

# CERTIFICATE OF COMPLIANCE

## Smart Contrat Audit

Hereby Proudly Presented to

# **LARGE CHAIN**

.....

For complying with

## **Blockchain Security Architecture**

This certifica is only valid for Binance Smart Chain Contrat

0x744C3e7CCff5A8e6B0EA8365Ae75Fe210BFF25A2



# SMART CONTRACT SECURITY AUDIT OF (LRG) LARGE CHAIN

## Audit Introduction

<b>Auditing Firm</b>	LARGE CHAIN PROTOCOL
<b>Audit Architecture</b>	LARGE CHAIN Echelon Auditing Standard
<b>Language</b>	Solidity
<b>Client Firm</b>	LARGE CHAIN
<b>Website</b>	<a href="https://largechain.net/">https://largechain.net/</a>
<b>Telegram</b>	<a href="https://t.me/largechainn">https://t.me/largechainn</a>
<b>Twitter</b>	<a href="https://twitter.com/lrgtokenn">https://twitter.com/lrgtokenn</a>
<b>Facebook</b>	<a href="https://www.facebook.com/Largechain">https://www.facebook.com/Largechain</a>
<b>Whitepaper</b>	<a href="https://largechain.net">https://largechain.net</a>
<b>Report Date</b>	Dec 26. 2022

### About LRG CHAIN

LARGE CHAIN WEB 3.0 PROTOCOL

Large Chain, (BinanceSmart Chain ile yaygınlaşan) blockchain tabanlı akıllı sözleşmelerle gerçek dünya uygulamaları arasında köprü vazifesi görmeyi hedefleyen bir platformdur. Blockchain'ler ağ dışındaki veriye erişim sağlayamadığı için, akıllı sözleşmelerde veri sağlayıcı olarak (WEB 3.0) ihtiyaç duyuluyor. LargeChain durumunda BinanceSmartChain ağına bağlı bulunuyor. ağın sağladığı (sıcaklık, hava durumu gibi) harici veri, önceden belirlenmiş koşullar gerçekleştiğinde akıllı sözleşmeleri tetikliyor sağlıyor.

## Audit Summary

large chain team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- LRG LARGE CHAIN solidity source code has **LOW RISK SEVERITY**
- LRG LARGE CHAIN smart contract has an **ACTIVE OWNERSHIP**
- LRG LARGE CHAIN centralization risk correlated to the active owner is **MEDIUM**
- Important owner privileges – **BURN, PAUSE, SET FEES, WITHDRAW**

Be aware that smart contracts deployed on the blockchain aren't resistant to internal exploit, external vulnerability, or hack. For a detailed understanding of risk severity, source code vulnerability, exploitability, and audit disclaimer, kindly refer to the audit.

✔ Contract address:

**0x744C3e7CCff5A8e6B0EA8365Ae75Fe210BFF25A2**

Blockchain: **Binance Smart Chain**

✔ Verify the authenticity of this report on InterFi's GitHub:

<https://github.com/LargeChain>

## Table Of Contents

### Audit Information

## Audit Scope

InterFi was consulted by LARGE CHAIN to conduct the smart contract security audit of their solidity source codes. The audit scope of work is strictly limited to the mentioned solidity file(s) only:

- lrg.sol

### Solidity Source Code On GitHub

<https://github.com/LargeCHAIN/LargeChainn>

### SHA-1 Hash

Solidity source code is audited at hash #  
0xb00022a72db628b124708b9beb7aac184ba815636e5d53c28799c037815e4d  
72

## Audit Methodology

The scope of this report is to audit the smart contract source code of LARGE CHAIN has scanned contracts and reviewed codes for common vulnerabilities, exploits, hacks, and backdoors. Due to being out of scope, LARGE CHAIN has not tested contracts on testnet to assess any functional flaws. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

### **Category**

#### **Smart Contract Vulnerabilities**

- Re-entrancy
- Unhandled Exceptions
- Transaction Order Dependency
- Integer Overflow
- Unrestricted Action
- Incorrect Inheritance Order
- Typographical Errors
- Requirement Violation

## **Source Code Review**

- Gas Limit and Loops
- Deployment Consistency
- Repository Consistency
- Data Consistency
- Token Supply Manipulation
- Access Control and Authorization
- Operations Trail and Event Generation
- Assets Manipulation
- Ownership Control
- Liquidity Access

## **Large Chain Echelon Audit Standard**

- The aim of Large Chain“Echelon” standard is to analyze smart contracts and identify the vulnerabilities and the hacks. Kindly note, large chain does not test smart contracts on testnet. It is recommended that smart contracts are thoroughly tested prior to the audit submission. Mentioned are the steps used by large chain to audit smart contracts:
  - 
  - Solidity smart contract source code reviewal:
  - Review of the specifications, sources, and instructions provided to Large chain to make sure we understand the size, and scope of the smart contract audit.

- Manual review of code, which is the process of reading source code line– by–line to identify potential vulnerabilities.
- Static, Manual, and Software analysis:
- Test coverage analysis is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
- Symbolic execution is analyzing a program to determine what inputs cause each part of a program to execute.
- Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
- Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts
- Automated 3P frameworks used to assess the smart contract vulnerabilities Consensys Tools
- SWC Registry
- Solidity Coverage
- Open Zeppelin Code Analyzer
- Solidity Code Compiler

## Risk Classification

Smart contracts are generally designed to manipulate and hold funds denominated in ETH/BNB. This makes them very tempting attack targets, as a successful attack may allow the attacker to directly steal funds from the contract. Below are the typical risk levels of a smart contract:

**Vulnerable:** A contract is vulnerable if it has been flagged by a static analysis tool as such. As we will see later, this means that some contracts may be vulnerable because of a false positive.

**Exploitable:** A contract is exploitable if it is vulnerable and the vulnerability could be exploited by an external attacker. For example, if the “vulnerability” flagged by a tool is in a function that requires owning the contract, it would be vulnerable but not exploitable.

**Exploited:** A contract is exploited if it received a transaction on the main network which triggered one of its vulnerabilities. Therefore, a contract can be vulnerable or even exploitable without having been exploited.

<b>Risk severity</b>	<b>Meaning</b>
----------------------	----------------

	This level vulnerabilities could be exploited easily and canlead to asset loss,
--	---

**! Low**

**! Medium**



## ! Low

## ! Informational

data loss, asset, or data manipulation. They should be fixed right away.

This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity

This level vulnerabilities should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.

This level vulnerabilities can be ignored. They are code style violations and informational statements in the code. They may not affect the smart contract execution

## Centralization Risk

Centralization risk is the most common cause of decentralized finance hacks. When a smart contract has an active contract ownership, the risk related to centralization is elevated. There are some well-intended reasons to be an active contract owner, such as:

- Contract owner can be granted the power to `pause()` or `lock()` the contract in case of an external attack.

- Contract owner can use functions like, `include()`, and `exclude()` to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale, and to list on an exchange.

Authorizing a full centralized power to a single body can be dangerous. Unfortunately, centralization related risks are higher than common smart

contract vulnerabilities. Centralization of ownership creates a risk of rug pull scams, where owners cash out tokens in such quantities that they become valueless. **Most important question to ask here is, how to mitigate centralization risk?** Here's InterFi's recommendation to lower the risks related to centralization hacks:

- Smart contract owner's private key must be carefully secured to avoid any potential hack.
- Smart contract ownership should be shared by multi-signature (multi-sig) wallets.
- Smart contract ownership can be locked in a contract, user voting, or community DAO can be introduced to unlock the ownership.

### **LRG LARGE CHAIN Centralization Status**

- **LARGE CHAIN** smart contract has an **active ownership**.
- Smart contract is **not deployed** on blockchain at the time of the audit.

## Static Analysis

Symbol	Meaning
	Function
	can modify
	state
	Function is
	payable
🔒	Function
	is locked
	Function
	can be
	accessed

!!Important functionality

	**Ownable**		Implementation		Context			
	└		<Constructor>		Public			NO
	└		owner		Public			NO
	└		_checkOwner		Internal			
	└		renounceOwnership		Public			onlyOwner
	└		transferOwnership		Public			onlyOwner
	└		_transferOwnership		Internal			
	**SafeMath**		Library					
	└		tryAdd		Internal			
	└		trySub		Internal			
	└		tryMul		Internal			
	└		tryDiv		Internal			
		└		tryMod		Internal		



	L		add		Internal			
	L		sub		Internal			
	L		mul		Internal			
	L		div		Internal			
	L		mod		Internal			
	L		sub		Internal			
	L		div		Internal			
	L		mod		Internal			

|||||

| **\*\*ReentrancyGuard\*\*** | Implementation | |||  
 | L | **<Constructor>** | Public | |NO |

|||||

| **\*\*IBEP2E\*\*** | Interface | |||  
**!!**| L | totalSupply | External | |NO |  
**!!**| L | decimals | External | |NO **!** |  
**!!**| L | symbol | External | |NO **!** |  
**!!**| L | name | External | |NO |

**!!**| L | getOwner | External | |NO **!** |  
**!!**| L | balanceOf | External | |NO |  
**!!**| L | transfer | External |  |NO **!** |  
**!!**| L | allowance | External | |NO |  
**!!**| L | approve | External |  |NO |  
**!!**| L | transferFrom | External | |NO |

|||||

			<b>**IPancakeswapV2Factory**</b>		Interface		
		L	feeTo		External		NO
		L	feeToSetter		External		NO
		L	getPair		External		NO
		L	allPairs		External		NO
		L	allPairsLength		External		NO
		L	createPair		External		NO
		L	setFeeTo		External		NO
	L		setFeeToSetter		External		NO

	**IPancakeSwapV2Pair**   Interface											
	L	name   External			NO							
	L		symbol   External				NO					
	L		decimals   External				NO					
	L		totalSupply   External						NO			
	L			balanceOf			External			NO		
	L			allowance			External			NO		

L	approve	External		NO
L	transfer	External		NO
L	transferFrom	External		NO
L	DOMAIN\_SEPARATOR	External		NO

	L		PERMIT_TYPEHASH		External		NO
--	---	--	-----------------	--	----------	--	----

	L		nonces   External		NO		
	L		permit   External		NO		
	L		MINIMUM_LIQUIDITY   External		NO		
	L		factory   External		NO		

!! | L | token0 | External | |NO

	L		token1   External		NO		
	L		getReserves   External		NO		
	L		price0CumulativeLast	External		NO	
	L		price1CumulativeLast	External		NO	
	L		kLast   External		NO		
	L		mint	External		NO	
	L		burn	External		NO	
	L		swap	External		NO	
	L		skim	External		NO	
	L		sync	External	!	NO	

! ! ! ! | L | initialize | External | |NO

|||||

| **\*\*IPancakeRouter01\*\*** | Interface | |||

!! | L | factory | External | |NO

| L | WETH | External | |NO

!! | L | addLiquidity | External | |NO

!! | L | addLiquidityETH | External | |NO

!! | L | removeLiquidity | External | |NO

! | L | removeLiquidityETH | External | |NO !

! | L | removeLiquidityWithPermit | External | |NO !

!! | L | removeLiquidityETHWithPermit | External | |NO

!! | L | swapExactTokensForTokens | External | |NO

	L		swapTokensForExactTokens   External			NO	
	L		swapExactETHForTokens   External			NO	
	L		swapTokensForExactETH   External			NO	
	L		swapExactTokensForETH   External			NO	
	L		swapETHForExactTokens   External			NO	
	L		quote   External		NO		

| L | getAmountOut | External | |NO

| L | getAmountIn | External | |NO

| L | getAmountsOut | External | |NO

| L | getAmountsIn | External | |NO

|||||

| **\*\*IPancakeRouter02\*\*** | Interface | IPancakeRouter01 |||

| L | removeLiquidityETHSupportingFeeOnTransferTokens | External | |NO

	L		removeLiquidityETHWithPermitSupportingFeeOnTransferTokens			NO	
	L		swapExactTokensForTokensSupportingFeeOnTransferTokens		NO		

	L		swapExactETHForTokensSupportingFeeOnTransferTokens			NO	
	External						
	L		swapExactTokensForETHSupportingFeeOnTransferTokens			NO	
	External						
			**BrnMetaverse**   Implementation   Ownable, IBEP2E, ReentrancyGuard				
	L		<Constructor>		Public		NO
	L		paused		Public		NO
	L		name		Public		NO
	L		symbol		Public		NO
	L		decimals		Public		NO
	L		totalSupply		Public		NO
	L		getOwner		Public		NO
	L		balanceOf		Public		NO
	L		transfer		Public		whenNotPaused
	L		transferFrom		Public		whenNotPaused
	L		allowance		Public		NO
	L		approve		Public		whenNotPaused
	L		increaseAllowance		Public		whenNotPaused
	L		decreaseAllowance		Public		whenNotPaused
	L		burn		Public		onlyOwner
	L		pause		Public		onlyOwner
	L		unpause		Public		onlyOwner
			whenPaused				
	L		createLiquidityPoolPair		Public		
			liquidityPairNotCreatedOnlyOwner				
	L		setRouterAddress		External		liquidityPairCreated onlyOwner
	L		setLiquidityFee		External		onlyOwner
!	L		setLiquidityPoolBuyFee		External		onlyOwner
!	L		setLiquidityPoolSellFee		External		onlyOwner
!	L		setSwapAndLiquifyEnabled		Public		onlyOwner
			swapIsNotEnabled				
	L		calculateLiquidityFee		Private		
	L		calculateTaxFee		Private		
	L		removeAllFee		Private		
	L		restoreAllFee		Private		
!	L		<Receive Ether>		External		NO
	L		swapAndLiquify		Private		lockTheSwap
	L		swapTokensForBnb		Private		
	L		addLiquidity		Private		
	L		removeLiquidity		Public		
			onlyOwner				
			excludeFromReward		Public		
			onlyOwner				

	L		includeInReward   External	
	L		onlyOwner   <b>withdraw</b>   Public	
	L		onlyOwner nonReentrant	
	L		_getChainID   Private	
	L		_transfer   Internal	
	L		_takeLiquidity   Private	
			takeFee   Private	
	L		_transferTokens   Private	
	L		takeTaxes   Internal	
	L		_getFeeAmountValues   Private	
		L		burn   Internal
		L		approve   Internal
		L		burnFrom   Internal
		L		_beforeTokenTransfer   Internal
		L		_afterTokenTransfer   Internal
		L		

## Software Analysis

### Function Signatures

39509351	=>	increaseAllowance(address,uint256)
75128141	=>	calculateTaxFee(uint256)
8da5cb5b	=>	owner()
53a72975	=>	checkOwner()
715018a6	=>	renounceOwnership()
f2fde38b	=>	transferOwnership(address)
d29d44ee	=>	_transferOwnership(address)
884557bf	=>	tryAdd(uint256,uint256)
a29962b1	=>	trySub(uint256,uint256)
6281efa4	=>	tryMul(uint256,uint256)
736ecb18	=>	tryDiv(uint256,uint256)
38dc0867	=>	tryMod(uint256,uint256)
771602f7	=>	add(uint256,uint256)
b67d77c5	=>	sub(uint256,uint256)
c8a4ac9c	=>	mul(uint256,uint256)
a391c15b	=>	div(uint256,uint256)
f43f523a	=>	mod(uint256,uint256)
e31bdc0a	=>	sub(uint256,uint256,string)
b745d336	=>	div(uint256,uint256,string)
71af23e8	=>	mod(uint256,uint256,string)
18160ddd	=>	totalSupply()
313ce567	=>	decimals()
95d89b41	=>	symbol()
06fdde03	=>	name()
893d20e8	=>	getOwner()
70a08231	=>	balanceOf(address)

a9059cbb	=>	transfer(address,uint256)
dd62ed3e	=>	allowance(address,address)
095ea7b3	=>	approve(address,uint256)
23b872dd	=>	transferFrom(address,address,uint256)
017e7e58	=>	feeTo()
094b7415	=>	feeToSetter()
e6a43905	=>	getPair(address,address)
1e3dd18b	=>	allPairs(uint256)
574f2ba3	=>	allPairsLength()
c9c65396	=>	createPair(address,address)
f4690led	=>	setFeeTo(address)
a2e74af6	=>	setFeeToSetter(address)
3644e515	=>	DOMAIN_SEPARATOR()
30adf81f	=>	PERMIT_TYPEHASH()
7ecebe00	=>	nonces(address)
d505accf	=>	permit(address,address,uint256,uint256,uint8,bytes32,bytes32)
ba9a7a56	=>	MINIMUM_LIQUIDITY()
c45a0155	=>	factory()
0dfe1681	=>	token0()

d21220a7	=>	token1()
0902f1ac	=>	getReserves()
5909c0d5	=>	price0CumulativeLast()
5a3d5493	=>	price1CumulativeLast()
7464fc3d	=>	kLast()
6a627842	=>	mint(address)
89afcb44	=>	burn(address)
022c0d9f	=>	swap(uint256,uint256,address,bytes)
bc25cf77	=>	skim(address)
fff6cae9	=>	sync()
485cc955	=>	initialize(address,address)
ad5c4648	=>	WETH()
e8e33700	=>	addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256)
f305d719	=>	addLiquidityETH(address,uint256,uint256,uint256,address,uint256)
baa2abde	=>	removeLiquidity(address,address,uint256,uint256,uint256,address,uint256)
02751cec	=>	removeLiquidityETH(address,uint256,uint256,uint256,address,uint256)



2195995 c	= >	
--------------	--------	--

removeLiquidityWithPermit (address, address, uint256, uint256, uint256, address, uint256

, bool, uint8, byte

s3 2, bytes32)

ded9382a =>

removeLiquidityETHWithPermit (address, uint256, uint256, uint256, address, uint256

, bool, uint8, bytes32, bytes32)

38ed1739

=>

swapExactTokensForTokens (uint256, uint256,

address [], address, uint256) 8803dbee

=>swapTokensForExactTokens (uint256, uint256,

054d50d4	=>	getAmountOut (uint256, uint256, uint256)
85f8c259	=>	getAmountIn (uint256, uint256, uint256)
d06ca61f	=>	getAmountsOut (uint256, address [])
1f00ca74	=>	getAmountsIn (uint256, address [])
af2979eb	=>	

4a25d94a =>

swapTokensForExactETH (uint256, uint256

, address [], address, uint256) 18cbafe5

=>

swapExactTokensForETH (uint256, uint256

, address [], address, uint256) fb3bdb41

=>

swapETHForExactTokens (uint256, address

[], address, uint256) ad615dec =>

quote (uint256, uint256, uint256)

removeLiquidityETHSupportingFeeOnTransferTokens (address, uint256, uint256, uint256,

address, uint256) 5b0d5984 =>

removeLiquidityETHWithPermitSupportingFeeOnTransferTokens (address, uint256, uint256

, uint256, address, u

int256, bool, uint8, bytes32, bytes32)

5c11d795 =>

swapExactTokensForTokensSupportingFeeOnTransferTokens (uint256, uint256

, address [], address, uint256) b6f9de95 =>

swapExactETHForTokensSupportingFeeOnTransferTokens (uint256, address [

], address, uint256) 791ac947 =>

a457c2d7	=>	decreaseAllowance (address, uint256)
9dc29fac	=>	burn (address, uint256)
8456cb59	=>	pause ()
3f4ba83a	=>	unpause ()

4dc2fe46	=>	createLiquidityPoolPair()
41cb87fc	=>	setRouterAddress(address)

swapExactTokensForETHSupportingFeeOnTransferTokens(uint256,uint256,address[],address,uint256) 5c975abb => paused()

357bf15c	=>	setLiquidityFee(uint256)
6ea23a34	=>	setLiquidityPoolBuyFee(uint256)
3dd2e789	=>	setLiquidityPoolSellFee(uint256)
dc38a03f	=>	setSwapAndLiquifyEnabled()
cc126a23	=>	calculateLiquidityFee(uint256)
301370af	=>	removeAllFee()
e7e3e3a7	=>	restoreAllFee()
173865ad	=>	swapAndLiquify(uint256)
528689ea	=>	swapTokensForBnb(uint256)
9cd441da	=>	addLiquidity(uint256,uint256)
9c8f9f23	=>	removeLiquidity(uint256)
52390c02	=>	excludeFromReward(address)
3685d419	=>	includeInReward(address)
2e1a7d4d	=>	withdraw(uint256)
660a00ed	=>	getChainID()
30e0789e	=>	transfer(address,address,uint256)
c432df5e	=>	takeLiquidity(uint256)
49026a97	=>	takeFee(uint256)
20d6115d	=>	transferTokens(address,address,uint256,bool)
aa0ffc5	=>	takeTaxes(address,address,uint256)
d4b80039	=>	getFeeAmountValues(uint256)
6161eb18	=>	burn(address,uint256)
104e81ff	=>	approve(address,address,uint256)
a22b35ce	=>	burnFrom(address,uint256)
cad3be83	=>	beforeTokenTransfer(address,address,uint256)
8f811a1c	=>	afterTokenTransfer(address,address,uint256)

## Inheritance Graph

## Manual Analysis

Function	Description	Available	Status
----------	-------------	-----------	--------

provides information about the total token

**Total Supply**

**Balance Of**

**Transfer**

**Approve**

**Allowance**

**Burn**

**Dividend**

**Lock / Pause**

**Contract Fees**

supply

provides account balance of the owner's account

executes transfers of a specified number of tokens to a specified

address allow a spender to withdraw a set number of tokens from

a specified account returns a set number of tokens from a spender

to the owner

executes transfers of a specified number of tokens to a

burn address executes transfers of a specified dividend

token to a specified address locks or pauses all or some  
function modules of the smart contract

executes fee collection from swap events and/or transfer events

Yes **Passed**

Yes **Passed**

Yes **Passed**

Yes **Passed**

Yes **Passed**

Yes **Passed**

Yes **Passed**

Yes **Passed**

Yes **Passed**

**Transfe**

**r**

## Owners

## hip

## Renoun

## ce

executes transfer of contract ownership to a specified

walletexecutes transfer of contract ownership to a

Yes **PassedOwnership**

dead address

Yes **Passed**

## Notable Information

- Smart contract owner can **pause or lock** the smart contract functionmodules.

```
function pause() public onlyOwner whenNotPaused {  
    _paused = true;  
function unpause() public onlyOwner whenPaused {  
    _paused = false;
```

- Smart contract owner can **burn** user assets.

Token allowancerequirements must be met to perform this transaction.

```
function burn(address _account, uint _amount) public onlyOwner
whenNotPaused returns (bool) {
    _burn(_account, _amount);
}
```

- Smart contract utilizes **safemath** function to avoid common smartcontract vulnerabilities.

```
string private _name = "BRNMETAVERSE";
library SafeMath {
function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a,
    b, "SafeMath:
    subtraction
    overflow");
    uint256 c = a
    * b;

    require(c / a == b, "SafeMath:
    multiplicatio
    n overflow");
    return c;
function div(uint256 a, uint256 b) internal pure returns (uint256) {
    return div(a, b, "SafeMath: division by zero");
function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    return mod(a, b, "SafeMath: modulo by zero");
}
```

- Smart contract utilizes **re-entrancy guard** to prevent re-entrant calls to `withdraw()`.
- Smart contract utilizes **redundant code** for `transferOwnership()`. Ideal transfer ownership code should look like:

```
function transferOwnership(address newOwner)
public virtual onlyOwner {
    require(newOwner != address(0), "Ownable:
    new owner is the zero address"); emit
    OwnershipTransferred(_owner, newOwner);
}
```

```
_owner = newOwner;
```

- Smart contract owner can **change transaction fees**. This function module can be used to impose extraordinary fees. Noarbitrary limit set.

```
function setLiquidityFee(uint256
    _liquidityFee) external
    onlyOwner() {
    require(_liquidityFee !=
    0, "BEP2E: Feecannot be
    zero"); liquidityFee =
    _liquidityFee;
function
    setLiquidityPoolBuyFee(uint256
    _fee) external onlyOwner{
    require( _fee != 0, "BEP2E:Fee
    cannot be zero");
    _buyFee = _fee;
function
    setLiquidityPoolSellFee(uint256
    _fee) external onlyOwner{
    require( _fee != 0, "BEP2E:Fee
    cannot be zero");
```

- Smart contract has a **low severity issue** which may or may not create any functional vulnerability.

**"severity": 8, (! Low Severity)**

**"Expected pragma, import directive or contract/interface/library definition"**

## SWC Attacks

SWC ID	Description	Status
SWC-101	Integer Overflow and Underflow	Passed

<b>SWC-102</b>	Outdated Compiler Version	<b>! Informational</b>
<b>SWC-103</b>	Floating Pragma	<b>Passed</b>
<b>SWC-104</b>	Unchecked Call Return Value	<b>Passed</b>
<b>SWC-105</b>	Unprotected Ether Withdrawal	<b>Passed</b>
<b>SWC-106</b>	Unprotected SELF-DESTRUCT Instruction	<b>Passed</b>
<b>SWC-107</b>	Re-entrancy	<b>Passed</b>
<b>SWC-108</b>	State Variable Default Visibility	<b>Passed</b>
<b>SWC-109</b>	Uninitialized Storage Pointer	<b>Passed</b>
<b>SWC-110</b>	Assert Violation	<b>Passed</b>
<b>SWC-111</b>	Use of Deprecated Solidity Functions	<b>Passed</b>
<b>SWC-112</b>	Delegate Call to Untrusted Callee	<b>Passed</b>
<b>SWC-113</b>	DoS with Failed Call	<b>Passed</b>
<b>SWC-114</b>	Transaction Order Dependence	<b>Passed</b>
<b>SWC-115</b>	Authorization through tx.origin	<b>Passed</b>
<b>SWC-116</b>	Block values as a proxy for time	<b>Passed</b>

<b>SWC-117</b>	Signature Malleability	<b>Passed</b>
----------------	------------------------	---------------



<b>SWC-118</b>	Incorrect Constructor Name	<b>Passed</b>
----------------	----------------------------	---------------

<b>SWC-119</b>	Shadowing State Variables	<b>Passed</b>
<b>SWC-120</b>	Weak Sources of Randomness from Chain Attributes	<b>Passed</b>
<b>SWC-121</b>	Missing Protection against Signature Replay Attacks	<b>Passed</b>
<b>SWC-122</b>	Lack of Proper Signature Verification	<b>Passed</b>
<b>SWC-123</b>	Requirement Violation	<b>Passed</b>
<b>SWC-124</b>	Write to Arbitrary Storage Location	<b>Passed</b>
<b>SWC-125</b>	Incorrect Inheritance Order	<b>Passed</b>
<b>SWC-126</b>	Insufficient Gas Griefing	<b>Passed</b>
<b>SWC-127</b>	Arbitrary Jump with Function Type Variable	<b>Passed</b>
<b>SWC-128</b>	DoS With Block Gas Limit	<b>Passed</b>
<b>SWC-129</b>	Typographical Error	<b>Passed</b>
<b>SWC-130</b>	Right-To-Left-Override control character(U+202E)	<b>Passed</b>
<b>SWC-131</b>	Presence of unused variables	<b>Passed</b>
<b>SWC-132</b>	Unexpected Ether balance	<b>Passed</b>
<b>SWC-133</b>	Hash Collisions With Multiple Variable LengthArguments	<b>Passed</b>
<b>SWC-134</b>	Message call with the hardcoded gas amount	<b>Passed</b>
<b>SWC-135</b>	Code With No Effects (Irrelevant/Dead Code)	<b>!</b>

		<b>Informational</b>
<b>SWC-136</b>	Unencrypted Private Data On-Chain	<b>Passed</b>

## Risk Status & Radar Chart

### Risk Severity

### Status

**Passed**

No high severity issues identified

**Medium**

No medium severity issues identified

**Low**

2 low severity

issues identified

**Informational** 2

informational severity

issues identified

### Centralization Risk

Active contract ownership

identified

## Score out of 100

Compiler Check

100  
95  
90  
85  
80  
75

Static Analysis

Manual Analysis

Software Analysis

## Auditor's Verdict

InterFi team has performed a line-by-line manual analysis and automated review of smart contracts. Smart contracts were analyzed mainly for common contract vulnerabilities, exploits, and manipulation hacks. According to the audit:

- LARGE CHAIN solidity source code has **LOW RISK SEVERITY**
- LARGE CHAIN smart contract has an **ACTIVE OWNERSHIP**
- LARGE CHAIN centralization risk correlated to the active owner is **MEDIUM**

**Note for  
stakehold  
ers**

- Be aware that active smart contract owner privileges constitute an elevated impact on smart contract safety and security.
- If the smart contract is not deployed on any blockchain at the time of the audit, the contract can be modified or altered before blockchain development. Verify contract's deployment status in the audit report.
- Make sure that the project team's KYC/identity is verified by an independent firm.
- Always check if the contract's liquidity is locked. A longer liquidity lock plays an important role in the project's longevity. It is recommended to have multiple liquidity providers.

- Examine the unlocked token supply in the owner, developer, or team's private wallets. Understand the project's tokenomics, and make sure the tokens outside of the LP Pair are vested or locked for a longer period.

## Important Disclaimer

LARGE CHAIN provides contract development, testing, auditing and project evaluation services for blockchain projects. The purpose of the audit is to analyze the on-chain smart contract source code and to provide a basic overview of the project. **This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purpose without InterFi's prior written consent.**

LARGE CHAIN provides the easy-to-understand assessment of the project, and the smart contract (otherwise known as the source code).

The audit makes no statements or warranties on the security of the code. It also cannot be considered as enough assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have used all the data at our disposal to provide the transparent analysis, it is important to note that you should not rely on this report only — we recommend

proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

**Be aware that smart contracts deployed on a blockchain aren't resistant to external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact on smart contract safety and security. Therefore, InterFi does not guarantee the explicit security of the audited smart contract.**

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

**This report should not be considered as an endorsement or disapproval of any project or team.**

The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. Do conduct your due diligence and consult your financial advisor before making any investment decisions.

## About LARGE CHAIN PROTOCOL

LARGE CHAIN provides intelligent blockchain solutions. LARGE CHAIN is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. **LARGE CHAIN mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy to use.**

InterFi is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members, and 10+ casual contributors. **LARGE CHAIN provides manual, static, and automatic smart contract analysis, to ensure that project is checked against known attacks and potential vulnerabilities.**

To learn more, visit <https://largechain.net/>

To view our audit portfolio, visit <https://github.com/LargeChain>

To book an audit, message <https://t.me/largechainn>

•

• • •

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN TURKEY