auditace.tech

AuditAce

FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

Martik

DATED : 24 FEB 23'

AUDIT SUMMARY

Project name – Listen AI

Date: 24 February, 2023

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

Audit Status: Passed with High Risk

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	1	1	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/0xcfcf24c5fb28dad1d2f0bbcc1f53df7ac467f9f4

Token Information

Token Name : Martik

Token Symbol: MTK

Decimals: 18

Token Supply: 10,000

Token Address:
0x116526135380E28836C6080f1997645d5A807FAE

Checksum:
4f73c6a95b91773c428b406bc946e473fcd2518e

Owner:
0xBF081317630FCD484068A50287620A58f38BbC33

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.

auditace.tech

VULNERABILITY CHECKLIST

- Return values of low-level calls
- Private modifier
- Multiple Sends
- Using Suicide
- Gas Limitand Loops
- Address hardcoded
- Exception Disorder
- Using inline assembly
- Divide before multiply
- Missing Zero Address Validation
- Compiler version not fixed
- **Gasless Send**
- Using block.timestamp
- Re-entrancy
- Tautology or contradiction
- Timestamp Dependence
- Revert/require functions
- Use of tx.origin
- Integer overflow/underflow
- Dangerous strict equalities
- Using SHA3
- 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	1
Medium-Risk	1
Low-Risk	0
Gas Optimization / Suggestions	2

INHERITANCE TREE

POINTS TO NOTE

- Owner is able to set buy/sell/transfer taxes up to 100%
- Owner is not able to set max tx or max wallet
- Owner is not able to disable trades
- Owner is not able to mint new tokens

CONTRACT ASSESMENT

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IDEXFactory	Interface			
└	createPair	External ❗	🛑	NO❗
IDEXRouter	Interface			
└	factory	External ❗		NO❗
└	WETH	External ❗		NO❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Martik	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
└	isExcludedFromFee	Public ❗		NO❗
└	burn	External ❗	🛑	NO❗
└	_burn	Internal 🔒	🛑	
└	_burnIN	Internal 🔒	🛑	
└	shouldSwapBack	Internal 🔒		
└	setecosystemFeeReceivers	External ❗	🛑	onlyOwner
└	setAutoCompoundFeeReceivers	External ❗	🛑	onlyOwner
└	setSwapBackSettings	External ❗	🛑	onlyOwner
└	value	Public ❗		NO❗
└	_isSell	Internal 🔒		
└	BURNFEE	Internal 🔒		
└	ECOFEE	Internal 🔒		
└	_transferFrom	Internal 🔒	🛑	

CONTRACT ASSESMENT

└	_basicTransfer	Internal 🔒	🛑	
└	_txTransfer	Internal 🔒	🛑	
└	swapBack	Internal 🔒	🛑	swapping
└	setFees	External ❗	🛑	onlyOwner
└	manualSend	External ❗	🛑	onlyOwner
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

STATIC ANALYSIS

```
Function IDEXRouter.WETH() (contracts/Token.sol#24) is not in mixedCase
Parameter Martik.setPair(address,bool)._pair (contracts/Token.sol#154) is not in mixedCase
Parameter Martik.setecosystemFeeReceivers(address)._ecosystemFeeReceiver (contracts/Token.sol#196) is not in mixedCase
Parameter Martik.setAutoCompoundFeeReceivers(address)._autoCompoundFeeReceiver (contracts/Token.sol#202) is not in mixedCase
Parameter Martik.setSwapBackSettings(bool)._enabled (contracts/Token.sol#207) is not in mixedCase
Function Martik.BURNFEE(bool) (contracts/Token.sol#226-232) is not in mixedCase
Function Martik.ECOFEE(bool) (contracts/Token.sol#234-240) is not in mixedCase
Parameter Martik.setFees(uint256,uint256,uint256,uint256)._ecoFee_B (contracts/Token.sol#322) is not in mixedCase
Parameter Martik.setFees(uint256,uint256,uint256,uint256)._burnFee_B (contracts/Token.sol#323) is not in mixedCase
Parameter Martik.setFees(uint256,uint256,uint256,uint256)._ecoFee_S (contracts/Token.sol#324) is not in mixedCase
Parameter Martik.setFees(uint256,uint256,uint256,uint256)._burnFee_S (contracts/Token.sol#325) is not in mixedCase
Constant Martik._name (contracts/Token.sol#36) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Martik._symbol (contracts/Token.sol#37) is not in UPPER_CASE_WITH_UNDERSCORES
Constant Martik._decimals (contracts/Token.sol#38) is not in UPPER_CASE_WITH_UNDERSCORES
Variable Martik._totalSupply (contracts/Token.sol#39) is not in mixedCase
Variable Martik._balances (contracts/Token.sol#40) is not in mixedCase
Variable Martik._allowances (contracts/Token.sol#41) is not in mixedCase
Variable Martik._isExcludedFromFee (contracts/Token.sol#42) is not in mixedCase
Variable Martik.ecoFee_BUY (contracts/Token.sol#48) is not in mixedCase
Variable Martik.burnFee_BUY (contracts/Token.sol#49) is not in mixedCase
Variable Martik.ecoFee_SELL (contracts/Token.sol#51) is not in mixedCase
Variable Martik.burnFee_SELL (contracts/Token.sol#52) is not in mixedCase
Variable Martik.WBNB (contracts/Token.sol#71) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Reentrancy in Martik._transferFrom(address,address,uint256) (contracts/Token.sol#242-276):
        External calls:
        - swapBack(ecoFeeAmount) (contracts/Token.sol#258)
                - address(ecosystemFeeReceiver).transfer(amountBNB) (contracts/Token.sol#318)
        State variables written after the call(s):
        - _balances[sender] = _balances[sender] - amount (contracts/Token.sol#271)
        - _balances[recipient] = _balances[recipient] + amountWithFee (contracts/Token.sol#272)
        - _burnIN(sender,burnFeeAmount) (contracts/Token.sol#267)
                - _totalSupply = _totalSupply - amount (contracts/Token.sol#183)
        Event emitted after the call(s):
        - Transfer(account,address(0),amount) (contracts/Token.sol#184)
                - _burnIN(sender,burnFeeAmount) (contracts/Token.sol#267)
        - Transfer(sender,recipient,amountWithFee) (contracts/Token.sol#273)
Reentrancy in Martik.manualSend() (contracts/Token.sol#335-342):
        External calls:
        - address(ecosystemFeeReceiver).transfer(address(this).balance) (contracts/Token.sol#336)
        State variables written after the call(s):
        - _basicTransfer(address(this),ecosystemFeeReceiver,balanceOf(address(this))) (contracts/Token.sol#337-341)
                - _balances[sender] = _balances[sender] - amount (contracts/Token.sol#284)
                - _balances[recipient] = _balances[recipient] + amount (contracts/Token.sol#285)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (contracts/Token.sol#286)
                - _basicTransfer(address(this),ecosystemFeeReceiver,balanceOf(address(this))) (contracts/Token.sol#337-341)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

Martik.feeDenominator (contracts/Token.sol#53) should be constant
Martik.router (contracts/Token.sol#69-70) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

Martik.WBNB (contracts/Token.sol#71) 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

Tests are done for current state of the token, some tests may fail depending on contract state, please refer to Findings sections

1- Adding Liquidity (Passed):
liquidity added on Pancakeswap V2:

https://testnet.bscscan.com/tx/0xddef5cd25629a6cec99c36344fc9564254897d2a007992a5d79857b532152bc1

no issue were found on adding liquidity.

2- Buying (10% tax, 5% burning and 5% fee receiver) (Passed):

https://testnet.bscscan.com/tx/0x209e6c3e3246923bf507925f61150c76458d6fab7d65b66974f924cb90bc8f66

3- Selling (10% tax, 5% burning and 5% swap back) (Passed):

https://testnet.bscscan.com/tx/0x6f17f3da3c64ac1ca982cd2e9a5106fc487c8e01643072f51f148af1c88dbbc7

4-Transferring (10% tax, 5% burning and 5% swap back) (Passed):

https://testnet.bscscan.com/tx/0x553a8238a0563821393fa4007c40c43c667b03b634ab9bd89505d99b24c50aa8

MANUAL TESTING

HIGH RISK FINDINGS

Centralization – Owner is able to set buy/sell/transfer taxes up to 100%:

```
function setFees(
    uint256 _ecoFee_B,
    uint256 _burnFee_B,
    uint256 _ecoFee_S,
    uint256 _burnFee_S
) external onlyOwner {
    ecoFee_BUY = _ecoFee_B;
    burnFee_BUY = _burnFee_B;
    ecoFee_SELL = _ecoFee_S;
    burnFee_SELL = _burnFee_S;
    buyTax = _ecoFee_B + _burnFee_B;
    sellTax = _ecoFee_S + _burnFee_S;
}
```

Suggestions –Set a reasonable limit for fees

MANUAL TESTING

MEDIUM RISK FINDINGS

Logical – setting ecosystem fee to 0 can disable sells, since at _transfeFrom function, there is no checks to ensure that contract token balance is greater than 0.

```
swapThreshold = balanceOf(address(this));
if (shouldSwapBack()) {
    swapBack(ecoFeeAmount);
} else {
    balances[address(this)] =
```

If ecosystem fee is zero, then balanceOf(address(this)) is zero, but swapback would still be called

Suggestions –before calling swapBack, ensure than: balanceOf(address(this)) > 0

MANUAL TESTING

SUGGESTIONS AND OPTIMIZATIONS

Optimization – there is not need to set swapThreshold to contract's balance again, since this is done before entering swapBack function

```
function swapBack(uint256 amount) internal swapping {
    uint256 a = amount;
    if (a <= swapThreshold) {
        a = amount;
    } else {
        a = swapThreshold;
    }
    swapThreshold = balanceOf(address(this));
```

Optimization – declare **"router"** and **"WETH"** variables as constant

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