Lord Of The Inu

Token Audit Report

CONTENTS

OVERVIEW

This document is a security audit of the contract Token.sol included with this delivery.

Following files / results are included with this delivery:

1. Original Contract Code
2. Report
3. Contract Overview diagram
4. Custom Test Result

GLOSSARY

The automated tests are performed against a knowledgebase of commonly known issues and assigned a SEVERITY as per the security issue.

Severity is categorized into three levels starting from 1 up to 3. Higher the number, higher is the threat.

- Severity 1 - Low threat
- Severity 2 - Medium threat
- Severity 3 - High threat

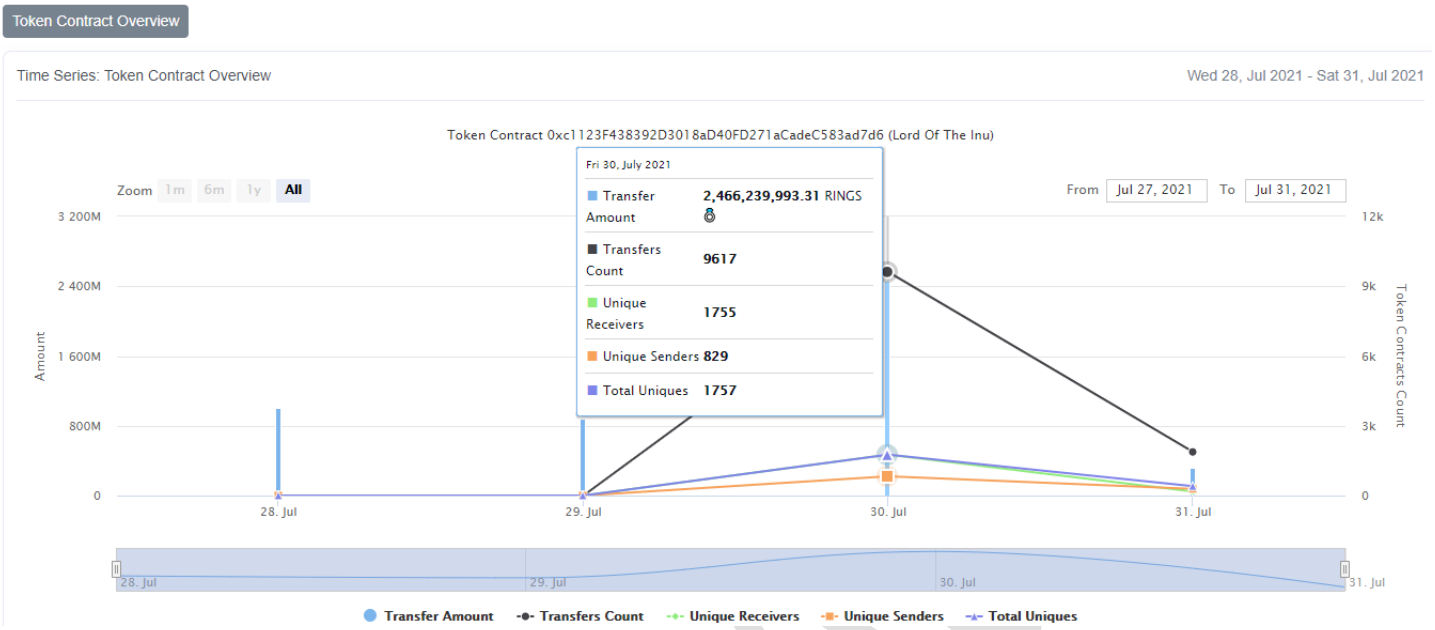
Apart from security issues, notes might also be added to point out a certain functionality.

DISCLAIMER

The audit makes no statements or warrants about the utility of the code, safety of the code, the suitability of the business model, regulatory regime for the business model, or any other statements about the fitness of the contracts to purpose, or their bug-free status. The audit documentation is for discussion purposes only.

CONTRACT OVERVIEW

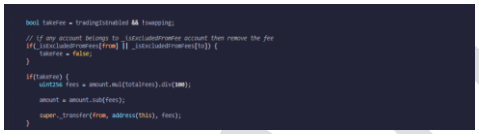
Note: Contract Diagram has been included with this delivery for tracking contract inheritance, functionality calls



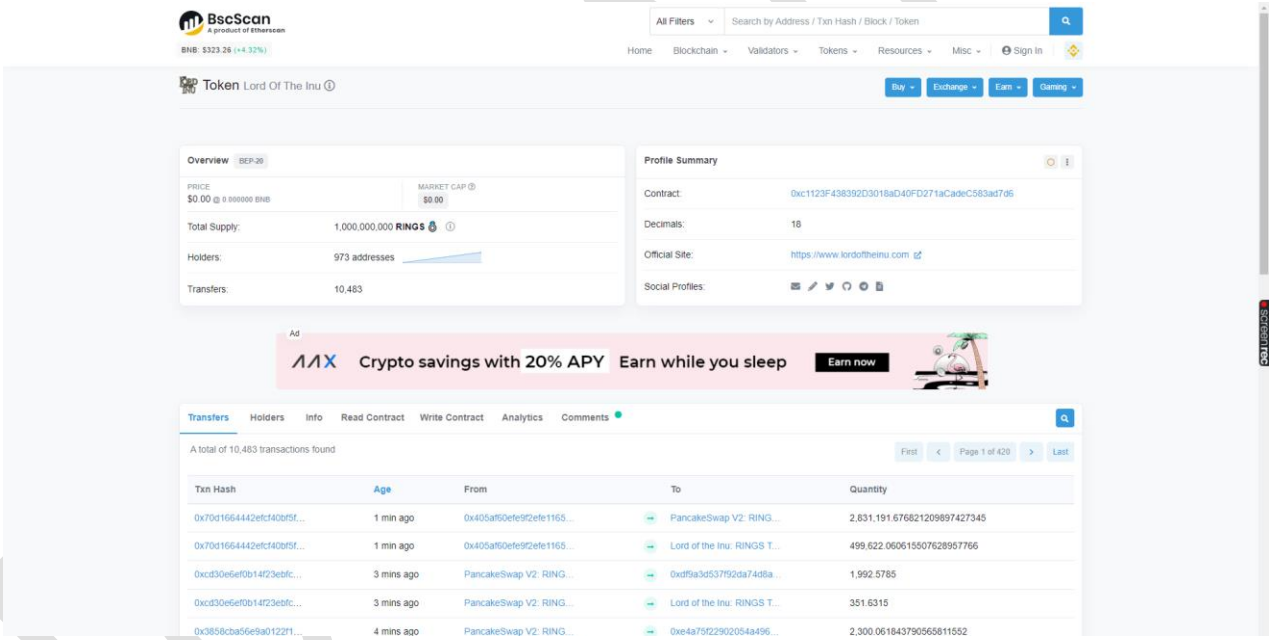
This document details **Load Of The Inu** token findings and recommended solutions. This audit was performed on July 31, 2021. We audited a deployed token.

Token Name	Load Of The Inu
Token Symbol	RINGS
Contract Address	0xc1123F438392D3018aD40FD271aCadeC583ad7d6
BSCScan Link	https://bscscan.com/token/0xc1123F438392D3018aD40FD271aCadeC583ad7d6

AUDIT RESULT

NO.	Audited Items	Function's role of Token	Audit Result
1	Overflow Audit		Passed
2	Safe Design Audit	Checking if the design of contract is safe	Passed
3	Gas Optimization Audit		Passed
4	Design Logic Audit	Checking if the logic of full contract	Passed
5	Malicious Event Log Audit	Checking if Malicious event log will be displayed	Passed
6	Uninitialized Storage Pointers Audit	Checking if uninitialized storage exists	Passed
7	Arithmetic Accuracy Deviation Audit	Checking if the all calculating is success	Passed
8	Compiler errors	Checking if compile is success	Passed
9	Private user data leaks	Checking if user data is safe	Passed
10	Possibly delays in data delivery	Checking if there is delay when data delivery	Passed
11	Methods execution permissions	Checking if the permission is allowed to exact user	Passed
12	_transfer()	Sending token between two accounts	Passed
13	swapAndLiquify()	Swapping token and Creating liquidity	Passed
14	addLiquidity()	Addition Liqudity(It calls frequently in farming and pools)	Passed
15	setMarketingFee()	Setting Fee amount that goes to Marketing wallet	Passed
16	setLiquidityFee()	Setting Fee amount that goes to Liqudity	Passed

		wallet	
17	setMaxSellTransactionAmount()	Setting the limitation amount that can sell at once	Failed
18	setMaxWalletToken ()	Setting the limitation of tokens that can be taken in a wallet	Passed
19	updateLiquidityWallet()	Updating Liqudity Wallet Address. Liqudity fee will be sent to this address	Passed
20	updateMarketingWallet()	Updating Marketing wallet Address. Marketing fee will be sent to this address	Passed



High Severity Issues

- Issues:

  Maximum amount of token can buy or sell at once has a problem.

  Because, there is also Maximum wallet token amount limitation.

  If people select the Maximum sell or buy amount below than maximum wallet token amount, it has no any problem. It will work exactly.

  But in otherwise, the maximum sell or buy amount has no meaning.

```
function updateLiquidityWallet(address newLiquidityWallet) public onlyOwner {
    require(newLiquidityWallet != liquidityWallet, "LordoftheInu: The liquidity wallet is already this address");
    excludeFromFees(newLiquidityWallet, true);
    emit LiquidityWalletUpdated(newLiquidityWallet, liquidityWallet);
    liquidityWallet = newLiquidityWallet;

}

function updateMarketingWallet(address newMarketingWallet) public onlyOwner {
    excludeFromFees(newMarketingWallet, true);
    emit MarketingWalletUpdated(newMarketingWallet, marketingWallet);
    marketingWallet = newMarketingWallet;

}

function setMaxWalletToken(uint256 value) external onlyOwner{
    require(value >= 20000000 * (10**18), "LordoftheInu: Minimum max wallet limit is 2 percent");
    _maxWalletToken = value;

}

function setMaxSellTransactionAmount(uint256 value) external onlyOwner{
     maxSellTransactionAmount = value;
}

function setBUSDRewardsFee(uint256 value) external onlyOwner{
    BUSDRewardsFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
}

function setLiquidityFee(uint256 value) external onlyOwner{
    liquidityFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
}

function setMarketingFee(uint256 value) external onlyOwner{
    marketingFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
```

- Comments:

  The setMaxSellTransactionAmount() function should be changed so that user can't set high value than maximum wallet token amount.

  It should be fixed like below image.

```
function updateLiquidityWallet(address newLiquidityWallet) public onlyOwner {
    require(newLiquidityWallet != liquidityWallet, "LordoftheInu: The liquidity wallet is already this address");
    excludeFromFees(newLiquidityWallet, true);
    emit LiquidityWalletUpdated(newLiquidityWallet, liquidityWallet);
    liquidityWallet = newLiquidityWallet;

}

function updateMarketingWallet(address newMarketingWallet) public onlyOwner {
    excludeFromFees(newMarketingWallet, true);
    emit MarketingWalletUpdated(newMarketingWallet, marketingWallet);
    marketingWallet = newMarketingWallet;

}

function setMaxWalletToken(uint256 value) external onlyOwner{
    require(value >= 20000000 * (10**18), "LordoftheInu: Minimum max wallet limit is 2 percent");
    _maxWalletToken = value;

}

function setMaxSellTransactionAmount(uint256 value) external onlyOwner{
     require(
          value <= _maxWalletToken,
          "Maximum Sell transaction Amount should be low than Maximum Wallet Token amount."
      );
     maxSellTransactionAmount = value;

}

function setBUSDRewardsFee(uint256 value) external onlyOwner{
    BUSDRewardsFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
}

function setLiquidityFee(uint256 value) external onlyOwner{
    liquidityFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);
}

function setMarketingFee(uint256 value) external onlyOwner{
    marketingFee = value;
    totalFees = BUSDRewardsFee.add(liquidityFee).add(marketingFee);

}

function updateGasForProcessing(uint256 newValue) public onlyOwner {
    require(newValue >= 200000 && newValue <= 500000, "LordoftheInu: gasForProcessing must be between 200,000 and 500,000");
    require(newValue != gasForProcessing, "LordoftheInu: Cannot update gasForProcessing to same value");
```

Medium Severity Issues

- Issue:

  Token Owner can change liquidity fee any time if he wants.

  Liqudity fee is used when people create liquidity or sending it to another wallet.

  But people don't often focus on fee. They will see it at beginning time at once.

  People may not know if the liquidity fee is changed

- Comments:

  The Liqudity Fee should be static value. So people will always be paid the same fee

Low Severity Issues

- None.

CONCLUSION

High severity issue and medium severity issue found. These issues are not so serious problems. But if user selects high maximum sell or buy amount then he can't sell or buy it, he can't understand it. I think it should be fixed. All other functions are working very well, it is great. It can be fixed by changed the contract code and re-deploy it.