INTERFI

SMART CONTRACT SECURITY ASSESSMENT

PREPARED FOR

DECENTRALIZED OTC [deOTC]

hello@interfi.network

interfinetwork

www.interfi.network

FOR DECENTRALIZED OTC

Date of Report	August 30, 2025
Code	0xFe7cbf216d6bbE781d78fC8AB02b91250D52D956
Platform	Ethereum
Website	https://deotc.io
Telegram	https://t.me/de_OTC1
X	https://x.com/DeOtcPLATFORM
Language	Solidity
Methodology	Automated Review, Unit Tests, Manual Review
Auditor	InterFi

Disclaimer: Smart contracts deployed on blockchains are inherently exposed to potential exploits, vulnerabilities, and security risks. Blockchain and cryptographic technologies are emerging and carry ongoing uncertainties. Please review the full audit report for detailed insights into risk severity, vulnerabilities, and audit scope limitations.

Centralization Warning: Centralized privileges—regardless of intent or access control—introduce elevated risks to contract security and user trust.

KYC Advisory: The project lacks verified third-party KYC of its owners, team, or deployers. Without independent KYC, transparency and accountability are reduced, increasing the risk of fraud or rug pulls.

Verification: Verify this report: https://www.github.com/interfinetwork

TABLE OF CONTENTS

1. SUMMARY

The audit resulted in the identification of issues across a range of severity levels, including logic flaws, access control oversights, and design inconsistencies. All high-impact findings were communicated to the development team with clear recommendations for remediation. Where applicable, the team has confirmed implementation of fixes or provided justifications for design choices.

1.1 Summary of Findings

Severity	Count
Critical	1 / FIXED
Major	0
Medium	2 / FIXED
Minor	6 / FIXED
Unknown	1 / ACKNOWLEDGED
Centralization	1 / ACKNOWLEDGED

1.2 Resolution Status

Status	Count
Fixed	7
Partially Fixed	1
Acknowledged	3
Pending	0

1.3 System Overview

This system is a decentralized protocol comprising a suite of smart contracts. These contracts collectively define the rules, permissions, and operational workflows for managing on-chain assets, executing user interactions, and enforcing protocol-level logic. Smart contracts in this context are self-executing code units that autonomously manage the state and behavior of digital assets based on predefined conditions. The protocol utilizes these contracts to enable key functionalities such as:

- Ownership and access control enforcement
- Permission and role-based actions
- Data storage and updates
- Event logging and auditability
- Batch processing and collection management

1.4 Files in Scope

InterFi was engaged by `Decentralized OTC` to perform a security audit of the smart contracts. The audit scope was strictly limited to the files explicitly listed under the “Files in Scope” section. No other files or components were reviewed unless otherwise stated.

File	Path	Notes
deOTC	`OTCTrade.sol`	

1.5 Out-of-Scope Assumptions

The following components and assumptions were explicitly excluded from this audit:

- Frontend or backend integration logic.
- Off-chain components, scripts, or oracles.
- External contracts or libraries unless explicitly stated.
- Compiler-level or EVM-specific behavior outside the contract's scope.
- Governance or tokenomics-related decisions not implemented in code.
- All third-party dependencies as discussed in findings.

2. METHODOLOGY

2.1 Audit Objectives

This audit aims to ensure that the smart contract system is predictable, and behaves as intended under normal conditions. Primary audit objectives are to:

- Identify potential vulnerabilities or logic errors in the implementation.
- Evaluate adherence to best practices in smart contract development.
- Assess the correctness of access controls and permission systems.
- Recommend remediations or enhancements for improved security and performance.

2.2 Methodologies

The audit follows a layered security approach using both automated tools and manual techniques. We review the contracts for functional correctness, exploitability, and adherence to smart contract best practices:

Type	Tools & Techniques
Manual Code Review	Line-by-line analysis to check logic, permissions, and edge cases
Automated Analysis	Tools like Slither, MythX, or custom linters to catch known patterns
Static Analysis	Identification of bugs without executing the code (compile-time checks)
Unit Test Inspection	Evaluation of existing test coverage, assumptions, and potential false positives/negatives (if applicable)
Architecture Review	Mapping of privileged roles, callable paths, and contract interdependencies (if applicable)

2.3 Risk Categorization

Each issue identified during the audit is assigned a severity level based on its potential impact, exploitability, and likelihood of real-world abuse. These categories help prioritize remediation efforts:

Risk Severity	Definition
Critical	Represents a severe vulnerability that may result in complete contract compromise, such as asset theft, permanent loss of functionality, or unrestricted access. These issues are often easily exploitable and require immediate resolution.
Major	Indicates significant risk that can affect core contract behavior, enable unauthorized operations, or create unintended financial exposure. While not as urgent as critical risks, they should be remediated promptly.
Medium	These are moderate-level risks that may become exploitable under specific conditions. They often relate to logic errors, insufficient validation, or architectural oversights that could escalate over time.
Minor	Denotes issues that have low security impact but may degrade code quality, performance, or maintainability. These include inefficiencies, style violations, or redundant logic. Fixes are recommended for robustness.
Unknown	Risks where the severity cannot be confidently determined due to limited context, external dependencies, or ambiguous design intent. It is advised to treat these conservatively and address them proactively.
Centralization	Any function controlled by a single privileged role is treated as a critical risk, regardless of its purpose, due to the potential for misuse, override, or total asset control.

2.4 Resolution Status Definitions

All identified issues are also assigned a resolution status, indicating the current handling and response from the development team:

Status	Definition
Fixed	The issue has been remediated and verified as resolved during the re-audit or final check.
Partially Fixed	The issue has been partially mitigated, but remnants or related concerns may still exist. Further attention may be required.
Acknowledged	The development team has accepted the finding but opted not to implement a fix.
Pending	The issue remains unresolved at the time of publication. It poses a potential risk and should be addressed.

3. FINDINGS

01	Unchecked ERC-20 transfers
Severity	Critical
Affected	`addSale, buy, cancelSale`
Description	`addSale`: `IERC20(token).transferFrom(msg.sender, address(this), amount);` does not check return value. A malicious or non-standard token may return false (or do nothing) and the sale would still be created without the contract receiving tokens. `buy`: `IERC20(token).transferFrom(msg.sender, address(this), amount);` does not check return value. Buyer receives no tokens, while seller already received payment. This enables a honeypot: seller lists a non-compliant token > buyers pay > payout silently fails. `cancelSale`: Refund uses `IERC20(...).transfer(...)` without checking return value. If token returns false (or `contract balance < required` because initial deposit was FoT), function continues, marks the sale *Cancelled*, and seller cannot retry or recover tokens.
Recommendation	Use `SafeERC20` everywhere. Ensure all token ops revert on failure (do not ignore return values).
Status	Fixed
	Using `SafeERC20` for `IERC20`

02	Inventory mismatch for FoT/deflationary tokens
Severity	Medium
Affected	`addSale, buy, cancelSale`
Description	`s.amount` is assumed fully held by the contract. For FoT tokens, the contract receives less than `amount` on deposit, making later payouts/refunds impossible or partial (even with `SafeERC20`).
Recommendation	On deposit, compute `received = after - before` and use `received` as the sale's `amount` Consider guarding `buy` against under-inventory by checking `IERC20(token).balanceOf(address(this))`.
Status	Partially Fixed
	`addSale` now measures `actualAmount = postBalance - preBalance` for ERC-20 and stores that, no runtime balance check before payouts. FoT may still deliver less to buyers than quantity, and there's no balance guard before payout.

03	Price adjusted flow bricks sales
Severity	Medium
Affected	`adjustPrice, buy`
Description	`adjustPrice` sets status to Adjusted, but `buy()` only allows `ForSale`. There's no method to revert to `ForSale`, so sales become unbuyable until cancel/re-list.
Recommendation	Keep status `ForSale` on price change, or allow `buy` when `status == Adjusted`, or add `resumeSale()`.
Status	Fixed
	`buy()` now permits `ForSale`

04	ETH transfer (fixed gas) can DoS
Severity	Minor
Affected	`buy, cancelSale`
Description	Using `transfer` may fail for contracts with complex receive/fallback (or future gas cost changes), blocking all buys/cancellations.
Recommendation	Use `call` pattern Keep `nonReentrant`, update state first and follow CEI.
Status	Fixed

05	Potential overflow when computing 10 ** decimals for malicious tokens
Severity	Minor
Affected	`buy (cost = quantity * pricePerToken / 10**decimals)`
Description	For `decimals > 77, 10**decimals` overflows and buy reverts (DoS). An `acceptedPayment` with absurd decimals can brick the sale.
Recommendation	Add require statement to keep `decimals` in check.
Status	Fixed
	`setTokenInfo` now enforces `decimals <= 18` for ERC-20s. But `_nativeDecimals` is not bounded in constructor.

06	knownTokens can contain duplicates
Severity	Minor
Affected	`knownTokens`
Description	It's keyed `off volumeByToken == 0`. Multiple `addSales` before first `buy` push duplicates.
Recommendation	Track `mapping(address => bool) isKnown` before pushing.
Status	Fixed

07	Metadata fetch try/catch only wraps name()
Severity	Minor
Affected	`symbol()`, `decimals()`
Description	`symbol()` / `decimals()` outside the `try` block may revert with generic messages.
Recommendation	Wrap all three calls in the same `try` block, or perform separate guarded calls.
Status	Fixed

08	NATIVE sentinel using address(1) is non-standard
Severity	Minor
Affected	`contract OTCTrade`
Description	NULL
Recommendation	Use a dedicated constant flag, or `address(0)` with separate handling, or a `bool isNative`.
Status	Acknowledged

09	Potential front-running
Severity	Minor
Affected	`contract OTCTrade`
Description	Owner could change fees between user signing and tx inclusion; not front-running in the traditional sense but a centralization risk.
Recommendation	Timelock fee changes Max-fee arg in `buy()` to protect buyers
Status	Fixed

10	Direct & Indirect Dependencies
Severity	Unknown
Affected	`contract OTCTrade`
Description	This contract relies on external components and environment assumptions: ▪ *OpenZeppelin* primitives (`Ownable`, `ReentrancyGuard`, `IERC20`) and their exact versions. ▪ External contracts: `discountContract` (assumed to implement `getSellerDiscount/getBuyerDiscount`). ▪ Environmental assumptions: ERC-20 tokens may be non-standard (USDT-style). Behavior varies by token. Any divergence from audited baselines can impact safety and liveness.
Recommendation	Pin and vendor audited dependency commits; prefer `SafeERC20` everywhere; whitelist immutable, well-known tokens; monitor upstream advisories; blocklist known non-standard tokens.
Status	Acknowledged

4. CENTRALIZATION

Centralization is one of the leading causes of smart contract-related asset losses. When a contract assigns critical powers to a privileged role—such as an `owner`, `admin`, or designated `controller`—the associated risk becomes elevated, especially if that role is tied to a single externally owned account (EOA). In many cases, privileged roles serve operational or safety functions, including:

- Emergency/Operational Controls: Ability to redirect payoffs with `setVault()` and remove inventory with `withdrawTokens()`.
- Contract Configuration: Ability to rotate the `operator` who can create orders for users.

4.1 Noteworthy Privileged Functions

`setDiscountActive(bool)`	onlyOwner
`setDiscountContract(address)`	onlyOwner
`setSellerFeeBps(uint256)` *(capped ≤ 3%)*	onlyOwner
`setBuyerFlatFee(uint256)`	onlyOwner
`setTreasury(address)`	onlyOwner
`cancelSale(uint256)`	`seller or `admin`

01	Centralized Privileges
Severity	Centralization
Description	EOAs with control can be compromised via phishing, private key leakage, or insider threats. Malicious or negligent use of privileges can lead to - token supply manipulation, disruption of trading via pausing, arbitrary fee changes or wallet exclusions, asset seizures or rerouting, etc. 4.1 Privileged Functions --- Owner can set unbounded `buyerFlatFee` and can redirect treasury to any address. This can confiscate buyer funds.
Recommendation	Using Multi-Signature Wallets: Assign privileged roles to a multi-sig contract requiring signatures from multiple trusted parties. This reduces the impact of any single compromised key. Time-Locked Functions: Introduce delays before executing sensitive operations, allowing time for community review or cancellation. Role Revocation or Transfer: If privileges are no longer needed post-deployment, renounce them or migrate them to DAO governance. Secure Key Management: Any private keys associated with privileged roles must be protected using hardware wallets, secret sharing schemes, or offline signing protocols.
Status	Acknowledged

5. DISCLAIMER

InterFi Network provides professional smart contract audits for blockchain-based codebases (commonly known as smart contracts). This audit assessed the reviewed contract(s) for common vulnerabilities, centralization risks, and logic flaws. However, no audit can guarantee the complete absence of bugs or vulnerabilities. This report does not constitute a security guarantee, endorsement, or assurance of business model soundness or legal compliance.

The review is limited strictly to the source code and its logic as provided, and does not extend to compiler behavior, off-chain components, or external integrations. Due to the evolving nature of blockchain technology and associated risks, users should understand that all materials, including this audit report, are provided strictly on an “as is”, “as available”, and “with all faults” basis.

5.1 Confidentiality

This report is confidential and intended solely for the client. It may not be disclosed, reproduced, or relied upon by third parties without prior written consent from InterFi Network. All terms, including confidentiality, liability limitations, and scope, are governed by the audit agreement.

5.2 No Financial Advice

This report is not financial, investment, tax, legal, or regulatory advice. It should not be relied upon for making investment decisions or assessing the value, viability, or safety of any token, product, or platform. No part of this document should be interpreted as an endorsement or recommendation. InterFi Network accepts no liability for any actions taken based on this report.

5.3 Technical Disclaimer

InterFi disclaims all warranties—express, implied, or statutory—including merchantability, fitness for a particular purpose, title, and non-infringement. We do not guarantee that the reviewed contracts are error-free, fully secure, or meet any specific requirements. Audit results may contain false positives or negatives, and findings are subject to the context and limitations of the review scope.

5.4 Timeliness & Accuracy

Audit results reflect the state of the code at the time of review. InterFi makes no commitment to update findings after publication. We do not warrant the accuracy, completeness, or timeliness of information delivered via this report.

5.5 Third-Party Links

This report may contain references or links to external websites and social media accounts. InterFi Network is not responsible for the content or operation of third-party platforms and assumes no liability for actions taken based on their content.

6. ABOUT

InterFi Network is a leading provider of intelligent blockchain solutions, offering secure, scalable, and production-ready smart contract services. Our team specializes in the development, testing, and auditing of smart contracts across a wide range of blockchain ecosystems.

We have delivered:

- 300+ smart contract systems developed
- 2,000+ smart contracts audited
- 500,000+ lines of code reviewed and analyzed

Our technical expertise spans multiple languages including:

- Solidity for EVM-compatible chains (Ethereum, BNB Chain, Polygon, Avalanche, Cronos, Fantom, Velas, Metis, and more)
- Move for next-generation platforms such as Sui and Aptos
- Rust for advanced ecosystems like Solana, Near, and Cosmos SDK-based chains

6.1 Connect with Us

InterFi Network is driven by a multidisciplinary team of engineers, developers, UI/UX specialists, and blockchain researchers. The core team consists of 3 senior members supported by 4+ expert contributors across code auditing, tooling, and protocol design.

- Website: interfi.network
- Email: hello@interfi.network
- GitHub: github.com/interfinetwork
- Telegram (Engineering): @interfiaudits
- Telegram (Onboarding): @interfisupport

THANK YOU