DogScan Security Audit Report

Project	**LeverageTradingDiamond**
Chain	BSC Testnet (ID 97)
Contract Address	0xBeDddDe4c61BBfD9f3a51F88c1Ad8466e9B9Ec0a
Audit Date	2025-07-06
Report Version	1.0

Table of Contents

1. Executive Summary

We conducted a security audit of the LeverageTradingDiamond contract system (a leverage trading protocol based on the EIP-2535 Diamond standard). The audit results show that the system's overall security status is good, with no serious security vulnerabilities discovered.

During the audit process, we identified one low-risk technical issue, primarily related to optimizing user trading experience. While the protocol employs centralized governance mechanisms that exhibit certain centralized characteristics, these are acceptable governance risks within the current design framework. The core business logic implementation is reasonable, with proper security protection measures in place.

We recommend that the project team consider the architectural recommendations outlined in this report before mainnet deployment, to further enhance the protocol's decentralization and user experience.

2. Audit Scope

The audit scope covers the LeverageTradingDiamond contract and its related Facets and libraries. The main proxy contract is located at `0xBeDddDe4c61BBfD9f3a51F88c1Ad8466e9B9Ec0a`.

The following source files were considered within the scope of this analysis: - `contracts/diamond/Diamond.sol` - `contracts/diamond/LibDiamond.sol` - `contracts/diamond/interfaces/IDiamondCut.sol` - `contracts/diamond/interfaces/IDiamondLoupe.sol` - `contracts/diamond/facets/DiamondCutFacet.sol` - `contracts/diamond/facets/DiamondLoupeFacet.sol` - `contracts/leverage/LeverageTradingDiamond.sol` - `contracts/leverage/LibLeverageTradingStorage.sol` - `contracts/leverage/facets/LeverageTradingOperationsFacet.sol` - `contracts/leverage/facets/LeverageTradingQueryFacet.sol` - `contracts/PriceFeed.sol` - `contracts/PancakeHelper.sol` - `contracts/Utils.sol`

3. Audit Methodology

This audit employed a multi-agent AI security analysis workflow. The process deploys multiple specialized AI agents to analyze smart contract code, design, and on-chain context. Each agent focuses on specific categories of vulnerabilities, such as reentrancy, access control, mathematical vulnerabilities, and economic model risks. A chief security strategist AI then collects and synthesizes the findings from these specialized agents, conducting a comprehensive review to identify systemic risks, prioritize findings by severity, and compile this final comprehensive report.

4. Findings Summary

ID	Title	Severity	Status
L-01	Trading functions lack slippage protection, potentially affecting user trading experience	**Low**	**For Optimization**

5. Detailed Findings

[L-01] Trading functions lack slippage protection, potentially affecting user trading experience (Low)

Severity: Low

Description

The `closePositionWithData` function in `LeverageTradingOperationsFacet.sol` hardcodes the `amountOutMinimum` parameter to 0 when interacting with the DEX router, or sets the `amountInMaximum` parameter to `type(uint256).max`. While this does not directly lead to security vulnerabilities, it may affect user trading experience during periods of high market volatility.

Impact

During extreme market conditions, users may encounter worse execution prices than expected, affecting the trading experience. However, the probability of this occurring is relatively low, and it does not directly result in fund loss.

Code Snippet

```
// File: contracts/leverage/facets/
LeverageTradingOperationsFacet.sol

allMarginAmount = lts.dexRouter.exactInputSingle(
    ISwapRouter.ExactInputSingleParams({
        ...
        amountOutMinimum: 0, // Could consider adding slippage
protection
        ...
    })
);
```

Recommendations

Consider the following optimization approaches to improve user trading experience: 1. **User-defined slippage**: Add slippage protection parameters to the `ClosePositionParams` struct. 2. **Reasonable defaults**: If user-customizable slippage is not allowed, set a reasonable default slippage protection value. 3. **Documentation**: Clearly explain the current trade execution mechanism in documentation.

6. Architecture and Design Recommendations

In addition to the above findings, we propose the following architectural and design recommendations to improve the protocol's robustness and user experience:

1. **Governance Mechanism Optimization**: The protocol currently employs centralized governance mechanisms, including `contractOwner` control over `diamondCut` and `DATA_PROVIDER_ROLE` signature requirements for transaction data. While this is acceptable within the current design framework, long-term consideration could be given to introducing more decentralized governance mechanisms, such as multi-signature wallets or timelock contracts.

2. **Data Provider Role Transparency**: The current system relies on `DATA_PROVIDER_ROLE` for signature verification of transaction parameters. We recommend clearly documenting the purpose and operation of this mechanism to enhance user trust in the system.

3. **Price Oracle Stability**: The system uses a 5-second time window for price calculations, which effectively prevents instantaneous price manipulation attacks. Consider documenting the design principles and security measures of the price oracle.

4. **Access Control Documentation**: The protocol contains multiple functions requiring special permissions. We recommend creating detailed permission management documentation that explains the responsibilities and authority scope of various roles.

5. **Signature Verification Mechanism Explanation**: The current signature verification mechanism ensures through the `_validatePositionForClosing` function that only position owners can operate their own positions. We recommend clearly explaining the implementation principles of this security mechanism in code comments.

6. **View Function Optimization**: For view functions that may return large amounts of data (such as `getUserPositions`), consider adding pagination mechanisms or data volume limits to improve query efficiency and user experience.

7. Conclusion

This audit found that the LeverageTradingDiamond contract system has good overall security status. While there are some design issues that could be optimized, none constitute serious security threats.

The protocol's Diamond architecture implementation complies with the EIP-2535 standard, with reasonable core business logic design and proper security protection measures. The identified issues mainly focus on user experience optimization and further improvement of governance mechanisms. We assess the protocol’s overall security status as **“Low Risk”**.

We recommend that the project team consider the architectural recommendations outlined in this report before mainnet launch, to further enhance the protocol’s decentralization and user experience.

8. Disclaimer

This audit report is for reference only and does not constitute investment advice. The analysis is based on smart contract source code provided at a specific point in time and is not an endorsement of the project or a guarantee of its security. Smart contracts have inherent risks, including the possibility of undiscovered vulnerabilities. Users should conduct their own research and exercise extreme caution when interacting with any smart contract.