InterFi NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

NFT LEND AUCTION CONTRACT

[VAULTLAYER]

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	VaultLayer
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0x4953D0e96D13b9802361102837f16F844a135EAb
Blockchain	Core
Centralization	Active Ownership
Commit	c56916d9e94dd986930df1512c3bfeeff7f6f77e
Website	
Telegram	
Report Date	January 07, 2025

ℹ Verify the authenticity of this report on our website: https://www.github.com/interfinetwork

EXECUTIVE SUMMARY

InterFi has performed the automated and manual analysis of solidity codes. Solidity codes were reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical 🔴	Major 🟠	Medium 🟡	Minor 🟢	Unknown 🟤
Open	0	0	0	2	0
Acknowledged	0	1	0	1	1
Resolved	1	0	2	4	1
Important Functions	`listLoan, placeBid, delistLoan, acceptLoan, repayLoan, cancelBid, claimDefaultedLoan`				
Noteworthy Privileges	`grantManagerRole, revokeManagerRole, setMaxActiveLoans, updateAllowedNFT, setProtocolFeeRate, withdrawProtocolFees, withdrawEther`				

ℹ️ Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

ℹ️ Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

TABLE OF CONTENTS

TABLE OF CONTENTS 4

SCOPE OF WORK 5

AUDIT METHODOLOGY 6

RISK CATEGORIES 8

CENTRALIZED PRIVILEGES 9

AUTOMATED ANALYSIS 10

MANUAL REVIEW 11

DISCLAIMERS 29

ABOUT INTERFI NETWORK 32

SCOPE OF WORK

InterFi was consulted by VaultLayer to conduct the smart contract audit of their solidity source codes.

The audit scope of work is strictly limited to mentioned solidity file(s) only:

- NFTLendAuction.sol

ℹ️ If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Verify the contract's deployment status below:

Public Contract Link	
https://scan.coredao.org/address/0x4953D0e96D13b9802361102837f16F844a135EAb#code	
Contract Name	NFTLendAuction
Compiler Version	0.8.23
License	MIT

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of InterFi's auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
  - Remix IDE Developer Tool
  - Open Zeppelin Code Analyzer
  - SWC Vulnerabilities Registry
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	○ Token Supply Manipulation ○ Access Control and Authorization ○ Assets Manipulation ○ Ownership Control ○ Liquidity Access ○ Stop and Pause Trading ○ Ownable Library Verification

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to solidity codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

ℹ️ It is important to note that there is no pass or fail in the audit, it is recommended to view the audit as an unbiased assessment of the safety of solidity codes.

RISK CATEGORIES

A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized:

Risk Type	Definition
Critical 🔴	These risks pose immediate and severe threats, such as asset theft, data manipulation, or complete loss of contract functionality. They are often easy to exploit and can lead to significant, irreparable damage. Immediate fix is required.
Major 🟠	These risks can significantly impact code performance and security, and they may indirectly lead to asset theft and data loss. They can allow unauthorized access or manipulation of sensitive functions if exploited. Fixing these risks are important.
Medium 🟡	These risks may create attack vectors under certain conditions. They may enable minor unauthorized actions or lead to inefficiencies that can be exploited indirectly to escalate privileges or impact functionality over time.
Minor 🟢	These risks may include inefficiencies, lack of optimizations, code-style violations. These should be addressed to enhance overall code quality and maintainability.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. Immediate fix is required to mitigate risk uncertainty.

All statuses which are identified in the audit report are categorized here:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

- Privileged roles can be granted the power to `pause()` the contract in case of an external attack.
- Privileged roles 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 privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- The client can lower centralization-related risks by implementing below mentioned practices:
- Privileged role's private key must be carefully secured to avoid any potential hack.
- Privileged role should be shared by multi-signature (multi-sig) wallets.
- Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.
- Renouncing the contract ownership, and privileged roles.
- Remove functions with elevated centralization risk.

ℹ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💸	Function is payable
🔒	Function is internal
🔐	Function is private
!	Function is important

/v1

```
└  <Constructor> Public !  🛑  Sets initial roles
└  grantManagerRole  External !  🛑  onlyRole(DEFAULT_ADMIN_ROLE)
└  revokeManagerRole  External !  🛑  onlyRole(DEFAULT_ADMIN_ROLE)
└  setMaxActiveLoans  External !  🛑  onlyRole(OWNER_ROLE)
└  updateAllowedNFT  External !  🛑  onlyRole(MANAGER_ROLE)
└  setProtocolFeeRate  External !  🛑  onlyRole(OWNER_ROLE)
└  listLoan  External !  🛑  nonReentrant isAllowedNFT onlyNftOwner
└  placeBid  External !  🛑  nonReentrant loanExists isNotAccepted
└  delistLoan  External !  🛑  nonReentrant loanExists isNotAccepted onlyBorrower
└  acceptLoan  External !  🛑  nonReentrant loanExists isNotAccepted onlyBorrower
└  getTotalRepayment  Public !  loanExists
└  repayLoan  External !  🛑  nonReentrant loanExists onlyBorrower
└  cancelBid  External !  🛑  nonReentrant loanExists isNotAccepted onlyLender
└  claimDefaultedLoan  External !  🛑  nonReentrant loanExists
└  getActiveLoans  External !
└  withdrawProtocolFees  External !  🛑  nonReentrant onlyRole(OWNER_ROLE)
└  withdrawEther  External !  🛑  nonReentrant onlyRole(OWNER_ROLE)
└  _removeActiveLoan  Private 🔐  🛑
```

MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized and controlled privileges	Major 🟠
CEN-01-01	Owner role has authority to grant manager role	
CEN-01-02	Owner role can withdraw fees and ether	

`onlyRole` controlled privileges are listed below:

```
grantManagerRole
setMaxActiveLoans
updateAllowedNFT
revokeManagerRole
setProtocolFeeRate
withdrawProtocolFees
withdrawEther
```


RECOMMENDATION

Securing private keys or access credentials of deployers, contract owners, managers, and other roles with privileged access is crucial to prevent single points of failure that can compromise contract security.

Use of multi-signature wallets is recommended – These wallets require multiple authorizations to execute sensitive contract functions, reducing the risk associated with single-party control.

Use of decentralized governance model is recommended – This model allows token holders and stakeholders to actively participate in decision-making, such as contract upgrades and parameter adjustments, enhancing overall security and resilience.

ACKNOWLEDGEMENT

VaultLayer team argued that centralized and controlled privileges are used as required. VaultLayer team will apply multi-sig ownership approach if possible.

Identifier	Definition	
CEN-03	Lack of circuit breaker	

Smart contract currently lacks a circuit breaker or emergency stop mechanism. Circuit breakers are crucial for halting contract functionality in response to active security breaches or critical bugs, allowing administrators to freeze all non-essential activities until issues are resolved.

NOTE

Use a circuit breaker mechanism with a simple state variable that can disable critical functionalities such as token transfers, staking, withdrawals, and reward distribution when activated.

ACKNOWLEDGEMENT

VaultLayer team commented that there is a `maxActiveLoans` parameter to circuit break the creation of new listed loans if required. There is no dedicated circuit breaker mechanism.

Identifier	Definition	Severity
LOG-02	Potential front-running	Minor

Potential front-running happens when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by front-running a transaction to purchase assets and make profits by back-running a transaction to sell assets.

`placeBid()`

Current implementation could allow a miner or observer to see a submitted transaction with a favorable bid and place another transaction with a slightly better offer before the original is confirmed, exploiting the public nature of pending transactions.

RECOMMENDATION

Implement commit-reveal schemes or transaction ordering to protect against front-running.

ACKNOWLEDGEMENT

Front-running is not avoidable on public blockchains. VaultLayer team commented that, most EVM chains are prone to some sort of front-running and external manipulation. VaultLayer team added that in reversed auction market, the behaviour of placing better bids is intended logic.

Identifier	Definition
LOG-03	Re-entrancy
LOG-04	Checks-Effects-Interactions

Below mentioned functions are using `nonReentrant` modifier to protect against re-entrancy:

`listLoan`
`placeBid`
`delistLoan`
`acceptLoan`
`repayLoan`
`cancelBid`
`claimDefaultedLoan`
`withdrawProtocolFees`
`withdrawEther`

Identifier	Definition	
LOG-04-01	Checks-Effects-Interactions	Medium 🟡

Below mentioned function should adhere to Checks-Effects-Interactions pattern:

`placeBid` – Performs an Ether transfer `(payable(loan.lender).transfer(escrowedFunds[loanId]))` before all state updates are finalized.

`delistLoan` – Can transfer funds before all state changes are committed.

`acceptLoan` – Funds transfer to the borrower happens before loan's state is fully updated to reflect that it has been accepted, which includes setting the `startTime` and changing the `isAccepted` flag.

RECOMMENDATION

Follow Checks-Effects-Interactions strictly, and make sure all state changes are committed (effects), before transfers (interactions).

RESOLUTION

State assignments are committed before transfers.

Identifier	Definition	Severity
COD-01	Dependency on NFT contracts	Unknown

Smart contract assumes that NFT contract it interacts with is compliant with the ERC721 standard without malicious behaviors. Malicious or poorly implemented NFT contracts can cause unexpected behaviors, and create transfer failures.

`listLoan`
`delistLoan`
`claimDefaultedLoan`

RECOMMENDATION

Provide strict checks on NFT contracts or maintain a list of approved and audited NFT contracts. Additionally, use safe transfer methods and verify transfer success.

RESOLUTION

VaultLayer team commented that there is a governance parameter `allowedNFTContracts` to control which NFT collections are vetted for use. Added adds stricter checks for NFT contracts. For example, `supportsInterface(0x80ac58cd)`, nonzero code size, etc. Smart contract also uses `isCollateralized[nftAddress][tokenId]` to avoid double-collateralizing the same NFT.

Identifier	Definition	Severity
COD-02	Timestamp dependence for loan calculation	Minor

Use of `block.timestamp` introduces potential risks due to the minor manipulability of timestamps by miners. It can affect interest calculations when they are based on block times.

`getTotalRepayment` in APR calculation.

RECOMMENDATION

Avoid relying solely on timestamp of the block for critical contract functions. Use an average block time metric for more predictable results when function allows for such approximation.

Identifier	Definition	Severity
COD-03	Potential denial of service (DoS)	Critical 🔴

When lender continually places bids slightly lower than the current interest rate and keeps cancelling them, it can lock out other lenders and cause a denial of service for practical loan progress.

`placeBid`	This function handles transferring or refunding large amounts of Ether. If this operation fails (for example, if the previous lender is a contract that rejects the transaction), the entire function reverts, which can prevent legitimate bids from being placed. (COD-13)
`delistLoan`	This function allows borrowers to remove their loan, which involves refunding the current lender and returning the NFT to the borrower. If either the fund transfer or the NFT transfer fails (for example, when NFT contract reverts or runs out of gas), this function will revert. (COD-01) (COD-13)
`repayLoan`	If smart contract balance is low or if gas costs fluctuate leading to out-of-gas exceptions, this function may fail. (COM-04)
`claimDefaultedLoan`	This function allows lenders to claim NFT collateral when a loan defaults. It involves transferring an NFT, which can fail if NFT contract does not behave as expected. (COD-01)

RECOMMENDATION

Add logical mechanisms that allow smart contract to handle transaction failures gracefully, for example allowing state updates even if a fund transfer fails, possibly by queuing failed transactions for manual or automated retry.

V1 UPDATE

While V1 introduces `try/catch` blocks around NFT transfers in a few places, it ultimately reverts on any failure. A malicious NFT contract or payee address can still block the entire operation in each of these functions, yielding a potential denial of service (DoS) scenario.

`placeBid`	Transfers Ether to previous bidder in a single step. If that fails, everything reverts. (No update)
`delistLoan`	Uses try/catch for the NFT transfer but immediately reverts if it fails; refunds previous bidder afterward. Can still lead to DoS. (No update)
`repayLoan`	Reverts if `safeTransferFrom` for the NFT fails. Also depends on immediate Ether transfer out. (No update)
`claimDefaultedLoan`	If the NFT transfer reverts, the entire function reverts. (No update)

V2 UPDATE

V2 now uses a pull payment approach `pendingWithdrawals` for refunds. This reduces DoS risk from direct transfers. A malicious NFT contract can still revert `safeTransferFrom`, blocking loan closures in `claimDefaultedLoan`. However, as per COD-01, VaultLayer team acknowledges that there is a governance parameter `allowedNFTContracts` to control which NFT collections are vetted for use.

RESOLUTION

Instead of immediately sending refunds to the previous lender, V2 stores escrowed amounts in `pendingWithdrawals`. This avoids transaction reverts and reduces DoS risks. V2 verifies NFT interfaces and sets `isCollateralized` flags to reduce unexpected transfer failures.

Identifier	Definition	Severity
COD-10	Direct and indirect dependencies	Unknown

`NFTLendAuction` contract relies on third-party *ERC721* (NFT) contracts and *OpenZeppelin* libraries (`ReentrancyGuard` and `AccessControl`). It assumes these external contracts and libraries function correctly and securely. However, vulnerabilities or updates in these dependencies could impact the contract's operations, introducing risks such as security flaws or changes in functionality. Continuous monitoring and regular updates are recommended to manage these dependencies effectively.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary.

ACKNOWLEDGEMENT

VaultLayer team will inspect third party dependencies regularly, and push upgrades whenever required.

Identifier	Definition	Severity
COD-11	Insufficient validation of loan conditions	Medium 🟡

In `listLoan` while there are checks on loan amount, interest rate, and duration, additional parameters like validity of NFT address or token ID are not explicitly validated. Add further checks to validate NFT ownership more robustly, ensuring that NFTs are not already collateralized in another loan.

Interest rate calculations in `getTotalRepayment` are straightforward but do not account for complex scenarios such as early repayments or late payments beyond the simple fixed APR.

When a loan is accepted by borrower via `acceptLoan`, smart contract does not validate if the full loan amount is still escrowed properly. Use explicit check that the full expected amount is available in escrow.

RECOMMENDATION

Add extensive checks to validate all loan conditions, and lending logic.

RESOLUTION

VaultLayer team validates acceptance properly in `acceptLoan`. V2 introduces `isCollateralized[nftAddress][tokenId]` to prevent double-collateralizing the same NFT. It uses `IERC721.ownerOf` for checks.

Identifier	Definition	Severity
COD-12	Lack of event-driven architecture	Minor

Some state changes are not accompanied by event emissions, making tracking changes off-chain more difficult.

`setMaxActiveLoans`
`setProtocolFeeRate`

RECOMMENDATION

Use events to track state changes. Events improve transparency and provide a more granular view of contract activity.

RESOLUTION

Smart contract uses events to track state changes.

Identifier	Definition	
COD-13	Note regarding escrow management risks	

COD-03 continuation... Funds are held in escrow within the contract, and management of these funds relies heavily on successful function execution.

When escrow handling fails (for example transfer failure due to denial of service, gas costs, etc), these funds can become stuck.

```
placeBid
delistLoan
```


RECOMMENDATION

Implement robust escrow management practices, such as periodic audits of balances and mechanisms to handle failed transfers appropriately.

Identifier	Definition	Severity
COD-14	Note regarding flash loan attacks	Minor

Loan bidding and acceptance: The logic of this smart contract involves placing bids on loans, potentially accepting them, and dealing with loan repayments. The critical logic of flash loan protection involves ensuring that state changes related to loan acceptance and repayment cannot be exploited to create conditions favorable for a flash loan attack.

Economic parameters: Smart contract logic around loan bidding, acceptance, and repayment includes economic parameters like interest rates. A flash loan attack can try to exploit these parameters if they can be influenced within one block. However, since smart contract primarily deals with fixed and pre-defined terms once a loan bid is accepted, the risk should be minimal.

Loan repayment: There's potential for a flash loan attack if a malicious actor can take out a loan and manipulate the repayment terms or timing using a flash loan. However, since smart contract enforces strict conditions on loan duration and repayment terms, and there isn't a direct way to profit from a quick open and close of a loan due to the non-fungible nature of the collateral (NFTs), typical risk should be minimal.

RECOMMENDATION

To mitigate flash loan attacks, use checks that assess loan bidding and acceptance parameters. Ensure that state changes related to loan acceptance and repayment cannot be exploited to create conditions favorable for a flash loan attack.

Identifier	Definition	Severity
COM-01	Floating pragma	Minor

Compiler is set to `^0.8.23`

RECOMMENDATION

Pragma should be fixed to stable compiler version. Fixing pragma ensures compatibility and prevents the contract from being compiled with incompatible compiler versions.

RESOLUTION

Smart contract is deployed with stable compiler.

Identifier	Definition	Severity
COM-03	Protocol fee calculation transparency	Minor

Protocol fee is calculated within several functions, but logic is somewhat duplicated and can lead to inconsistencies if updated incorrectly.

`repayLoan`
`claimDefaultedLoan`

RECOMMENDATION

Centralize protocol fee calculation logic into a single internal function to reduce errors.

RESOLUTION

Smart contract centralizes fee logic in `calculateProtocolFee(uint256 amount)`.

Identifier	Definition	Severity
COM-04	Gas optimization	Minor 🟢

Smart contract frequently loops over `activeLoanIds`, which can lead to unbounded gas costs, particularly in the functions like `_removeActiveLoan`.

RECOMMENDATION

Restrict the maximum array size or implement a more gas-efficient method for managing active loans.

RESOLUTION

Smart contract uses `maxActiveLoans` limit for the array to manage maximum active loans.

DISCLAIMERS

InterFi Network provides the easy-to-understand audit of solidity source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. 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. This audit report should not be used in any way

to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, INTERFI NETWORK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, INTERFI NETWORK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. InterFi Network does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than InterFi Network. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites' and social accounts' owners. You agree that InterFi Network is not responsible for the content or operation of such websites and social accounts and that InterFi Network shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT INTERFI NETWORK

InterFi Network provides intelligent blockchain solutions. We provide solidity development, testing, and auditing services. We have developed 150+ solidity codes, audited 1000+ smart contracts, and analyzed 500,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Velas, Oasis, etc.

InterFi Network is built by engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 4 core members, and 6+ casual contributors.

Website: https://interfi.network

Email: hello@interfi.network

GitHub: https://github.com/interfinetwork

Telegram (Engineering): https://t.me/interfiaudits

Telegram (Onboarding): https://t.me/interfisupport

interfinetwork

hello@interfi.network

https://interfi.network

SMART CONTRACT AUDITS | SOLIDITY DEVELOPMENT AND TESTING

RELENTLESSLY SECURING PUBLIC AND PRIVATE BLOCKCHAINS