



## SMART CONTRACT SECURITY AUDIT OF



SMART CONTRACT AUDIT | TEAM KYC | PROJECT EVALUATION

RELENTLESSLY SECURING THE PUBLIC BLOCKCHAIN | MADE IN CANADA 

# Summary

<b>Auditing Firm</b>	InterFi Network
<b>Architecture</b>	InterFi "Echelon" Auditing Standard
<b>Smart Contract Audit Approved By</b>	Chris   Blockchain Specialist at InterFi Network
<b>Project Overview Approved By</b>	Albert   Marketing Specialist at InterFi Network
<b>Platform</b>	Solidity
<b>Mandatory Audit Check</b>	Static, Software, Auto Intelligent & Manual Analysis
<b>Consultation Request Date</b>	November 13, 2021
<b>Report Date</b>	November 14, 2021

## Audit Summary

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

- ❖ **RexFinance's smart contract source code has **LOW RISK SEVERITY**.**
- ❖ **RexFinance has **PASSED** the smart contract audit.**

For the detailed understanding of risk severity, source code vulnerability, and functional test, kindly refer to the audit.

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



# Table Of Contents

## **Project Information**

Overview .....	4
----------------	---

## **InterFi “Echelon” Audit Standard**

Audit Scope & Methodology .....	6
InterFi’s Risk Classification.....	8

## **Smart Contract Risk Assessment**

Static Analysis.....	9
Software Analysis .....	11
Manual Analysis.....	13
SWC Attacks.....	15
Risk Status & Radar Chart.....	17

## **Report Summary**

Auditor’s Verdict .....	18
-------------------------	----

## **Legal Advisory**

Important Disclaimer .....	19
About InterFi Network.....	20



# Project Overview

InterFi was consulted by RexFinance to conduct the smart contract security audit of their solidity source code.

## About RexFinance

Introducing Rex Finance which aims to bridge the connection between defi and fiat currencies.

<b>Project</b>	RexFinance
<b>Blockchain</b>	Binance Smart Chain
<b>Language</b>	Solidity
<b>Contract</b>	0xD5D4D47922422789c550773024F20c936C6DFDAd
<b>Website</b>	<a href="https://rexfinance.net/">https://rexfinance.net/</a>
<b>Telegram</b>	<a href="https://t.me/rexfinancecomm">https://t.me/rexfinancecomm</a>
<b>Twitter</b>	<a href="https://twitter.com/_rexfinance?s=11">https://twitter.com/_rexfinance?s=11</a>

## Public logo



## **Solidity Source Code On Blockchain** (Verified Contract Source Code)

<https://bscscan.com/address/0xD5D4D47922422789c550773024F20c936C6DFDAd#code>

Contract Name: RexFinance (AntiBotStandardERC20)

Symbol: REX

Compiler Version: v0.7.6

Optimization Enabled: Yes with 200 runs

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

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

## **SHA-1 Hash**

Solidity source code is audited at hash #8a0ae3e570d8106cff0bd829490578eab2eedc29



# Audit Scope & Methodology

The scope of this report is to audit the smart contract source code of RexFinance. InterFi has scanned the contract and reviewed the project for common vulnerabilities, exploits, hacks, and back-doors. 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

- ❖ Ownership Takeover
- ❖ Gas Limit and Loops

### Source Code Review

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

### Functional Assessment

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



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

The aim of InterFi's "Echelon" standard is to analyze the smart contract and identify the vulnerabilities and the hacks in the smart contract. Mentioned are the steps used by ECHELON-1 to assess the smart contract:

1. Solidity smart contract source code reviewal:
  - ❖ Review of the specifications, sources, and instructions provided to InterFi to make sure we understand the size, scope, and functionality of the smart contract.
  - ❖ Manual review of code, which is the process of reading source code line-byline to identify potential vulnerabilities.
2. Static, Manual, and Software analysis:
  - ❖ Test coverage analysis, which 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, which is analysing a program to determine what inputs causes 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**

- ❖ Slither
- ❖ Consensys MythX
- ❖ Consensys Surya
- ❖ Open Zeppelin Code Analyzer
- ❖ Solidity Code Compiler



# InterFi's 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 which requires to own 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
<b>! Critical</b>	This level vulnerabilities could be exploited easily, and can lead to asset loss, data loss, asset manipulation, or data manipulation. They should be fixed right away.
<b>! High</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 critical risk severity
<b>! Medium</b>	This level vulnerabilities are 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>! Low</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





# Smart Contract – Static Analysis

Symbol	Meaning
	Function can be modified
	Function is payable
	Function is locked
	Function can be accessed
	Important functionality

```

**IERC20Upgradeable** | Interface | |||
| L | totalSupply | External ! | NO ! | |
| L | balanceOf | External ! | NO ! |
| L | transfer | External ! | NO ! |
| L | allowance | External ! | NO ! |
| L | approve | External ! | NO ! |
| L | transferFrom | External ! | NO ! |
| |||||
**AddressUpgradeable** | Library | |||
| L | isContract | Internal | | |
| L | sendValue | Internal | |
| L | functionCall | Internal | |
| L | functionCall | Internal | |
| L | functionCallWithValue | Internal | |
| L | functionCallWithValue | Internal | |
| L | functionStaticCall | Internal | |
| L | functionStaticCall | Internal | |
| L | _verifyCallResult | Private | |
| |||||
**Initializable** | Implementation | |||
| L | _isConstructor | Private | |
| |||||
**ContextUpgradeable** | Implementation | Initializable |||
| L | __Context_init | Internal | | initializer |
| L | __Context_init_unchained | Internal | | initializer |
| L | _msgSender | Internal | |
| L | _msgData | Internal | |
| |||||
**OwnableUpgradeable** | Implementation | Initializable, ContextUpgradeable |||
| L | __Ownable_init | Internal | | initializer |
| L | __Ownable_init_unchained | Internal | | initializer |

```



```

| L | owner | Public ! | |NO ! |
| L | renounceOwnership | Public ! | ● | onlyOwner |
| L | transferOwnership | Public ! | ● | onlyOwner |
|||||
| SafeMath | Library | |||
| L | tryAdd | Internal 🔒 | | |
| L | trySub | Internal 🔒 | | |
| L | tryMul | Internal 🔒 | | |
| L | tryDiv | Internal 🔒 | | |
| L | 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 🔒 | | |
|||||
| IPinkAntiBot | Interface | |||
| L | setTokenOwner | External ! | ● |NO ! |
| L | onPreTransferCheck | External ! | ● |NO ! |
|||||
| IAntiBotStandardERC20 | Interface | |||
| L | initialize | External ! | ● |NO ! |
|||||
| AntiBotStandardERC20 | Implementation | IERC20Upgradeable, OwnableUpgradeable,
IAntiBotStandardERC20 |||
| L | initialize | External ! | ● | initializer |
| L | setEnableAntiBot | External ! | ● | onlyOwner |
| L | name | Public ! | |NO ! |
| L | symbol | Public ! | |NO ! |
| L | decimals | Public ! | |NO ! |
| L | totalSupply | Public ! | |NO ! |
| L | balanceOf | Public ! | |NO ! |
| L | transfer | Public ! | ● |NO ! |
| L | allowance | Public ! | |NO ! |
| L | approve | Public ! | ● |NO ! |
| L | transferFrom | Public ! | ● |NO ! |
| L | increaseAllowance | Public ! | ● |NO ! |
| L | decreaseAllowance | Public ! | ● |NO ! |
| L | _transfer | Internal 🔒 | ● | |
| L | _mint | Internal 🔒 | ● | |
| L | _burn | Internal 🔒 | ● | |
| L | _approve | Internal 🔒 | ● | |
| L | _setupDecimals | Internal 🔒 | ● | |
| L | _beforeTokenTransfer | Internal 🔒 | ● | |

```



# Smart Contract – Software Analysis

## Function Signatures

```

16279055 => isContract(address)
39509351 => increaseAllowance(address,uint256)
48760858 => onPreTransferCheck(address,address,uint256)
18160ddd => totalSupply()
70a08231 => balanceOf(address)
a9059cbb => transfer(address,uint256)
dd62ed3e => allowance(address,address)
095ea7b3 => approve(address,uint256)
23b872dd => transferFrom(address,address,uint256)
24a084df => sendValue(address,uint256)
a0b5ffb0 => functionCall(address,bytes)
241b5886 => functionCall(address,bytes,string)
2a011594 => functionCallWithValue(address,bytes,uint256)
d525ab8a => functionCallWithValue(address,bytes,uint256,string)
c21d36f3 => functionStaticCall(address,bytes)
dbc40fb9 => functionStaticCall(address,bytes,string)
18c2c6a2 => _verifyCallResult(bool,bytes,string)
45549c1f => _isConstructor()
f08d647e => __Context_init()
ab96f671 => __Context_init_unchained()
119df25f => _msgSender()
8b49d47e => _msgData()
0142eb11 => __Ownable_init()
5ce29e24 => __Ownable_init_unchained()
8da5cb5b => owner()
715018a6 => renounceOwnership()
f2fde38b => 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)
18e02bd9 => setTokenOwner(address)
e39cba87 => initialize(address,string,string,uint8,uint256,address)
1f46b1c6 => setEnableAntiBot(bool)
06fdde03 => name()
95d89b41 => symbol()

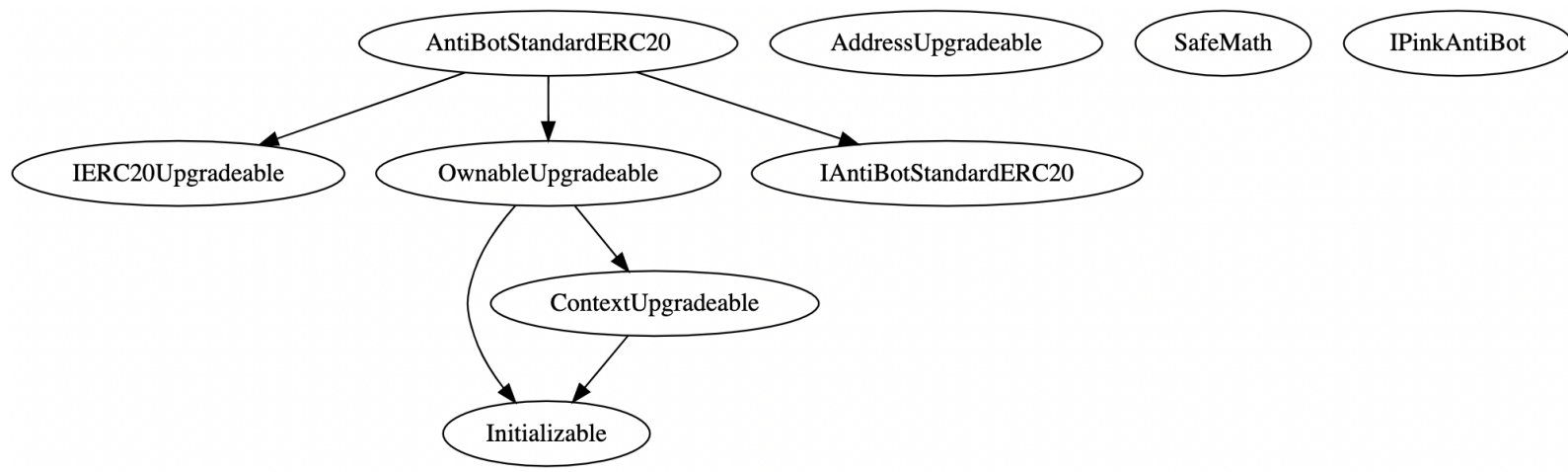
```



```

313ce567 => decimals()
a457c2d7 => decreaseAllowance(address,uint256)
30e0789e => _transfer(address,address,uint256)
4e6ec247 => _mint(address,uint256)
6161eb18 => _burn(address,uint256)
104e81ff => _approve(address,address,uint256)
61e9edb2 => _setupDecimals(uint8)
cad3be83 => _beforeTokenTransfer(address,address,uint256)
  
```

## Inheritance Graph



Security Audit



# Smart Contract – Manual Analysis

Function	Description	Tested	Verdict
<b>Total Supply</b>	provides information about the total token supply	Yes	Passed
<b>Balance Of</b>	provides account balance of the owner's account	Yes	Passed
<b>Transfer</b>	executes transfers of a specified number of tokens to a specified address	Yes	Passed
<b>Approve</b>	allow a spender to withdraw a set number of tokens from a specified account	Yes	Passed
<b>Allowance</b>	returns a set number of tokens from a spender to the owner	Yes	Passed
<b>Buy Back</b>	is an action in which the project buys back its tokens from the existing holders usually at a market price	NA	NA
<b>Burn</b>	executes transfers of a specified number of tokens to a burn address	NA	NA
<b>Mint</b>	executes creation of a specified number of tokens and adds it to the total supply	NA	NA
<b>Rebase</b>	circulating token supply adjusts (increases or decreases) automatically according to a token's price fluctuations	NA	NA
<b>Blacklist</b>	stops specified wallets from interacting with the smart contract function modules	NA	NA
<b>Lock</b>	stops or locks all function modules of the smart contract	NA	NA



## Review

- ❖ Active smart contract owner: 0xa731af05d014804ddcdcd0505bc2e564ed031622
- ❖ PinkAntiBot: 0x8efdb3b642eb2a20607ffe0a56cfeff6a95df002
- ❖ Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's safety and security.
- ❖ Owner can-not lock or burn user assets.
- ❖ Owner can-not stop or pause the smart contract.
- ❖ Owner can-not mint tokens after launch.
- ❖ The smart contract utilizes "SafeMath" function to avoid common smart contract vulnerabilities.

```

string private _name = "RexFinance";
string private _symbol = "REX";

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 – SWC Attacks

SWC ID	Description	Verdict
SWC-101	Integer Overflow and Underflow	Passed
SWC-102	Outdated Compiler Version	! Low
SWC-103	Floating Pragma	Passed
SWC-104	Unchecked Call Return Value	Passed
SWC-105	Unprotected Ether Withdrawal	Passed
SWC-106	Unprotected SELFDESTRUCT 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



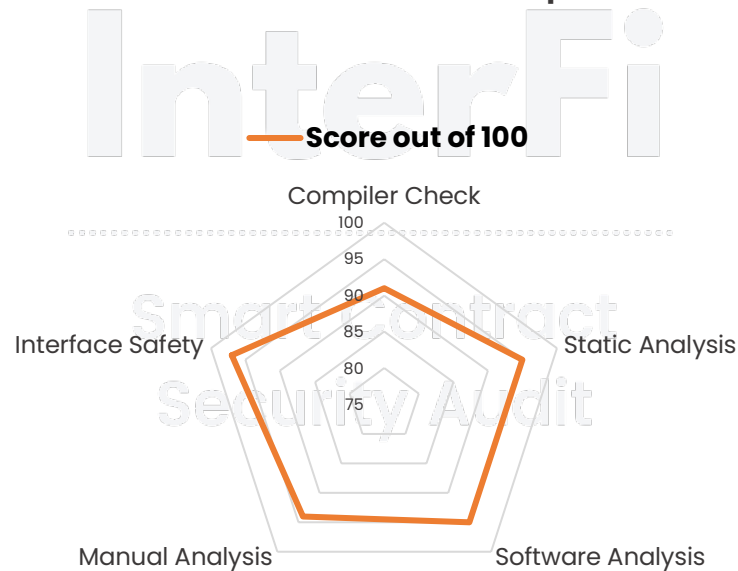
<b>SWC-119</b>	Shadowing State Variables	<b>Passed</b>
<b>SWC-120</b>	Weak Sources of Randomness from Chain Attributes	<b>Passed</b>
<b>SWC-121</b>	Missing Protection against Signature Replay Attacks	<b>Passed</b>
<b>SWC-122</b>	Lack of Proper Signature Verification	<b>Passed</b>
<b>SWC-123</b>	Requirement 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 hardcoded gas amount	<b>Passed</b>
<b>SWC-135</b>	Code With No Effects (Irrelevant/Dead Code)	<b>Passed</b>
<b>SWC-136</b>	Unencrypted Private Data On-Chain	<b>Passed</b>





# Smart Contract - Risk Status & Radar Chart

Risk Severity	Status
<b>! Critical</b>	None critical severity issues identified
<b>! High</b>	None high severity issues identified
<b>! Medium</b>	None medium severity issues identified
<b>! Low</b>	<b>1 low severity issue identified</b>
<b>Passed</b>	<b>41 functions and instances verified and passed</b>



Compiler Check	91
Static Analysis	95
Software Analysis	95
Manual Analysis	94
Interface Safety	97



## Auditor's Verdict

InterFi team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analyzed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks.

**RexFinance's smart contract source code has **LOW RISK SEVERITY**.**

**RexFinance has **PASSED** the smart contract audit.**

# InterFi

.....

## Smart Contract Security Audit

### Note for stakeholders

- ❖ Be aware that active smart contract owner privileges constitute an elevated impact on smart contract's safety and security.
- ❖ Make sure that the project team's KYC/identity is verified by an independent firm, e.g., InterFi.
- ❖ Always check if the contract's liquidity is locked. A longer liquidity lock plays an important role in 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 of time.
- ❖ Ensure that the project's official website is hosted on a trusted platform, and is using an active SSL certificate. The website's domain should be registered for a longer period of time.



# Important Disclaimer

InterFi Network provides contract auditing and project verification services for blockchain projects. The purpose of the audit is to analyse the on-chain smart contract source code, and to provide basic overview of the project. **This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purposes 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 an 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 from external vulnerability, or a hack. Be aware that active smart contract owner privileges constitute an elevated impact to smart contract's 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 on 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 own 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 🇨🇦**