Cyberscope

# Audit Report

# I'm Meta Trader

April 2024

Network BSC

Address 0x15bd44875f349f23bd1ebbee8a137bce52cd0b98

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 |
|---|---|---|---|
| Critical | IAF | Inadequate AddLiquidity Function | Unresolved |
| Critical | IRF | Inadequate RemoveLiquidity Function | Unresolved |
| Minor / Informative | IDI | Immutable Declaration Improvement | Unresolved |
| Minor / Informative | RF | Redundant Function | Unresolved |
| Minor / Informative | L02 | State Variables could be Declared Constant | Unresolved |
| Minor / Informative | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| Minor / Informative | L20 | Succeeded Transfer Check | Unresolved |

# Table of Contents

# Review

| | |
|---|---|
| **Contract Name** | BEP20Token |
| **Compiler Version** | v0.5.16+commit.9c3226ce |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0x15bd44875f349f23bd1ebbee8a137bce52cd0b98 |
| **Address** | 0x15bd44875f349f23bd1ebbee8a137bce52cd0b98 |
| **Network** | BSC |
| **Symbol** | IMMT |
| **Decimals** | 8 |
| **Total Supply** | 1,000,000,000 |
| **Badge Eligibility** | Yes |

## Audit Updates

| | |
|---|---|
| **Initial Audit** | 17 Apr 2024 |
| **Corrected Phase 2** | 22 Apr 2024 |

## Source Files

| Filename | SHA256 |
| --- | --- |
| **immt.sol** | 0db0b60e1bdb8d4b78dba5a84f071b742e5a38ab371efd42b1b8e110a9b9f629 |
| **SafeMath.sol** | bb8e16d39f118eda96b401e503c227b0fb42f305de7528d2779f9a158234f488 |
| **IERC20.sol** | 15eb3fb624fd5f079490a80ceab72bfafedc5d1ff55eea910f8960c994cef110 |
| **IBEP20.sol** | 1523ec45a4e33044959922147b3d4838193cfbd8166451004b55959aec63e76e |

# Findings Breakdown


| | |
|---|---|
| Critical | 2 |
| Medium | 0 |
| Minor / Informative | 5 |

| Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|
| Critical | 2 | 0 | 0 | 0 |
| Medium | 0 | 0 | 0 | 0 |
| Minor / Informative | 5 | 0 | 0 | 0 |

## IAF - Inadequate AddLiquidity Function

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | immt.sol#L199 |
| **Status** | Unresolved |

## Description

The function `addLiquidity` within the contract has multiple issues that could potentially lead to unexpected behavior and security vulnerabilities. The primary concerns are related to the validation of token transfers and the improper use of the `addLiquidity` function.

1. The `token` address is not initialized.
2. Inadequate Allowance Validation:
    - The function checks if `msg.sender` has provided sufficient allowance to `PANCAKE_ROUTER` using the `allowance` function. However, the said check is unnecessary, because the tokens are transferred from `msg.sender` to the contract.
    - However, it fails to verify if the actual transfer of tokens from `msg.sender` to the contract `transferFrom` is successful or not.
3. Misuse of router's `addLiquidity` Function:
    - The function is making use of router's function `addLiquidity`, implying that it adds liquidity to a trading pair involving two ERC20 tokens. However, the parameters suggest an attempt to add liquidity using native coins (BNB).
    - This inconsistency in function naming and functionality could cause confusion for developers and users interacting with the contract, leading to unintended usage or exploitation.

```
function addLiquidity(uint256 tokenAmount, uint256 bnbAmount) external {

    require(_balances[msg.sender] >= tokenAmount, "Insufficient token
balance");
    require(allowance(msg.sender, PANCAKE_ROUTER) >= tokenAmount,
"Increase token allowance to PancakeSwap Router");

    token.transferFrom(msg.sender, address(this), tokenAmount);
    token.approve(PANCAKE_ROUTER, tokenAmount);

    pancakeRouter.addLiquidity.value(bnbAmount)(
          address(this),
          tokenAmount,
          tokenAmount,
          bnbAmount,
          msg.sender,
          block.timestamp
    );
}
```

## Recommendation

To address these issues and enhance the security and usability of the smart contract, the following measures are recommended:

1. Ensure Validation of Token Allowance:
    - Verify that the `msg.sender` has given the appriopriate allowance to the contract before proceeding to use `token.transferFrom`. Also, verify the success of the token transfer using appropriate error handling mechanisms such as revert statements or SafeMath operations.
2. Ensure Correct Function Call:
    - Ensure that the correct router's function is called and also that the parameters used are the appriopriate.
    - Consider renaming the function to accurately reflect its intended functionality, whether it involves adding liquidity with ERC20 tokens or native coins (BNB).
    - Provide clear documentation or comments within the code to elucidate the purpose and proper usage of the function, mitigating confusion for developers and users.
3. Handle BNB Transactions Appropriately:

- If the intention is to receive BNB for liquidity provision, ensure the function is marked as payable and implement appropriate logic to handle BNB transfers securely.
- Alternatively, if liquidity provision is solely intended with ERC20 tokens, remove any references to BNB transactions and adjust function parameters and logic accordingly.

By addressing these findings and implementing the recommended measures, the smart contract can achieve improved reliability, security, and user experience, reducing the risk of potential exploits and enhancing trust among stakeholders.

## IRF - Inadequate RemoveLiquidity Function

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | immt.sol#L217 |
| **Status** | Unresolved |

## Description

The function `removeLiquidity` within the contract exhibits a vulnerability that could potentially lead to unexpected behavior.

The primary concern arises from the use of the `uint liquidity` variable as the parameter for the `approve` function. The `approve` function is associated with ERC20 tokens to grant permission for another address to spend tokens on behalf of the owner. However, in this context, the `liquidity` parameter is being treated as an ERC20 token address, which is incorrect.

Upon closer inspection, it becomes apparent that the intention behind this implementation might have been to approve the PancakeSwap Router contract `pancakeRouter` to spend a certain amount of liquidity tokens. However, instead of passing the token address to `approve`, the function is using the `liquidity` variable directly, assuming it represents an ERC20 token address.

This issue is further compounded as the same `liquidity` variable is subsequently used as the amount of liquidity to be removed when calling the `removeLiquidity` function of the PancakeSwap Router.

```
function removeLiquidity(address tokenA, address tokenB, uint liquidity)
external {

IERC20(liquidity).approve(address(pancakeRouter), liquidity);

    pancakeRouter.removeLiquidity(
        tokenA,
        tokenB,
        liquidity,
        0,
        0,
        address(this),
        block.timestamp
    );
}
```

## Recommendation

To address this vulnerability and ensure the security of the smart contract, the following steps are recommended:

1. **Correct Parameter Usage**: Ensure that the `approve` function is called with the correct ERC20 token address as the first parameter, followed by the spender address ( `pancakeRouter` ) and the amount of liquidity tokens to approve.
2. **Verify Token Addresses**: Prior to invoking any ERC20-related functions, validate that the provided token addresses ( `tokenA` , `tokenB` , and `liquidity` ) are indeed ERC20-compliant contract addresses to prevent unintended interactions.
3. **Input Validation**: Implement robust input validation mechanisms to verify the integrity and validity of user-supplied parameters before executing critical functions, mitigating the risk of incorrect behavior and potential exploitation.

# IDI - Immutable Declaration Improvement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | immt.sol#L387,388,389 |
| **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.

```
uint8 private _decimals;
string private _symbol;
string private _name;
```

## 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.

# RF - Redundant Function

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | immt.sol#L311,480 |
| **Status** | Unresolved |

## Description

The contract contains a redundant function `getOwner()` that mirrors the functionality of the `owner()` function. Both functions return the same value - the address of the contract owner. While redundancy itself may not necessarily pose a security risk, it can introduce unnecessary complexity.

Redundant code segments like `getOwner()` may confuse developers or users interacting with the contract. Additionally, redundant functions can increase gas costs and bloating of the contract size without providing any tangible benefit.

```
function owner() public view returns (address) {
    return _owner;
}
...
function getOwner() external view returns (address) {
    return owner();
}
```

## Recommendation

Consider removing the redundant function `getOwner()` and encouraging users to utilize the existing `owner()` function directly. Simplifying the contract interface by eliminating unnecessary duplication enhances readability and improves overall contract efficiency.

## L02 - State Variables could be Declared Constant

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | immt.sol#L149 |
| **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.

```
IBEP20 public token
```

### 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** | immt.sol#L168 |
| **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.

```
uint _tokenAmount
address[] calldata _path
uint _expectedMinOut
```

### 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.

## L20 - Succeeded Transfer Check

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | immt.sol#L173,204 |
| **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.transferFrom(msg.sender, address(this), _tokenAmount)
token.transferFrom(msg.sender, address(this), 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

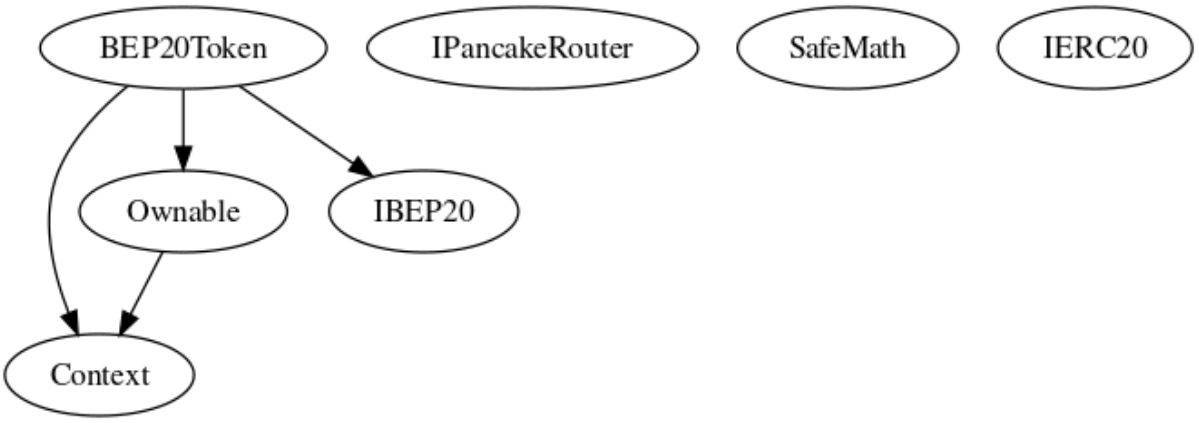
| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **Context** | Implementation | | | |
| | | Internal | ✓ | |
| | _msgSender | Internal | | |
| | _msgData | Internal | | |
| | | | | |
| **Ownable** | Implementation | Context | | |
| | | Internal | ✓ | |
| | owner | Public | | - |
| | renounceOwnership | Public | ✓ | onlyOwner |
| | transferOwnership | Public | ✓ | onlyOwner |
| | _transferOwnership | Internal | ✓ | |
| | | | | |
| **IPancakeRouter** | Interface | | | |
| | getAmountsOut | External | | - |
| | swapExactTokensForTokens | External | ✓ | - |
| | addLiquidity | External | Payable | - |
| | removeLiquidity | External | ✓ | - |
| | | | | |

| | | | | |
|---|---|---|---|---|
| **BEP20Token** | Implementation | Context, IBEP20, Ownable | | |
| | | Public | ✓ | - |
| | swapTokens | External | ✓ | - |
| | addLiquidity | External | ✓ | - |
| | removeLiquidity | External | ✓ | - |
| | getOwner | External | | - |
| | decimals | External | | - |
| | symbol | External | | - |
| | name | External | | - |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | Public | | - |
| | approve | External | ✓ | - |
| | _approve | Internal | ✓ | |
| | transferFrom | External | ✓ | - |
| | _transfer | Internal | ✓ | |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | burn | Public | ✓ | - |
| | _burn | Internal | ✓ | |
| | | | | |
| **SafeMath** | Library | | | |

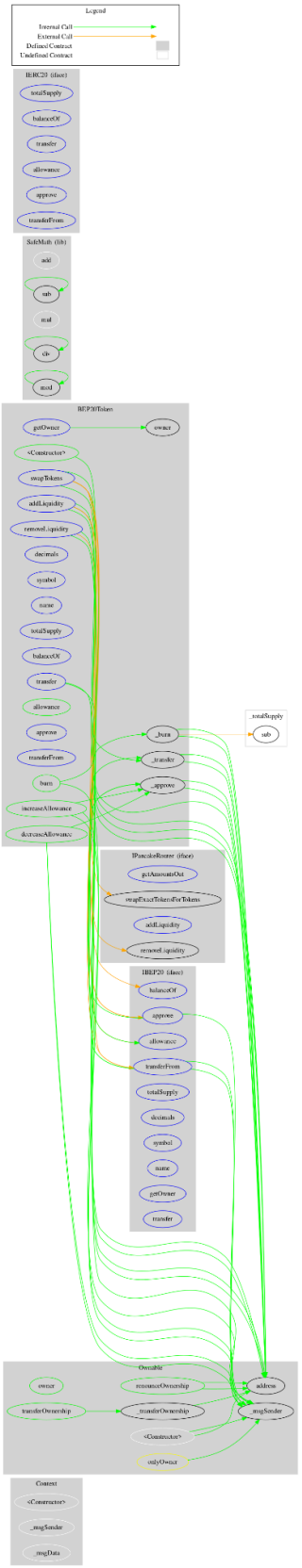
| | | | | |
|---|---|---|---|---|
| | add | Internal | | |
| | sub | Internal | | |
| | sub | Internal | | |
| | mul | Internal | | |
| | div | Internal | | |
| | div | Internal | | |
| | mod | Internal | | |
| | mod | Internal | | |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **IBEP20** | Interface | | | |
| | totalSupply | External | | - |
| | decimals | External | | - |
| | symbol | External | | - |
| | name | External | | - |
| | getOwner | External | | - |

| | | | | |
|---|---|---|---|---|
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
SafeMath (lib)
add
sub
mul
div
mod
BEP20Token
getOwner
owner
<Constructor>
swapTokens
addLiquidity
removeLiquidity
decimals
symbol
name
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
burn
increaseAllowance
decreaseAllowance
_totalSupply
sub
_burn
_transfer
_approve
IPancakeRouter (iface)
getAmountsOut
swapExactTokensForTokens
addLiquidity
removeLiquidity
IBEP20 (iface)
balanceOf
approve
allowance
transferFrom
totalSupply
decimals
symbol
name
getOwner
transfer
Ownable
owner
renounceOwnership
transferOwnership
address
msgSender
<Constructor>
onlyOwner
Context
<Constructor>
_msgSender
_msgData

# Summary

I'm Meta Trader contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.

# 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