InterFi NETWORK

SMART CONTRACT AUDIT

interfinetwork

hello@interfi.network

https://interfi.network

PREPARED FOR

VAULTER CONTRACT

[VAULTLAYER]

INTRODUCTION

Auditing Firm	InterFi Network
Client Firm	VaultLayer
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
Contract	0x2589f07C0239A0075c61f7316898596c23D4c44f
Blockchain	Core
Centralization	Active Ownership
Commit	f9a23366e8b5fb1f6578ad0e838fdc1908004a02
Website	
Report Date	March 31, 2025

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

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.

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	2	0
Acknowledged	0	1	1	2	1
Resolved	0	0	3	3	0
Noteworthy Privileges	`setGrade, setReserveRatio, setDefaultValidator, setUseGrades, setRewardRatios, setStakeHub, setBitcoinStake, setCoreAgent, setTargetRatio, setAutoStake, setUseRewardRatios, setRoundingReserve, setCoreLib, setMaxActiveTxsPerBtcPubKey, setMaxBtcRounds, setAllowedDelegators, setProtocolFeeRate, pause, unpause, transferBtcDelegation, stakeCore, stakeCoreTo, unstakeCore, unstakeCoreFrom, transferCoreDelegation, withdrawProtocolFees`				

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

ℹ️ Please note that the absence of public KYC verification of the project owners, team members, or deployers associated with VaultLayer. Typically, third-party KYC processes are instrumental in ensuring the transparency and accountability of a project's leadership, thereby enhancing user trust and regulatory compliance. Without external KYC verification by reputable providers, users may face increased risks related to rug pull.

TABLE OF CONTENTS

SCOPE OF WORK

InterFi was consulted by VaulterCore 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:

- BitcoinHelper.sol
- CoreHelper.sol
- DualDelegator.sol
- VaulterCore.sol

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/0x2589f07C0239A0075c61f7316898596c23D4c44f#code	
Contract Name	VaulterCore
Compiler Version	0.8.23
License	None

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.

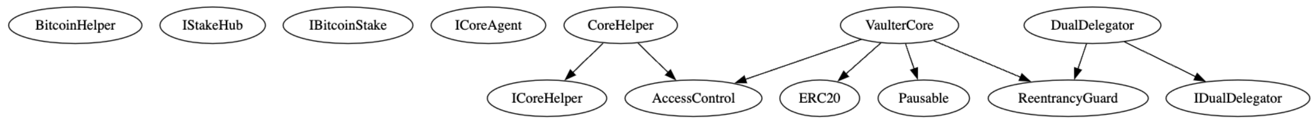
ℹ️ Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

AUTOMATED ANALYSIS

Symbol	Definition
🛑	Function modifies state
💵	Function is payable
🔒	Function is internal
🔐	Function is private
❗	Function is important

```
| **BitcoinHelper** | Library |  |||
| └ | calculateTxId | Internal 🔒 |   | |
| └ | convertEthToBtcPubKeyHash | Public ❗ |   |NO❗ |
| └ | recoverEthereumSigner | Public ❗ |   |NO❗ |
| └ | deriveAddress | Public ❗ |   |NO❗ |
| └ | verifyEthPubKeySignature | Public ❗ |   |NO❗ |
| └ | extractBitcoinAddress | Internal 🔒 |   | |
| └ | reverseUint256 | Internal 🔒 |   | |
| └ | uintToStr | Internal 🔒 |   | |
| └ | toAsciiString | Internal 🔒 |   | |
| └ | char | Internal 🔒 |   | |
| └ | compressBtcPubKey | Public ❗ |   |NO❗ |
| └ | toString | Internal 🔒 |   | |
| └ | bytesContains | Internal 🔒 |   | |
||||||
| **IStakeHub** | Interface |  |||
| └ | claimReward | External ❗ | 🛑  |NO❗ |
||||||
| **IBitcoinStake** | Interface |  |||
| └ | btcTxMap | External ❗️ |   |NO❗️ |
| └ | receiptMap | External ❗️ |   |NO❗️ |
| └ | transfer | External ❗️ | 🛑  |NO❗️ |
||||||
| **ICoreAgent** | Interface |  |||
| └ | roundTag | External ❗️ |   |NO❗️ |
| └ | getDelegator | External ❗️ |   |NO❗️ |
| └ | getCandidateListByDelegator | External ❗️ |   |NO❗️ |
| └ | delegateCoin | External ❗️ |  💵 |NO❗️ |
| └ | undelegateCoin | External ❗️ |  💵 |NO❗️ |
| └ | transferCoin | External ❗️ | 🛑  |NO❗️ |
||||||
| **ICoreHelper** | Interface |  |||
| └ | getBtcRewardRatio | External ❗️ |   |NO❗️ |
| └ | getValidator | External ❗️ |   |NO❗️ |
||||||
| **CoreHelper** | Implementation | ICoreHelper, AccessControl |||
| └ | <Constructor> | Public ❗️ | 🛑  |NO❗️ |
| └ | grantManagerRole | External ❗️ | 🛑  | onlyRole |
| └ | revokeManagerRole | External ❗️ | 🛑  | onlyRole |
| └ | setGrade | External ❗️ | 🛑  | onlyRole |
| └ | setReserveRatio | External ❗️ | 🛑  | onlyRole |
| └ | setDefaultValidator | External ❗️ | 🛑  | onlyRole |
| └ | setUseGrades | External ❗️ | 🛑  | onlyRole |
| └ | setRewardRatios | External ❗️ | 🛑  | onlyRole |
| └ | getBtcRewardRatio | External ❗️ |   |NO❗️ |
| └ | getValidator | External ❗️ |   |NO❗️ |
||||||
| **IDualDelegator** | Interface |  |||
| └ | stakeCore | External ❗️ |  💵 |NO❗️ |
| └ | unstakeCore | External ❗️ | 🛑  |NO❗️ |
| └ | unstakeCoreFromValidator | External ❗️ | 🛑  |NO❗️ |
| └ | claimRewards | External ❗️ |  💵 |NO❗️ |
| └ | transferCoreDelegation | External ❗️ | 🛑  |NO❗️ |
| └ | transferBtcDelegation | External ❗️ | 🛑  |NO❗️ |
| └ | getPendingStakingRewards | External ❗️ |   |NO❗️ |
||||||
| **DualDelegator** | Implementation | IDualDelegator, ReentrancyGuard |||
| └ | <Constructor> | Public ❗️ | 🛑  |NO❗️ |
| └ | stakeCore | External ❗️ |  💵 | nonReentrant onlyVaulterCore |
| └ | unstakeCore | External ❗️ | 🛑  | nonReentrant onlyVaulterCore |
| └ | unstakeCoreFromValidator | External ❗️ | 🛑  | nonReentrant onlyVaulterCore |
| └ | transferCoreDelegation | External ❗️ | 🛑  | nonReentrant onlyVaulterCore |
| └ | transferBtcDelegation | External ❗️ | 🛑  | nonReentrant onlyVaulterCore |
| └ | claimRewards | External ❗️ |  💵 | nonReentrant onlyVaulterCore |
| └ | getPendingStakingRewards | External ❗️ |   |NO❗️ |
| └ | <Receive Ether> | External ❗️ |  💵 |NO❗️ |
||||||
| **VaulterCore** | Implementation | ERC20, ReentrancyGuard, AccessControl, Pausable |||
| └ | <Constructor> | Public ❗️ | 🛑  | ERC20 |
| └ | grantManagerRole | External ❗️ | 🛑  | onlyRole |
| └ | revokeManagerRole | External ❗️ | 🛑  | onlyRole |
| └ | setStakeHub | External ❗️ | 🛑  | onlyRole |
| └ | setBitcoinStake | External ❗️ | 🛑  | onlyRole |
| └ | setCoreAgent | External ❗️ | 🛑  | onlyRole |
| └ | setTargetRatio | External ❗️ | 🛑  | onlyRole |
| └ | setAutoStake | External ❗ | 🛑  | onlyRole |
| └ | setUseRewardRatios | External ❗ | 🛑  | onlyRole |
| └ | setRoundingReserve | External ❗ | 🛑  | onlyRole |
| └ | setCoreLib | External ❗ | 🛑  | onlyRole |
| └ | setMaxActiveTxsPerBtcPubKey | External ❗ | 🛑  | onlyRole |
| └ | setMaxBtcRounds | External ❗ | 🛑  | onlyRole |
| └ | setAllowedDelegators | External ❗ | 🛑  | onlyRole |
| └ | setProtocolFeeRate | External ❗ | 🛑  | onlyRole |
| └ | pause | External ❗ | 🛑  | onlyRole |
| └ | unpause | External ❗ | 🛑  | onlyRole |
| └ | availableAssets | Public ❗ |   |NO❗ |
| └ | totalAssets | Public ❗ |   |NO❗ |
| └ | convertToShares | Public ❗ |   |NO❗ |
| └ | convertToAssets | Public ❗ |   |NO❗ |
| └ | maxDeposit | Public ❗ |   |NO❗ |
| └ | maxWithdraw | Public ❗ |   |NO❗ |
| └ | getPricePerShare | Public ❗ |   |NO❗ |
| └ | isAllowedDelegator | Public ❗ |   |NO❗ |
| └ | depositCore | External ❗ |  💵 | nonReentrant whenNotPaused |
| └ | withdrawCore | External ❗ | 🛑  | nonReentrant |
| └ | recordBtcStake | External ❗ | 🛑  | nonReentrant whenNotPaused |
| └ | getBtcStakerTxIds | Public ❗ |   |NO❗ |
| └ | getBtcStakedPerRound | Public ❗ |   |NO❗ |
| └ | getBtcStaked | Public ❗ |   |NO❗ |
| └ | transferBtcDelegation | External ❗ | 🛑  | nonReentrant onlyRole |
| └ | transferBtcDelegationWithPubKey | External ❗ | 🛑  | nonReentrant |
| └ | _claimBtcRewards | Internal 🔒 | 🛑  | |
| └ | getPendingBTCRewards | Public ❗ |   |NO❗ |
| └ | claimBtcRewardsWithSig | External ❗ | 🛑  | nonReentrant |
| └ | claimBtcRewards | External ❗ | 🛑  | nonReentrant |
| └ | _stakeCore | Internal 🔒 | 🛑  | |
| └ | stakeCore | External ❗ | 🛑  | nonReentrant onlyRole |
| └ | stakeCoreTo | External ❗ | 🛑  | nonReentrant onlyRole |
| └ | getDelegatorCoreStaked | External ❗ |   |NO❗ |
| └ | getDelegatorBtcStakedPerRound | Public ❗ |   |NO❗ |
| └ | unstakeCore | External ❗ | 🛑  | nonReentrant onlyRole |
| └ | unstakeCoreFrom | External ❗ | 🛑  | nonReentrant onlyRole |
| └ | _unstakeCore | Internal 🔒 | 🛑  | |
| └ | transferCoreDelegation | External ❗ | 🛑  | nonReentrant onlyRole |
| └ | getPendingStakingRewards | External ❗ |   |NO❗ |
| └ | claimStakingRewards | External ❗ | 🛑  | nonReentrant |
| └ | _rebalanceRewardRatios | Internal 🔒 | 🛑  | |
| └ | withdrawProtocolFees | External ❗ | 🛑  | nonReentrant onlyRole |
| └ | <Receive Ether> | External ❗ |  💵 |NO❗ |
| └ | <Fallback> | External ❗ |  💵 |NO❗ |
```

INHERITANCE GRAPH



MANUAL REVIEW

Identifier	Definition	Severity
CEN-01	Centralized Privileges	Major 🟠
CEN-01-01	Authorization and Access Control	

Important `onlyOwner` centralized privileges are listed below:

```
grantManagerRole
revokeManagerRole
setGrade
setReserveRatio
setDefaultValidator
setUseGrades
setRewardRatios
grantManagerRole
revokeManagerRole
setStakeHub
setBitcoinStake
setCoreAgent
setTargetRatio
setAutoStake
setUseRewardRatios
setRoundingReserve
setCoreLib
setMaxActiveTxsPerBtcPubKey
setMaxBtcRounds
setAllowedDelegators
setProtocolFeeRate
pause
unpause
transferBtcDelegation
stakeCore
stakeCoreTo
unstakeCore
unstakeCoreFrom
transferCoreDelegation
withdrawProtocolFees
```

RECOMMENDATION

Securing private keys or access credentials of deployers, contract owners, operators, 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.

Identifier	Definition	Severity
LOG-01	Assembly Usage Without Bounds Checking	Minor 🟢

Functions like `convertEthToBtcPubKeyHash`, `recoverEthereumSigner`, and `extractBitcoinAddress` use inline assembly to manipulate memory without explicit bounds checking beyond the initial `require` statements. For example, in `convertEthToBtcPubKeyHash, mload(add(ethPubKey, 33))` assumes the input is at least 65 bytes, but if the memory layout is corrupted or the input is malformed, it could read unintended data. When caller passes an invalid or shorter `ethPubKey` that somehow bypasses the length check (via a misconfigured proxy), it can lead to runtime errors.

RECOMMENDATION

Add additional safety checks within the assembly blocks (e.g., validate memory offsets explicitly). Alternatively, replace assembly with high-level Solidity where possible to reduce risk.

ACKNOWLEDGEMENT

VaultLayer team has increased `extractBitcoinAddress: require(script.length >= 32, "Invalid script length");`. VaultLayer team acknowledged that use of assembly code is noted, but parsing the Bitcoin script is harder in solidity.

Identifier	Definition	Severity
LOG-02	Insufficient validation	Medium 🟡

Below mentioned functions does not validate inputs or ranges:

```
setGrade
recordBtcStake
btcRewardRatio
coreRewardRatio
```

RECOMMENDATION

Establish checks to validate inputs and enforce input boundaries.

RESOLUTION

VaultLayer team has added checks to validate inputs and enforce input boundaries in mentioned functions.

Identifier	Definition	Severity
LOG-03	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.

RECOMMENDATION

Functions that execute critical state changes should enforce minimum output thresholds. Setting these minimums above zero can deter malicious actors by reducing the predictability and profitability of front-running strategies.

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.

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

Below mentioned functions should be verified for Checks-Effects-Interactions:

`withdrawCore`

RECOMMENDATION

Use Checks-Effects-Interactions (CEI) pattern when transferring control to external entities. This design pattern ensures that all state changes are completed before external interactions occur.

RESOLUTION

VaultLayer team has updated `withdrawCore` and `_unstakeCore` functions to follow Checks-Effects-Interactions (CEI) pattern.

Identifier	Definition	Severity
LOG-05	Unchecked External Calls	Medium 🟡

`unstakeCore` function calls `coreAgent.getCandidateListByDelegator` and `coreAgent.undelegateCoin` without sufficient error handling for edge cases (when candidates are empty or `undelegateCoin` reverts). If `undelegateCoin` fails midway, the loop may leave the contract in an inconsistent state.

RECOMMENDATION

Use robust pattern, such as accumulating `unstakeAmount` in a temporary variable and performing a single transfer after all unstaking is confirmed. Add explicit checks for `candidates.length > 0` and handle partial failures properly.

Identifier	Definition	Severity
COD-01	Unbounded Loops	Medium 🟡

- `_claimBtcRewards` function iterates over `txIds` in reverse to claim rewards and remove expired stakes. If a user has many active transactions (up to `maxActiveTxsPerBtcPubKey`), this can exceed block gas limits, preventing reward claims. Users with many stakes could be unable to claim rewards, effectively locking their funds. An attacker could spam transactions to DoS others.
- `claimStakingRewards` function loops over `allowedDelegators` to claim rewards, with no upper bound on array length. A large number of delegators could make the transaction exceed block gas limits, preventing reward distribution. Reward distribution could become impossible, stalling the system and locking user funds.

RECOMMENDATION

Set `maxActiveTxsPerBtcPubKey` at a lower, gas-safe value, e.g., 50.

Redesign to aggregate rewards off-chain and submit in a single call.

RESOLUTION

VaultLayer team has:

- Reduced `maxActiveTxsPerBtcPubKey` to 10 by default.
- Added new variable `maxAllowedDelegators`.

Identifier	Definition	Severity
COD-02	Timestamp dependence	Minor ●

Be aware that the timestamp of the block can be manipulated by miners. Since miners can slightly adjust the timestamp, they may influence contract outcomes to their advantage.

RECOMMENDATION

Avoid relying solely on timestamp of the block for critical contract functions. Follow 15 seconds rule, and scale time dependent events accordingly.

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

Below mentioned functions are missing zero address input validation:

```
recoverEthereumSigner
deriveAddress
```

RECOMMENDATION

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

RESOLUTION

VaultLayer team has added zero address checks.

Identifier	Definition	Severity
COD-10	Direct and indirect dependencies	Unknown ●

`VaulterCore` relies on *OpenZeppelin* libraries (*ERC20*, *ReentrancyGuard*, etc.) without a fixed version and external contracts (`IStakeHub`, `IBitcoinStake`, `ICoreAgent`) that could break or be compromised if upgraded or malicious. 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.

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-12	Lack of event-driven architecture	Minor 🟢

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

```
grantManagerRole
revokeManagerRole
```

RECOMMENDATION

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

RESOLUTION

VaultLayer team has added events to track state changes in mentioned functions.

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`.

COMMENT

VaultLayer team comments that the keccak256 collision has little effect on the functionality as it’s related to signed data, except for the storage layout that can be solved by creating test files to check storage slot collisions. `keccak256` function is widely adapted in cryptography, and its use is relatively safe.

Identifier	Definition	Severity
COM-01	Floating Pragma	Minor ●
COM-02	Multiple Pragma Directives	

Compiler is set to:

```
pragma solidity ^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 contracts are deployed with stable compiler.

Identifier	Definition	Severity
COM-03	Division Before Multiplication	Minor ●

`getBtcRewardRatio` function uses `Math.mulDiv` for fixed-point arithmetic, but the order of operations in calculating `D_fixed (Math.mulDiv(currentRatio, 1000, targetRatioReserve))` can lead to precision loss if `currentRatio` is small relative to `targetRatioReserve`.

RECOMMENDATION

Rearrange the calculation to multiply first, then divide, and use intermediate variables to ensure precision.

Identifier	Definition	Severity
COM-04	Gas Optimization	Medium 🟡

Below mentioned functions are gas-intensive due to repeated memory allocations and byte manipulations:

```
toAsciiString
uintToStr
getBtcRewardRatio
unstakeCore
verifyEthPubKeySignature
```

RECOMMENDATION

Optimize functions, arrays, and loops identified as high gas consumers by simplifying logic, minimizing state changes, and limiting loop iterations where possible.

ACKNOWLEDGED

VaultLayer team has acknowledged this finding and commented that, “We left the `CoreHelper` as a separate modular contract, with a clear interface, so we can improve the delegator selection over time, without having to upgrade the main `vltCORE` contract.”

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