verichains

SECURITY AUDIT OF

BEAN EXCHANGE DLMM

Public Report

Oct 20, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Oct 20, 2025. We would like to thank the Bean Exchange for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Bean Exchange DLMM. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified no vulnerable issue in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Bean Exchange DLMM

Bean Exchange DLMM is Bean Exchange's custom-built architecture for spot trading on **Monad**. Inspired by systems like **Trader Joe**'s Liquidity Book and **Meteora**, it is engineered to optimize capital efficiency, execution quality, and LP profitability in a performant and composable DeFi environment.

Bean Exchange DLMM implements an adaptive fee system that automatically adjusts to market volatility and trading patterns. This mechanism ensures optimal compensation for liquidity providers while maintaining competitive pricing for traders through real-time market-responsive fee adjustments.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Bean Exchange DLMM.

The latest versions of the following files from the git repository `https://github.com/BeanExchange/dlmm-audit` at commit `6d1e64159c715bb33fcf77796c5cbce44bba396a` were audited:

SHA256 Sum	File
4549b3cb39cd4b275a5d76feb80719a663b49c6dc2cdc8b6eb001ddd35ab6f1e	src/interfaces/IDexLens.sol
d7649811013282eed02180a3a4a20fd13db7017223fa76e683b7c853ea4384ac	src/interfaces/ILBFactory.sol
4f07a31be3c441cef49d3460789b934a664ebc5d754d3da6910e13c27c499e37	src/interfaces/ILBFlashLoanCallback.sol
f6fd0c21c0aef7c841fcbb7feb509d66822dacfd987d5203cca7e46cac12c363	src/interfaces/ILBHooks.sol
9fd927e06575ce2acd7507a1d098cc67f411f4a6e0ca42d0575e201aa860e01c	src/interfaces/ILBPair.sol
7720b6a546d30e2c7fce23e222a4bab3e7feb604291951e7b5b40d220da5de7e	src/interfaces/ILBRewarder.sol
fd8b730b3b8eef743fc891360f13a2ad2fffca46f87e726dd51d3f736a33cd70	src/interfaces/ILBRewarderVault.sol

f0348e01e92e1132755849bb1bc4a36344fc2da25967107053f632728ecf8d18	src/interfaces/ILBRouter.sol
4892fffb18c81d5306eb2cb01b2ffc50a50b1f32b02d61eadbca14df242fa6cb	src/interfaces/ILBToken.sol
9d78ff9c300436b7043a4e45e9127834225ef12914cb2a33212f15723d1c8e4f	src/interfaces/IWNATIVE.sol
3d5517aa1b1e08b7b5ada310941f62e75168f94cf27e629ea673aee9a2d894b3	src/LBBaseHooks.sol
58bd1c7dc7dc0ba3a0ed4859bca8c3355525e4e644dce6864b124f90ed100550	src/LBFactory.sol
65051d64df993c239b6b6f90e14716971267c5260df80495cf3d550d00dc4008	src/LBPair.sol
654a5a9172a6df67f9705a8ad6aeb6b259c348c52791580f501ee05db001ab82	src/LBQuoter.sol
da87214255fdab093e5a27df0af3a6a22b675bf161daffad0f550086ddf1ee78	src/LBRewarder.sol
1e7f6dc7669bdc222b1148ab1c75869574ba62c431bd65a2aa7ecf3bedf26f50	src/LBRewarderVault.sol
c9f7333c39f6bcb88a8fa39de28d4641d16edad3bef67a0dcef6d8f0007e992a	src/LBRouter.sol
2176836617c8d5834c14051714505586f45f50673045cc2841f4a7a62793209e	src/LBToken.sol
14cb466e35697fcfc89c3e3110bbe27f685cfed8502121e29051053e9ee1189a	src/libraries/BinHelper.sol
45e8e764e7509f7791aa5837adbf8d949fa0a48a75e39e0feee779a90ba74ea5a	src/libraries/Clone.sol
65bcd1320d9136aa53bde33676b668c8a18f0e066dcff7d9e9f6422e280078f2	src/libraries/Constants.sol
c16b77a6f5821ffe4f67c37dce57a02910d1a5d0c5c11d3024e4e9899c7ac54d	src/libraries/FeeHelper.sol
ebd04bb9fc6ac2481f7ea2bef5adf74b0a129ba04fde0fe7f48fdc0623439ef4	src/libraries/Hooks.sol

4be0a942f8a32d3cea27b2ae44785711a9057b018a87b39af7008e37d08c74a1	src/libraries/ImmutableClone.sol
d6932050af50ea4161b9dcaffca07c2be0ed003bad969fee1d6d1beb733729cd	src/libraries/JoeLibrary.sol
48ae5547d6963f55f241bacb76959af13955e4670446df7946565e63b46eb2a4	src/libraries/math/BitMath.sol
58a5b13426c603c23bc9889b6d92ce99648b24df465779b9ea2c511a8384489d	src/libraries/math/Encoded.sol
f9a185682b43b2404e4c54e53d1f5c0b53fb4fa8131d10f3f192d19146fb2be4	src/libraries/math/LiquidityConfigurations.sol
18940420025707d1727cbf3398bb750bc6150676e8daa3737a77b2dd9c9bee85	src/libraries/math/PackedUint128Math.sol
e117c36aced3fa0fb2c71b67e452a963835e3b763bbe414ac9468182a2fc86b9	src/libraries/math/SafeCast.sol
bcf3955a3d098681be61489dba9757b5db8236a166061ae049ede34fc8f89e33	src/libraries/math/SafeMath.sol
758293af2bdee11b9635190ecbebe687163f770c9e2ced735f9e7b390daf1747	src/libraries/math/SampleMath.sol
b3536d3fb2b3c3f7ac7ff5f6855a921705631f70ab84dbd03362b8c479db5bd2	src/libraries/math/TreeMath.sol
c092f230ef4b13b95ac115701015ecc8eb9be08ab72091eebaedb23768fc2161	src/libraries/math/Uint128x128Math.sol
aca625e3691d01fd8f56362fdd23b45c4b3242dfda3a78aebc0e2543c5aea1c6	src/libraries/math/Uint256x256Math.sol
38dcdb529924d7570ac486a1f6f79a3d709454e28383e7ef36253608b039ac1d	src/libraries/PairParameterHelper.sol
15cd510359cf8d9a6db4ad6b9a3682a66d8b65a7d319c38734794cc073d00965	src/libraries/PriceHelper.sol
87e21b9392c58ca1440cba96274cbf6a9de1218341190c0492c78014fdb8c1c9	src/libraries/ReentrancyGuardUpgradeable.sol
e2a4ec6d791f1ff2ef551b2df6cd2a1c4cc615fd908376744c36757c61347da9	src/libraries/TokenHelper.sol

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Bean Exchange acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Bean Exchange understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Bean Exchange agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Bean Exchange will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Bean Exchange, the final report will be considered fully accepted by the Bean Exchange without the signature.

2. AUDIT RESULT

2.1. Overview

The Bean Exchange DLMM contract is implemented in `Solidity` and targets compiler version `^0.8.20`.

Its architecture and design are inspired by open-source projects such as **Trader Joe** and **Meteora**, but the implementation has been substantially adapted to meet Bean Exchange's specific goals and constraints. The project removes several external dependencies and changes core behaviors to better align with the protocol's risk profile and incentive model.

Key modifications and characteristics:

- **Oracle integrations have been removed**, eliminating reliance on external price feeds within the contract itself.
- **Flash loans are disabled by default**; the factory owner retains the ability to enable flash loan functionality when desired.
- **An adaptive fee mechanism has been introduced**. Fees are designed so that liquidity providers can receive fee income without having to remove their liquidity positions from the pool.
- **A rewards mechanism has been added** to further incentivize liquidity provision. Rewards are distributed based on the contract's emission rate and each user's proportional contribution of liquidity to the pool, so participation is rewarded according to relative stake.
- **The pair contract is upgradeable**: this enables migration for fixes or mitigations but introduces a centralization risk — privileged admins can perform upgrades/migrations that, if misused or compromised, could result in unauthorized fund transfers or loss of user assets.

Together, these changes reduce external attack surface, provide owner-controlled risk options for flash loans, and align economic incentives for liquidity providers through both fee accrual and emission-based rewards.

2.2. Findings

During the audit process, the audit team found no vulnerability in the given version of Bean Exchange DLMM. However, there are some additional information notes.

2.3. Additional Notes

2.3.1. INFORMATIVE - Upgradeable Proxy Implementation Risk in LBPair

Scope

- Commit: `6d1e64159c715bb33fcf77796c5cbce44bba396a`
- Files:
  - `src/LBFactory.sol`

Description

The LBPair contracts are deployed using `OpenZeppelin`'s `TransparentUpgradeableProxy` with the factory owner as the `ProxyAdmin`. This creates significant centralization risks and potential attack vectors that could compromise user funds and protocol integrity.

```
// In LBFactory.createLBPair()
TransparentUpgradeableProxy proxy = new TransparentUpgradeableProxy(
    implementation,
    owner(), // ProxyAdmin contract - CENTRALIZATION RISK
    data
);

pair = ILBPair(address(proxy));
```

RECOMMENDATION

While this is a design choice, it is recommended to implement robust governance mechanisms around the `ProxyAdmin` role. This could include multi-signature wallets, time-locked upgrades, or decentralized governance to mitigate risks associated with centralized control over contract upgrades.

2.3.2. INFORMATIVE - Dust Fee Loss During Settlement Due to Precision Truncation

Scope

- Commit: `6d1e64159c715bb33fcf77796c5cbce44bba396a`
- Files:
  - `src/LBToken.sol`

Description

When users interact with positions that have accumulated very small fee amounts, the fee settlement process can result in permanent loss of "dust" fees due to precision truncation in the `_calculateFeesEarned` function. This occurs when the calculated fee amount is too small after the `mulDivRoundDown` operation, causing the earned fees to be rounded down to zero while the fee growth checkpoint is still updated.

```
function _calculateFeesEarned(
    uint256 liquidity,
    uint256 currentGrowth,
    uint256 lastGrowth
) internal pure returns (uint256 earned) {
    if (currentGrowth >= lastGrowth) {
        uint256 growthDelta = currentGrowth - lastGrowth;
        // ISSUE: mulDivRoundDown truncates small amounts to 0
        earned = liquidity.mulDivRoundDown(growthDelta, Constants.ONE);
    }
}

function _settleFees(address account, uint256 binId) internal {
    // ... existing code ...

    uint256 earnedX = _calculateFeesEarned(
        balance,
        binFee.feeGrowthX128,
        position.feeGrowthX128Last
    );

    // If earnedX = 0 due to dust, fees are lost
    if (earnedX > 0 || earnedY > 0) {
        position.tokensOwedX = newOwedX;
        position.tokensOwedY = newOwedY;
    }

    // PROBLEM: Checkpoints updated regardless, losing future claim opportunity
    position.feeGrowthX128Last = binFee.feeGrowthX128;
    position.feeGrowthY128Last = binFee.feeGrowthY128;
}
```

RECOMMENDATION

This is an informational issue that users can mitigate by understanding the precision limitations and timing their fee collections appropriately. The impact is minimal on a per-transaction basis but represents a design characteristic users should be aware of.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 20, 2025*	Public Report	Verichains Lab

Table 2. Report versions history