verichains

SECURITY AUDIT OF

FERRA DLMM SMART CONTRACT

Ferra

Public Report

Sep 26, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Sui Blockchain	Sui is an innovative, decentralized Layer 1 blockchain that redefines asset ownership. Sui Move feels like a paradigm change in web3 development. Treating objects as 1st class citizens brings composability to a whole new level. Polymedia. We are thrilled to be building on Sui.
Sui Object	The basic unit of storage in Sui is object. In contrast to many other blockchains where storage is centered around accounts and each account contains a key-value store, Sui's storage is centered around objects.
Move	Move is a new programming language that implements all the transactions on the Aptos/Sui blockchain.
Move Module	A Move module defines the rules for updating the global state of the Aptos/Sui blockchain. In the Aptos/Sui protocol, a Move module is a smart contract.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Sep 26, 2025. We would like to thank the Ferra 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 Ferra DLMM Smart Contract. 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 team identified several areas in the smart contracts code that could be improved. The Ferra team has addressed some of these findings and acknowledged the remaining points.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Ferra DLMM Smart Contract

Ferra DLMM Smart Contract is written in the Sui programming language and introduces Dynamic Liquidity Market Maker (DLMM) technology to optimize automated market making and liquidity provision. The platform is designed to address the inefficiencies of traditional constant product market makers by implementing dynamic fee structures and concentrated liquidity mechanisms.

The Ferra DLMM protocol enables liquidity providers to optimize their capital efficiency through intelligent liquidity distribution across price ranges. Unlike traditional AMMs that spread liquidity uniformly across all price ranges, Ferra's DLMM concentrates liquidity where trading activity is most likely to occur, resulting in reduced slippage for traders and enhanced yield opportunities for liquidity providers.

Key features of the Ferra DLMM include:

- **Dynamic Fee Adjustment**: Fees automatically adjust based on market volatility and trading volume to optimize returns for liquidity providers
- **Concentrated Liquidity**: Liquidity is strategically positioned in active price ranges to maximize capital efficiency
- **Impermanent Loss Mitigation**: Advanced algorithms help reduce the impact of impermanent loss on liquidity providers
- **Multi-Asset Support**: The protocol supports various token pairs with optimized liquidity management for each

The Ferra DLMM Smart Contract are built on the Sui blockchain, leveraging Sui's high-performance infrastructure and Move programming language to deliver fast, secure, and cost-effective DeFi operations.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Ferra DLMM Smart Contract.

It was conducted on commit `0643e07c7908d1b479484680ea25f97150b0ffb7` from git repository link: *https://github.com/Ferra-Labs/ferra_dlmm*.

SHA256 Sum	File
0f167d8050079964b506b337badd0c425029be92973c0bf937a9fae4d8cde6ea	sources/acl.move
628dcd972ca20eb3c05bd8556fb2bb9cfeecb59034b82a64248b848676dc66f2	sources/bin.move
55dc3a0fd49a8704c3b001ad85267018eb02401d0824d0525787a7ec6b113325	sources/config.move
130e8df9496855d90a04e2d4cc2b0140418318357a348611bc30ca5e5df16967	sources/lb_factory.move
aee9f4e7f250643d9557f521e25c70cbe59745680d812eff2ab87e774294ad58	sources/lb_pair.move
37004c6bf4a5c0f6b8d0e7f7b065b3ca1e9495438fc1d5098c30c4b008e0ff5a	sources/lb_position.move
ff84e0487315b38cda29b05105124c21ac91d24b3c5da20a1cc6ee8fda00395b	sources/libraries/constants.move
8272966981774bea9f9256bee8e1b76c2584b9568652abf51902a0e0f517d11f	sources/libraries/fee_helper.move
ac80d1771794e820a270f0960d9dd5e471837a6b9a3ede35296ed947a83f48ae	sources/libraries/math/bit_math.move
cc8e6a32245cd4785cbc79bc66bd621fd583c4fe6ba863f94dde77ba3538e607	sources/libraries/math/q64x64.move
4077b583d408bf5b2bbca5570df561a45a0ecd3c0ce27e1c53b3fa87b21ee0d4	sources/libraries/math/safe_math.move
71424ab2def18fde38cdd3a90f934d5901b9dbd984e6637742482e8f97981b5d	sources/libraries/math/tree_math.move
1714a801ae6ada6b6659b15988692019e5c0e77587dc304874e6f5baf1b92ad5	sources/libraries/pair_parameter_helper.move
73f8edcd8cf64325f6b34d21204c8b52c6ba1c37f54f035abc35109a88fb9c1e	sources/libraries/price_helper.move

ff4562949dca1907e98afc93b71ae4f89fc08054fde1ad187da11325d548682b	sources/rewarder.move

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:

- Numerical precision errors
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- Gas Usage, Gas Limit and Loops
- 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

Ferra 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. Ferra understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Ferra 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 Ferra will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Ferra, the final report will be considered fully accepted by the Ferra without the signature.

2. AUDIT RESULT

2.1. Overview

The Ferra DLMM Smart Contract was developed using the Move programming language and deployed on the Sui Blockchain. The codebase is structured to provide modular and reusable components for managing access control, liquidity pools, reward distribution, and other core functionalities of a decentralized liquidity market.

Key Components:

- **Access Control (`acl.move`)**: Implements role-based access control mechanisms to manage permissions within the system.
- **Bin Management (`bin.move`)**: Handles the creation, management, and operations related to liquidity bins.
- **Configuration (`config.move`)**: Handles system-wide configuration settings and parameters.
- **Liquidity Pool Factory (`lb_factory.move`)**: Manages the creation and initialization of liquidity pools.
- **Liquidity Pool Pair (`lb_pair.move`)**: Represents and manages individual liquidity pool pairs.
- **Liquidity Pool Position (`lb_position.move`)**: Tracks and manages user positions within liquidity pools.
- **Reward Distribution (`rewarder.move`)**: Implements mechanisms for distributing rewards to liquidity providers.

Libraries: The `libraries/` directory contains utility modules that support the core functionality:

- **Constants (`constants.move`)**: Defines system-wide constants.
- **Fee Helper (`fee_helper.move`)**: Provides utilities for calculating and managing fees.
- **Pair Parameter Helper (`pair_parameter_helper.move`)**: Manages parameters specific to liquidity pool pairs.
- **Price Helper (`price_helper.move`)**: Contains utilities for price calculations.
- **Math Utilities (`math/`)**:
  - **Bit Math (`bit_math.move`)**: Provides bitwise mathematical operations.
  - **Q64x64 (`q64x64.move`)**: Implements fixed-point arithmetic for high-precision calculations.
  - **Safe Math (`safe_math.move`)**: Ensures safe arithmetic operations to prevent overflows and underflows.
  - **Tree Math (`tree_math.move`)**: Implements tree-based mathematical operations, possibly for data structures like Merkle trees.

2.2. Findings

During the audit process, the audit team found some issues and recommendations in the given version of Ferra DLMM Smart Contract.

#	Issue	Severity	Status
1	Flaw in Type Ordering Checking Algorithm	LOW	ACKNOWLEDGED
2	Unsafe Casts in Liquidity Functions	LOW	FIXED
3	Wrong Package Version Checking	LOW	ACKNOWLEDGED
4	Missing Maximum Liquidity Per Bin Check	LOW	ACKNOWLEDGED
5	Liquidity Lock Expiration Prevents Addition	LOW	FIXED
6	Incorrect Function Visibility	LOW	FIXED
7	Missing Input Validation in Fee Parameters	INFORMATIVE	ACKNOWLEDGED
8	Unused AdminCap Structure	INFORMATIVE	ACKNOWLEDGED

2.2.1. LOW - Flaw in Type Ordering Checking Algorithm

Scope

- Commit: cd692f9926cb963ec53971d9914dabe0e81eeaea
- Files:
  - `sources/lb_factory.move`#new_pair_key

Description

The `new_pair_key<X, Y>` function in `lb_factory.move` contains a fundamentally flawed algorithm that prevents legitimate pair creation and corrupts type name data during the ordering process.

The function attempts to enforce lexicographic ordering of type names but uses the wrong approach. During the comparison loop, the algorithm appends `bytes_b` to `bytes_a`:

```
while (i < vector::length<u8>(&bytes_b)) {
    let byte_b = *vector::borrow<u8>(&bytes_b, i);
    let in_bounds = !is_greater && i < vector::length<u8>(&bytes_a);

    if (in_bounds) {
        let byte_a = *vector::borrow<u8>(&bytes_a, i);
        if (byte_a < byte_b) {
            abort 6
        };

        if (byte_a > byte_b) {
            is_greater = true;
        };
    };

    vector::push_back<u8>(&mut bytes_a, byte_b);  //CORRUPTS original data
    i = i + 1;
};
```

This operation mutates `bytes_a` by appending all elements of `bytes_b`, corrupting the original data. As a result, `bytes_a` will always have a length greater than or equal to `bytes_b`, which allways bypass the following checking:

```
if (!is_greater) {
    if (vector::length<u8>(&bytes_a) < vector::length<u8>(&bytes_b)) {
        abort 6
    };

    if (vector::length<u8>(&bytes_a) == vector::length<u8>(&bytes_b)) {
        abort 3
    };
};
```

The broken type ordering algorithm lets pairs be created without making sure tokens are sorted in the right order. This can lead to duplicate or unclear pair records, which breaks the system's rules and may allow attacks that take advantage of confusing pair identification.

Example scenarios:

- Representation's bytes A is `ABCD` and bytes B is `ABCDE`. The expected behavior is to abort with error code 6 since `ABCD` is lexicographically smaller than `ABCDE`.
- During the loop, `bytes_a` becomes `ABCDABCDE`.
- The length check `if (vector::length<u8>(&bytes_a) < vector::length<u8>(&bytes_b))` fails because `bytes_a` is now longer, allowing the function to proceed incorrectly.

RECOMMENDATION

Replace with proper lexicographic comparison and canonical type ordering without data corruption.

UPDATES

- ***Sep 26,2025***: Issue has been acknowledged by the Ferra team.

2.2.2. LOW - Unsafe Casts in `q64x64.move`

Scope

- Commit: e3cb33436267c1b11c0133fd1b3d6390e4614447
- Files:
  - `sources/q64x64.move`#L53
  - `sources/q64x64.move`#L258

Description

The code contains unsafe casts that could lead to data truncation or unexpected behavior if the values being cast exceed the target type's range. These issues occur in the following locations:

- **Line 53**: Casting from `u128` to `u64` without checking if the value fits within `u64` limits.

```
((y_shifted / price) as u64)  // Unsafe cast from u128 to u64
```

- **Line 258**: Casting from `u256` to `u128` without ensuring the value is within `u128` bounds.

```
(((x_256 * y_256) / d_256) as u128)  // Unsafe cast from u256 to u128
```

The cast could result in a smaller value than expected, causing incorrect calculations in swap operations. The truncation could lead to imbalances in liquidity or incorrect token amounts during swaps.

RECOMMENDATION

To mitigate these risks, it is recommended to implement checks before performing the casts to ensure that the values are within the acceptable range of the target type.

UPDATES

- ***Sep 26,2025***: Issue has been fixed by the Ferra team.

2.2.3. LOW - Wrong Package Version Checking

Scope

- Commit: cd692f9926cb963ec53971d99914dabe0e81eeaea
- Files:
  - `sources/config.move`#checked_package_version

Description

Version check uses `<=` instead of `==`. In scenarios such as a critical vulnerability, administrators may downgrade the object version to a lower value. However, with the current check, the downgraded object can still be accepted by a higher package version, potentially exposing it to exploitation by the vulnerable package.

```
public fun checked_package_version(config: &GlobalConfig) {
    assert!(config.package_version <= VERSION, 10); //Should use equal comparison
}
```

Example scenarios: If the latest package (version 3) contains a critical bug, administrators need to downgrade the config to use version 2 logic while still running the version 3 package. However, the user/attacker still can pass package version check in the package version 3, which contains the critical bug.

RECOMMENDATION

Change to strict equality check: `assert!(config.package_version == VERSION, 10);`

UPDATES

- ***Sep 26,2025***: Issue has been acknowledged by the Ferra team.

2.2.4. LOW - Missing Maximum Liquidity Per Bin Check

Scope

- Commit: cd692f9926cb963ec53971d9914dabe0e81eeaea
- Files:
  - `sources/config.move`#get_shares_and_effective_amounts_in

Description

The function validates individual share amounts against `MAX_SHARES_PER_MINT` but missed to check if the total liquidity per bin exceeds, allowing potential liquidity over the defined maximum, which can lead to wrongful arithmetic operations.

```
public(friend) fun get_shares_and_effective_amounts_in(
    bin: &Bin,
    amount_x: u64,
    amount_y: u64,
    price: u128,
): (u128, u64, u64) {
    //...

    // First mint case
    if (bin_liquidity == 0 || total_supply == 0) {
        let share = u128::max(sqrt_u128(user_liquidity), MIN_INITIAL_SHARE);
        assert!(share <= MAX_SHARES_PER_MINT, E_SHARES_TOO_LARGE);
        return (share, amount_x, amount_y) //Missing max liquidity per bin check
    };
    //...
}
```

RECOMMENDATION

Add maximum liquidity bounds check like `assert!(new_liquidity <= constants::max_liquidity_per_bin(), E_MAX_LIQUIDITY_PER_BIN_EXCEEDED);`

UPDATES

- ***Sep 26,2025***: Issue has been acknowledged by the Ferra team.

2.2.5. LOW - Liquidity Lock Expiration Prevents Addition

Scope

- Commit: cd692f9926cb963ec53971d9914dabe0e81eeaea
- Files:
  - `sources/lb_pair.move`#add_liquidity

Description

Only allows liquidity addition when position is never locked (lock_until = 0). Blocks liquidity addition even after lock period has expired.

```
public fun add_liquidity<X, Y>(
    config: &GlobalConfig,
    pair: &mut LBPair<X, Y>,
    position: &mut LBPosition,
    // ... other parameters
    clock: &Clock,
    ctx: &mut TxContext,
) {
    // ... other checks ...

    assert!(lb_position::get_lock_until(position) == 0, E_POSITION_LOCKED); //Missing check

    // ... rest of function
}
```

RECOMMENDATION

Allow liquidity addition when lock period has expired.

UPDATES

- ***Sep 26,2025***: Issue has been fixed by the Ferra team.

2.2.6. LOW - Incorrect Function Visibility

Scope

- Commit: cd692f9926cb963ec53971d9914dabe0e81eeaea
- Files:
  - `sources/libraries/pair_parameter_helper.move`

Description

Function should be friend-only but is public, **POTENTIALLY** allowing unauthorized access to internal calculations.

```
    public fun update_time_of_last_update(params: &mut PairParameters, timestamp: u64) {}
    public fun update_volatility_reference(params: &mut PairParameters) {}
    public fun update_volatility_accumulator(params: &mut PairParameters, active_id: u32)
{}

// and more ...
```

RECOMMENDATION

Change to `public(friend)` to restrict access to authorized modules only.

UPDATES

- ***Sep 26,2025***: Issue has been fixed by the Ferra team.

2.2.7. INFORMATIVE - Missing Input Validation in Fee Parameters

Scope

- Commit: cd692f9926cb963ec53971d9914dabe0e81eeaea
- Files:
  - sources/config.move#add_update_bin_step

Description

Critical fee parameters lack bounds validation, allowing invalid states that could break calculations.

```
public fun add_update_bin_step(
    config: &mut GlobalConfig,
    bin_step: u16,
    base_factor: u32,  // No validation
    protocol_share: u16,  // Could exceed 100%
```

RECOMMENDATION

Add comprehensive bounds checking for all fee parameters before state updates.

UPDATES

- ***Sep 26,2025***: Issue has been acknowledged by the Ferra team.

2.2.8. INFORMATIVE - Unused AdminCap Structure

Scope

- Commit: cd692f9926cb963ec53971d99914dabe0e81eeaea
- Files:
  - `sources/config.move`#AdminCap

Description

AdminCap is defined but never used anywhere in the contract, missing emergency override capabilities.

```
struct AdminCap has key, store {
    id: UID,
}
```

RECOMMENDATION

Implement AdminCap usage for emergency functions or remove unused structure to reduce code complexity.

UPDATES

- ***Sep 26,2025***: Issue has been acknowledged by the Ferra team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Sep 26, 2025*	Public Report	Verichains Lab

Table 2. Report versions history