



Smart Contract Audit

FOR

Peg-Martik

DATED : 16 May 23'



AUDIT SUMMARY

Project name – Peg-Martik

Date: 16 May, 2023

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

Audit Status: Passed with **Critical Risk**

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	1	1	2	0	2
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 done on BSC Test network, each test has its transaction has attached to it.

3- Slither : Static Analysis

Testnet Link: all tests were done using this contract, tests are done on BSC Testnet

<https://testnet.bscscan.com/token/0xab1ed8473404ee2e447e797363d15c8162c72cc4>

Payment Mode:

<https://bscscan.com/tx/0x307bd19710ebac7b72688bd384d933eaf24ed7c86132343814e6d1ec475f47be>



Token Information

Token Name : Peg-Martik

Token Symbol: MTKP

Decimals: 18

Token Supply:100

Token Address:

0x820599Ed2c90868256501B0169cd11c845E933Cc

Checksum:

e584b16580214639fc96ca41ab0b42108f2494ec

Owner:

0xe5726F44AF0cEfD6831e10ebD3Eb6c0aB15233E9



TOKEN OVERVIEW

Fees:

Buy Fees: 0-10 %

Sell Fees: 0-10 %

Transfer Fees: 0-10%

Fees Privilege: owner

Ownership : owned

Minting: No mint function

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

Other Privileges: 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 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 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

1

◆ High-Risk

1

◆ Medium-Risk

2

◆ Low-Risk

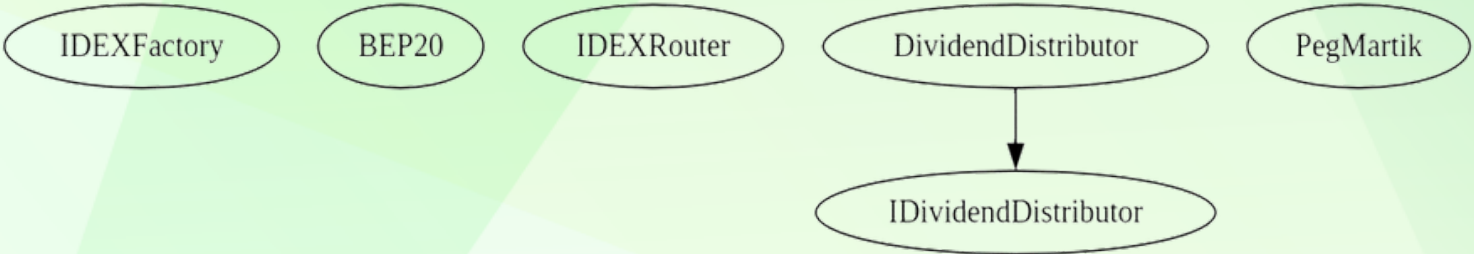
0

◆ 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			
:-----: :-----: :-----: :-----: :-----:					
L	**Function Name**	**Visibility**	**Mutability**	**Modifiers**	
	IDEXFactory	Interface			
L	createPair	External	!	⊗	NO!
	BEP20	Interface			
L	balanceOf	External	!		NO!
L	transfer	External	!	⊗	NO!
L	approve	External	!	⊗	NO!
L	transferFrom	External	!	⊗	NO!
	IDEXRouter	Interface			
L	factory	External	!		NO!
L	WETH	External	!		NO!
L	addLiquidityETH	External	!	👤	NO!
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External	!	⊗	NO!
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External	!	👤	NO!
L	getAmountsOut	External	!		NO!
	IDividendDistributor	Interface			
L	setDistributionCriteria	External	!	⊗	NO!
L	setShare	External	!	⊗	NO!
L	deposit	External	!	👤	NO!
L	process	External	!	⊗	NO!
	DividendDistributor	Implementation	IDividendDistributor		
L	<Constructor>	Public	!	⊗	NO!
L	setDistributionCriteria	External	!	⊗	onlyToken
L	setShare	External	!	⊗	onlyToken
L	deposit	External	!	👤	onlyToken
L	process	External	!	⊗	onlyToken
L	shouldDistribute	Internal	🔒		
L	distributeDividend	Internal	🔒	⊗	
L	claimDividend	External	!	⊗	onlyToken
L	getUnpaidEarnings	Public	!		NO!
L	getCumulativeDividends	Internal	🔒		
L	addShareholder	Internal	🔒	⊗	
L	removeShareholder	Internal	🔒	⊗	
L	setDividendTokenAddress	External	!	⊗	onlyToken

CONTRACT ASSESMENT



```
| **PegMartik** | Implementation | |||
| | <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! |
```



CONTRACT ASSESMENT

^L	manualSend	External !		onlyOwner
^L	setDistributionCriteria	External !		onlyOwner
^L	claimDividend	External !		NO !
^L	getUnpaidEarnings	Public !		NO !
^L	setDistributorSettings	External !		onlyOwner
^L	setTXBNBgas	External !		onlyOwner
^L	setDistributorBuyGas	External !		onlyOwner
^L	setLiquidifyGas	External !		onlyOwner
^L	setDividendToken	External !		onlyOwner
^L	renounceOwnership	Public !		onlyOwner
^L	transferOwnership	Public !		onlyOwner
^L	_transferOwnership	Internal 		

Legend

Symbol	Meaning
	Function can modify state
	Function is payable



```
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.setDistributorBuyGas(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 = 1000000000000000000000000000 (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>

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



FUNCTIONAL TESTING

Router (PCS V2):

0xD99D1c33F9fC3444f8101754aBC46c52416550D1

All the functionalities have been tested, no issues were found

1- Adding liquidity (passed):

<https://testnet.bscscan.com/tx/0x986e69ee82242736fb73489e24fb7b28ac6f5f2d3794d11054049413d4775438>

2- Buying when excluded (0% tax) (passed):

<https://testnet.bscscan.com/tx/0xe8278606a206a11386e14aaf6a86c891a1bcb83afbb92a71bcf3de7e6a644d94>

3- Selling when excluded (0% tax) (passed):

<https://testnet.bscscan.com/tx/0xf9b2be964bc43a621b889002071167402acc4317b1b1455773d325b97c898804>

4- Transferring when excluded (0% tax) (passed):

<https://testnet.bscscan.com/tx/0x2fabf34902f5f6633d2b8ff8b68ebcc54f105c43f03397780bde508187c93ec0>

5- Buying when not excluded (0-10% tax) (passed):

<https://testnet.bscscan.com/tx/0x9a335f5b38f3dc55edcb48dab0c05bab0fd0af3f824e1c9cf121a9bfde3d28ea>

6- Selling when not excluded (0- 10% tax) (passed):

<https://testnet.bscscan.com/tx/0x9b6b7c3d6b29aa947b081594479609bbd2c2278357f65d83fbb8f2c327efd6b4>



FUNCTIONAL TESTING

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

<https://testnet.bscscan.com/tx/0x30bb1b54cd7999aca73fa411edac447381529a766e711fa27aacf9fd91f9c08d>

8- Internal swap (marketing + liquidity + rewards) (passed):

<https://testnet.bscscan.com/tx/0x30bb1b54cd7999aca73fa411edac447381529a766e711fa27aacf9fd91f9c08d>

MANUAL TESTING

Logical – Insufficient BNB for auto-liquidity

Severity: **Critical**

function: swapBack

Status: Not Resolved

Overview:

The function 'swapBack' erroneously treats the variable 'amountToLiquidify' as ETH when adding liquidity to the pair. This is problematic as 'amountToLiquidify' constitutes half of the peg-martik tokens employed for liquidity addition, not ETH. As a result, auto-liquidity will consistently be reverted due to the imbalance.

When 'amountToLiquidify' surpasses the ETH balance of the contract, the transaction will be reverted due to insufficient ETH balance. Even if 'amountToLiquidify' is less than the ETH balance of the contract, the transaction will still be reverted. This is because 'amountBNBLiquidity' is calculated using the 'getAmountsOut' function of PancakeSwap with an input of 'liquidity_tokens / 2'. Subsequently, the entire token balance of the contract is swapped into BNB. This implies that the 'amountBNBLiquidity' value is based on a higher price (due to the smaller input) than the actual ETH in the contract, causing 'amountBNBLiquidity' to exceed the balance of the contract and the transaction to be reverted.

It is noteworthy that the same issue is also present with other types of fees like marketing and reflections, where the calculated BNB through 'getAmountsOut' is less than the actual amount in the contract.

```
if (amountToLiquidify > 0) {  
  router.addLiquidityETH(  
    value: amountToLiquidify <= address(this).balance  
      ? amountBNBLiquidity  
      : address(this).balance,  
    gas: LiquidifyGas  
  )(  
    address(this),  
    amountToLiquidify,  
    0,  
    0,  
    address(this),  
    block.timestamp  
  );  
}
```

Suggestion

To mitigate this logical issue:

- Use address(this).balance as the proper amount to add liquidity with peg-martik tokens
-

MANUAL TESTING

Logical – Wallets can not be excluded from rewards

Severity: **High**

function: setDividendExempt

Status: Not Resolved

Overview:

setDividendExempt is used to exclude an account from receiving rewards from dividend tracker. However if account has already tokens inside it, excluding it from rewards would be not possible since total shares of the account is still set to the previous balance.

```
function setDividendExempt(address account, bool b) public onlyOwner {
    isDividendExempt[account] = b;
}
```

Suggestion

To mitigate this logical issue ensure that you set shares of “account” to zero

```
function setDividendExempt(address account, bool b) public onlyOwner {
    isDividendExempt[account] = b;
    if(b == true){
        distributor.setShare(account, 0);
    }else{
        distributor.setShare(account, balanceOf(account));
    }
}
```

MANUAL TESTING

Logical – Dividend token can be updated

Severity: **Medium**

function: setDividendToken

Status: Not Resolved

Overview:

setDividendToken can be used to update reward token. If reward token is not a contract or is not compatible with ERC20 interface reward system of peg-martik would be affected which could disable all transfers

```
function setDividendToken(address _newContract) external onlyOwner {  
    require(_newContract != address(0));  
    distributor.setDividendTokenAddress(_newContract);  
}
```

Suggestion

It's suggested to transfer ownership of the contract to a multi-signature wallet which means updating the state of the contract has to be approved by several wallets.

MANUAL TESTING

Logical – Flashload attack

Severity: **Medium**

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 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 (with zero fees using the 'toMartik' function if sufficient MTK tokens are available in the contract), and subsequently selling these tokens at a substantially higher price. This opens the possibility for a flash loan attack.

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");  
    _;  
}
```

MANUAL 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 arise 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.

MANUAL 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 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://twitter.com/auditace_



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