verichains

SECURITY AUDIT OF

BEAN EXCHANGE DLMM

Public Report

Nov 11, 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 Nov 11, 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 some vulnerable issues 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. Specifically, the team analyzed the codebase changed from the originally Trader Joe's `dlmm` codebase.

The latest versions of the following files from the git repository `https://github.com/BeanExchange/dlmm-audit` at commit `8213cd593ab36ab5dbbe71ff810b467a2282cc93` were audited:

SHA256 Sum	File
4549b3cb39cd4b275a5d76feb80719a663b49c6dc2cdc8b6eb001ddd35ab6f1e	src/interfaces/IDexLens.sol
351c43ffd65dbd60d1bff01597f876b27ba3c61f7a709561c69e7e18fe8d31ab	src/interfaces/ILBFactory.sol
4f07a31be3c441cef49d3460789b934a664ebc5d754d3da6910e13c27c499e37	src/interfaces/ILBFlashLoanCallback.sol
f6fd0c21c0aef7c841fcbb7feb509d66822dacfd987d5203cca7e46cac12c363	src/interfaces/ILBHooks.sol
458817dc92027e2b1b361497c927d1a3829fb88dd948d47628bbf97b7acd9c21	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
8dbac3cefdbd2386850fe6bc1c316271a0ad9ee5fa50add42039f1170e3cd8c8	src/LBBeacon.sol

4844968523bf64bdbe7dc1498147c235f71d27751d6f3cb24e97762ca6fc9e1a	src/LBFactory.sol
dc1a2d56fb25db1921756a56a1937b0288ed59a848eebbe7a5f00825e07b05be	src/LBPair.sol
4b157401e6c5d4c6bdc10a0fcacf26e1973af74db284a7cefdfb2191fecd992b	src/LBQuoter.sol
29279a8ce36ace174cadc7fd30efe6cb99733d0fc28749dfda9ae2f56d1db279	src/LBRewarderUpgrade.sol
1e7f6dc7669bdc222b1148ab1c75869574ba62c431bd65a2aa7ecf3bedf26f50	src/LBRewarderVault.sol
6edd6d3db040fb281fde5c8eb80568180c3ae11ed23a8913978074be7b0d13c7	src/LBRouter.sol
82d09a0d7726f57d1d37b5e6767c75b17d2a5ab4aa4b9134409e2f03972ac7f8	src/LBTokenUpgrade.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
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
b4dbafbbdb87bd3dc569a172671afbea55374c084b9e07f8b7bf0859b1a575a0	src/libraries/OracleHelper.sol
bbcbd94d51f1b77ee07c37e60275e386392cdbc91eb8b707d4c96e00776d9d99	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:

- **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 some vulnerability in the given version of Bean Exchange DLMM.

#	Title	Severity	Status
1	Arbitrary Call in `increaseOracleLength`	MEDIUM	FIXED
2	Upgradeable Proxy Implementation Risk in LBPair	INFORMATIVE	ACKNOWLEDGED
3	Missing `AccessControlUpgradeable` in Upgradeable Contract	INFORMATIVE	FIXED

2.2.1. MEDIUM - Arbitrary Call in `increaseOracleLength` function

Scope

- Commit: `9420f1427e9947f70a0239eb10cca408c6710c6d`
- Files:
  - `src/LBPair.sol`

Description

The `increaseOracleLength` function allows any user to increase the oracle array size up to 65,535 samples without access control or validation. This creates a time-delayed oracle manipulation vector where an attacker can gradually corrupt the price oracle by filling the expanded storage with manipulated data over weeks, ultimately compromising `TWAP` calculations used by the protocol and integrated systems.

```
function increaseOracleLength(uint16 newLength) external override nonReentrant { // No
access control - anyone can call

}
```

The impact includes price oracle manipulation, cross-protocol contagion, and irreversible protocol degradation. However, the exploit requires significant time and resources, making it a medium-severity issue.

RECOMMENDATION

Implement access control to restrict who can call `increaseOracleLength`. Additionally, consider adding validation to ensure the new length is within reasonable bounds.

UPDATES

- ***Nov 11, 2025***: The issue has been fixed by the Bean Exchange DLMM development team in commit `8213cd593ab36ab5dbbe71ff810b467a2282cc93`.

2.2.2. 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(), //@auditor 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.

UPDATES

- ***Oct 21, 2025***: The issue has been acknowledged by the Bean Exchange DLMM development team.

2.2.3. INFORMATIVE - Missing `AccessControlUpgradeable` in Upgradeable Contract

Scope

- Commit: `13414e2f330c3cf5fdb8a40eb9e868aaf1a551bb`
- Files:
  - `src/LBPair.sol`

Description

The `LBPair` contract inherits from multiple upgradeable contracts (`LBRewarderUpgradeable`, `LBTokenUpgradeable`, `ReentrancyGuardUpgradeable`) but uses the non-upgradeable `AccessControl` from `OpenZeppelin` instead of `AccessControlUpgradeable`. This creates potential storage layout conflicts and initialization issues in the upgradeable proxy pattern, which could lead to unpredictable behavior or storage corruption during upgrades.

```
contract LBPair is LBRewarderUpgradeable, LBTokenUpgradeable, ReentrancyGuardUpgradeable,
Clone, ILBPair, AccessControl {

// ^^^^^^^^^^^^ @auditor: Should be AccessControlUpgradeable
```

RECOMMENDATION

Replace AccessControl with AccessControlUpgradeable and ensure proper initialization:

```
import "@openzeppelin/contracts-upgradeable/access/AccessControlUpgradeable.sol";
contract LBPair is LBRewarderUpgradeable, LBTokenUpgradeable, ReentrancyGuardUpgradeable,
Clone, ILBPair, AccessControlUpgradeable {
    function initialize(...) public initializer {
        //... other initializations
        __AccessControl_init();
        //... rest of initialization
    }
}
```

UPDATES

- ***Nov 11, 2025***: The issue has been fixed by the Bean Exchange DLMM development team in commit `9420f1427e9947f70a0239eb10cca408c6710c6d`.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Oct 20, 2025*	Public Report	Verichains Lab
1.1	*Oct 21, 2025*	Public Report	Verichains Lab
1.2	*Nov 11, 2025*	Public Report	Verichains Lab
1.3	*Nov 11, 2025*	Public Report	Verichains Lab

Table 2. Report versions history