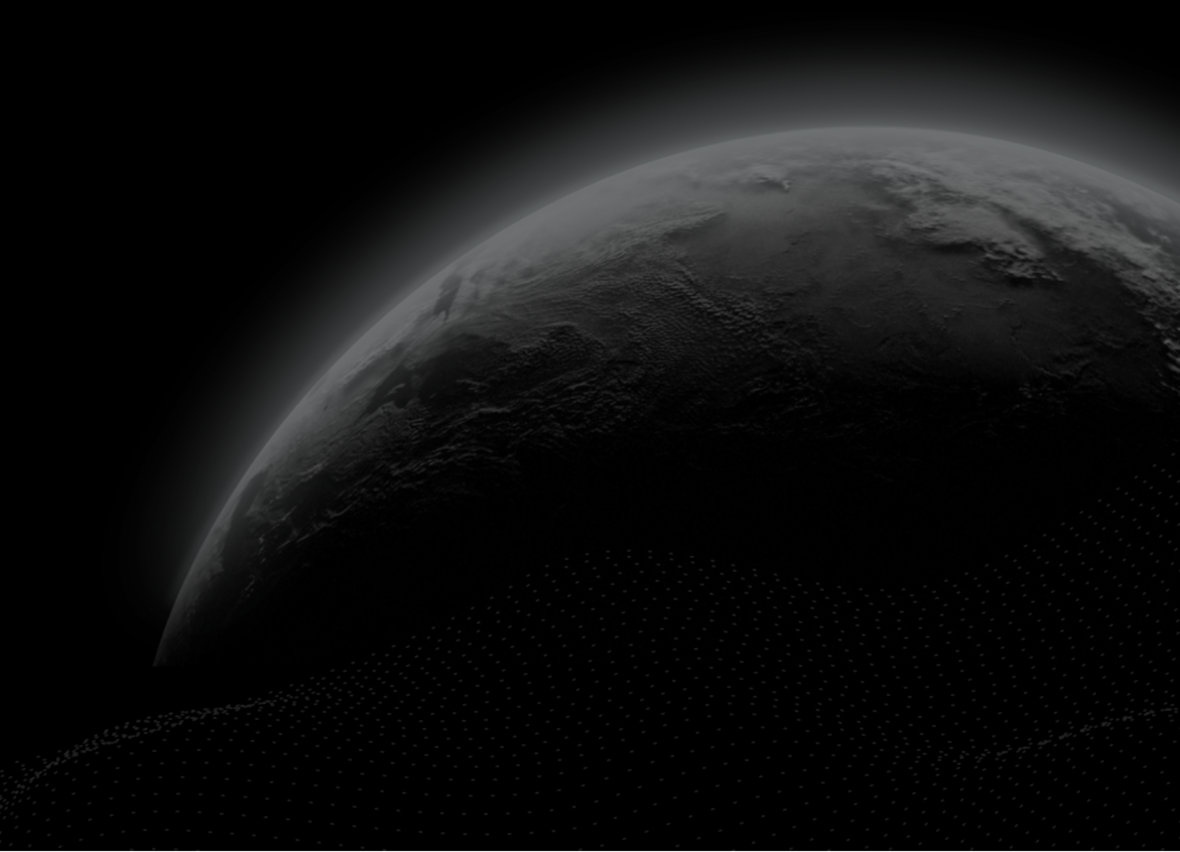
Security Assessment

Mirolab - Audit

CertiK Assessed on Aug 25th, 2024



CertiK Assessed on Aug 25th, 2024

Mirolab - Audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

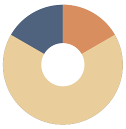
Executive Summary

TYPES	ECOSYSTEM	METHODS
Exchange	Mirolab WEB 3.0	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 08/25/2024	N/A

CODEBASE	COMMITS
https://github.com/mirolabgroup/mainnet-contracts/	3fe685b94654cebe96cc17e6dac4cc8fc7b6f82d bfcbb018b1add466804163dc6e72e9c9eed8628b

Vulnerability Summary



6	1	1	0	4	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

🟥 0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
🟧 1	Major	1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
🟨 0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨 4	Minor	1 Resolved, 3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
🟦 1	Informational	1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | MIROLAB - AUDIT

CODEBASE | MIROLAB - AUDIT

Repository

https://github.com/mirolabgroup/mainnet-contracts/contracts/core

Commit

3fe685b94654cebe96cc17e6dac4cc8fc7b6f82c
bfcbb018b1add466804163dc6e72e9c9eed8628d

AUDIT SCOPE | MIROLAB - AUDIT

6 files audited ● 6 files without findings

ID	Repo	File	SHA256 Checksum
● MHZ	mirolabgroup/mainnet- - contracts	contracts/core/MetadataHelper.sol	cfb667415ef7bda4df885c43f6c3cf29a827cdaff348d6586126547c37ae1697
● MLF	mirolabgroup/mainnet-contracts	contracts/core/MLFactory.sol	94d412cb84dc2768bce2dc37cfb1da5aaa3e4b2b37ba64237e8519d7943493bc
● MLL	mirolabgroup/mainnet-contracts	contracts/core/MLLibrary.sol	d14adc072ad57dec786e41eb12e127ccd3a7a796ecc9df42207ce1a1c1de9c0d
● MLP	mirolabgroup/mainnet-contracts	contracts/core/MLPair.sol	c707eefbdb93f3372b193e3476bdd13604abc8cbde6cb1996d0849d54bb978c4
● MLR	mirolabgroup/mainnet-contracts	contracts/core/MLRouter.sol	844e56c111f5fb6463b3fb27f7d24c26b682ec8519fe2915d7d9b307aa479567
● MLI	mirolabgroup/mainnet-contracts	contracts/core/MLRouterInternal.sol	6f01cfa9be93739e53019295ba4ab6e748ab43cbfbabeb07de1f36b0319e662b

APPROACH & METHODS | MIROLAB - AUDIT

This report has been prepared for Mirolab to discover issues and vulnerabilities in the source code of the Mirolab - Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

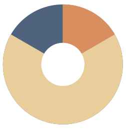
The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | MIROLAB - AUDIT



6	0	1	0	4	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Mirolab - Audit. Through this audit, we have uncovered 6 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
COR -01	**Centralization Related Risks**	**Centralization**	**Major**	● **Mitigated**
COR-02	Missing Zero Address Validation	Volatile Code	Minor	● Acknowledged
MLF-01	Lack Of Reasonable Limit	Logical Issue	Minor	● Acknowledged
MLP-04	Unsafe Integer Cast	Incorrect Calculation	Minor	● Acknowledged
MLR-02	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	● Resolved
MLZ-01	`indexedPairs` Not Update When Users Remove Liquidity	Logical Issue	Informational	● Acknowledged

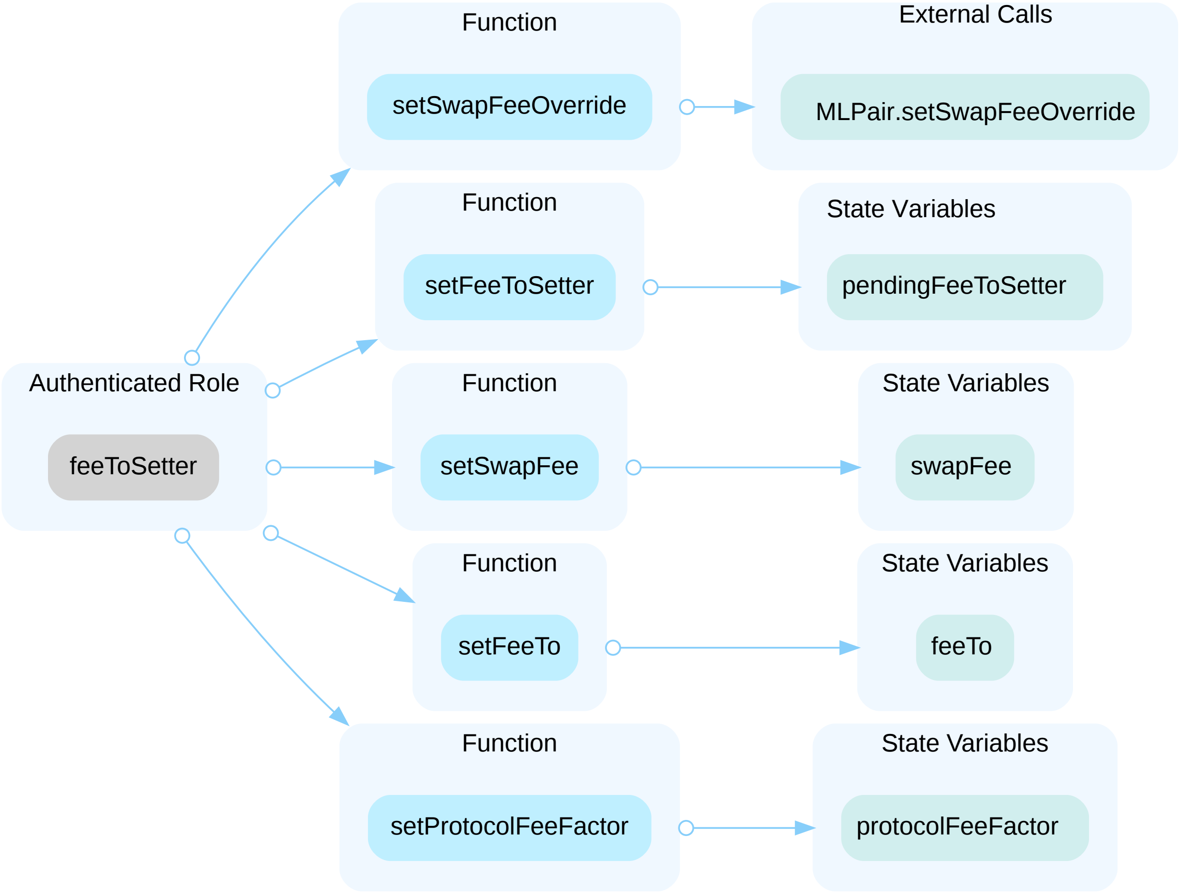
COR-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	● Major	MLFactory.sol (3fe68 - 11/30); MLPair.sol (3fe68 - 11/30)	● Mitigated

Description

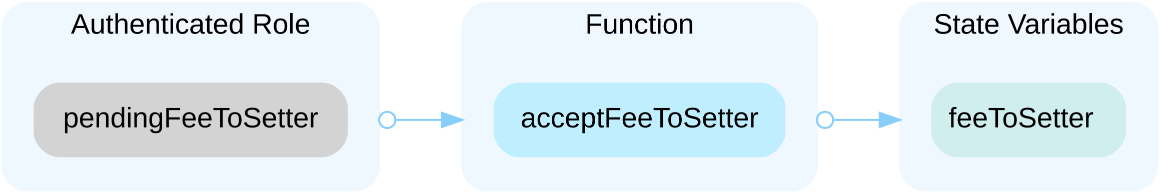
In the contract `MLFactory` the role `feeToSetter` has authority over the functions shown in the diagram below. Any compromise to the `feeToSetter` account may allow the hacker to take advantage of this authority.

- set the address of `feeTo`
- set `swapFee`
- set protocol fee factor
- set the address of `pendingFeeToSetter`, who can accept the `feeToSetter` role
- set swap fee point override for a pair

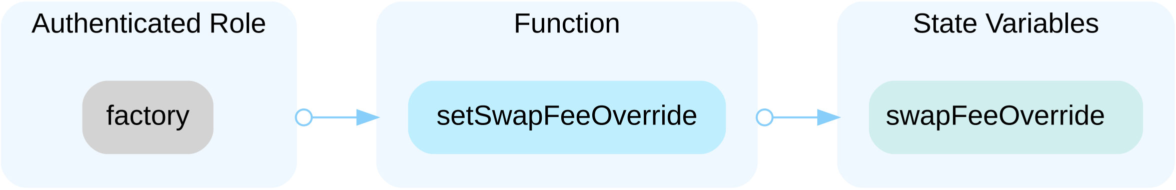


In the contract `MLFactory` the role `pendingFeeToSetter` has authority over the functions shown in the diagram below.

- accept the `feeToSetter` role



In the contract `MLPair` the role `factory` has authority over the functions shown in the diagram below.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. In general, we strongly recommend centralized privileges or roles in the protocol be improved

via a decentralized mechanism or smart-contract- based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible

suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove this functionality.

Alleviation

[MIROLAB FINANCE TEAM 08/20/2024]:

Considering that our DAO is currently in the process of maturing, we have opted for the Short-Term solution

- Time-lock with reasonable latency: We have instituted a time lock to allow for awareness of privileged operations.
- Privileged roles assigned to multi-signature wallets: To mitigate the risk of a single point of failure resulting from compromised private keys, we have assigned privileged roles to multi-signature wallets.
- All relevant information has been publicly disclosed in our documentation.

The privileged roles of the factory, feeToSetter and pendingFeeToSetter, have been transferred to the

timelock.

Signer 1: mlsync:0xe9D5791Be827F092109C41F5eBFD48FF66d21b92

Signer 2: mlsync:0x67cd008DB78a667A8983e8196F2a2C7D38bD6744

Signer 3: mlsync:0xA74A66219a08D6346c512c50a5d0648a65a9183c

Signer 4: mlsync:0x4700347E98C9c8A0c63a865575dFf34088C473d2

Signer 5: mlsync:0x13BD7a61b46950fF0e9b41571Dc4C503eE854041

It requires 3 out of 5 signers to sign the transaction to execute.

COR-02 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	MLFactory.sol (3fe68 - 11/30): 34, 69, 86; MLPair.sol (3fe68 - 11/30): 53, 54; MLRouter.sol (3fe68 - 11/30): 19, 20	● Acknowledged

Description

Addresses are not validated before assignment or external calls, potentially allowing the use of zero addresses and leading to unexpected behavior or vulnerabilities. For example, transferring tokens to a zero address can result in a permanent loss of those tokens.

```
34          feeToSetter = _feeToSetter;
```

- `_feeToSetter` is not zero-checked before being used.

```
69          feeTo = _feeTo;
```

- `_feeTo` is not zero-checked before being used.

```
86          pendingFeeToSetter = _feeToSetter;
```

- `_feeToSetter` is not zero-checked before being used.

```
53          token0 = _token0;
```

- `_token0` is not zero-checked before being used.

```
54          token1 = _token1;
```

- `_token1` is not zero-checked before being used.

```
19          factory = _factory;
```

- `_factory` is not zero-checked before being used.

```
20          WETH = _WETH;
```

- `_WETH` is not zero-checked before being used.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[MIROLAB FINANCE TEAM 08/20/2024]

We thank Certik for identifying these volatile codes. After a thorough investigation of this issue, we found that:

- MLFactory.sol: 34 → This volatile code was used only once during the initial deployment of the contract. Consequently, it does not pose any risks, considering that our core contracts have been deployed and used for several months. Essentially, this does not impact the safety of the contracts or user funds.

- MLFactory.sol: 69 → The volatile code at line 69 can only be executed by the FeeToSetter address, currently set as the timelock controller under the multisig wallet. The likelihood of setting the zero-address as feeTo is very low. Even if such an event occurs, the FeeToSetter can easily rectify this mistake without causing any issues to the operations of the other involved contracts or risking user funds. Essentially, this does not affect the safety of the contracts or user funds.

- MLFactory.sol: 86 → Similar to the issue mentioned at line 69, this volatile code can only be executed by the FeeToSetter address. The probability of setting the zero-address as FeeToSetter is minimal. In the rare event of such a mistake, it will not compromise user funds or disrupt the operations of the other involved contracts. The only consequence is the inability to set the swap fee, equivalent to the feeToSetter role renouncement. Essentially, this does not pose a risk to the safety of the contracts or user funds.

- MLPair.sol: 53, 54 → These lines of code are within the constructor function and are used only once by the MLFactory to create and initialize the pair. At that moment, there is no existing liquidity in this pool as it is being created. Therefore, it does not introduce any risk to the safety of the contracts or user funds.

- MLRouter.sol: 19, 20 → Similarly, these two lines of code are within the constructor function of the MLRouter contract. Essentially, this does not impact the safety of the contracts or user funds.

MLF-01 | LACK OF REASONABLE LIMIT

Category	Severity	Location	Status
Logical Issue	● Minor	MLFactory.sol (3fe68 - 11/30): 80	● Acknowledged

Description

The `setProtocolFeeFactor()` function allows the `feeToSetter` to set the minimum `protocolFeeFactor` as 2, which means half of the fee will be charged and sent to the `_feeTo`.

```
    function _getFeeLiquidity(uint _totalSupply, uint _rootK2, uint _rootK1, uint8
_feeFactor) private pure returns (uint) {
        uint numerator = _totalSupply * (_rootK2 - _rootK1);
        uint denominator = (_feeFactor - 1) * _rootK2 + _rootK1;
        return numerator / denominator;
    }
```

Recommendation

We would like to confirm with the client whether the current implemenation aligns with the project design.

Alleviation

[MIROLAB FINANCE TEAM 08/20/2024]

We hereby confirm that the current implementation aligns with our project design.

MLP-04 | UNSAFE INTEGER CAST

Category	Severity	Location	Status
Incorrect Calculation	● Minor	MLPair.sol (3fe68 - 11/30): 109, 110, 118, 119	● Acknowledged

Description

Type casting refers to changing an variable of one data type into another. The code contains an unsafe cast between integer types, which may result in unexpected truncation or sign flipping of the value.

```
109                 principal0: uint112(liquidity * _reserve0 / _totalSupply),
```

Casted expression `liquidity * _reserve0 / _totalSupply` has estimated range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935] but target type `uint112` has range [0, 5192296858534827628530496329220095].

```
110                 principal1: uint112(liquidity * _reserve1 / _totalSupply),
```

Casted expression `liquidity * _reserve1 / _totalSupply` has estimated range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935] but target type `uint112` has range [0, 5192296858534827628530496329220095].

```
118                 principal0: uint112(liquidity * _reserve0 / _totalSupply),
```

Casted expression `liquidity_scope_0 * _reserve0 / _totalSupply` has estimated range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935] but target type `uint112` has range [0, 5192296858534827628530496329220095].

```
119                 principal1: uint112(liquidity * _reserve1 / _totalSupply),
```

Casted expression `liquidity_scope_0 * _reserve1 / _totalSupply` has estimated range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935] but target type `uint112` has range [0, 5192296858534827628530496329220095].

Recommendation

It is recommended to check the bounds of integer values before casting. Alternatively, consider using the `SafeCast` library from OpenZeppelin to perform safe type casting and prevent undesired behavior.

Reference: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/cf86fd9962701396457e50ab0d6cc78aa29a5ebc/contracts/utils/math/SafeCast.sol

Alleviation

[MIROLAB FINANCE TEAM 08/20/2024]

MLPair.sol: 109, 110, 118, 119 → In reality, for an unexpected truncation to occur due to these unsafe integer castings, the total liquidity of a pool must reach an unrealistic value.

- To demonstrate this, let’s consider ETH-USDC pool of our DEX at the address:

  currently valued at 557K USD, the total supply of LP token is _totalSupply=5041968077308680, corresponding _reserve0=279126019242, _reserve1=118709123971826255802 (all these values are readable on chain). Thus, the maximum value of the variable principal0=279126019242 and maximum value of principal1=118709123971826255802. These two value is significantly below the limit of uint112 type, which is 5192296858534827628530496329220095 or 5.2*10^33)
- Additionally, for this unsafe integer cast issue to occur, the variables _reserve0 and _reserve1 need to a reach a minimum amount of 5.2*10^15 tokens in a liquidity pool, assuming that this token has 18 decimals. This number is unreasonably large for normal tokens.
- Hence, we think that the conversions to unint112 in MLPair.sol: 109, 110, 118, 119 won’t cause any issues in reality.

MLR-02 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	● Minor	MLRouterInternal.sol (3fe68 - 11/30): 129	● Resolved

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

```
129         IMLPair(pair).transferFrom(msg.sender, pair, liquidity);
```

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[MIROLAB FINANCE TEAM 08/20/2024]

Issue acknowledged. The IMLPair utilizes the transfer and transferFrom functions from the ERC20.sol contract (located within the subfolder libraries/token/ERC20.sol). It's important to note that the transfer and transferFrom functions within this ERC20.sol always either return true or throw an error. Consequently, this does not pose an issue.

MLZ-01 | `indexedPairs` NOT UPDATE WHEN USERS REMOVE LIQUIDITY

Category	Severity	Location	Status
Logical Issue	● Informational	MLRouter.sol (3fe68 - 11/30): 134	● Acknowledged

Description

We note that the variable `indexedPairs` is used to keep track of users who add liquidity, but does not remove the user from the variable `indexedPairs` after the user removes liquidity completely.

Recommendation

We would like to confirm with the client if the current implementation aligns with the original project design.

Alleviation

The team acknowledged this issue and they will leave it as it is for now.

APPENDIX | MIROLAB - AUDIT

Finding Categories

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK 's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.