



# SMART CONTRACT SECURITY AUDIT OF SERA VESTING CONTRACTS



SMART CONTRACT AUDIT | SOLIDITY DEVELOPMENT & TESTING | PROJECT EVALUATION

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN

# Audit Introduction

<b>Auditing Firm</b>	InterFi Network
<b>Audit Architecture</b>	InterFi Echelon Auditing Standard
<b>Language</b>	Solidity
<b>Client Firm</b>	SERA
<b>Website</b>	<a href="https://www.seraproject.org">https://www.seraproject.org</a>
<b>Telegram</b>	<a href="https://t.me/Sera_Project">https://t.me/Sera_Project</a>
<b>Twitter</b>	<a href="https://twitter.com/Project_SERA">https://twitter.com/Project_SERA</a>
<b>LinkedIn</b>	<a href="https://www.linkedin.com/company/sera-project">https://www.linkedin.com/company/sera-project</a>
<b>YouTube</b>	<a href="https://bit.ly/SERA_Project">https://bit.ly/SERA_Project</a>
<b>Report Date</b>	May 05, 2022

## About SERA

SERA is an ERP that runs on a layer-2 blockchain solution, combining all of the existing benefits of a regular ERP with the added privacy and security of a public blockchain. And all of this comes for free. The only condition here is getting a share of SERA's token.



# Audit Summary

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:

- ❖ SERA's vesting solidity source codes have **LOW RISK SEVERITY**
- ❖ SERA's vesting contracts have **ACTIVE OWNERSHIP**
- ❖ Centralization risk correlated to the active owner is **HIGH**
- ❖ TokenPresale.sol important privileges – **SET BUSD AND USDT EXCHANGE PRICE, END SALE, WITHDRAW, TRANSFER ACCIDENTALY LOCKED TOKENS**
- ❖ TokenPreVesting.sol important privileges – **CREATE VESTING SCHEDULE**
- ❖ TokenPreTimelock.sol important privileges – **DEPOSIT TOKENS, TRANSFER ACCIDENTALY LOCKED TOKENS**

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.

🔴 TokenPreSale.sol contract address: **0x3BE190258C362c979E7fF64679BD8bAF3c5d0969**

🔴 TokenPreVesting.sol contract address: **0x83a62d0fd10Be5Ae6915Aec16ed9690cF1d150cF**

🔴 TokenPreTimelock.sol contract address: **0xb2200513bA103E49180078e806BC09cC34bDcdeB**

🔗 Blockchain: **Binance Smart Chain**

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



# Table Of Contents

## **Audit Information**

Audit Scope.....	5
------------------	---

## **Echelon Audit Standard**

Audit Methodology .....	6
Risk Classification.....	8
Centralization Risk.....	9

## **Smart Contract Risk Assessment**

Static Analysis.....	10
Software Analysis .....	12
Manual Analysis.....	14
SWC Attacks.....	16
Risk Status & Radar Chart.....	18

## **Audit Summary**

Auditor's Verdict .....	19
-------------------------	----

## **Legal Advisory**

Important Disclaimer .....	20
About InterFi Network.....	21



# Audit Scope

InterFi was consulted by SERA Vesting 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:

- ❖ TokenPreSale.sol
- ❖ TokenPreVesting.sol
- ❖ TokenPreTimeLock.sol

## **Solidity Source Code On Blockchain**

- ❖ TokenPreSale.sol

<https://bscscan.com/address/0x3BE190258C362c979E7fF64679BD8bAF3c5d0969#code>

- ❖ TokenPreVesting.sol

<https://bscscan.com/address/0x83a62d0fd10Be5Ae6915Aec16ed9690cF1d150cF#code>

- ❖ TokenPreTimeLock.sol

<https://bscscan.com/address/0xb2200513bA103E49180078e806BC09cC34bDcdeB#code>

## **Solidity Source Code On InterFi GitHub**

<https://github.com/interfinetwork/audited-codes/blob/main/SERAVesting.sol>

## **SHA-1 Hash**

Solidity source code is audited at hash #cd8b19ff1b9635a33f6509d58b7c6ef22f6df543



# Audit Methodology

The scope of this report is to audit the smart contract source code of SERA Vesting. InterFi has scanned contracts and reviewed codes for common vulnerabilities, exploits, hacks, and backdoors. Due to being out of scope, InterFi 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

### ❖ Gas Limit and Loops

- ❖ Deployment Consistency
- ❖ Repository Consistency
- ❖ Data Consistency
- ❖ Token Supply Manipulation

### Source Code Review

- ❖ Access Control and Authorization
- ❖ Operations Trail and Event Generation
- ❖ Assets Manipulation
- ❖ Ownership Control
- ❖ Liquidity Access



## **InterFi's Echelon Audit Standard**

The aim of InterFi's "Echelon" standard is to analyze smart contracts and identify the vulnerabilities and the hacks. Kindly note, InterFi 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 InterFi to audit smart contracts:

1. Solidity smart contract source code reviewal:
  - ❖ Review of the specifications, sources, and instructions provided to InterFi 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.
2. 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.
3. 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.
4. 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.

## Smart Contract Security Audit

Risk severity	Meaning
<b>! High</b>	This level vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
<b>! Medium</b>	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
<b>! Low</b>	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.
<b>! Informational</b>	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.

## Centralization Status

- ❖ SERA's vesting contracts have **active ownership**.



# Static Analysis

Symbol	Meaning
	Function can modify state
	Function is payable
	Function is locked
	Function can be accessed
!	Important functionality

```

||||| |
| **TokenPreSale** | Implementation | Ownable |||
| L | <Constructor> | Public ! |  | NO ! |
| L | setExchangePriceUSDT | External ! |  | onlyOwner |
| L | setExchangePriceBUSD | External ! |  | onlyOwner |
| L | setDuration | External ! |  | onlyOwner |
| L | setCliff | External ! |  | onlyOwner |
| L | setTimeStamp | External ! |  | onlyOwner |
| L | setSaleStatus | External ! |  | onlyOwner |
| L | setAvailableAtTGE | External ! |  | onlyOwner |
| L | transferAccidentallyLockedTokensInTimeLock | External ! |  | onlyOwner |
| L | setBuyAmountRangeBUSD | External ! |  | onlyOwner |
| L | setBuyAmountRangeUSDT | External ! |  | onlyOwner |
| L | buyTokensUsingBUSD | External ! |  | onSale |
| L | buyTokensUsingUSDT | External ! |  | onSale |
| L | computeTokensForBUSD | Public ! | NO ! |
| L | computeTokensForUSDT | Public ! | NO ! |
| L | withdrawBUSD | Public ! |  | onlyOwner |
| L | withdrawUSDT | Public ! |  | onlyOwner |
| L | withdrawFromVesting | Public ! |  | onlyOwner |
| L | transferAccidentallyLockedTokensFromTimeLock | Public ! |  | onlyOwner |
| L | revoke | External ! |  | onlyOwner |
| L | endSale | External ! |  | onlyOwner |
|||||
| **TokenPreVesting** | Implementation | Ownable, ReentrancyGuard |||
| L | <Constructor> | Public ! |  | NO ! |
| L | setTimestamp | External ! |  | onlyOwner onlyIfLaunchTimestampNotSet |
| L | getVestingSchedulesCountByBeneficiary | External ! | NO ! |
| L | getVestingIdAtIndex | External ! | NO ! |
| L | getVestingScheduleByAddressAndIndex | External ! | NO ! |
| L | getVestingSchedulesTotalAmount | External ! | NO ! |

```



```

| L | getToken | External ! | | NO ! |
| L | createVestingSchedule | Public ! | ● | incomingDepositsStillAllowed onlyOwner |
| L | createVestingSchedule | External ! | ● | incomingDepositsStillAllowed onlyOwner |
| L | revoke | Public ! | ● | onlyOwner onlyIfVestingScheduleNotRevoked |
| L | withdraw | Public ! | ● | nonReentrant onlyOwner |
| L | release | Public ! | ● | nonReentrant onlyIfVestingScheduleNotRevoked |
| L | getVestingSchedulesCount | Public ! | | NO ! |
| L | computeReleasableAmount | Public ! | | onlyIfVestingScheduleNotRevoked |
| L | getVestingSchedule | Public ! | | NO ! |
| L | getWithdrawableAmount | Public ! | | NO ! |
| L | computeNextVestingScheduleIdForHolder | Public ! | | NO ! |
| L | getLastVestingScheduleForHolder | Public ! | | NO ! |
| L | computeVestingScheduleIdForAddressAndIndex | Public ! | | NO ! |
| L | _computeReleasableAmount | Internal 🔒 | | |
| L | getCurrentTime | Public ! | | NO ! |
|||||
| **TokenPreTimeLock** | Implementation | Ownable |||
| L | <Constructor> | Public ! | ● | NO ! |
| L | getToken | External ! | | NO ! |
| L | setTimestamp | Public ! | ● | onlyOwner timestampNotSet |
| L | depositTokens | Public ! | ● | onlyOwner incomingDepositsStillAllowed |
| L | bulkDepositTokens | External ! | ● | onlyOwner incomingDepositsStillAllowed |
| L | transferTimeLockedTokensAfterTimePeriod | Public ! | ● | timestampIsSet noReentrant |
| L | transferAccidentallyLockedTokens | Public ! | ● | onlyOwner noReentrant |
| L | getCurrentTime | Public ! | | NO ! |

```

.....

## Smart Contract Security Audit



# Software Analysis

## Function Signatures

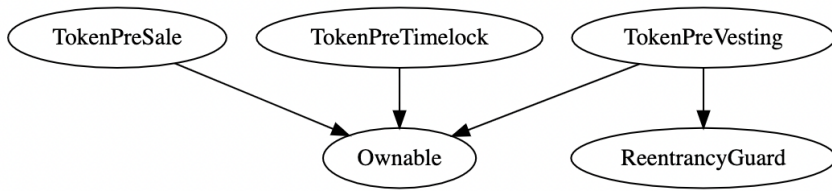
```

13083617 => getVestingSchedulesCount()
8ddeaf39 => setExchangePriceUSDT(uint256)
da499ad0 => setExchangePriceBUSD(uint256)
f6be71d1 => setDuration(uint256)
ff4d1f25 => setCliff(uint256)
13e43bc6 => setTimeStamp(uint256)
b1cefa28 => setSaleStatus(SaleStatus)
9a99e3c5 => setAvailableAtTGE(uint256)
111c1056 => transferAccidentallyLockedTokensInTimeLock(IERC20,uint256)
0e6a2f0a => setBuyAmountRangeBUSD(uint256,uint256)
ec41f7dd => setBuyAmountRangeUSDT(uint256,uint256)
f3fe4809 => buyTokensUsingBUSD(uint256)
b937f1e0 => buyTokensUsingUSDT(uint256)
7540cd63 => computeTokensForBUSD(uint256)
77cdf957 => computeTokensForUSDT(uint256)
5b0b8596 => withdrawBUSD()
362e496b => withdrawUSDT()
a5dee4eb => withdrawFromVesting(uint256)
6977454e => transferAccidentallyLockedTokensFromTimelock(IERC20,uint256)
b75c7dc6 => revoke(bytes32)
380d831b => endSale()
a0a2b573 => setTimeStamp(uint256)
5a7bb69a => getVestingSchedulesCountByBeneficiary(address)
f9079b37 => getVestingIdAtIndex(uint256)
f51321d7 => getVestingScheduleByAddressAndIndex(address,uint256)
48deb471 => getVestingSchedulesTotalAmount()
21df0da7 => getToken()
530ff4e5 => createVestingSchedule(address,uint256,uint256,uint256,bool,uint256,uint256)
d1f3c24d =>
createVestingSchedule(address[],uint256[],uint256[],uint256[],bool[],uint256[],uint256[])
2e1a7d4d => withdraw(uint256)
66afd8ef => release(bytes32,uint256)
ea1bb3d5 => computeReleasableAmount(bytes32)
9ef346b4 => getVestingSchedule(bytes32)
90be10cc => getWithdrawableAmount()
f7c469f0 => computeNextVestingScheduleIdForHolder(address)
7e913dc6 => getLastVestingScheduleForHolder(address)
8af104da => computeVestingScheduleIdForAddressAndIndex(address,uint256)
1be9dc72 => _computeReleasableAmount(VestingSchedule)
29cb924d => getCurrentTime()
66168bd7 => depositTokens(address,uint256)
d1484298 => bulkDepositTokens(address[],uint256[])
1ace373e => transferTimeLockedTokensAfterTimePeriod(IERC20,address,uint256)
1b503821 => transferAccidentallyLockedTokens(IERC20,uint256)

```



## Inheritance Graph



# InterFi

Smart Contract  
Security Audit



# Manual Analysis


## Notable Information

- ❖ TokenPreVesting.sol smart contract **utilizes** reentrancy guards to prevent known reentrant vulnerabilities. Reentrancy Guard is a contract module that helps prevent reentrant calls to a function.
- ❖ SERA's smart contracts TokenPreTimelock.sol and TokenPreVesting.sol have a low severity issue which may not create any functional vulnerability.

### Utilization of block.timestamp

"severity": 8, (! Low Severity)

- ❖ Smart contracts utilize **safemath** function to avoid common smart contract vulnerabilities.



```
string private _name = "SERAVesting";
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");
```

- ❖ TokenPreSale.sol smart contract owner can `withdrawFromVesting()`, `withdrawUSDT()`, `withdrawBUSD()`, and `transferAccidentallyLockedTokensFromTimelock()`.
- ❖ TokenPreVesting.sol smart contract owner can `withdraw()` funds.

```
function withdraw(uint256 amount) public nonReentrant onlyOwner {
    require(this.getWithdrawableAmount() >= amount, "TokenPreVesting: not enough withdrawable funds");
```



- ❖ TokenPreTimelock.sol owner can `transferAccidentallyLockedTokensFromTimelock()`.
- ❖ Smart contract owner can **set min – max USDT, and min – max BUSD buy amount.**

```
function setBuyAmountRangeBUSD(uint256 _min, uint256 _max) external onlyOwner {
    minBuyAmountBUSD = _min;
    maxBuyAmountBUSD = _max;
function setBuyAmountRangeUSDT(uint256 _min, uint256 _max) external onlyOwner {
    minBuyAmountUSDT = _min;
    maxBuyAmountUSDT = _max;
```

# InterFi

## Smart Contract Security Audit



# 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



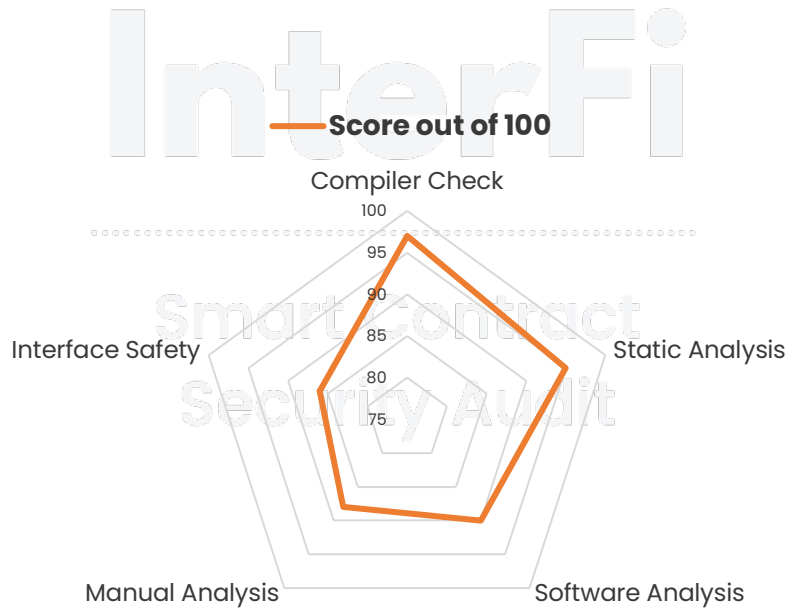


<b>SWC-119</b>	Shadowing State Variables	<b>Passed</b>
<b>SWC-120</b>	Weak Sources of Randomness from Chain Attributes	<b>Passed</b>
<b>SWC-121</b>	Missing Protection against Signature Replay Attacks	<b>Passed</b>
<b>SWC-122</b>	Lack of Proper Signature Verification	<b>Passed</b>
<b>SWC-123</b>	Requirement 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 Length Arguments	<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>! Low</b>
<b>SWC-136</b>	Unencrypted Private Data On-Chain	<b>Passed</b>



# Risk Status & Radar Chart

Risk Severity	Status
High	No high severity issues identified
Medium	No medium severity issues identified
Low	2 low severity issues identified
Informational	1 informational severity issue identified
Centralization Risk	Active contract ownership identified



## 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:

- ❖ SERA Vesting's smart contract source code has **LOW RISK SEVERITY**
- ❖ SERA Vesting's smart contract has an **ACTIVE OWNERSHIP**
- ❖ SERA Vesting's centralization risk correlated to the active owner is **HIGH**

# InterFi

### Note for stakeholders

## Smart Contract Security Audit

- ❖ 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

InterFi Network 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.**

InterFi 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 InterFi Network

InterFi Network provides intelligent blockchain solutions. InterFi is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. **InterFi's 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. **InterFi 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://interfi.network>

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

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





**@INTERFINETWORK**

**RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA **