# AuditAce
## FROM INCEPTION TO SUCCESS

# Smart Contract Audit

## FOR

# MTKP

**DATED : 17 MAY 23'**

# AUDIT SUMMARY

**Project name** – MTKP

**Date**: 17 May, 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 | 0 | 0 | 3 | 2 |
| Acknowledged | 0 | 0 | 0 | 0 | 0 |
| Resolved | 0 | 0 | 0 | 0 | 0 |

# USED TOOLS

## Tools:

**1.Manual Review:** The code has undergone a line-by-line review by the **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/0xb8373e226b8a6ed1c91845cbffe308703a0da4e0

# Token Information

**Name :** Peg-Martik

**Symbol :** MTKP

**Decimals**: 18

**Network**: BSC

**Token Type**: BEP20

**Token Address:**
0x9Df92A74bC31B40C34Bb5e92Bd48a18d3887ee12

**Owner:**
0xe5726F44AF0cEfD6831e10ebD3Eb6c0aB15233E9

**Deployer**:0xe5726F44AF0cEfD6831e10ebD3Eb6c0a
B15233E9

# Token Information

**Fees:**

Buy Fees: 0-10%

Sell Fees: 0-10%

Transfer Fees: 0-10%

---

**Fees Privilige:** Owner

---

**Ownership** : Owned

---

**Minting:** None

---

**Max Tx Amount/ Max Wallet Amount:** No

---

**Blacklist:** No

---

**Other Priviliges**: Including and excluding form fee - changing dividend token - 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 isexercised 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 Limitand 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

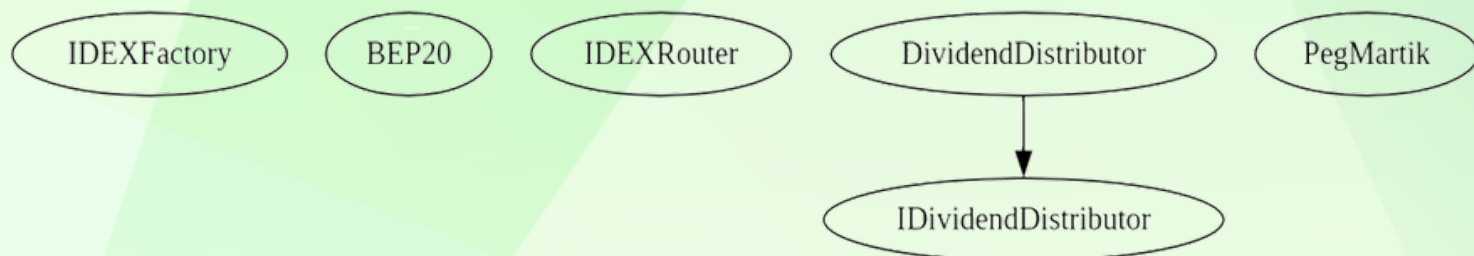| Severity | Found |
|---|---|
| ◆ Critical | 0 |
| ◆ High-Risk | 0 |
| ◆ Medium-Risk | 0 |
| ◆ Low-Risk | 3 |
| ◆ Gas Optimization / Suggestions | 2 |

# INHERITANCE TREE

# POINTS TO NOTE

---

- Owner is not able to change buy/sell/transfer fees over 10% each
- Owner is not able to blacklist an arbitrary address.
- Owner is not able to disable trades
- Owner is not able to set max buy/sell/transfer/hold amount to 0
- Owner is not able to mint new tokens

# CONTRACT ASSESMENT

| Contract | Type | Bases | | | |
|:---------:|:------------------:|:---------------:|:---------------:|:---------------:|
| └ | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
|||||
| **IDEXFactory** | Interface | ||| |
| └ | createPair | External ❗ | 🔴 | NO ❗ |
|||||
| **BEP20** | Interface | ||| |
| └ | balanceOf | External ❗ | | NO ❗ |
| └ | transfer | External ❗ | 🔴 | NO ❗ |
| └ | approve | External ❗ | 🔴 | NO ❗ |
| └ | transferFrom | External ❗ | 🔴 | NO ❗ |
|||||
| **IDEXRouter** | Interface | ||| |
| └ | factory | External ❗ | | NO ❗ |
| └ | WETH | External ❗ | | NO ❗ |
| └ | addLiquidityETH | External ❗ | 💲 | NO ❗ |
| └ | swapExactTokensForETHSupportingFeeOnTransferTokens | External ❗ | 🔴 | NO ❗ |
| └ | swapExactETHForTokensSupportingFeeOnTransferTokens | External ❗ | 💲 | NO ❗ |
| └ | getAmountsOut | External ❗ | | NO ❗ |
|||||
| **IDividendDistributor** | Interface | ||| |
| └ | setDistributionCriteria | External ❗ | 🔴 | NO ❗ |
| └ | setShare | External ❗ | 🔴 | NO ❗ |
| └ | deposit | External ❗ | 💲 | NO ❗ |
| └ | process | External ❗ | 🔴 | NO ❗ |
|||||
| **DividendDistributor** | Implementation | IDividendDistributor ||| |
| └ | <Constructor> | Public ❗ | 🔴 | NO ❗ |
| └ | setDistributionCriteria | External ❗ | 🔴 | onlyToken |
| └ | setShare | External ❗ | 🔴 | onlyToken |
| └ | deposit | External ❗ | 💲 | onlyToken |
| └ | process | External ❗ | 🔴 | onlyToken |
| └ | shouldDistribute | Internal 🔒 | | |
| └ | distributeDividend | Internal 🔒 | 🔴 | |
| └ | claimDividend | External ❗ | 🔴 | onlyToken |
| └ | getUnpaidEarnings | Public ❗ | | NO ❗ |
| └ | getCumulativeDividends | Internal 🔒 | | |
| └ | addShareholder | Internal 🔒 | 🔴 | |
| └ | removeShareholder | Internal 🔒 | 🔴 | |
| └ | setDividendTokenAddress | External ❗ | 🔴 | onlyToken |
|||||
| **PegMartik** | Implementation | ||| |

# CONTRACT ASSESMENT

| └ | \<Constructor\> | Public ❗ | 🔴 | NO ❗ |
| └ | \<Receive Ether\> | External ❗ | 💵 | NO ❗ |
| └ | totalSupply | External ❗ | | NO ❗ |
| └ | owner | Public ❗ | | NO ❗ |
| └ | decimals | External ❗ | | NO ❗ |
| └ | symbol | External ❗ | | NO ❗ |
| └ | name | External ❗ | | NO ❗ |
| └ | getOwner | External ❗ | | NO ❗ |
| └ | balanceOf | Public ❗ | | NO ❗ |
| └ | allowance | External ❗ | | NO ❗ |
| └ | transfer | External ❗ | 🔴 | NO ❗ |
| └ | approve | Public ❗ | 🔴 | NO ❗ |
| └ | transferFrom | External ❗ | 🔴 | NO ❗ |
| └ | setPair | Public ❗ | 🔴 | onlyOwner |
| └ | excludeFromFee | Public ❗ | 🔴 | onlyOwner |
| └ | includeInFee | Public ❗ | 🔴 | onlyOwner |
| └ | setDividendExempt | Public ❗ | 🔴 | onlyOwner |
| └ | isExcludedFromFee | Public ❗ | | NO ❗ |
| └ | _burn | Internal 🔐 | 🔴 | |
| └ | toMartik | External ❗ | 🔴 | NO ❗ |
| └ | toPegMartik | External ❗ | 🔴 | NO ❗ |
| └ | setmigrate | External ❗ | 🔴 | onlyOwner |
| └ | _burnIN | Internal 🔐 | 🔴 | |
| └ | shouldSwapBack | Internal 🔐 | | |
| └ | setmarketingFeeReceivers | External ❗ | 🔴 | onlyOwner |
| └ | setbuytokensReceiver | External ❗ | 🔴 | onlyOwner |
| └ | setSwapBackSettings | External ❗ | 🔴 | onlyOwner |
| └ | value | Public ❗ | | NO ❗ |
| └ | _isSell | Internal 🔐 | | |
| └ | BURNFEE | Internal 🔐 | | |
| └ | MKTFEE | Internal 🔐 | | |
| └ | LIQUIFYFEE | Internal 🔐 | | |
| └ | REFPOOLFEE | Internal 🔐 | | |
| └ | _transferFrom | Internal 🔐 | 🔴 | |
| └ | _basicTransfer | Internal 🔐 | 🔴 | |
| └ | _txTransfer | Internal 🔐 | 🔴 | |
| └ | getamount | Internal 🔐 | | |
| └ | swapBack | Internal 🔐 | 🔴 | swapping |
| └ | setFees | External ❗ | 🔴 | onlyOwner |
| └ | multiTransfer | External ❗ | 🔴 | NO ❗ |
| └ | manualSend | External ❗ | 🔴 | onlyOwner |
| └ | setDistributionCriteria | External ❗ | 🔴 | onlyOwner |

# CONTRACT ASSESMENT

| └ | claimDividend | External ❗ | 🔴 |NO ❗ |
| └ | getUnpaidEarnings | Public ❗ | |NO ❗ |
| └ | setDistributorSettings | External ❗ | 🔴 | onlyOwner |
| └ | setTXBNBgas | External ❗ | 🔴 | onlyOwner |
| └ | setDistribuitorBuyGas | External ❗ | 🔴 | onlyOwner |
| └ | setLiquidifyGas | External ❗ | 🔴 | onlyOwner |
| └ | setDividendToken | External ❗ | 🔴 | onlyOwner |
| └ | renounceOwnership | Public ❗ | 🔴 | onlyOwner |
| └ | transferOwnership | Public ❗ | 🔴 | onlyOwner |
| └ | _transferOwnership | Internal 🔒 | 🔴 | |

Legend

| Symbol | Meaning |
|:--------:|-----------|
| 🔴 | Function can modify state |
| 💵 | Function is payable |

# STATIC ANALYSIS
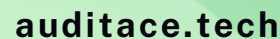
```
Reentrancy in PegMartik.manualSend() (contracts/Token.sol#790-797):
        External calls:
        - address(marketingFeeReceiver).transfer(address(this).balance) (contracts/Token.sol#791)
        State variables written after the call(s):
        - _basicTransfer(address(this),marketingFeeReceiver,balanceOf(address(this))) (contracts/Token.sol#792-796)
                - _balances[sender] = _balances[sender] - amount (contracts/Token.sol#622)
                - _balances[recipient] = _balances[recipient] + amount (contracts/Token.sol#623)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (contracts/Token.sol#624)
                - _basicTransfer(address(this),marketingFeeReceiver,balanceOf(address(this))) (contracts/Token.sol#792-796)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

PegMartik.constructor() (contracts/Token.sol#338-367) uses literals with too many digits:
        - _allowances[address(this)][address(router)] = 100000000 * (10 ** 50) * 100 (contracts/Token.sol#340-343)
PegMartik.setDistributorSettings(uint256) (contracts/Token.sol#816-819) uses literals with too many digits:
        - require(bool)(gas < 3000000) (contracts/Token.sol#817)
PegMartik.setTXBNBgas(uint256) (contracts/Token.sol#821-824) uses literals with too many digits:
        - require(bool)(gas < 100000) (contracts/Token.sol#822)
PegMartik.setDistribuitorBuyGas(uint256) (contracts/Token.sol#826-829) uses literals with too many digits:
        - require(bool)(gas < 1000000) (contracts/Token.sol#827)
PegMartik.setLiquidifyGas(uint256) (contracts/Token.sol#831-834) uses literals with too many digits:
        - require(bool)(gas < 1000000) (contracts/Token.sol#832)
PegMartik.slitherConstructorVariables() (contracts/Token.sol#276-869) uses literals with too many digits:
        - _totalSupply = 100000000000000000000 (contracts/Token.sol#285)
PegMartik.slitherConstructorVariables() (contracts/Token.sol#276-869) uses literals with too many digits:
        - distributorGas = 300000 (contracts/Token.sol#311)
PegMartik.slitherConstructorVariables() (contracts/Token.sol#276-869) uses literals with too many digits:
        - distributorBuyGas = 400000 (contracts/Token.sol#313)
PegMartik.slitherConstructorVariables() (contracts/Token.sol#276-869) uses literals with too many digits:
        - LiquidifyGas = 500000 (contracts/Token.sol#314)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

DividendDistributor.dividendsPerShareAccuracyFactor (contracts/Token.sol#112) should be constant
PegMartik.MTK (contracts/Token.sol#281) should be constant
PegMartik.PoolFee (contracts/Token.sol#299) should be constant
PegMartik.feeDenominator (contracts/Token.sol#310) should be constant
PegMartik.router (contracts/Token.sol#278-279) should be constant
PegMartik.sellPoolFee (contracts/Token.sol#306) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

DividendDistributor.WBNB (contracts/Token.sol#92) should be immutable
DividendDistributor._token (contracts/Token.sol#91) should be immutable
DividendDistributor.router (contracts/Token.sol#102) should be immutable
PegMartik.WBNB (contracts/Token.sol#335) should be immutable
PegMartik.distributor (contracts/Token.sol#277) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

## Static Analysis

an static analysis of the code were performed using slither. No issues were found

# FUNCTIONAL TESTING

**Router (PCS V2):**
**0xD99D1c33F9fC3444f8101754aBC46c52416550D1**

**1- Adding liquidity** (passed):
https://testnet.bscscan.com/tx/0x4b8cf1d429fc4280c40056e45b
cedb2f6187e49d0cfffcce28c195dfbb6c073a

**2- Buying when excluded (0% tax)** (passed):
https://testnet.bscscan.com/tx/0x1291768086170ef0be92f1be0c21
cdf830ae5d943b4e7c8ca5883674c10ed13d

**3- Selling when excluded (0% tax)** (passed):
https://testnet.bscscan.com/tx/0xd1c28057e33f43b4abbb417353
eba63c7add00dad83e5d95b6bb74344be29805

**4- Transferring when excluded from fees (0% tax)** (passed):
https://testnet.bscscan.com/tx/0x96f5af12fbee7bf7ed93364eb36
44be44e81af0943b9204062a96cb4390f3c9e

**5- Buying when not excluded from fees (0-10% tax)** (passed):
https://testnet.bscscan.com/tx/0xb72ae08f770ee1b5cf0a5890e49
71ced4776c36ded6b46d92d0ec5825857db1b

**6- Selling when not excluded from fees (0-10% tax)** (passed):
https://testnet.bscscan.com/tx/0x9ac734eb5fb6061d5466258abd
c017abab3f59272c6cb7718302fb2b37180dc8

**7- Transferring when not excluded from fees (0-10% tax)**
(passed):
https://testnet.bscscan.com/tx/0x0f86bef79275746fecde1498720
03962413e10658e20aa6eeb507625e4c6f114

# FUNCTIONAL TESTING

**8- Internal swap (marketing + rewards)** (passed):
https://testnet.bscscan.com/tx/0x0d1e137a0b30063263c3b23bee3b12b9575dd34438d6de292970b22e0f6f5a45

**9- Bridging** (passed):
**to Matrik**
https://testnet.bscscan.com/tx/0xe4a302af4da6a5647d56b2f81855bb1db3417c2ebf38d0c601059cd7608e38fb
**to peg-matrik**
https://testnet.bscscan.com/tx/0x39568006a416ade0ee8e7bf018d50413094068a40fe21fdd229928622fe44258

# FUNCTIONAL TESTING

## Logical – Flashload attack

**Severity:** Low

**function:** toPegMatric and toMatrik

**Status:** Not Resolved

**Overview:**

The functions 'toPegMartik' and 'toMartik' can be utilized to convert peg-martik to matrik and vice versa. If a significant price or supply disparity exists between the two tokens, a malicious actor could exploit this by purchasing a large quantity of peg-martik, promptly converting them to MTK tokens (or vice versa), and subsequently selling these tokens at a substantially higher price or buying at a lower price. This opens the possibility for a flash loan attack and other arbitrage opportunities that may affect token priec.

## Suggestion

- **Conversion Limit:** Establish a limit on the number of peg-martik tokens that can be converted to MTK (and vice versa) per account within a specified timeframe. This limit could dynamically adjust based on the price gap between the two tokens.

- **Conversion Fee:** Implement a fee for converting between the two tokens. This fee could discourage malicious actors from exploiting the price gap, as it would reduce their potential profit.

- **Disallow Contract Senders:** To prevent flash loan bots from exploiting this vulnerability, ensure that 'msg.sender' is not a contract. This could be implemented with a modifier that checks if the calling address is a contract.

Example of a modifier which only allows calls from an EOA:

```
modifier onlyEOA() {
   require(msg.sender == tx.origin, "Contracts not allowed");
    _;
}
```

# FUNCTIONAL TESTING

## Logical – Locked MTK tokens

**Severity**: **Low**

**function**: toPegMatric and toMatrik and _burnIN

**Status:** Not Resolved

**Overview:**

peg-martik token has burn tax which means a portion of peg-matrik is burned on each transaction which could leave some MTK tokens being having no backed tokens of peg-matrik and be locked in the contract

## Suggestion

since contract acts as a smart bridge, ensure that peg tokens are not burnt

# FUNCTIONAL TESTING

## Logical – Calculation of BNB fees

**Severity**: Low

**function**: swapBack

**Status**: Not Resolved

**Overview:**

swapBack function is using getAmountsOut function of pancake router to retrieve amount of BNB that marketing and reflection contract receive, however this amounts may not be accurate or even be higher than current ETH in the contract

**Suggestion : use below approach to calculate bnb share of marketing and reflections**

```
        uint256 balanceBefore = address(this).balance;

    router.swapExactTokensForETHSupportingFeeOnTransferTokens(

        a,
        0,
        path,
        address(this),
        block.timestamp
    );
    uint256 received = address(this).balance - balanceBefore;
    if (marketing > 0) {
        (bool success, ) = payable(marketingFeeReceiver).call{
            value: (marketing * received) / a,
            gas: txbnbGas
        }("");
        require(success, "Failed to send Ether to marketing fee receiver");
    }
    if (reflection > 0) {
        try
            distributor.deposit{
                value: received - ((marketing * received) / a),
                gas: distributorBuyGas
            }()
        {} catch {}
    }
```

# FUNCTIONAL TESTING

## Informational— Use of MTK as Reward Token in Smart Contract

**Severity:** Informational

**Status:** Not Applicable

### Overview:

The audited smart contract employs **MTK** (**0x116526135380E28836C6080f1997645d5A807FAE**) as a reward token. The specific operations involving the **MTK** token include transferring the token to users and monitoring its balance in the contract.

However, it is important to note that this audit does not cover the MTK token itself. The MTK token has not been evaluated for its functionality, security, or any potential issues that might aris from its use.

### Implications:

The use of an external token like MTK as a reward presents potential risks, such as the reliance on the functionality and security of the MTK token. If the MTK token has vulnerabilities or issues, it could potentially impact the operations of the audited contract. Users interacting with the contract could also be affected.

### Suggestion

While the MTK token is not within the scope of this audit, it is strongly advised to conduct a separate comprehensive audit of the MTK token contract. This will help ensure its security and functionality, thus mitigating potential risks associated with its use in the audited contract.

Furthermore, developers and users should be made aware that the MTK token has not been audited in conjunction with the current contract, and they should exercise caution and due diligence when interacting with it.

# FUNCTIONAL TESTING

## Suggestion – Lack of event emission:

some functions are not emitting any events, this included but not limited to:
1. `toMartik`

2 `toPegMartik`

3. `setPair`

4. `excludeFromFee`

5. `includeInFee`

6. `setDividendExempt`

7. `setmarketingFeeReceivers`

8. `setbuytokensReceiver`

9. `setSwapBackSettings`

10. `setFees`

11. `multiTransfer`

12. `manualSend`

13. `setDistributionCriteria`

14. `claimDividend`

15. `setDistributorSettings`

16. `setTXBNBgas`

17. `setDistribuitorBuyGas`

18. `setLiquidifyGas`

19. `setDividendToken`

20. `renounceOwnership`

21. `transferOwnership`

# 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 specializes 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://twitter.com/auditace_**

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