)._liquidityFee (contracts/Token.sol#1061) is not in mixedCase  
Constant MemeMillionaires.BoysClub.deadAddress (contracts/Token.sol#891) is not in UPPER_CASE_WITH_UNDERSCORES  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,address,uint256).amountDesired (contracts/Token.sol#721) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,address,uint256).amountBDesired (contracts/Token.sol#722)  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar
```

```
MemeMillionaires.BoysClub.constructor() (contracts/Token.sol#943-1007) uses literals with too many digits:  
- _totalSupply = 1000000000000 * 1e18 (contracts/Token.sol#948)  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
MemeMillionaires.BoysClub._decimals (contracts/Token.sol#897) should be immutable  
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

**Result => A static analysis of contract's source code has been performed using slither,  
No major issues were found in the output**



# FUNCTIONAL TESTING

## 1- Adding liquidity (**passed**):

<https://testnet.bscscan.com/tx/0x4dd7acc32dafe5b4fe15abc45ad8870abe6e8ada1e486ec58767be94bf2288f1>

## 2- Buying when excluded from fees (0% tax) (**passed**):

<https://testnet.bscscan.com/tx/0xc25198c384ac2178fe0bc933f9b08a48fd11f8eee7785a25c0021efc2fee464f>

## 3- Selling when excluded from fees (0% tax) (**passed**):

<https://testnet.bscscan.com/tx/0xdda48122b2058b9916793ccfd7ddad0a101c272fbb5ef005bf71782352dc6723>

## 4- Transferring when excluded from fees (0% tax) (**passed**):

<https://testnet.bscscan.com/tx/0x72a1e30409c97afe7eb6613a8aab99c98beab68ebabc4f57a790b470a58598a4>

## 5- Buying when not excluded from fees (0-10% tax) (**passed**):

<https://testnet.bscscan.com/tx/0x8e7ba4f5141ebbc04671aaa4a4adadbpcf57f2cabeffde2088781afea39903e0>

## 6- Selling when not excluded from fees (0-10% tax) (**passed**):

<https://testnet.bscscan.com/tx/0xb75dc46f3464c4ff1ce2f294855de08eb022cc4200a3ee7f6ac3d5830ead51af>



# FUNCTIONAL TESTING

## 7- Transferring when not excluded from fees (0-10% tax) (passed):

<https://testnet.bscscan.com/tx/0xa8fece385caafbfbef285265b7cc2e072f0afd367aac038f7b46a5efa7018cd9>

## 8- Internal swap (passed):

- Tax converted to BNB and sent to tax wallet
- Tax converted to BNB and added to liquidity pool along with an equivalent (in value) number of tokens

<https://testnet.bscscan.com/tx/0xa8fece385caafbfbef285265b7cc2e072f0afd367aac038f7b46a5efa7018cd9>

## 9- Rescue tax (passed):

a feature that directly sends all collected fees to tax wallet instead of swapping those tokens for BNB.

<https://testnet.bscscan.com/tx/0x11ad9227d17a98a25f2c27f20277613958c0d84698b4e1474cb4161635d86ecc>



# HIGH RISK FINDINGS

## Centralization – Trades must be enabled

**Severity:** High

**function:** enableTrading

**Status:** Open

### Overview:

The smart contract owner must enable trades for holders. If trading remain disabled, no one would be able to buy/sell/transfer tokens.

```
function enableTrading() external onlyOwner {  
    tradingActive = true;  
    swapEnabled = true;  
}
```

### Suggestion

To mitigate this centralization issue, we propose the following options:

1. Renounce Ownership: Consider relinquishing control of the smart contract by renouncing ownership. This would remove the ability for a single entity to manipulate the router, reducing centralization risks.
2. Multi-signature Wallet: Transfer ownership to a multi-signature wallet. This would require multiple approvals for any changes to the mainRouter, adding an additional layer of security and reducing the centralization risk.
3. Transfer ownership to a trusted and valid 3<sup>rd</sup> party in order to guarantee enabling of the trades



# HIGH RISK FINDINGS

## Logical – Setting tax wallet to

0x00  
000

Severity: **High**

**function:** updateTaxWallet

**Status:** Open

### Overview:

Setting the tax wallet to the null address (0x00000000000000000000000000000000) may lead to the disruption of sale transactions when the "rescueSwap" feature is active. Specifically, accumulated fees, which should be transferred to the tax wallet, would fail to be processed since the null address is not a valid recipient. Consequently, the entire sale transaction would be reverted due to this internal swap failure.

```
function updateTaxWallet(address newTaxWallet) external onlyOwner {  
    taxWallet = newTaxWallet;  
    emit TaxWalletUpdated(newTaxWallet, taxWallet);  
}  
  
function swapBack() private {  
    uint256 contractBalance = balanceOf(address(this));  
  
    if (rescueSwap) {  
        if (contractBalance > 0) {  
            super._transfer(address(this), taxWallet, contractBalance);  
        }  
        return;  
    }  
    //rest of the code  
}
```

### Suggestion

Prevent the tax wallet from being set to the null address by introducing a requirement check within the updateTaxWallet function. A modified version of the function would look like this:

```
function updateTaxWallet(address newTaxWallet) external onlyOwner {  
    require(newTaxWallet != address(0), "can not set tax wallet to address 0");  
    taxWallet = newTaxWallet;  
  
    emit TaxWalletUpdated(newTaxWallet, taxWallet);  
}
```

# HIGH RISK FINDINGS

## Logical – fees are not resetted

Severity: **High**

**function:** swapBack

**Status:** Open

### Overview:

Currently, the variables **tokensForLiquidity** and **tokensForTax** represent the amount of tokens accumulated for each type of tax. However, the swapBack function fails to reset these two variables to zero after sending the contract token balance to the tax wallet.

While the owner can manually reset these variables to zero using the **resetTaxAmount** function, a failure to perform this action could cause an internal swap to revert. This would occur when the **rescueSwap** flag is set to false and an attempt is made to swap more tokens to BNB than what the contract has.

```
function resetTaxAmount() public onlyOwner {  
    tokensForLiquidity = 0;  
    tokensForTax = 0;  
}  
  
function swapBack() private {  
    uint256 contractBalance = balanceOf(address(this));  
  
    if (rescueSwap) {  
        if (contractBalance > 0) {  
            super._transfer(address(this), taxWallet, contractBalance);  
        }  
        return;  
    }  
    //rest of the code  
}
```

### Suggestion

Ensure that **tokensForLiquidity** and **tokensForTax** are reset to zero in the **swapBack** function. A modified version of the function is as follows:

```
function swapBack() private {  
    uint256 contractBalance = balanceOf(address(this));  
  
    if (rescueSwap) {  
        if (contractBalance > 0) {  
            super._transfer(address(this), taxWallet, contractBalance);  
        }  
        tokensForLiquidity = 0;  
        tokensForTax = 0;  
        return;  
    }  
    //rest of the code  
}
```



# MANUAL TESTING

## Logical – Stuck ETH and Tokens

**Severity:** Medium

**Status:** Open

### Overview:

This contract can receive both ETH and all types of ERC20 tokens. However, there are currently no functions available to withdraw these stuck tokens or ETH. This could result in assets being permanently locked within the contract.

### Suggestion

Recommendation: To resolve this issue, implement withdrawal functions for both ETH and ERC20 tokens. This will allow the contract owner to recover any assets mistakenly sent to the contract address. Here is an example of how such functions might look:

javascript

```
// For ERC20 Tokens
function withdrawTokens(address _tokenContract) external onlyOwner {
    require(_tokenContract != address(this), "can not withdraw native tokens");
    ERC20 token = ERC20(_tokenContract);
    uint256 balance = token.balanceOf(address(this));
    token.transfer(owner, balance);
    emit WithdrawalTokens(owner, balance);
}

// For ETH
function withdrawETH(uint256 amount) external onlyOwner {
    require(amount <= address(this).balance, "Not enough ETH balance");
    payable(owner).transfer(amount);
    emit WithdrawalETH(owner, amount);
}
```

In these functions, `onlyOwner` is a modifier to ensure only the contract owner can execute these functions, preventing unauthorized withdrawals. The `WithdrawalTokens` and `WithdrawalETH` events are emitted after the successful transfer of tokens or ETH to provide transparency and trackability.



# DISCLAIMER

---

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.



# ABOUT AUDITACE

---

We specialize in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.



**<https://auditace.tech/>**



**[https://t.me/Audit\\_Ace](https://t.me/Audit_Ace)**



**[https://twitter.com/auditace\\_](https://twitter.com/auditace_)**



**<https://github.com/Audit-Ace>**

---