SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

SATOSHI PEER CONTRACTS

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	Satoshi's Vision
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	Multiple Contracts
Blockchain	
Centralization	Active ownership
Commit	be577d09968e65d62c0023d339131cd23967b237
Website	https://www.satoshisvision.cash/
Telegram	https://t.me/SatoshisVisionERC20/
Twitter	https://twitter.com/SatoshiVision75/
Medium	https://medium.com/@satoshisvision/
Report Date	April 14, 2023

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	1	1	2
Acknowledged	0	0	0	1	0
Resolved	0	0	1	1	0

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 Satoshi's Vision 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:

- ETHPeer.sol
- MerkleTreeWithHistory.sol
- Peer.sol
- Verifier.sol

If source codes are not deployed on the main net, they can be modified or altered before main-net deployment.

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.

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.

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

```
| **ETHPeer** | Implementation | Peer |||
| └ | <Constructor> | Public ❗️ | 🛑  | Peer |
| └ | _processDeposit | Internal 🔒 | 🛑  | |
| └ | _processWithdraw | Internal 🔒 | 🛑  | |
||||||
| **IVerifier** | Interface |  |||
| └ | verifyProof | External ❗️ | 🛑  |NO❗️ |
||||||
| **Peer** | Implementation | MerkleTreeWithHistory, ReentrancyGuard |||
| └ | <Constructor> | Public ❗️ | 🛑  | MerkleTreeWithHistory |
| └ | deposit | External ❗️ |  💵 | nonReentrant |
| └ | _processDeposit | Internal 🔒 | 🛑  | |
| └ | withdraw | External ❗️ |  💵 | nonReentrant |
| └ | _processWithdraw | Internal 🔒 | 🛑  | |
| └ | isSpent | Public ❗️ |   |NO❗️ |
| └ | isSpentArray | External ❗️ |   |NO❗️ |
||||||
| **IHasher** | Interface |  |||
```

| └ | MiMCSponge | External ❗ | |NO❗ |

||||||

| **MerkleTreeWithHistory** | Implementation | |||

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

| └ | hashLeftRight | Public ❗ | |NO❗ |

| └ | _insert | Internal 🔒 | 🛑 | |

| └ | isKnownRoot | Public ❗ | |NO❗ |

| └ | getLastRoot | Public ❗ | |NO❗ |

| └ | zeros | Public ❗ | |NO❗ |

| **Pairing** | Library | |||

| └ | P1 | Internal 🔒 | | |

| └ | P2 | Internal 🔒 | | |

| └ | negate | Internal 🔒 | | |

| └ | addition | Internal 🔒 | | |

| └ | scalar_mul | Internal 🔒 | | |

| └ | pairing | Internal 🔒 | | |

| └ | pairingProd2 | Internal 🔒 | | |

| └ | pairingProd3 | Internal 🔒 | | |

| └ | pairingProd4 | Internal 🔒 | | |

||||||

| **Verifier** | Implementation | |||

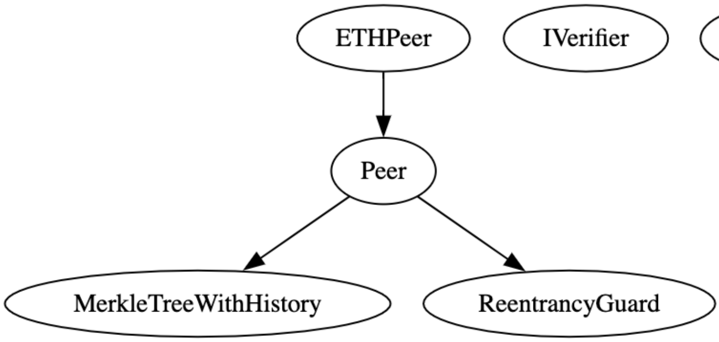
| └ | verifyingKey | Internal 🔒 | | |

| └ | verify | Internal 🔒 | | |

| └ | verifyProof | Public ❗ | |NO❗ |

| └ | verifyProof | External ❗ | |NO❗ |

INHERITANCE GRAPH



MANUAL REVIEW

Identifier	Definition
SAT-01	Code uniqueness

Smart contracts provide privacy solution on ETH called Peer, which uses *zkSNARK** to enable private transactions. Peer provides a way for users to deposit and withdraw funds in a private manner, meaning the amounts and the involved addresses are concealed.

Verifier contract uniqueness:

- Verifies a proof of knowledge of a solution to a specific set of commitments given as input. It uses the Pairing library, which provides functions to work with elliptic curve pairings.

Peer contract uniqueness:

- Takes a SNARK verifier contract, a hasher contract, a denomination, and a Merkle tree height as constructor parameters.
- Provides a `deposit()` function where users can deposit funds by submitting a commitment.
- Provides a `withdraw()` function that accepts a *zkSNARK* proof and its public inputs, verifies the proof, and handles the withdrawal process.

**zkSNARK* stands for "Zero-Knowledge Succinct Non-Interactive Argument of Knowledge," and refers to a proof construction where one can prove possession of certain information, e.g., a secret key, without revealing that information, and without any interaction between the prover and verifier.

Identifier	Definition	Severity
CEN-01	Centralization with *zk-SNARK*	Medium

zkSNARK construction requires a trusted setup to generate the initial parameters. If this setup is compromised, an attacker may be able to create fake proofs that are accepted as valid by the system.

RECOMMENDATION

To mitigate this, a transparent setup process can be used, or a multi-party computation (MPC) setup can be performed where no single party has access to the full setup.

Identifier	Definition	Severity
SAT-02	Side channel attacks	Unknown

Side-channel attacks exploit information leaked during the computation of a *zkSNARK* proof to reveal secrets or compromise the system. *Side channel attacks are difficult to detect and may require physical access to the device or system being attacked.*

Identifier	Definition
LOG-03	Re-entrancy

Below mentioned functions are used with `nonReentrant` modifier to protect against re-entrancy attacks:

```
deposit()
withdraw()
```

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.

RECOMMENDATION

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

Identifier	Definition	Severity
COD-06	Use of fixed history size for storing Merkle tree	Minor

`roots` mapping stores the roots of the Merkle tree for the most recent `ROOT_HISTORY_SIZE` deposits. However, if an attacker is able to mount a successful attack after `ROOT_HISTORY_SIZE` deposits have been made, they could potentially change the state of the contract without detection.

RECOMMENDATION

Use an unbounded list to store Merkle tree roots, and add a function to trim list periodically to a maximum length.

Identifier	Definition	Severity
COD-07	Conformance to solidity coding conventions	Minor

`getLastRoot()` returns the current root of the Merkle Tree, therefore it should probably be named `getCurrentRoot()`.

Group related functions together, especially the `internal` and `external` functions, can improve code readability. It's not a strict requirement, but it can be helpful for others who read the code.

RECOMMENDATION

Follow coding conventions for writing solidity code. Learn more:
https://docs.soliditylang.org/en/v0.8.16/style-guide.html

RESOLUTION

Satoshi's Vision team has changed `getLastRoot()` function name.

Identifier	Definition	Severity
COD-08	Lack of fallback function	

Fallback functions are usually executed in one of the following cases: If a function identifier doesn't match any of the available functions in a smart contract. If there was no data supplied along with the function call.

RECOMMENDATION

Use fallback function with empty data, and mark it external, and payable.

RESOLUTION

Satoshi’s Vision team has added callback function to ETHPeer code.

Identifier	Definition	Severity
COD-10	Third Party Dependencies	Unknown

Codes are interacting with third party protocols e.g., MiMC Hash contract through `IHasher` interface, *OpenZeppelin* tools e.g., Re-entrancy Guard. 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	Use `SafeMath` for arithmetic operations	

Although `SafeMath` is not strictly necessary in this code, it's a good practice to use `SafeMath` when dealing with `uint256` data types to prevent arithmetic overflows and underflows.

Identifier	Definition	Severity
COM-02	Outdated compiler version	Medium
COM-01	Floating compiler status	

ETHPeer, MerkleTreeWithHistory and Peer codes uses Solidity version `^0.7.0`

Verifier code uses Solidity version `^0.5.0`

RECOMMENDATION

It is recommended to use the latest stable version of Solidity, which provides better security, optimization, and updated solidity language features.

RESOLUTION

Satoshi's Vision team has changed compiler versions to `^0.8.0`.

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