



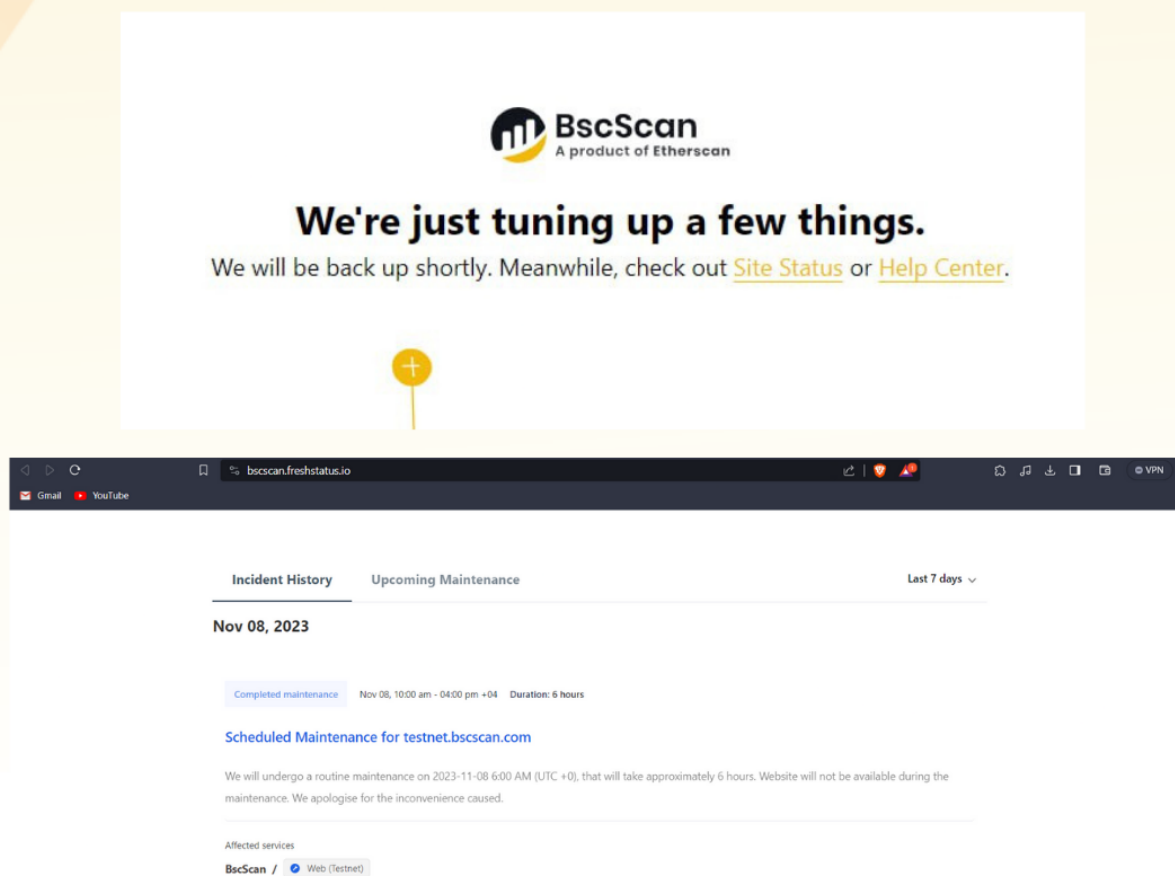
Smart Contract Audit

FOR
Cake Inu

DATED : 08 November 23'

NOTE

This Contract is Not Tested on BSC testnet, because the BSC Testnet maintenance is going on and the servers are down.



All the risks mentioned are of Manual Review.

Functional testnet links will be updated in the report after the Testnet servers are back to Normal.

MANUAL TESTING

Centralization – Enabling Trades

Severity: High

function: EnableTrading

Status: Open

Overview:

The EnableTrading function permits only the contract owner to activate trading capabilities. Until this function is executed, no investors can buy, sell, or transfer their tokens. This places a high degree of control and centralization in the hands of the contract owner.

```
function EnableTrading() external onlyOwner {  
    require(!tradingEnabled, "Cannot re-enable trading");  
    tradingEnabled = true;  
    providingLiquidity = true;  
    genesis_block = block.number;  
}
```

Suggestion

To reduce centralization and potential manipulation, consider one of the following approaches:

1. Automatically enable trading after a specified condition, such as the completion of a presale, is met.
2. If manual activation is still desired, consider transferring the ownership of the contract to a trustworthy, third-party entity like a certified "PinkSale Safu" developer. This can provide investors with more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad faith actions by the original owner



AUDIT SUMMARY

Project name – Cake Inu

Date: 08 November 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	0	2	1
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- Slither :

The code has undergone static analysis using Slither.



Token Information

Token Address:

0x985Ff63b4b727448F60B03173b69917DbC700096

Name: Cake Inu

Symbol: Cake

Decimals: 18

Network: Binance smart chain

Token Type: ERC20

Owner: 0x3177D066290A278a9182C53C29fC2B3a5eAC0307

Deployer:

0x3177D066290A278a9182C53C29fC2B3a5eAC0307

Token Supply: 1,000,000

Checksum: 3b75f27d547a7defa2c94855c3e389b8

Testnet version:-



TOKEN OVERVIEW

buy fee: 12%

Sell fee: 12%

transfer fee: 0%

Fee Privilege: Owner

Ownership: Owned

Minting: None

Max wallet: Yes

Max Trx: Yes

Blacklist: No

Other Privileges:

- Initial distribution of the tokens
- Modifying fees
- Enabling trades
- bulk exempts fee
- Modify amm, router,

Liquidity is dding to owner wallet



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

POINTS TO NOTE

- Owner can renounce ownership
 - Owner can transfer the ownership
 - Owner can change the swap threshold of not more than 1% of total supply
 - Owner can enable trading only once
 - Owner can update the deadline not more than 5 blocks
 - Owner can enable/disable wallet limit
 - Owner can update tax buy not more than 5 and sell not more than 15
 - Owner can exclude wallets from maximum transaction limit.
 - Owner can exclude multiple address from fees
 - Owner can claim ETH from the contract
 - Owner can claim stuck tokens
-



STATIC ANALYSIS

```
INFO:Detectors:
CakeInu.transfer(address,uint256) (cakeInu.sol#713-797) ignores return value by EIP2612(tokenAddr).transfer(token!,amount) (cakeInu.sol#716)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#no-return-transfer

INFO:Detectors:
CakeInu.liquify(uint256,CakeInu.Taxes) (cakeInu.sol#592-631) performs a multiplication on the result of a division:
- unitBalance * (tokenAmount / (currentTaxes * newTaxes.liquidity)) (cakeInu.sol#627-628)
- ethFunds.liquidityWith * unitBalance * newTaxes.liquidity (cakeInu.sol#612)
CakeInu.liquify(uint256,CakeInu.Taxes) (cakeInu.sol#592-631) performs a multiplication on the result of a division:
- unitBalance * (tokenAmount / (currentTaxes * newTaxes.liquidity)) (cakeInu.sol#612-613)
- amount * unitBalance * 2 * newTaxes.div (cakeInu.sol#624)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

INFO:Detectors:
CakeInu.transfer(address,address,uint256) (cakeInu.sol#487-488) is a local variable never initialized
CakeInu.transfer(address,address,uint256) (cakeInu.sol#488) is a local variable never initialized
CakeInu.transfer(address,address,uint256) (cakeInu.sol#488) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

INFO:Detectors:
CakeInu.addLiquidity(uint256,uint256) (cakeInu.sol#403-404) ignores return value by router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#403-403)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#no-return

INFO:Detectors:
CakeInu.transfer(address,address,uint256) (cakeInu.sol#400) is written in both
fee = 0 (cakeInu.sol#400)
fee = (amount * fee) / 100 (cakeInu.sol#400)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#write-after-write

INFO:Detectors:
CakeInu.updateLiquidityThreshold(uint256) (cakeInu.sol#678-679) should emit an event for:
- token.liquidityThreshold = newAmount + 10 == decValue() (cakeInu.sol#678)
CakeInu.updateLiquidityThreshold(uint256) (cakeInu.sol#678) should emit an event for:
- deadline = _deadline (cakeInu.sol#681)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-after-events

INFO:Detectors:
CakeInu.liquify(uint256,CakeInu.Taxes) (cakeInu.sol#592-631) does not always execute ;, or revertReference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-assert

INFO:Detectors:
Reentrancy in CakeInu.liquify(uint256,CakeInu.Taxes) (cakeInu.sol#592-631):
External calls:
- router.swapExactETHForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (cakeInu.sol#602-602)
- addLiquidity(tokenFunds.addLiquidityWith,ethFunds.addLiquidityWith) (cakeInu.sol#623)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#600-601)
External calls sending eth:
- addLiquidity(tokenFunds.addLiquidityWith,ethFunds.addLiquidityWith) (cakeInu.sol#623)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#600-601)
State variables written after the call(s):
- ethFunds.addLiquidityWith (cakeInu.sol#623)
- tokenAmount (cakeInu.sol#600)
Reentrancy in CakeInu.transferFrom(address,address,uint256) (cakeInu.sol#485-486):
External calls:
- _transfer(sender,recipient,amount) (cakeInu.sol#486)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#485-485)
- (success) = recipient.call{value: amount}() (cakeInu.sol#485)

Reentrancy in CakeInu.liquify(uint256,CakeInu.Taxes) (cakeInu.sol#592-631):
External calls:
- router.swapExactETHForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (cakeInu.sol#602-602)
- addLiquidity(tokenFunds.addLiquidityWith,ethFunds.addLiquidityWith) (cakeInu.sol#623)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#600-601)
Event emitted after the call(s):
- Approval(owner,spender,amount) (cakeInu.sol#628)
- addLiquidity(tokenFunds.addLiquidityWith,ethFunds.addLiquidityWith) (cakeInu.sol#623)
Reentrancy in CakeInu.transfer(address,address,uint256) (cakeInu.sol#485-486):
External calls:
- Liquify(feeSwap,currentTaxes) (cakeInu.sol#579)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#485-485)
- (success) = recipient.call{value: amount}() (cakeInu.sol#485)
- router.swapExactETHForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (cakeInu.sol#602-602)
- address(deWallet).sendValue(deValue) (cakeInu.sol#628)
External calls sending eth:
- Liquify(feeSwap,currentTaxes) (cakeInu.sol#579)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#485-485)
- (success) = recipient.call{value: amount}() (cakeInu.sol#485)
Event emitted after the call(s):
- Transfer(sender,recipient,amount) (cakeInu.sol#486)
- Super._transfer(sender,recipient,this,feeAmount) (cakeInu.sol#487)
- Transfer(sender,recipient,amount) (cakeInu.sol#486)
- Super._transfer(sender,recipient,amount - fee) (cakeInu.sol#482)
Reentrancy in CakeInu.transferFrom(address,address,uint256) (cakeInu.sol#485-486):
External calls:
- transfer(sender,recipient,amount) (cakeInu.sol#486)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#485-485)
- (success) = recipient.call{value: amount}() (cakeInu.sol#485)
- router.swapExactETHForTokensSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (cakeInu.sol#602-602)
- address(deWallet).sendValue(deValue) (cakeInu.sol#628)
External calls sending eth:
- transfer(sender,recipient,amount) (cakeInu.sol#486)
- router.addLiquidityETH(value: ethAmount)(address(this),tokenAmount,0,0,deadWallet,block.timestamp) (cakeInu.sol#485-485)
- (success) = recipient.call{value: amount}() (cakeInu.sol#485)
Event emitted after the call(s):
- Approval(owner,spender,amount) (cakeInu.sol#628)
- approve(sender,spender,currentAllowance - amount) (cakeInu.sol#487)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-5

INFO:Detectors:
Context: _msgData() (cakeInu.sol#9-12) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

INFO:Detectors:
Pragma version "0.9.19" (cakeInu.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.8.18.

INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (cakeInu.sol#333-344):
- (success) = recipient.call{value: amount}() (cakeInu.sol#339)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

INFO:Detectors:
Function Router.WETH() (cakeInu.sol#397) is not in mixedCase
Function CakeInu.liquify(uint256,CakeInu.Taxes) (cakeInu.sol#592-631) is not in mixedCase
Parameter CakeInu.updateLiquidityThreshold(uint256) newAmount (cakeInu.sol#678) is not in mixedCase
Function CakeInu.EnableTrading() (cakeInu.sol#678-683) is not in mixedCase
Parameter CakeInu.updateDeadline(uint256) _deadline (cakeInu.sol#685) is not in mixedCase
Parameter CakeInu.updateExemptFee(address,bool) _address (cakeInu.sol#714) is not in mixedCase
Variable CakeInu.genesis_block (cakeInu.sol#432) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

INFO:Detectors:
Redundant expression "this (cakeInu.sol#10)" inContext (cakeInu.sol#4-13)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

INFO:Detectors:
CakeInu.launchtax (cakeInu.sol#434) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

INFO:Detectors:
CakeInu.pair (cakeInu.sol#424) should be immutable
CakeInu.router (cakeInu.sol#423) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable

INFO:Slither:cakeInu.sol analyzed (9 contracts with 93 detectors), 32 result(s) found
```

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

MANUAL TESTING

Centralization – Enabling Trades

Severity: High

function: EnableTrading

Status: Open

Overview:

The EnableTrading function permits only the contract owner to activate trading capabilities. Until this function is executed, no investors can buy, sell, or transfer their tokens. This places a high degree of control and centralization in the hands of the contract owner.

```
function EnableTrading() external onlyOwner {  
    require(!tradingEnabled, "Cannot re-enable trading");  
    tradingEnabled = true;  
    providingLiquidity = true;  
    genesis_block = block.number;  
}
```

Suggestion

To reduce centralization and potential manipulation, consider one of the following approaches:

1. Automatically enable trading after a specified condition, such as the completion of a presale, is met.
2. If manual activation is still desired, consider transferring the ownership of the contract to a trustworthy, third-party entity like a certified "PinkSale Safu" developer. This can provide investors with more confidence in the eventual activation of trading capabilities, mitigating concerns of potential bad faith actions by the original owner

MANUAL TESTING

Severity: Low

subject: floating Pragma Solidity version

Status: Open

Overview:

It is considered best practice to pick one compiler version and stick with it. With a floating pragma, contracts may accidentally be deployed using an outdated.

```
pragma solidity ^0.8.19;
```

Suggestion

Adding the latest constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.

MANUAL TESTING

Severity: Low

subject: Missing Events

Status: Open

Overview:

They serve as a mechanism for emitting and recording data onto the blockchain, making it transparent and easily accessible.

```
function updateLiquidityTreshhold(uint256 new_amount) external
onlyOwner {
    require(
        new_amount <= 1e7,
        "Swap threshold amount should be lower or equal to 1% of tokens"
    );
    tokenLiquidityThreshold = new_amount * 10 ** decimals();
}
```

```
function EnableTrading() external onlyOwner {
    require(!tradingEnabled, "Cannot re-enable trading");
    tradingEnabled = true;
    providingLiquidity = true;
    genesis_block = block.number;
}
```

```
function updatedecline(uint256 _deadline) external onlyOwner {
    require(!tradingEnabled, "Can't change when trading has started");
    require(_deadline < 5, "Deadline should be less than 5 Blocks");
    deadline = _deadline;
}
```



MANUAL TESTING

```
function updateDevWallet(address newWallet) external  
onlyOwner  
    require(newWallet != address(0), "Fee Address cannot be zero  
address");  
    devWallet = newWallet;  
}
```

```
function updateTax(  
    uint256 buyDevTax,  
    uint256 buyLiquidityTax,  
    uint256 sellDevTax,  
    uint256 sellLiquidityTax  
) external onlyOwner {  
    require(  
        (buyDevTax + buyLiquidityTax) <= 5,  
        "Can't set tax greater than 5%"  
    );  
    require(  
        (sellDevTax + sellLiquidityTax) <= 15,  
        "Can't set tax greater than 15%"  
    );  
    taxes = Taxes(buyDevTax, buyLiquidityTax);  
    sellTaxes = Taxes(sellDevTax, sellLiquidityTax);  
}
```

```
function updateExemptFee(address _address, bool state) external  
onlyOwner {  
    exemptFee[_address] = state;  
}
```

MANUAL TESTING

Severity: Optimization

subject: Remove unused code.

Status: Open

Overview:

Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice. though to avoid them

```
function _msgData() internal view virtual returns (bytes  
calldata) {  
    this; // silence state mutability warning without generating  
bytecode - see  
https://github.com/ethereum/solidity/issues/2691  
    return msg.data;  
}  
}
```

Suggestion

To reduce high gas fees. It is suggested to remove. unused code from the contract.



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>
