MDEX

AUDIT REPORT

Version 1.0.0

Serial No. 20210112000012012

Presented by Fairyproof

Jan 12, 2021

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the MDEX project.

Audit Start Time:

Dec 25, 2020

Audit End Time:

Jan 12, 2021

Project Token's Name:

Mdex

Audited Code's Github Repository:

https://github.com/mdexSwap/contracts

Audited Code's Github Commit Number When Audit Started:

f7590ea34540179c946f37e1d71a0ece8e9520e8

Audited Code's Github Commit Number When Audit Ended:

90ab23fbd268610ddfca9a706fc6615d4a1f6735

The goal of this audit is to review MDEX's solidity implementation for its DEX application, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the MDEX team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

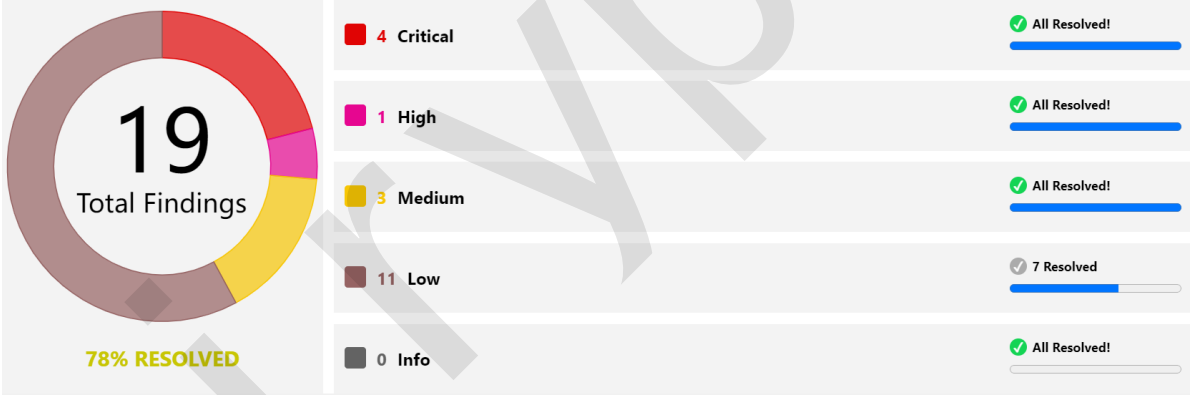
For this audit, we used the following sources of truth about how the DEX application should work:

https://mdex.com

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the MDEX team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Result
2021011200012012	Fairyproof Security Team	2020.12.25 - 2021.01.12	Low



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, 4 risks of critical-severity, 1 risk of high-severity, 3 risks of medium-severity and 11 risks of low-severity were uncovered. 4 risks of critical-severity, 1 risk of high-severity, 3 risks of medium-severity and 7 risks of low-severity have been fixed, 4 risk of low-severity have been confirmed.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Major functions of audited code

The audited code mainly implements a DEX application.

04. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- Replay Attack
- Reordering Attack
- Miner's Advantage
- Rollback Attack
- DDoS Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Fake Deposit
- Shadow Variable
- Design Vulnerability
- Token Issuance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration

- Code Improvement
- Misc

05. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

06. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Use of Inappropriate Price Feeds	Design Vulnerability	Critical	✓ Fixed
FP-2	Incorrect Algorithm	Design Vulnerability	Critical	✓ Fixed
FP-3	Edge Case Error	Design Vulnerability	Critical	✓ Fixed
FP-4	Inappropriate Minter Setting	Design Vulnerability	Critical	✓ Fixed
FP-5	Inappropriate Setting	Design Vulnerability	High	✓ Fixed
FP-6	Missing `require` Check	Implementation Vulnerability	Medium	✓ Fixed
FP-7	Missing `require` Check	Implementation Vulnerability	Medium	✓ Fixed
FP-8	Missing `require` Check	Implementation Vulnerability	Medium	✓ Fixed
FP-9	Deprecated Usage	Code Improvement	Low	✓ Fixed
FP-10	Deprecated Usage	Code Improvement	Low	✓ Fixed
FP-11	Inappropriately Setting Variable	Implementation Vulnerability	Low	✓ Fixed
FP-12	Inappropriate Variable Naming	Code Improvement	Low	✓ Fixed
FP-13	Unnecessary Function	Code Improvement	Low	✓ Fixed
FP-14	Inappropriate Function Naming	Code Improvement	Low	✓ Fixed
FP-15	Inappropriate Variable Naming	Code Improvement	Low	✓ Fixed
FP-16	Inappropriate Assumption	Design Vulnerability	Low	Confirmed
FP-17	Incorrect Use of Price Feeds	Design Vulnerability	Low	Confirmed
FP-18	Repeated Use of Number Literal	Design Vulnerability	Low	Confirmed
FP-19	Repeated Use of Number Literal	Design Vulnerability	Low	Confirmed

07. Issue descriptions

[FP-1] Use of Inappropriate Price Feeds Critical ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In line 326 of `SwapMining.sol`, the code section was as follows:

```
price = IMdexPair(IMdexFactory(factory).getPair(token,
anchorToken)).price(token, baseDecimal);
```

And in lines 333 - 334, the code section was as follows:

```
uint256 price0 = IMdexPair(IMdexFactory(factory).getPair(token,
base)).price(token, baseDecimal);
uint256 price1 = IMdexPair(IMdexFactory(factory).getPair(base,
anchorToken)).price(base, decimal);
```

These code sections used the function `price` in `Factory.sol` as price feeds. However the price returned by the `price` function could be easily manipulated.

Recommendation:

Considering changing the implementation of the function `price` in `Factory.sol` .

Update/Status:

It has been fixed by the MDEX team.

[FP-2] Incorrect Algorithm Critical ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In line 254 of `SwapMining.sol`, the code section was as follows:

`uint256 quantity = price.mul(amount).div(10 ** uint256(IERC20(targetToken).decimals()));`

The algorithm was incorrect.

Recommendation:

Consider changing `targetToken` to `output`. The recommended change is as follows:

`uint256 quantity = price.mul(amount).div(10 ** uint256(IERC20(output).decimals()));`

Update/Status:

It has been fixed by the MDEX team.

[FP-3] Edge Case Error Critical ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In line 139 of `CoinChef.sol`, in the implementation of the `updatePool` function, when `block.number > endBlock`, execution of `updatePool` would set `pool.lastRewardBlock` to the current block's `block.number` in line 165 as follows:

`pool.lastRewardBlock = block.number;`

And this would cause `updatePool` to fail in lines 158 - 159 shown as follows in all subsequent calls:

```
uint256 number = block.number > endBlock ? endBlock : block.number;
uint256 multiplier = number.sub(pool.lastRewardBlock);
```

Recommendation:

Consider changing the implementation of `updatePool`, the recommended change is as follows:

```
// Update reward variables of the given pool to be up-to-date.
function updatePool(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    uint256 number = block.number > endBlock ? endBlock : block.number;
    if (number <= pool.lastRewardBlock) {
        return;
    }
    uint256 lpSupply;
    if (isSushiLP(address(pool.lpToken))) {
        if (pool.totalAmount == 0) {
            pool.lastRewardBlock = number;
            return;
        }
        lpSupply = pool.totalAmount;
    } else {
        lpSupply = pool.lpToken.balanceOf(address(this)); //1000000000000000000
        if (lpSupply == 0) {
            pool.lastRewardBlock = number;
            return;
        }
    }
    uint256 multiplier = number.sub(pool.lastRewardBlock);
    uint256 mdxReward =
multiplier.mul(mdxPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
    bool minRet = mdx.mint(address(this), mdxReward);
    if (minRet) {
        pool.accMdxPerShare =
pool.accMdxPerShare.add(mdxReward.mul(1e12).div(lpSupply));
    }
    pool.lastRewardBlock = number;
}
```

Update/Status:

It has been fixed by the MDEX team.

[FP-4] Inappropriate Minter Setting Critical ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In `MdxToken.sol`, the function `setMinter` had a modifier `onlyOwner` which only allowed the owner to call this function. Since the `MdxToken.sol` contract was `Ownable`, the owner himself/hersef or an attacker who compromised the owner right could exploit the contract by changing `minter` to mint tokens at will.

Recommendation:

Consider changing the `minter` setting and removing `Ownable` to reduce the attack surface. The recommended change is as follows:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MdxToken is ERC20("MDX Token", "MDX") {
    uint256 private constant maxSupply = 30000000 * 1e18;     // the total
supply
    address public minter;

    // mint with max supply
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool)
{
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    // set minter only once
    function setMinter(address _newMinter) external {
        require(minter == address(0), "has set up");
        require(_newMinter != address(0), "is zero address");
        minter = _newMinter;
    }

    // modifier for mint function
    modifier onlyMinter() {
        require(msg.sender == minter, "caller is not the minter");
        _;
    }
}
```

Update/Status:

It has been fixed by the MDEX team.

[FP-5] Inappropriate Setting High ✓ Fixed

Issue/Risk: Design Vulnerability

Description:

In line 151 of `SwapMining.sol`, the function `setTargetToken` changed the target token that was used as the unit of token quantity. When this function changed the target token it might change the existing `quantity` of a token that used this target token as the unit. Affected lines included lines 258 and 261 as follows:

```
user.quantity = user.quantity.add(quantity);
```

Recommendation:

Consider removing the function `setTargetToken` and using a uniform token as the target token.

Update:

the function `setTargetToken` has been removed.

Status:

It has been fixed by the MDEX team.

[FP-6] Missing `require` Check Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In line 77 of `SwapMining.sol`, the implementation of the function `addPair` needed a `require` check for the `_pair` input parameter. If `_pair` was set to `address(0)` by the caller of the function `addPair`, execution of the function `massMintPools` would fail.

Recommendation:

Consider adding the following statement before the conditional check `if (_withUpdate) {`:

`require(_pair != address(0), "_pair is the zero address");`。

Update/Status:

It has been fixed by the MDEX team.

[FP-7] Missing `require` Check Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In line 149 of `HecoPool.sol`, the function `add` needed a "require" check for the `_lpToken` input parameter. If `_lpToken` was set to `address(0)` by the caller of the function `add`, execution of the function `massUpdatePools` would fail.

Recommendation:

Consider adding the following statement before the conditional check `if (_withUpdate) {`:

`require(address(_lpToken) != address(0), "_lpToken is the zero address");`。

Update: the MDEX team added a `require` check.

Status:

It has been fixed by the MDEX team.

[FP-8] Missing `require` Check

Medium ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In line 98 of `CoinChef.sol`, the implementation of the function `add` needed a "require" check for the `_lpToken` input parameter. If `_lpToken` was set to `address(0)` by the caller of the function `add`, execution of the function `massUpdatePools` would fail.

Recommendation:

Consider adding the following statement before the conditional check `if (_withUpdate) {`:

`require(address(_lpToken) != address(0), "_lpToken is the zero address");`。

Update:

the MDEX team added a `require` check for `address(_lpToken) != address(0)`.

Status:

It has been fixed by the MDEX team.

[FP-9] Deprecated Usage

Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In line 199 of `GovernorAlpha.sol`, the usage of `.value(...)` was deprecated. Here was the code section:

```
timelock.executeTransaction.value(proposal.values[i])(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], proposal.eta);
```

Recommendation:

Consider using `{value: ...}` instead. The recommended change is as follows:

```
timelock.executeTransaction{value:proposal.values[i]}(proposal.targets[i], proposal.values[i], proposal.signatures[i], proposal.calldatas[i], proposal.eta);
```

Update/Status: it has been fixed by the MDEX team.

[FP-10] Deprecated Usage

Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In line 99 of `GovernorAlpha.sol`, the code section was as follows:

```
(bool success, bytes memory returnData) = target.call.value(value)(callData);
```

The usage of `call.value(value)(callData)` was deprecated.

Compiler warnings were generated.

Recommendation:

Consider changing

```
(bool success, bytes memory returnData) = target.call.value(value)(callData);
```

to

```
(bool success, bytes memory returnData) = target.call{value:value}(callData);
```

Update/Status:

It has been fixed by the MDEX team.

[FP-11] Inappropriately Setting Variable Low ✓ Fixed

Issue/Risk: Implementation Vulnerability

Description:

In `Factory.sol`, two functions were used to retrieve and set the value of `initCodeHash` respectively. Only the `feeToSetter` was allowed to set the value and only set it once. If the value was inappropriately set there would be no chance to reset it. This could cause potential risks.

Recommendation:

Consider setting the value in the constructor by adding the following statement in the constructor:

```
initCodeHash = keccak256(abi.encodePacked(type(MdexPair).creationCode));
```

and commenting out some of the lines that set the value in other functions of this contract and the `IMdexFactory.sol` contract.

The recommened code changes are as follows:

```
constructor(address _feeToSetter) public {
      feeToSetter = _feeToSetter;
    initCodeHash = keccak256(abi.encodePacked(type(MdexPair).creationCode));
}
```

and consider commenting out the following lines as well:

Line 330: `bool public initCode = false;`

Line 381: `setInitCodeHash`

Line 397: `getInitCodeHash`

Line 12: `function initCodeHash() external view returns (bytes32);` in the `interface/IMdexFactory.sol` contract

Line 28: `function setInitCodeHash(bytes32) external;` in the `interface/IMdexFactory.sol` contract

Update/Status:

It has been fixed by the MDEX team.

[FP-12] Inappropriate Variable Naming Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In line 319 of `SwapMining.sol`, the variables `base` , `baseDecimal` and `decimal` were inappropriately named in the function `getPrice` . The code was as follows:

```
uint256 baseDecimal = 10 ** uint256(IERC20(token).decimals());
address base = getWhitelist(index);
uint256 decimal = 10 ** uint256(IERC20(base).decimals());
```

These variables were named in a way that didn't describe their behavior.

Recommendation:

Consider renaming `baseDecimal` to `tokenDecimas` , `base` to `intermediate` and `decimal` to `interDecimal` and making changes in all places where these variables were used accordingly.

Update/Status:

It has been fixed by the MDEX team.

[FP-13] Unnecessary Function Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In line 146 of `SwapMining.sol` , the `factory` contract had state variables(data), therefore it was very unlikely the contract would be replaced by calling the function `setFactory` . And this function was unnecessary.

Recommendation:

Consider removing the function `setFactory` .

Update:

The MDEX team removed the function.

Status:

It has been fixed by the MDEX team.

[FP-14] Inappropriate Function Naming Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In line 305 of `SwapMining.sol`, the function `getPoolList` wasn't named in a way that described its behavior.

Recommendation:

Consider renaming it to `getPoolDetail`.

Update:

The MDEX team renamed `getPoolList` to `getPoolInfo`.

Status:

It has been fixed by the MDEX team.

[FP-15] Inappropriate Variable Naming Low ✓ Fixed

Issue/Risk: Code Improvement

Description:

In line 474 of `HecoPool.sol`, the function modifier `notPause()` had a variable `pause` which was not named in a way that described its behavior.

In addition, `notPause` literally means "not paused". However what it did was to set `pause == true` which meant "pause".

Recommendation:

Consider renaming `pause` to `paused`, setting "pause" to "false" and changing the implementation of the function modifier `notPause` to make it behave what it literally means.

In addition, consider changing

`require(pause == true)` to

`require(pause)` and changing

`require(pause == false)` to

`require(!false)`

Update/Status:

These have been fixed by the MDEX team.

[FP-16] Inappropriate Assumption Low Confirmed

Issue/Risk: Design Vulnerability

Description:

In line 23 of `GovernorAlpha.sol`, the code section was as follows:

```
function votingPeriod() public pure returns (uint) { return 86400; } // ~3 days in blocks (assuming 3s blocks)
```

This implementation assumed blocks on Ethereum were generated once every 3 seconds. However this assumption didn't always hold true. If the rate of block generation was much lower than that, execution of this code might not follow the expected logic.

Recommendation:

Consider changing `86400` to `17280` (assuming blocks are generated once every 15 seconds) if this implementation will be deployed on Ethereum and changing `86400` to other values accordingly if it will be deployed on other EVM compatible blockchains.

Update:

The MDEX team prefers to keep it for now since the implementation will be deployed on a non-Ethereum blockchain.

Status:

It has been confirmed by the MDEX team.

[FP-17] Incorrect Use of Price Feeds Low Confirmed

Issue/Risk: Design Vulnerability

Description:

In line 311 of `Factory.sol`, the function `price` took an instant price feed as the prices for trading two tokens. However the prices could move significantly in a single block and this could cause the prices to be manipulated by an attacker.

Recommendation:

Consider using time weighted average prices from multiple blocks. Good examples can be found in Uniswap's `ExampleSlidingWindowOracle.sol` and `ExampleOracleSimple.sol`.

Update:

The MDEX team prefers to keep it for now since the function is only used as price inquiry rather than trading. It will not cause risks.

Status:

It has been confirmed by the MDEX team.

[FP-18] Repeated Use of Number Literal Low Confirmed

Issue/Risk: Code Improvement

Description:

The number literal `1e12` in `HecoPool.sol` was repeatedly used in multiple lines in the contract.

Recommendation:

Consider defining a constant and using that constant instead of `1e12` in all the lines where `1e12` was used.

Update:

It has been acknowledged by the MDEX team. And the team prefers to keep it for now and may make a change later.

Status:

It has been confirmed by the MDEX team.

[FP-19] Repeated Use of Number Literal Low

Confirmed

Issue/Risk: Code Improvement

Description:

In `CoinChef.sol`, the number literal `1e12` was repeatedly used in multiple lines in the contract.

Recommendation:

Consider defining a constant and using that constant instead of `1e12` in all the lines where `1e12` is used.

Update: it has been acknowledged by the MDEX team. The team prefers to keep it for now and may make a change later.

Status:

It has been confirmed by the MDEX team.

08. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- N/A

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech