InterFi
NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

PRESALE CONTRACTS

(FLASH-LAUNCH)

INTERFI SMART CONTRACT AUDIT

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	Flash Technologies
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Presale	
Factory	
Blockchain	
Centralization	Active ownership
Commit	2f153217e5c56d6c83ac619bc7865baeb90784bd
Website	https://flash-launch.com/
Report Date	March 23, 2024

ℹ️ 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	2	7	1
Acknowledged	1	0	1	0	0
Resolved	0	0	0	1	0
Noteworthy Privileges	Check PAGE 14 for centralized and controlled privileges of presale and factory contracts				

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 Flash 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:

- BestPresale.sol
- BestPresaleFactory.sol

ℹ 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	
Contract Name	BestPresale
Compiler Version	0.7.6
License	Unlicensed

Public Contract Link	
Contract Name	BestPresaleFactory
Compiler Version	0.7.6
License	Unlicensed

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.

ℹ 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.

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

BestPresale

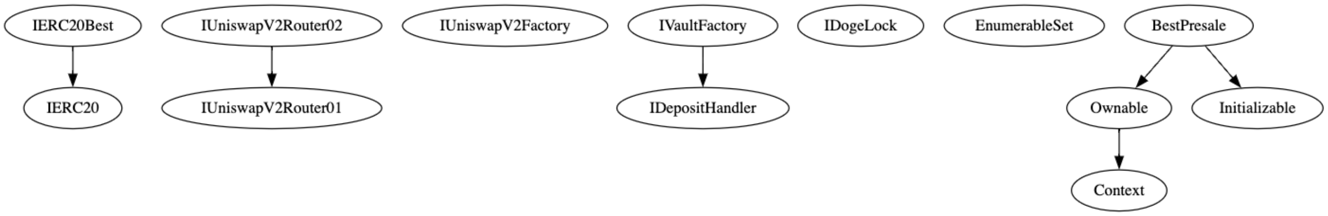
```
| **IDepositHandler** | Interface |  |||
||||||
| **IVaultFactory** | Interface | IDepositHandler |||
| └ | createVault | External ❗ |  💵 |NO❗ |
||||||
| **IDogeLock** | Interface |  |||
| └ | lock | External ❗ | 🛑  |NO❗ |
||||||
| **BestPresale** | Implementation | Initializable, Ownable |||
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | initialize | External ❗ | 🛑  | initializer |
| └ | <Fallback> | External ❗ |  💵 |NO❗ |
| └ | userContribute | External ❗ |  💵 | presaleLive |
| └ | emergencyWithdraw | External ❗ |  💵 | presaleLive |
| └ | totalParticipants | External ❗ |   |NO❗ |
| └ | claimTokens | External ❗ | 🛑  |NO❗ |
| └ | claimableTokens | Public ❗ |   |NO❗ |
| └ | withdrawAfterCancelled | External ❗ |  💵 |NO❗ |
| └ | affiliateClaim | External ❗ |  💵 |NO❗ |
| └ | getAffiliateClaimableAmount | Public ❗ |   |NO❗ |
| └ | cancelPresale | External ❗ | 🛑  | onlyOwner |
| └ | updateAffiliateProgram | External ❗ | 🛑  | onlyOwner |
| └ | updatePresalePeriod | External ❗ | 🛑  | onlyOwner |
| └ | updateWhitelistSetting | External ❗ | 🛑  | onlyOwner |
| └ | isUserInWhitelist | External ❗ |   |NO❗ |
| └ | addWhitelist | External ❗ | 🛑  | onlyOwner |
| └ | removeWhitelist | External ❗ | 🛑  | onlyOwner |
| └ | sqrt | Internal 🔒 |   | |
| └ | finalize | External ❗ |  💵 | onlyOwner |
| └ | getPairInfo | Public ❗ |   |NO❗ |
| └ | calculateQuotes | Public ❗ |   |NO❗ |
| └ | getCurrencyDecimals | Internal 🔒 |   | |
| └ | sendServiceFees | Internal 🔒 | 🛑  | |
| └ | sendLiquidityToDex | Internal 🔒 | 🛑  | |
| └ | calcSqrtPriceX96 | Internal 🔒 |   | |
| └ | sendLiquidityToV3 | Internal 🔒 | 🛑  | |
| └ | getUserContributeAmounts | External ❗ |   |NO❗ |
| └ | getReferrerCount | External ❗ |   |NO❗ |
| └ | getRealtimeRewardPercentage | External ❗ |   |NO❗ |
| └ | getAllRewardList | External ❗ |   |NO❗ |
```

BestPresaleFactory

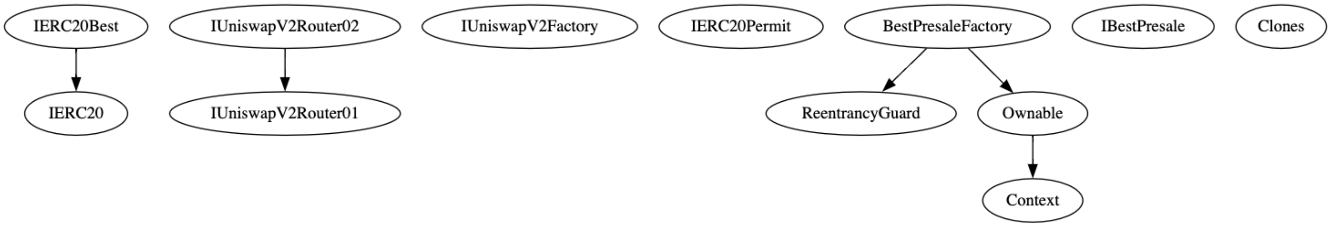
```
| **BestPresaleFactory** | Implementation | Ownable, ReentrancyGuard |||
| └ | <Constructor> | Public ❗ | 🛑  |NO❗ |
| └ | setImplementation | External ❗ | 🛑  | onlyOwner |
| └ | setFeeReceiver | External ❗ | 🛑  | onlyOwner |
| └ | setBurnInfo | External ❗ | 🛑  | onlyOwner |
| └ | setCreationFee | External ❗ | 🛑  | onlyOwner |
| └ | setMinLockerPeriod | External ❗ | 🛑  | onlyOwner |
| └ | setLockerAddresses | External ❗ | 🛑  | onlyOwner |
| └ | setV3Addresses | External ❗ | 🛑  | onlyOwner |
| └ | setDogeLockerAddress | External ❗ | 🛑  | onlyOwner |
| └ | setFeeOptionValues | External ❗ | 🛑  | onlyOwner |
| └ | refundExcessiveFee | Internal 🔒 | 🛑  | |
| └ | create | External ❗ |  💵 | enoughFee nonReentrant |
| └ | getCurrencyDecimals | Internal 🔒 |   | |
```

INHERITANCE GRAPH

BestPresale



BestPresaleFactory



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized privileges	Critical 🔴

BestPresale

Important Presale owners’ centralized privileges are listed below:

```
cancelPresale
updateAffiliateProgram
updatePresalePeriod
updateWhitelistSetting
addWhitelist
removeWhitelist
finalize
```

BestPresaleFactory

Important factory owners’ centralized privileges are listed below:

```
setImplementation
setFeeReceiver
setBurnInfo
setCreationFee
setMinLockerPeriod
setLockerAddresses
setV3Addresses
setDogeLockerAddress
setFeeOptionValues
```

RECOMMENDATION

Deployers’, presale and factory owners’, factory administrators’, and all other privileged roles’ private-keys/access-keys/admin-keys should be secured carefully. These entities can have a single point of failure that compromises the security of the project. Manage centralized and privileged roles carefully, review PAGE 09 for more information.

Implement multi-signature wallets: Require multiple signatures from different parties to execute certain sensitive functions within contracts. This spreads control and reduces the risk of a single party having complete authority.

Use a decentralized governance model: Implement a governance model that enables token holders or other stakeholders to participate in decision-making processes. This can include voting on contract upgrades, parameter changes, or any other critical decisions that impact the contract's functioning.

ACKNOWLEDGEMENT

Project team acknowledged to secure factory contract deployer, administrator, and owner private keys carefully. Project team acknowledged to use multi-signature validation approach to manage centralization roles whenever possible.

Project team argued that, presale feature can be used by any user, hence it is user’s responsibility to manage ownership of their own Presale contract.

Identifier	Definition	Severity
CEN-09	Use of proxy and upgradeable contracts	Medium 🟡
FLS-01	Use of `clones.clone` for presale creation	

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

When smart contracts are intended to be upgradeable, make sure that upgrade paths are secure, storage layouts are maintained, and proxy patterns are properly implemented to prevent clashes and vulnerabilities.

`BestPresale`

`Initializable`

`BestPresaleFactory`

`setImplementation`
`create`

Factory contract uses EIP-1167 for creating new presale contracts. New implemented contract address must be secure and immutable post-deployment, as any vulnerabilities in the implementation contract would affect all subsequent clones.

RECOMMENDATION

Test and validate contracts thoroughly before deployment. Future contract upgradeability negatively elevates centralization risk. New implemented presale contracts must be secure and immutable post-deployment.

ACKNOWLEDGEMENT

Project team argued that contracts use proxy mechanism to have future upgradeability, and flexibility.

Identifier	Definition	Severity
LOG-01	Lack of input validation	Minor 🟢

Below mentioned functions should be provided enough input validation to allow value change within pre-set parameters:

BestPresale

```
updatePresalePeriod
```

BestPresaleFactory

```
setCreationFee
setMinLockerPeriod
setFeeOptionValues
```

RECOMMENDATION

These functions should be provided appropriate input boundaries.

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

Potential front-running also classified as – sandwich attack 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. Below mentioned functions are called without setting restrictions on slippage or minimum output:

BestPresale

```
sendLiquidityToDex
initialize
userContribute
```

BestPresaleFactory

```
create
```

RECOMMENDATION

These functions should be provided reasonable minimum output amounts, instead of zero. Use commit-reveal schemes to deter front-runners whenever possible.

Identifier	Definition	Severity
LOG-03	Re-entrancy	Medium 🟡

Mentioned functions interact with external contracts but are not protected against re-entrancy attacks:

`BestPresale`

```
userContribute
emergencyWithdraw
withdrawAfterCancelled
affiliateClaim
```

RECOMMENDATION

These functions mostly follow Checks-Effects-Interactions pattern. Since re-entrancy can occur in unusual ways, it is recommended to use `nonReentrant` modifier from OpenZeppelin `ReentrancyGuard` to protect these functions.

Identifier	Definition	Severity
COD-05	Missing zero address validation	Minor ●

Below mentioned functions are missing zero address input validation:

`BestPresale`

```
initialize
addWhitelist
```

`BestPresaleFactory`

```
setImplementation
setFeeReceiver
setLockerAddresses
setV3Addresses
setDogeLockerAddress
```

RECOMMENDATION

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

Identifier	Definition	Severity
COD-06	External addresses	Minor 🟢

BestPresaleFactory

Uniswap V3's Nonfungible Position Manager

```
        v3PositionManagerAddress = 0xC36442b4a4522E871399CD717aBDD847Ab11FE88;
```

WETH contract

```
        WETH9Address = 0xB4FBF271143F4FBf7B91A5ded31805e42b2208d6;
```

RECOMMENDATION

Only interact with secure and trusted contracts.

Identifier	Definition	Severity
COD-10	Direct and indirect dependencies in both contracts	Unknown ●
COD-11	Reliance on liquidity locking logic with `flokiLocker` and `dogeLocker`	
COD-12	Reliance on DEX for execution prices	
COD-13	Reliance on `babyDogeSwapRounter`	

Smart contracts are interacting with third party protocols e.g., DEX Router Contracts, External Contracts, Web 3 Applications, Uniswap tools, Open Zeppelin tools. The scope of the audit treats these entities as black boxes and assumes their functional correctness. However, in the real world, all of them can be compromised, and exploited. Moreover, upgrades in these entities can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

`BestPresale`

Logic for locking liquidity involves external calls to contracts, `flokiLocker` and `dogeLocker`. It's impossible to ensure they securely handle locked liquidity. Malicious locker contracts could compromise the funds.

Smart contract interacts with Uniswap for adding liquidity, but it does not explicitly manage slippage or ensure favorable execution prices. This could lead to poor execution rates, especially during volatile market conditions.

`BestPresaleFactory`

Make sure `babyDogeSwapRounter` is a trusted contract address. Add sanity checks for the addresses involved in external calls to mitigate potential risks from malicious contract logic.

RECOMMENDATION

Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary. Only interact with trusted ERC20 addresses, contracts, and tokens.

Identifier	Definition
COD-11	Note about affiliate system

`BestPresale`

Affiliate system calculates rewards based on currency amounts without validating actual purchase or legitimacy of the referral. Malicious users could potentially game this system by using self-referrals or circular referrals to inflate rewards unfairly.

Identifier	Definition	Severity
COD-12	Lack of event-driven architecture in both contracts	Minor 🟢

Smart contracts use function calls to update state, which can make it difficult to track and analyze changes to the contract over time.

RECOMMENDATION

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

Identifier	Definition	Severity
COD-13	Note regarding `keccak256` secure hashing	Minor 🟢

Note that the `keccak256` function is not collision-resistant, and therefore there is a possibility of two different messages producing the same hash. Generating strong random input data, and properly securing and managing keys is recommended for fortification of `keccak256`.

Identifier	Definition	Severity
COM-01	Assembly code in both contracts	Minor 🟢

Inline assembly is a way to access the Ethereum Virtual Machine (EVM) at low level. This bypasses several important safety features and checks of Solidity. Moreover, automated and manual checks are not confidently possible for inline assembly codes.

RECOMMENDATION

Use high level Solidity constructs instead of `assembly`.

RESOLUTION

Project team has commented that – main assembly code is used for gas savings in byte manipulation and was written by *Consensys*, and is considered safe.

Identifier	Definition	Severity
COM-02	Outdated compiler version in both contracts	Medium 🟡

Compilers are set to outdated version.

```
pragma solidity =0.7.6;
```

RECOMMENDATION

Set Compiler to version 0.8.12 or above.

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

Below mentioned loops may throw out of gas errors upon executing:

BestPresale

```
totalParticipants
whitelist
referralAddresses
```

RECOMMENDATION

Set limits to stop arrays from getting too big.

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