auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

JOP

DATED : 23 MAY 23'

AUDIT SUMMARY

Project name – JOP

Date: 23 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	0	1	2	0
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.ETH Test Network: All tests were conducted on the ETH 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/0x179e0e68934f304fa27b4217fb006a020605b3db#code

Token Information

Name : JOKER PEPE

Symbol : JOP

Decimals: 18

Network: BSC

Token Type:BEP20

Token Address: ---

Owner: ---(at time of writing the audit)

Deployer:---

Token Information

Fees:

Buy Fees: 0-99%

Sell Fees: 0-99%

Transfer Fees: 0%

Fees Privilige: Owner

Ownership : Owned

Minting: None

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

Other Priviliges:- Modifying fees

- including in fees

- excluding from fees

- initial distribution of the tokens

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
- 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	1
High-Risk	0
Medium-Risk	1
Low-Risk	2
Gas Optimization / Suggestions	0

INHERITANCE TREE

POINTS TO NOTE

- Owner is able to change fees 0-99% for buy and sells (0% transfer fee)
- Owner is not able to blacklist an arbitrary address.
- Owner is not able to disable trades
- Owner is not able to limit buy/sell/transfer/wallet amounts
- Owner is not able to mint new tokens

CONTRACT ASSESMENT

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	transfer	External ❗	🛑	NO ❗
└	allowance	External ❗		NO ❗
└	approve	External ❗	🛑	NO ❗
└	transferFrom	External ❗	🛑	NO ❗
Token	Interface			
└	transferFrom	External ❗	🛑	NO ❗
└	transfer	External ❗	🛑	NO ❗
IUniswapV2Factory	Interface			
└	createPair	External ❗	🛑	NO ❗
IUniswapV2Router02	Interface			
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
└	factory	External ❗		NO ❗
└	WETH	External ❗		NO ❗
└	addLiquidityETH	External ❗	💵	NO ❗
Context	Implementation			
└	_msgSender	Internal 🔒		
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO ❗
└	owner	Public ❗		NO ❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
JOKERPEPE	Implementation	Context, IERC20, Ownable		
└	<Constructor>	Public ❗	🛑	NO ❗

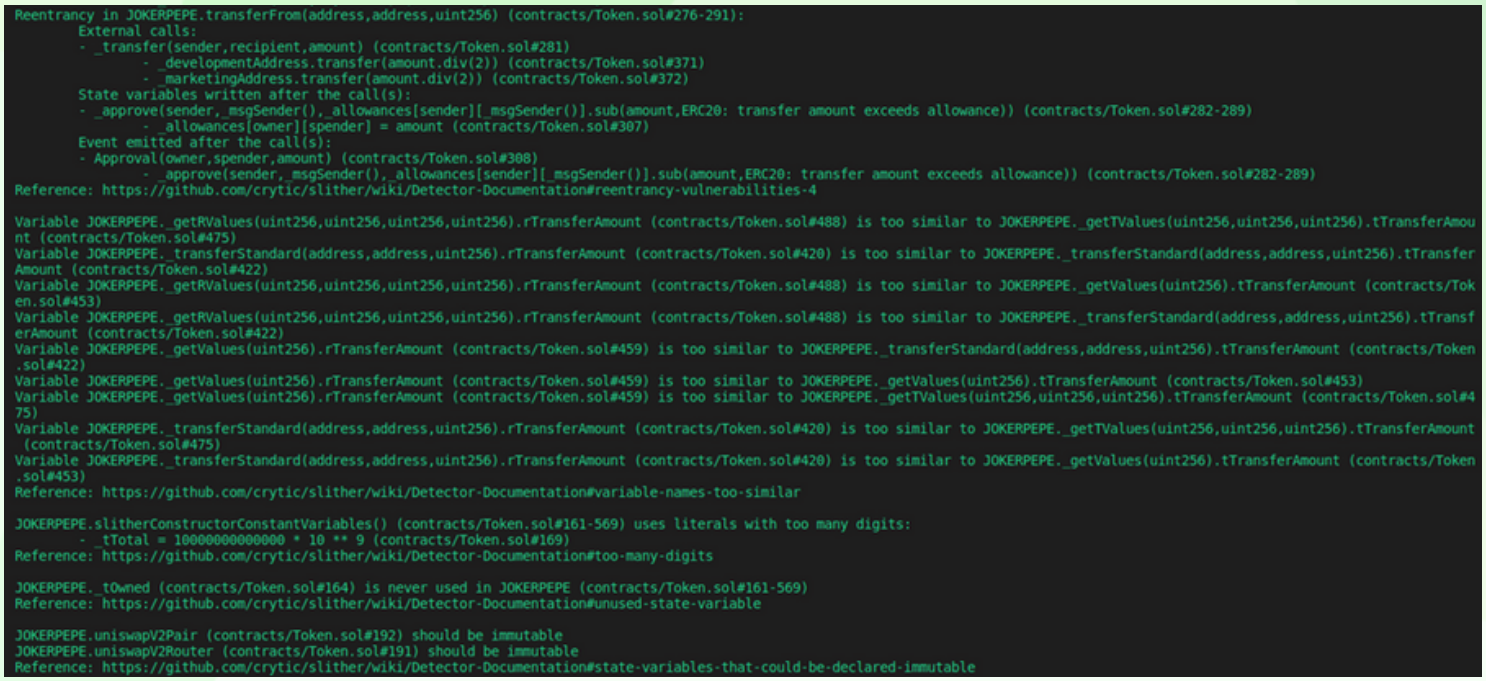
CONTRACT ASSESMENT

┗	name	Public ❗	|NO ❗
┗	symbol	Public ❗	|NO ❗
┗	decimals	Public ❗	|NO ❗
┗	totalSupply	Public ❗	|NO ❗
┗	balanceOf	Public ❗	|NO ❗
┗	transfer	Public ❗	🛑 |NO ❗
┗	allowance	Public ❗	|NO ❗
┗	approve	Public ❗	🛑 |NO ❗
┗	transferFrom	Public ❗	🛑 |NO ❗
┗	tokenFromReflection	Private 🔐		
┗	_approve	Private 🔐	🛑	
┗	_transfer	Private 🔐	🛑	
┗	swapTokensForEth	Private 🔐	🛑	lockTheSwap
┗	sendETHToFee	Private 🔐	🛑	
┗	_tokenTransfer	Private 🔐	🛑	
┗	rescueForeignTokens	Public ❗	🛑	onlyDev
┗	setNewDevAddress	Public ❗	🛑	onlyDev
┗	setNewMarketingAddress	Public ❗	🛑	onlyDev
┗	_transferStandard	Private 🔐	🛑	
┗	_takeTeam	Private 🔐	🛑	
┗	_reflectFee	Private 🔐	🛑	
┗	<Receive Ether>	External ❗	💵 |NO ❗
┗	_getValues	Private 🔐		
┗	_getTValues	Private 🔐		
┗	_getRValues	Private 🔐		
┗	_getRate	Private 🔐		
┗	_getCurrentSupply	Private 🔐		
┗	manualswap	External ❗	🛑 |NO ❗
┗	manualsend	External ❗	🛑 |NO ❗
┗	setFee	Public ❗	🛑	onlyDev
┗	toggleSwap	Public ❗	🛑	onlyDev
┗	excludeMultipleAccountsFromFees	Public ❗	🛑	onlyOwner
┗	claimShare	Public ❗	🛑 |NO ❗
┗	getShare	Public ❗	|NO ❗

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

STATIC ANALYSIS



```
Reentrancy in JOKERPEPE.transferFrom(address,address,uint256) (contracts/Token.sol#276-291):
        External calls:
        - _transfer(sender,recipient,amount) (contracts/Token.sol#281)
                - _developmentAddress.transfer(amount.div(2)) (contracts/Token.sol#371)
                - _marketingAddress.transfer(amount.div(2)) (contracts/Token.sol#372)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (contracts/Token.sol#282-289)
                - _allowances[owner][spender] = amount (contracts/Token.sol#307)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (contracts/Token.sol#308)
                - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (contracts/Token.sol#282-289)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

Variable JOKERPEPE._getRValues(uint256,uint256,uint256,uint256).rTransferAmount (contracts/Token.sol#488) is too similar to JOKERPEPE._getTValues(uint256,uint256,uint256).tTransferAmou
nt (contracts/Token.sol#475)
Variable JOKERPEPE._transferStandard(address,address,uint256).rTransferAmount (contracts/Token.sol#420) is too similar to JOKERPEPE._transferStandard(address,address,uint256).tTransfer
Amount (contracts/Token.sol#422)
Variable JOKERPEPE._getRValues(uint256,uint256,uint256,uint256).rTransferAmount (contracts/Token.sol#488) is too similar to JOKERPEPE._getValues(uint256).tTransferAmount (contracts/Tok
en.sol#453)
Variable JOKERPEPE._getRValues(uint256,uint256,uint256,uint256).rTransferAmount (contracts/Token.sol#488) is too similar to JOKERPEPE._transferStandard(address,address,uint256).tTransf
erAmount (contracts/Token.sol#422)
Variable JOKERPEPE._getValues(uint256).rTransferAmount (contracts/Token.sol#459) is too similar to JOKERPEPE._transferStandard(address,address,uint256).tTransferAmount (contracts/Token
.sol#422)
Variable JOKERPEPE._getValues(uint256).rTransferAmount (contracts/Token.sol#459) is too similar to JOKERPEPE._getValues(uint256).tTransferAmount (contracts/Token.sol#453)
Variable JOKERPEPE._getValues(uint256).rTransferAmount (contracts/Token.sol#459) is too similar to JOKERPEPE._getTValues(uint256,uint256,uint256).tTransferAmount (contracts/Token.sol#4
75)
Variable JOKERPEPE._transferStandard(address,address,uint256).rTransferAmount (contracts/Token.sol#420) is too similar to JOKERPEPE._getTValues(uint256,uint256,uint256).tTransferAmount
 (contracts/Token.sol#475)
Variable JOKERPEPE._transferStandard(address,address,uint256).rTransferAmount (contracts/Token.sol#420) is too similar to JOKERPEPE._getValues(uint256).tTransferAmount (contracts/Token
.sol#453)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

JOKERPEPE.slitherConstructorConstantVariables() (contracts/Token.sol#161-569) uses literals with too many digits:
        - _tTotal = 10000000000000 * 10 ** 9 (contracts/Token.sol#169)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

JOKERPEPE._tOwned (contracts/Token.sol#164) is never used in JOKERPEPE (contracts/Token.sol#161-569)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

JOKERPEPE.uniswapV2Pair (contracts/Token.sol#192) should be immutable
JOKERPEPE.uniswapV2Router (contracts/Token.sol#191) 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/0xe7c6b891e474e0d3552645bd709e457f76a372441f05fe8cb7859c2b200194ff

2- Buying when excluded (0% tax) (passed):
https://testnet.bscscan.com/tx/0x283e455dc792be8ab20faa48a18a817b20987529c797ca1db0cc13c662ee888d

3- Selling when excluded (0% tax) (passed):
https://testnet.bscscan.com/tx/0x07bbd33d9b73872760dc1caee19af472d9e68b84df87f5ad0d92447b5d72b987

4- Transferring when excluded from fees (0% tax) (passed):
https://testnet.bscscan.com/tx/0xc0140316663cf7362c77e7b1b350826ff5e6fa7eb42a039e8b7c55df82276427

5- Buying from a regular wallet (0-99% tax) (passed):
https://testnet.bscscan.com/tx/0x3fdece0137da97649c83a7e7996752a2acefcd4ead47840456cb7f59a8e8fcf0

6- Selling from a regular wallet (0-99% tax) (passed):
https://testnet.bscscan.com/tx/0x3676b31bc6df9ff22ea5c1cf009193f3cb56c0b810d03aa9fc21c707aff1915e

7- Transferring from a regular wallet (0-99% tax) (passed):
https://testnet.bscscan.com/tx/0x8d799d9ed5f910c7e818721eb50c55bf282786b119aaf87e5bbf7b7784c54cdb

FUNCTIONAL TESTING

7- Internal swap (marketing and development bnb)(passed):
https://testnet.bscscan.com/address/0xB68c9FdD918bf7a186a1b236B30eDf0eaBdAdaFc#internaltx

FUNCTIONAL TESTING

Issue Category: Centralization – Excessive Fee

Severity: Critical

Function: setFee

Status: Not Resolved

Overview: The function setFee allows the owner to set the transaction fees for buying and selling. However, it is possible for the owner to set the fee to an excessive amount (up to 99%) which can be detrimental to the users of the token.

Code:

```
solidity

function setFee(
    uint256 redisFeeOnBuy,
    uint256 redisFeeOnSell,
    uint256 taxFeeOnBuy,
    uint256 taxFeeOnSell
) public onlyDev {
    require(redisFeeOnBuy < 100);
    require(redisFeeOnSell < 100);
    require(taxFeeOnBuy < 100);
    require(taxFeeOnSell < 100);
    _redisFeeOnBuy = redisFeeOnBuy;
    _redisFeeOnSell = redisFeeOnSell;
    _taxFeeOnBuy = taxFeeOnBuy;
    _taxFeeOnSell = taxFeeOnSell;
}
```

Suggestion: Limit the maximum fee that can be set to prevent the owner from setting an excessively high fee. This can be done by modifying the require statements to a reasonable percentage.

Buy Fee <= 10

Sell Fee <= 10

Tranfser Fee <= 10

FUNCTIONAL TESTING

Issue Category: Efficiency - High Slippage Risk

Severity: Medium

Function: _transfer

Status: Not Resolved

Overview: In the _transfer function, it appears that the contract will try to swap all tokens in its balance for ETH whenever a transfer occurs that does not involve the owner. This can potentially lead to a situation where, if a large amount of tokens are accumulated in the contract, a huge amount of tokens will be swapped at once, leading to a high slippage. Its expected that during launch of the token, a huge amount of token get accumulated in the contract depending on the buy volume which leads to a high slippage in sell transactions (8-49%).

Code:

solidity

```
function _transfer(address from, address to, uint256 amount) private {
  require(from != address(0), "ERC20: transfer from the zero address");
  require(to != address(0), "ERC20: transfer to the zero address");
  require(amount > 0, "Transfer amount must be greater than zero");

  _redisFee = 0;
  _taxFee = 0;

  if (from != owner() && to != owner()) {
    uint256 contractTokenBalance = balanceOf(address(this));
    if (
      !inSwap &&
      from != uniswapV2Pair &&
      swapEnabled &&
      contractTokenBalance > 0
    ) {
      swapTokensForEth(contractTokenBalance);
      uint256 contractETHBalance = address(this).balance;
      if (contractETHBalance > 0) {
        sendETHToFee(address(this).balance);
      }
    }
  }
}
```

Suggestion: To mitigate the risk of high slippage, you might consider setting a swap threshold, i.e., a maximum amount of tokens that can be swapped at any single transaction. This can help prevent the contract from swapping a massive amount of tokens at once, thus preventing an excessive impact on the token's price.

FUNCTIONAL TESTING

Issue Category: Efficiency – Unused Code

Severity: Informational

Function: claimShare, getShare

Status: Not Resolved

Overview: The contract contains unused code related to token holders' shares. This can lead to confusion and inefficiency in contract execution.

Code:

```
mapping(address => uint256) private _tokenHolders;

function claimShare() public {
   uint256 share = _tokenHolders[msg.sender];
   require(share > 0, "No share available to claim");
   _tokenHolders[msg.sender] = 0;
   _transfer(address(this), msg.sender, share);
}

function getShare(address holder) public view returns (uint256) {
   return _tokenHolders[holder];
}
```

Suggestion: Consider removing or updating unused code to improve contract efficiency and readability. If this is legacy code, ensure it is appropriately deprecated to avoid future misunderstandings.

FUNCTIONAL TESTING

Issue Category: Centralization – Unrestricted Withdrawal

Severity: Informational

Function: rescueForeignTokens

Status: Not Applicable

Overview: The owner has unrestricted access to withdraw any tokens from the contract. This poses a risk as it allows the owner to withdraw native tokens from the contract

Code:

```
function rescueForeignTokens(
    address _tokenAddr,
    address _to,
    uint _amount
) public onlyDev {
    emit tokensRescued(_tokenAddr, _to, _amount);
    Token(_tokenAddr).transfer(_to, _amount);
}
```

Suggestion: Implement checks and balances on the owner's ability to withdraw tokens from the contract. This could be achieved by establishing multisig control, time locks, or by setting a withdrawal limit.

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