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

0xa85f8864842BF6ce0DB298681559CaBF83c65883

VERIFIED
SOURCE CODE

SMART CONTRACT SECURITY AUDIT OF

(LRG) LARGE CHAIN

Audit Introduction

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

About LRG CHAIN

LARGE CHAIN WEB 3.0 PROTOCOL

Large Chain, (BinanceSmart Chain ile yaygınlaşan) blockchain tabanlı akıllı sözleşmelerle gerçek dünyauygulamaları 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: **0xa85f8864842bf6ce0db298681559cabf83c65883**

⛓️ Blockchain: **Binance Smart Chain**

✅ Verify the authenticity of this report on InterFi's GitHub:
https://github.com/LargeChain

Table Of Contents

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 # 0x0c3caaa2275ea6a123045ebfe39654dbf30aa4886ee3a5e245b85dbbf24a6b18

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 Chaın Echelon Audit Standard

- The aim of Large Chaın“Echelon” standard is to analyze smart contracts and identify the vulnerabilities and the hacks. Kindly note, large chaın 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 chaın to audit smart contracts:
-
- Solidity smart contract source code reviewal:
- Review of the specifications, sources, and instructions provided to Large chaın 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 Complier

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.

Risk severity	Meaning
! Low	This level vulnerabilities could be exploited easily and can lead to asset loss,
! 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			
└	add	Internal			
└	sub	Internal			
└	mul	Internal			
└	div	Internal			
└	mod	Internal			
└	sub	Internal			
└	div	Internal			
└	mod	Internal			
ReentrancyGuard	Implementation			
└	<Constructor>	Public		NO	
IBEP2E	Interface			
└	totalSupply	External		NO	
└	decimals	External		NO !	
└	symbol	External		NO !	
└	name	External		NO	
└	getOwner	External		NO !	
└	balanceOf	External		NO	
└	transfer	External	🛑	NO !	
└	allowance	External		NO	
└	approve	External	🛑	NO	
└	transferFrom	External		NO	
IPancakeswapV2Factory	Interface			
└	feeTo	External		NO	
└	feeToSetter	External		NO	
└	getPair	External		NO	
└	allPairs	External		NO	
└	allPairsLength	External		NO	
└	createPair	External		NO	
└	setFeeTo	External		NO	
└	setFeeToSetter	External		NO	

||||||

IPancakeSwapV2Pair	Interface			
└	name	External		NO
└	symbol	External		NO
└	decimals	External		NO
└	totalSupply	External		NO
└	balanceOf	External		NO
└	allowance	External		NO
└	approve	External		NO
└	transfer	External		NO
└	transferFrom	External		NO
└	DOMAIN_SEPARATOR	External		NO
└	PERMIT_TYPEHASH	External		NO
└	nonces	External		NO
└	permit	External		NO
└	MINIMUM_LIQUIDITY	External		NO
└	factory	External		NO
❗❗| └	token0	External	|NO
└	token1	External		NO
└	getReserves	External		NO
└	price0CumulativeLast	External		NO
└	price1CumulativeLast	External		NO
└	kLast	External		NO
└	mint	External		NO
└	burn	External		NO
└	swap	External ❗		NO
└	skim	External		NO
└	sync	External ❗		NO
🛑❗🛑❗| └ | initialize | External | |NO |

||||||

| **IPancakeRouter01** | Interface | |||
‼️| └ | factory | External | |NO |
‼️🛑| └ | WETH | External | |NO |
‼️| └ | addLiquidity | External | |NO |

‼️| └ | addLiquidityETH | External | 💵 |NO |
‼️| └ | removeLiquidity | External | 🛑 |NO |
❗| └ | removeLiquidityETH | External | 🛑 |NO ❗ |
❗| └ | removeLiquidityWithPermit | External | 🛑 |NO ❗ |
‼️| └ | removeLiquidityETHWithPermit | External | 🛑 |NO |
‼️| └ | swapExactTokensForTokens | External | |NO |
└	swapTokensForExactTokens	External		NO
└	swapExactETHForTokens	External		NO
└	swapTokensForExactETH	External		NO
└	swapExactTokensForETH	External		NO
└	swapETHForExactTokens	External		NO
└	quote	External		NO

└	getAmountOut	External		NO
└	getAmountIn	External		NO
└	getAmountsOut	External		NO
└	getAmountsIn	External		NO
IPancakeRouter02	Interface	IPancakeRouter01		
└	removeLiquidityETHSupportingFeeOnTransferTokens	External		NO
└	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External		NO
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External		NO
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External		NO
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External		NO
BrnMetaverse	Implementation	Ownable, IBEP2E, ReentrancyGuard		
└	<Constructor>	Public		NO
└	paused	Public		NO
└	name	Public		NO
└	symbol	Public		NO
└	decimals	Public		NO
└	totalSupply	Public		NO
└	getOwner	Public		NO
└	balanceOf	Public		NO
└	transfer	Public		whenNotPaused
└	transferFrom	Public		whenNotPaused
└	allowance	Public		NO
└	approve	Public		whenNotPaused
└	increaseAllowance	Public		whenNotPaused
└	decreaseAllowance	Public		whenNotPaused
└	burn	Public		onlyOwner whenNotPaused
└	pause	Public		onlyOwner whenNotPaused
└	unpause	Public		onlyOwner whenPaused
└	createLiquidityPoolPair	Public		liquidityPairNotCreated onlyOwner
└	setRouterAddress	External		liquidityPairCreated onlyOwner
└	setLiquidityFee	External		onlyOwner
!| └	setLiquidityPoolBuyFee	External	🛑	onlyOwner
!| └	setLiquidityPoolSellFee	External	🛑	onlyOwner
!| └	setSwapAndLiquifyEnabled	Public	🛑	onlyOwner swapIsNotEnabled
🔒| └	calculateLiquidityFee	Private 🔒		
└	calculateTaxFee	Private		

| └ | removeAllFee | Private 🔒 | 🛑 | |
| └ | restoreAllFee | Private 🔒 | 🛑 | |
‼| └ | <Receive Ether> | External | 💵 |NO |
└	swapAndLiquify	Private 🔒	🛑	lockTheSwap
└	swapTokensForBnb	Private 🔒	🛑	
└	addLiquidity	Private		

└	removeLiquidity	Public		onlyOwner
└	excludeFromReward	Public		onlyOwner
└	includeInReward	External		onlyOwner
└	withdraw	Public		onlyOwner nonReentrant
└	_getChainID	Private		
└	_transfer	Internal		
└	_takeLiquidity	Private		
└	_takeFee	Private		
└	_transferTokens	Private		
└	takeTaxes	Internal		
└	_getFeeAmountValues	Private		

	└		_burn | Internal | |
	└		_approve | Internal | |
	└		_burnFrom | Internal | |
	└		_beforeTokenTransfer | Internal | |
	└		_afterTokenTransfer | Internal | |

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)
f46901ed	=>	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)
2195995c	=>	

removeLiquidityWithPermit(address,address,uint256,uint256,uint256,address,uint256
,bool,uint8,bytes3 2,bytes32)
ded9382a =>
removeLiquidityETHWithPermit(address,uint256,uint256,uint256,address,uint25
6,bool,uint8,bytes32,byt es32)
38ed1739 =>
swapExactTokensForTokens(uint256,uint256,address[],add
ress,uint256) 8803dbee =>
swapTokensForExactTokens(uint256,uint256,address[],add
ress,uint256) 7ff36ab5 =>
swapExactETHForTokens(uint256,address[],address,uint25
6)

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

4a25d94a =>
swapTokensForExactETH(uint256,uint256,address[],ad
dress,uint256) 18cbafe5 =>
swapExactTokensForETH(uint256,uint256,address[],ad
dress,uint256) fb3bdb41 =>
swapETHForExactTokens(uint256,address[],address,ui
nt256) ad615dec => quote(uint256,uint256,uint256)

removeLiquidityETHSupportingFeeOnTransferTokens(address,uint256,uint256,uin
t256,address,uint256) 5b0d5984 =>
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(address,uint256,uint256
,uint256,address,u int256,bool,uint8,bytes32,bytes32)
5c11d795 =>
swapExactTokensForTokensSupportingFeeOnTransferTokens(uint256,uint256,addre
ss[],address,uint256) b6f9de95 =>
swapExactETHForTokensSupportingFeeOnTransferTokens(uint256,address[],addre
ss,uint256) 791ac947 =>

a457c2d7	=>	decreaseAllowance(address,uint256)
9dc29fac	=>	burn(address,uint256)
8456cb59	=>	pause()
3f4ba83a	=>	unpause()
4dc2fe46	=>	createLiquidityPoolPair()
41cb87fc	=>	setRouterAddress(address)

swapExactTokensForETHSupportingFeeOnTransferTokens(uint256,uint256,addres
s[],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)
aa0ffca5	=>	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
Total Supply	provides information about the total token
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**

Transfer Ownership

Renounce

executes transfer of contract ownership to a specified wallet

executes transfer of contract ownership to a

Yes **Passed**

Ownership

dead address Yes **Passed**

Notable Information

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

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

- Smart contract owner can **burn** user assets. Token allowance requirements 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 smart contract 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:
        multiplication 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 be written 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. No arbitrary limit set.

```
function setLiquidityFee(uint256
    _liquidityFee) external onlyOwner() {
    require(_liquidityFee != 0,"BEP2E: Fee
    cannot 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
SWC-102	Outdated Compiler Version	! Informational
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELF-DESTRUCT Instruction	Passed
SWC-107	Re-entrancy	Passed
SWC-108	State Variable Default Visibility	Passed
SWC-109	Uninitialized Storage Pointer	Passed
SWC-110	Assert Violation	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed
SWC-112	Delegate Call to Untrusted Callee	Passed
SWC-113	DoS with Failed Call	Passed
SWC-114	Transaction Order Dependence	Passed
SWC-115	Authorization through tx.origin	Passed
SWC-116	Block values as a proxy for time	Passed

SWC-117	Signature Malleability	**Passed**
SWC-118	Incorrect Constructor Name	**Passed**

SWC-119	Shadowing State Variables	**Passed**
SWC-120	Weak Sources of Randomness from Chain Attributes	**Passed**
SWC-121	Missing Protection against Signature Replay Attacks	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**
SWC-123	Requirement Violation	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**
SWC-129	Typographical Error	**Passed**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**
SWC-131	Presence of unused variables	**Passed**
SWC-132	Unexpected Ether balance	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**
SWC-134	Message call with the hardcoded gas amount	**Passed**
SWC-135	Code With No Effects (Irrelevant/Dead Code)	**! Informational**
SWC-136	Unencrypted Private Data On-Chain	**Passed**

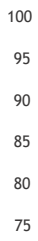
Risk Status & Radar Chart

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

LARGE CHAIN Safety

Score out of 100

Compiler Check



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 stakeholders

- 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**