InterFi NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

REMATIC FINANCE

[FSP FACTORY & GOVERNANCE CONTRACTS]

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	Rematic Finance LLC
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
FSP Factory Contract	0x687A406be05b7E0418fe2DF60fD0c730bBbB7e8E
FSP Factory Proxy	0x0CfF1ad6373B17dabc2fA5B3CFa1479F14500B01
Governance Contract	0x4d7810d260C4790535617665c2A08351b229FC4e
Governance Proxy	0x6e19135E0f9C511e7689Db15976096216B8961D7
Blockchain	Binance Smart Chain
Centralization	Active ownership
Website	https://rematic.finance/
Telegram	https://t.me/rematicfinance/
Twitter	https://twitter.com/RematicFinance/
Discord	https://discord.gg/nBj4TEHbRa/
Report Date	October 17, 2022

Verify the authenticity of this report on our website: https://www.interfi.network/audits

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	1	5	1
Acknowledged	1	1	0	3	0
Resolved	0	0	0	0	0
Noteworthy Privileges	Refer to PAGE 17 for centralization related comments				

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

SCOPE OF WORK

InterFi was consulted by Rematic Finance 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:

- FSPFactory.sol
- Governance.sol

ℹ️ If source codes are not deployed on the main net, they can be modified or altered before main-net deployment. Please note, upgradeable contracts can be modified or altered regardless of their deployment status.

FSP Factory Public Contract Link
https://bscscan.com/address/0x687A406be05b7E0418fe2DF60fD0c730bBbB7e8E#code
FSP Factory Proxy Contract Link
https://bscscan.com/address/0x0CfF1ad6373B17dabc2fA5B3CFa1479F14500B01#code

Governance Public Contract Link
https://bscscan.com/address/0x4d7810d260C4790535617665c2A08351b229FC4e#code
Governance Proxy Contract Link
https://bscscan.com/address/0x6e19135E0f9C511e7689Db15976096216B8961D7#code

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

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. 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 here for the reader to review:

Risk Type	Definition
Critical 🔴	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major 🟠	These risks 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.
Medium 🟡	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

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.

ℹ️ 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

FSP FACTORY

IERC20Metadata	Interface	IERC20		
└	name	External !		NO !
└	symbol	External !		NO !
└	decimals	External !		NO !
IRematic	Interface	IERC20		
└	adminContract	External !		NO !
└	burnWallet	External !		NO !
└	stakingWallet	External !		NO !
└	txFeeRate	External !		NO !
└	burnFeeRate	External !		NO !
└	stakingFeeRate	External !		NO !
└	setBurnWallet	External !	🛑	NO !
└	setStakingWallet	External !	🛑	NO !
└	setTxFeeRate	External !	🛑	NO !
└	setBurnFeeRate	External !	🛑	NO !
└	setStakingFeeRate	External !	🛑	NO !
└	setIsOnBurnFee	External !	🛑	NO !
└	setIsOnStakingFee	External !	🛑	NO !

└	transferTokenFromPool	External ❗	🛑	NO❗
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_setOwner	Private 🔐	🛑	
ReentrancyGuard	Implementation			
└	<Constructor>	Public ❗	🛑	NO❗
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	
└	functionDelegateCall	Internal 🔒	🛑	
└	_verifyCallResult	Private 🔐		
SafeERC20	Library			
└	safeTransfer	Internal 🔒	🛑	
└	safeTransferFrom	Internal 🔒	🛑	

| └ | safeApprove | Internal 🔒 | 🛑 | | |

| └ | safeIncreaseAllowance | Internal 🔒 | 🛑 | | |

| └ | safeDecreaseAllowance | Internal 🔒 | 🛑 | | |

| └ | _callOptionalReturn | Private 🔐 | 🛑 | | |

||||||

| **FSPPool** | Implementation | Ownable, ReentrancyGuard |||

| └ | <Constructor> | Public ❗ | 🛑 | NO❗ |

| └ | initialize | External ❗ | 🛑 | NO❗ |

| └ | rewardTokenTransfer | External ❗ | 🛑 | onlyOwner |

| └ | deposit | External ❗ | 💵 | nonReentrant isPoolActive |

| └ | claimReflections | External ❗ | 💵 | nonReentrant |

| └ | claimReward | External ❗ | 💵 | nonReentrant |

| └ | withdraw | External ❗ | 💵 | nonReentrant |

| └ | isStakedUser | Internal 🔒 | | |

| └ | stopReward | External ❗ | 🛑 | NO❗ |

| └ | updatePoolLimitPerUser | External ❗ | 🛑 | onlyOwner |

| └ | getDepositFee | Public ❗ | | NO❗ |

| └ | getEarlyWithdrawFee | Public ❗ | | NO❗ |

| └ | getCanceledWithdrawFee | Public ❗ | | NO❗ |

| └ | getRewardClaimFee | Public ❗ | | NO❗ |

| └ | getReflectionFee | Public ❗ | | NO❗ |

| └ | getMaxStakeTokenAmount | Public ❗ | | NO❗ |

| └ | getTotalStaked | Public ❗ | | NO❗ |

| └ | getTotalReward | Public ❗ | | NO❗ |

| └ | pendingReward | Public ❗ | | NO❗ |

| └ | pendingReflectionReward | Public ❗ | | NO❗ |

| └ | _getRewardAmount | Internal 🔒 | | |

| └ | _getReflectionAmount | Internal 🔒 | | |

| └ | _calculateReflections | Public ❗ | 🛑 | NO❗ |

| └ | emergencyWithdrawByPoolOwner | Public ❗ | 🛑 | onlyOwner |

| └ | emergencyWithdrawByPlatformOwner | Public ❗ | 🛑 | NO❗ |

└	hasUserLimit	Public ❗		NO❗
└	getPoolLifeTime	External ❗		NO❗
└	getDepositAmount	Public ❗		NO❗
└	getPoolStatus	Public ❗		NO❗
FSPFactory	Implementation	Initializable, OwnableUpgradeable		
└	<Constructor>	Public ❗	🛑	NO❗
└	initialize	Public ❗	🛑	initializer
└	deployPool	External ❗	💵	NO❗
└	getDepositFee	Public ❗		NO❗
└	getRewardClaimFee	Public ❗		NO❗
└	getEarlyWithdrawFee	Public ❗		NO❗
└	getCanceledWithdrawFee	Public ❗		NO❗
└	getReflectionFee	Public ❗		NO❗
└	getCreationFee	Public ❗		NO❗
└	updatePoolCreateFee	External ❗	🛑	onlyOwner
└	updateReflectionFees	External ❗	🛑	onlyOwner
└	updateNonReflectionFees	External ❗	🛑	onlyOwner
└	setPlatformOwner	External ❗	🛑	onlyOwner
└	isPlatformOwner	Public ❗		NO❗
└	updateRFTXAddress	External ❗	🛑	onlyOwner
└	updateTotalDepositAmount	Public ❗	🛑	NO❗
└	addAdmin	Public ❗	🛑	onlyOwner
└	removeAdmin	Public ❗	🛑	onlyOwner
└	_safeTransferETH	Internal 🔒	🛑	
└	withdraw	Public ❗	🛑	onlyOwner
└	withdrawToken	External ❗	🛑	NO❗
└	getAllPools	Public ❗		NO❗
└	<Receive Ether>	External ❗	💵	NO❗

GOVERNANCE

ILoan	Interface			
└	owner	External !		NO!
└	accept	External !	🛑	NO!
└	token	External !		NO!
└	tokenAmount	External !		NO!
└	duration	External !		NO!
└	paymentPeriod	External !		NO!
└	aPRInerestRate	External !		NO!
└	status	External !		NO!
└	getAcceptLoanMapLengthOf	External !		NO!
└	initialize	External !	🛑	NO!
ILoanDeployer	Interface			
└	createLoan	External !	🛑	NO!
IterableMapping	Library			
└	get	Public !		NO!
└	getIndexOfKey	Public !		NO!
└	getKeyAtIndex	Public !		NO!
└	size	Public !		NO!
└	set	Public !	🛑	NO!
└	remove	Public !	🛑	NO!
Governance	Implementation	UUPSUpgradeable, OwnableUpgradeable		
└	initialize	Public !	🛑	initializer
└	_authorizeUpgrade	Internal 🔒	🛑	onlyOwner
└	whitelistLender	Public !	🛑	onlyOwner
└	whitelistBorrower	Public !	🛑	onlyOwner
└	blackListUser	Public !	🛑	onlyOwner
└	isWhitelistedLender	Public !		NO!

| └ | isWhitelistedBorrower | Public ❗ | | NO❗ |

| └ | isBlacklistedUser | Public ❗ | | NO❗ |

| └ | createLoan | Public ❗ | 🛑 | onlyWhiteListedLender notBlackListed |

| └ | getOwnedLoanListOfUser | Public ❗ | | NO❗ |

| └ | getBorrowedLoanListOfUser | Public ❗ | | NO❗ |

| └ | updateCreditAmountOfUser | Public ❗ | 🛑 | onlyOwner |

| └ | increaseCreditAmountOfUser | Public ❗ | 🛑 | onlyLoan |

| └ | accept | Public ❗ | 🛑 | onlyWhiteListedBorrower notBlackListed |

| └ | upateTeamWallet | Public ❗ | 🛑 | onlyOwner |

| └ | getLimitAmountOf | Public ❗ | | NO❗ |

| └ | getLoanList | Public ❗ | | NO❗ |

| └ | isWhiteListedLender | Public ❗ | | NO❗ |

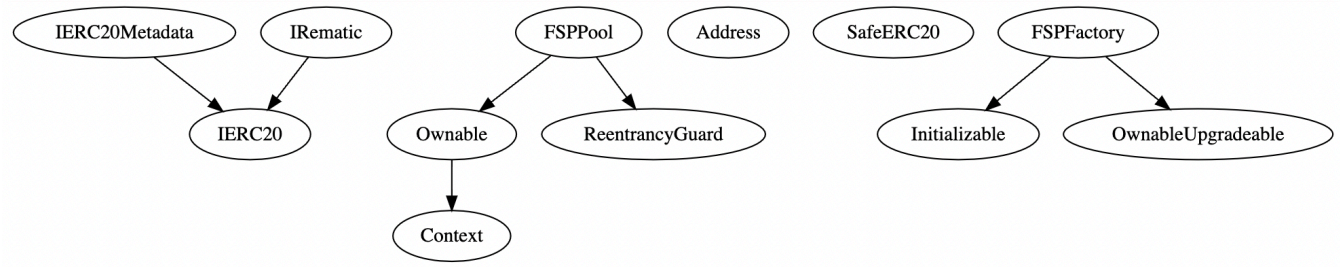
| └ | isWhiteListedBorrower | Public ❗ | | NO❗ |

| └ | isBlackListed | Public ❗ | | NO❗ |

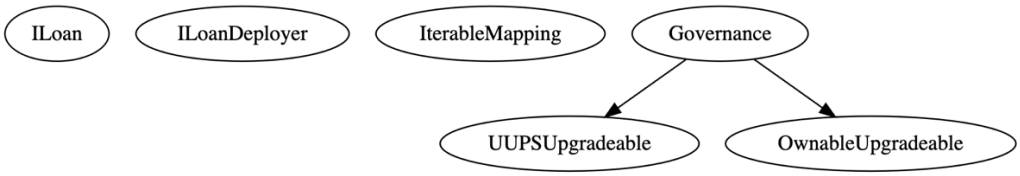
| └ | setLoanDeployer | Public ❗ | 🛑 | onlyOwner |

INHERITANCE GRAPH

FSP FACTORY



GOVERNANCE



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralization privileges of FSP Factory and Governance contracts	Major 🟠
CEN-07	Access controls in FSP Factory and Governance contracts	Minor 🟢

FSP FACTORY

`onlyOwner` centralized privileges are listed below:

```
rewardTokenTransfer
updatePoolLimitPerUser
emergencyWithdrawByPoolOwner
updatePoolCreateFee
updateReflectionFees
updateNonReflectionFees
setPlatformOwner
updateRFTXAddress
addAdmin
removeAdmin
withdraw
renounceOwnership
transferOwnership
```

Other noteworthy access control privileges are listed below:

```
initialize msg.sender == SMART_CHEF_FACTORY
stopReward msg.sender == owner() ||
FSPFactory(payable(address(SMART_CHEF_FACTORY))).isPlatformOwner(msg.sender
```

GOVERNANCE

`onlyOwner` centralized privileges are listed below:

```
_authorizeUpgrade
whitelistLender
whitelistBorrower
```

```
blackListUser
updateCreditAmountOfUser
upateTeamWallet
setLoanDeployer
```

Other noteworthy access control privileges are listed below:

```
increaseCreditAmountOfUser onlyLoan
createLoan onlyWhiteListedLender notBlackListed
accept onlyWhiteListedBorrower notBlackListed
```


CEN-01 RECOMMENDATION

Deployer, contract owner, and privileged roles private keys must be secured carefully. Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding.

CEN-07 RECOMMENDATION

Access control privileges must be authenticated adequately. Privileged roles such as admin, smart chef factory private keys must be secured carefully.

Identifier	Definition	Severity
CEN-03	Privileged role performing blacklist in Governance contract	Medium 🟡

In Governance contract, privileged role can call `blacklist()`

```
function blackListUser(address _account) public onlyOwner {
    blackList.set(_account, 1);
}
```


RECOMMENDATION

Remove blacklist – as it can intentionally stop an address from accessing smart contract function modules.

Identifier	Definition	Severity
CEN-09	Use of proxy and upgradeable contracts	Critical 🔴

Privileged role can initiate contract implementation. Contract upgradeability allows privileged roles to change current contract implementation.

```
contract FSPFactory is Initializable, OwnableUpgradeable {
contract Governance is UUPSUpgradeable, OwnableUpgradeable {
```


RECOMMENDATION

Test and validate current contract thoroughly before deployment. Future contract upgradeability negatively elevates centralization risk.

ALLEVIATION

Rematic Finance team will utilize contract upgradeability for contract testing and/or vulnerability surveillance purposes. Upgradable contracts are easier to upgrade and push on chain with minimal down-time.

Identifier	Definition	Severity
LOG-01	Lack of arbitrary limits	Minor 🟢

In FSP Factory contract, below mentioned functions are set without any arbitrary limits.

`updatePoolCreateFee`
`updateReflectionFees`
`updateNonReflectionFees`

RECOMMENDATION

These functions should be provided arbitrary limits, e.g., put a `require` check that allows maximum tax change up to 25%.

Identifier	Definition	Severity
LOG-02	Potential front-running	Minor 🟢

Mentioned functions in FSP Factory and Governance contracts can be manipulated by front-running a transaction to deposit assets and make profits by back-running a transaction to withdraw assets.

```
deposit
claimReflections
claimReward
withdraw
createLoan
accept
```


RECOMMENDATION

Sufficient strategies should be used to deter front-running transactions. Learn more: https://coinmarketcap.com/alexandria/article/3-minute-tips-avoiding-front-runners-on-decentralized-exchanges

Identifier	Definition	Severity
LOG-03	Re-entrancy	Unknown

Mentioned functions in FSP Factory and Governance contracts are used without re-entrancy guard:

`deployPool`

RECOMMENDATION

Re-entrancy guard is used to prevent re-entrant calls. Learn about re-entrancy guard: https://consensys.github.io/smart-contract-best-practices/attacks/reentrancy/

Identifier	Definition	Severity
LOG-04	Improper function logic	Minor 🟢

In FSP Factory contract, function `withdrawToken` is not callable by the contract owner to remove stuck tokens.

In Governance contract, function `_authorizeUpgrade` is never used.

RECOMMENDATION

Remove unused functions and/or fix logic flow.

Identifier	Definition	Severity
COD-02	Timestamp manipulation via `block.timestamp`	Minor 🟢

Be aware that the timestamp of the block can be manipulated by a miner. When the contract uses the timestamp to seed a random number, the miner can actually post a timestamp within 15 seconds of the block being validated, effectively allowing the miner to precompute an option more favorable to their chances, this is a critical exploit for contracts calculating random numbers, e.g., lottery.

RECOMMENDATION

To maintain block integrity, follow 15 seconds rule, and scale time dependent events accordingly.

Identifier	Definition	Severity
COD-03	Minimize function visibility Improper use of external and/or public attributes	Minor 🟢

For example,

In FSP Factory contract, function `_calculateReflections` is can be set to internal.

In Governance contract, function `_authorizeUpgrade` is never used, and can be set to internal.

RECOMMENDATION

To maintain logic integrity and gas optimization, use proper visibility attributes. To deter potential exploit, minimize function visibility.

Identifier	Definition	Severity
COD-05	Missing zero address validation	

Mentioned function inputs in FSP Factory and Governance contracts can be verified for zero address validation:

`setPlatformOwner`
`updateRFTXAddress`
`addAdmin`
`removeAdmin`
`withdraw`
`whitelistLender`
`whitelistBorrower`
`blackListUser`
`upateTeamWallet`
`setLoanDeployer`

RECOMMENDATION

Validate if the modified address is `dead(0)` or not.

Identifier	Definition	Severity
COD-10	Third Party Dependencies	Minor 🟢

Smart contract is interacting with third party protocols e.g., Pancakeswap, Uniswap. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised, and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

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

Identifier	Definition	Severity
VOL-01	Irrelevant code	

Mentioned functions in FSP Factory and Governance contracts serve similar logic success:

```
isWhitelistedLender
isWhitelistedBorrower
isBlacklistedUser
isWhiteListedLender
isWhiteListedBorrower
isBlackListed
```


RECOMMENDATION

Remove redundant and dead code.

Identifier	Definition	Severity
VOL-02	Typographical Error	

Mentioned functions in Governance contract have typographical errors:

`upateTeamWallet`
`aPRInerestRate`

RECOMMENDATION

Fix typographical errors.

Identifier	Definition	Severity
COM-04	Potential resource exhaustion errors	Minor 🟢

Mentioned functions in FSP Factory contract may throw out-of-gas errors when called:

`_calculateReflections`
`emergencyWithdrawByPoolOwner`

RECOMMENDATION

Set an upper limit for multi-address calls, e.g., 100 wallets can be airdropped in a single transaction.

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