Lumia Morphex V3

Security Audit Report

February 12, 2025

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

BugBlow utilized a widely adopted approach to performing a security audit. Below is a breakout of how our team was able to identify and exploit vulnerabilities.

1.2.1 Code and architecture review:

- Review the source code.
- Read the project's documentation.
- Review the project’s architecture.

Stage goals

- Build a good understanding of how the application works.

1.2.2 Check code against vulnerability checklist:

- Understand the project's critical assets, resources, and security requirements.
- Manual check for vulnerabilities based on the auditor's checklist.
- Run static analyzers.

Stage goals

- Identify and eliminate typical vulnerabilities (gas limit, replay, flash-loans attack, etc.)

1.2.3 Threat & Attack Simulation:

- Analyze the project's critical assets, resources and security requirements.
- Exploit the found weaknesses in a safe local environment.
- Document the performed work.

Stage goals

- Identify vulnerabilities that are not listed in static analyzers that would likely be exploited by hackers.
- Develop Proof of Concept (PoC).

1.2.4 Report found vulnerabilities:

- Discuss the found issues with developers
- Show remediations.
- Write an initial audit report.

Stage goals

- Verify that all found issues are relevant.

1.2.5 Fix bugs & re-audit code:

- Help developers fix bugs.

- Re-audit the updated code again.

Stage goals

- Double-check that the found vulnerabilities or bugs were fixed and were fixed correctly.

1.2.4. Deliver final report:

- The Client deploys the re-audited version of the code
- The final report is issued for the Client.

Stage goals

- Provide the Client with the final report that serves as security proof.

Severity classification

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Vulnerabilities leading to assets theft, fund access locking, or any other loss of funds.
High	Vulnerabilities that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Vulnerabilities that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Vulnerabilities that do not have a significant immediate impact and could be easily fixed.

1.3 Project Overview

MorhexV3 is a decentralized exchange (DEX) protocol built on an automated market maker (AMM) model of concentrated liquidity, enabling users to swap any two BEP-20 tokens instantly and directly from their wallets. It features liquidity pools funded by liquidity providers (LPs) that not only facilitate seamless trading but also power yield farming and staking opportunities. Inspired by PancakeSwap V3, MorhexV3 offers a censorship-resistant and efficient trading experience while rewarding users for actively participating in the ecosystem.

1.4 Project Dashboard

Project Summary

Title	Description
Client name	DLumia
Project name	Morphex V3
Timeline	05.02.2025 - 12.02.2025
Number of Auditors	3

Project Last Log

Date	Data Hash	Note
04.01.2025	245ecc286c1c81e77bde61ba0d5d3c0a2b992f94	https://github.com/morphex-exchange/MorphexV3_v2

Project Scope

The audit covered the changes introduced in Morphex V3 based on Pancake Swap V3 (commit 7d2cb5700651b77d140b81d00c561ef6fc4b9f8e) in the following files

File name
hardhat.config.js
projects/masterchef-v3/contracts/MasterChefV3.sol
projects/masterchef-v3/contracts/interfaces/ILMPool.sol
projects/masterchef-v3/contracts/interfaces/ILMPool.sol
projects/masterchef-v3/contracts/interfaces/IMorphexV3Pool.sol
projects/masterchef-v3/contracts/interfaces/INonfungiblePositionManager.sol
projects/v3-core/contracts/MorphexV3Factory.sol
projects/v3-core/contracts/MorphexV3Pool.sol
projects/v3-core/contracts/interfaces/IMorphexV3Factory.sol
projects/v3-lm-pool/contracts/MorphexV3LmPool.sol
projects/v3-lm-pool/contracts/interfaces/IMasterChefV3.sol
projects/v3-lm-pool/contracts/interfaces/IMorphexV3LmPool.sol
projects/v3-lm-pool/contracts/libraries/LmTick.sol
projects/v3-periphery/contracts/MorphexLens.sol
projects/v3-periphery/contracts/NFTDescriptorEx.sol
projects/v3-periphery/contracts/NonfungibleTokenPositionDescriptorOffChain.sol
projects/v3-periphery/contracts/lens/TickLens.sol
projects/v3-periphery/contracts/libraries/ChainId.sol
projects/v3-periphery/contracts/libraries/NFTDescriptor.sol
. projects/v3-periphery/contracts/libraries/NFTSVG.sol
projects/v3-periphery/contracts/libraries/OracleLibrary.sol

projects/v3-periphery/contracts/libraries/PoolAddress.sol
projects/v3-periphery/contracts/libraries/PoolTicksCounter.sol
projects/v3-periphery/contracts/libs/NZGuard.sol
projects/v3-periphery/contracts/libs/SafeOPS.sol

The Client is aware of the known “dangerous tokens” issue in token rewards. The Client is planning to filter all tokens before adding them to the protocol.

1.5 Summary of findings

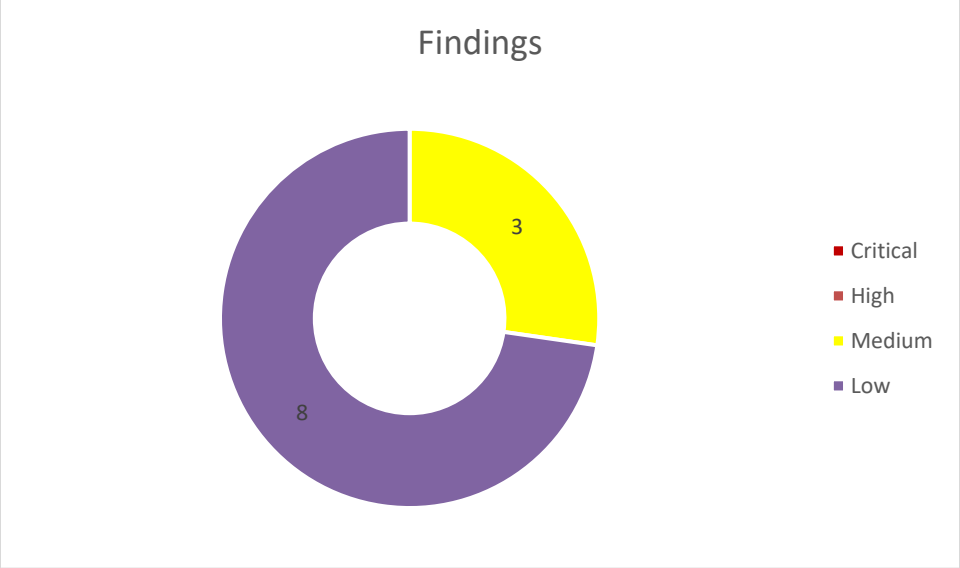

Figure 1. Findings chart

Severity	# of Findings
CRITICAL	0
HIGH	0
MEDIUM	3
LOW	8

2. FINDINGS REPORT

2.1 Critical

Not found

2.2 High

Not found

2.3 Medium

2.3.1 EVM Compatibility

Status: Fixed

Description

Your hardhat configuration does not specify the EVM's version. By default, hardhat sets the *Paris* EVM version. Your Lumia chain supports a different version (*Shanghai*), this may lead to significant problems in your contracts and unexpected behavior, because the OP codes and precompiles have different behavior in different EVM versions.

Remediation

Specify the EVM version in the hardhat configuration.

```
module.exports = {
  solidity: {
    version: "0.8.24",
    settings: {
      evmVersion: "shanghai",
    },
  },
};
```

An example of differences in different EVM versions can be found here [1].

2.3.2 Current code cannot be compiled

Status: Fixed

Description

The current attempt to compile leads to an error, because the contract uses an old version of the Uniswap library and should be updated.

```
TypeError: Explicit type conversion not allowed from "address" to "uint256".
  --> @uniswap/lib/contracts/libraries/AddressStringUtil.sol:11:27:
   |
11 |         uint256 addrNum = uint256(addr);
   |                           ^^^^^^^^^^^^

Error HH600: Compilation failed
```

Remediation

Replace the library dependency with a new version.

```
- “@uniswap/lib": "^2.1.0”,
+ “@uniswap/lib": "5.0.0-alpha”,
```

2.3.3 Insufficient Validation of Reward Tokens Array

Status: Fixed

Description

In the contract’s constructor (MasterChefV3), the reward tokens are initialized with the following code:

```
rewardTokensLength = _rewardTokens.length;
require(_rewardTokens.length <= MAX_REWARD_TOKENS_NUMBER, "Reward tokens number gt max");
```

There are two validation issues here:

1. Empty Array Check: There is no requirement ensuring that _rewardTokens.length > 0. As a result, it would be possible to deploy the contract with an empty reward tokens array,

which might lead to unintended behavior in functions that assume at least one reward token exists.

2. Uniqueness of Token Addresses: There is no check to ensure that all provided token addresses are unique. A deployer could provide an array filled with the same token address, and the contract would accept it without complaints.

Remediation

There are two validation issues here:

1. Add a check to ensure that the reward tokens array is not empty

```
require(_rewardTokens.length > 0, "Empty reward tokens");
```

2. Enforce Uniqueness (Optional). If it is neccesary that each reward token is unique, iterate over the _rewardTokens array and build a mapping to detect duplicates

```
mapping(address => bool) seen;
...
for (uint256 i = 0; i < _rewardTokens.length; i++) {
require(!seen[_rewardTokens[i]], "Duplicate reward token");
seen[_rewardTokens[i]] = true;
}
```

2.4 Low

2.4.1 Deprecated OpenZeppelin Import for IERC20Permit

Status: Ackowledged

Description

Your contract imports the draft version of the IERC20Permit interface from OpenZeppelin using:

```
import '@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol';
```

This draft version has been deprecated and removed in newer releases of OpenZeppelin. Using the old library significantly increases the risk. The interface definition for permit functionality (EIP-2612) has been updated or finalized in later versions, and relying on the outdated draft version will

lead to discrepancies between your contract's expectations and the behavior of tokens implementing the final standard.

Remediation

Replace the deprecated import with the newest version.

```
import '@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol';
```

2.4.2 Uninitialized Return Variables in checkUpkeep

Status: Fixed

Description

In the *checkUpkeep* function, the return variables (*upkeepNeeded* and the unnamed bytes variable) are not explicitly assigned on all code paths. In Solidity, if a return variable isn't explicitly set, it defaults to its zero value (i.e., false for booleans and an empty byte array for bytes). While this behavior isn't a security vulnerability at the moment, it may lead to subtle bugs when the code is changed.

```
function checkUpkeep(bytes calldata) external view override  returns (bool upkeepNeeded, bytes
memory)
```

Remediation

Explicitly assign values to all return variables before exiting the function.

2.4.3 Missing Index Bounds Check in getRewardTokenByIndex

Status: Fixed

Description

The function *getRewardTokenByIndex* is used to retrieve a reward token from the *rewardTokens* array based on the provided index. However, unlike the *setRewardsPerSecond* function, it lacks an explicit index bounds check. While an out-of-bound array access will revert in Solidity, adding an explicit check improves clarity and provides a consistent error message.

Remediation

Add a check at the beginning of the function, similar to setRewardsPerSecond:

```
function getRewardTokenByIndex(uint256 _index) external view returns (IERC20) {
    if (_index > rewardTokensLength - 1)
        revert IndexNotCorrect(_index);
    return rewardTokens[_index];
}
```

2.4.4 Unused Parameter in verifyCallback

Status: Fixed

Description

The function verifyCallback includes a parameter of type address that is never used in its body.

```
function verifyCallback(
address,
address tokenA,
address tokenB,
uint24 fee ) internal view returns (IMorphexV3Pool pool) {
    pool = getPool(tokenA, tokenB, fee); require(msg.sender == address(pool));
}
```

Remediation

Remove the unused parameter if it is not required.

2.4.5 Redundant code

Status: Acknowledged

Description

In the original version of Pancake, the `feeProtocol` used to depend on the `fee` parameter passed into MorphexV3Pool in `initialize` function. Since your fee is now constant, you don't need the following code.

```
        if (fee == 100) {
            slot0.feeProtocol = 131074000; // value for 2000:2000, store 2 uint32 inside
        } else if (fee == 500) {
            slot0.feeProtocol = 131074000; // value for 2000:2000, store 2 uint32 inside
        } else if (fee == 2500) {
            slot0.feeProtocol = 131074000; // value for 2000:2000, store 2 uint32 inside
        } else if (fee == 10000) {
            slot0.feeProtocol = 131074000; // value for 2000:2000, store 2 uint32 inside
        }
```

Remediation

Remove the unused code.

2.4.6 Protocol-based values should be constants

Status: Fixed

Description

In several places the following code uses a protocol-based value 10 (the number of rewards). For clairty, it should be used as a public constant everywhere, this number is used, instead of literal numbers.

```
uint256[10] public rewardsGrowthGlobalX128;
...
for (uint256 i; i < 10; i++) {
...
```

Remediation

Replace 10 with a public constant

```
 uint256 public constant MAX_REWARD_TOKENS_NUMBER = 10;
```

2.4.7 Using a deprecated interface

Status: Acknowledged

Description

The KeeperCompatibleInterface is deprecated. There is no need to use it.

```

```

```
import {AutomationCompatibleInterface as KeeperCompatibleInterface} from "./AutomationCompatibleIn-
terface.sol";
```

Also, we recommend not downloading libraries and using them locally, but instead using their latest versions. There are many issues found in the old libraries.

Remediation

Use the AutomationCompatible interface directly.

```
uint256 public constant MAX_REWARD_TOKENS_NUMBER = 10;
```

2.4.8 Potential failures with overflows

Status: Fixed

Description

There are 2 issues with calculations:

1. The following block of code performs the multiplication of two uint256 values. In extreme cases, this multiplication may overflow a 256-bit integer even if the final result (after division by Q128) should fit in a uint256

```
rewards[i] = (rewardGrowthInsideDelta * positionInfo.boostLiquidity) / Q128;
```

2. In places where you convert a uint32 into an int32, the result may be a negative number. Because the uint32 range doesn't fit into the int32 range (0 to 4,294,967,295 doesn't fit into −2147483648 to 2147483647).

```
if (tickCumulativesDelta < 0 && (tickCumulativesDelta % int32(secondsAgo) != 0)) arithmeticMeanTick--;
```

Although, we have not found where this calculation is critical (now), underestimating such things in the future will lead to critical vulnerabilities.

Remediation

1. Replace the multiplication and division with a call to FullMath.mulDiv [3] as follows:

```
rewards[i] = FullMath.mulDiv(rewardGrowthInsideDelta, positionInfo.boostLiquidity, Q128);
```

2. When you convert a uint32 into an integer, the best way is to cast secondsAgo to a signed type that can represent all possible uint32 values. Since int256 has a very wide range, you can do:

```
arithmeticMeanTick = int24(tickCumulativesDelta / int256(uint256(secondsAgo));
```

This ensures that no matter how big secondsAgo is (up to the maximum uint32 value), the cast won't overflow because a uint32 maximum (≈4.29e9) easily fits within an int256.

CONSCLUSION

The MorphexV3 protocol delivers key decentralized exchange functionalities—including swapping, yield farming, and staking using concentrated liquidity. We have not found any critical vulnerabilities; however the audit has uncovered several configuration, compilation, and validation issues that may affect the protocol in the future.

Medium severity concerns include an unspecified EVM version in the Hardhat configuration (defaulting to Paris instead of the target Shanghai version), and an outdated Uniswap library dependency that leads to compilation errors, both of which may cause unexpected behavior in

production. Additionally, insufficient validation of the reward tokens array—such as missing checks for non-emptiness and uniqueness increases the risk of misconfiguration and exploitation.

On the lower severity side, deprecated interfaces (e.g., the draft IERC20Permit and KeeperCompatibleInterface), uninitialized return variables, missing index bounds checks, and potential arithmetic overflows in reward calculations have been identified.

Addressing these issues by updating dependencies, enforcing robust validations, and adopting safer arithmetic practices will significantly enhance the protocol's overall reliability and security.

REFERENCES

[1] https://docs.linea.build/get-started/build/ethereum-differences

[2] https://hardhat.org/hardhat-runner/docs/config#default-evm-version

[3] https://docs.uniswap.org/contracts/v3/reference/core/libraries/FullMath