MDEX Audit Report

Version 1.0.0

Presented by Fairyproof

January 12th, 2021

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the mdex project, at the request of the mdex team.

The audited code can be found in the public mdex Github repository, and the version used for this report is commit

f7590ea34540179c946f37e1d71a0ece8e9520e8

The goal of this audit is to review mdex's solidity implementation for its decentralized exchange, 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.

— 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 smart contract 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. 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

Mdex's codebase 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 smart contracts.
   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 smart contracts 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 different aspects of the project: Governance, Heco and Mainnet. 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 mdex system should work:

https://mdex.com

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

— Comments from Auditee

All the bugs with critical, high and medium severities found in the mdex's codebase were fixed and the codebase passed the audit performed by the Fairyproof team.

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 smart contract systems.

03. 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.

04. List of issues by severity

A. Critical

Governance

- GovernorAlpha.sol

Compiler Errors

Type Mismatch

- Timelock.sol

Compiler Version Mismatch

Heco

- Factory.sol

Incorrect Use of Price Feeds

- SwapMining.sol

Redefinition Error

Incorrect Use of Price Feeds

Number of Input Parameters Mismatch

Incorrect Calculation

- HecoPool.sol

Undefined Variable

Mainnet

- CoinChef.sol

Edge Case Error

- MdxToken.sol

Inappropriately Setting Minter

B. High

Governance

- N/A

Heco

- SwapMining.sol

Unexpected User Experience Brought by Inappropriate Setting

Mainnet

- N/A

C. Medium

Governance

- N/A

Heco

- HecoPool.sol

Missing "Require" Check

- SwapMing.sol

Missing "Require" Check

Mainnet

- CoinChef.sol

Missing "Require" Check

D. Low

Governance

- GovernorAlpha.sol

Deprecated Usage

Inappropriate Assumption

- Timelock.sol

Deprecated Usage

Heco

- Factory.sol

Inappropriately Setting Variable

- SwapMining.sol

Inappropriate Naming of Variable

Unnecessary Functionality

Inappropriate Naming of Function

- HecoPool.sol

Inappropriate Naming of Variable

Repeated Use of Constant Number

Mainnet

- CoinChef.sol

Repeated Use of Constant Number

05. List of issues by topic

A. Governance

- GovernorAlpha.sol

Compiler Errors: Critical

Type Mismatch: Critical

Deprecated Usage: Low

Inappropriate Assumption: Low

- Timelock.sol

Compiler Version Mismatch: Critical

Deprecated Usage: Low

B. Heco

- Factory.sol

Incorrect Use of Price Feeds: Critical

Inappropriately Setting Variable: Low

- SwapMining.sol

Redefinition Error: Critical

Incorrect Use of Price Feeds: Critical

Number of Input Parameters Mismatch: Critical

Incorrect Calculation: Critical

Unexpected User Experience Brought by Inappropriate Setting: High

Missing "Require" Check: Medium

Inappropriate Naming of Variable: Low

Unnecessary Functionality: Low

Inappropriate Naming of Function: Low

- HecoPool.sol

Undefined Variable: Critical

Missing "Require" Check: Medium

Inappropriate Naming of Variable: Low

Repeated Use of Constant Number: Low

C. Mainnet

- CoinChef.sol

Edge Case Error: Critical
Missing "Require" Check: Medium

Repeated Use of Constant Number: Low

- MdxToken.sol

Inappropriately Setting Minter: Critical

06. Issue descriptions and recommendations by topic

A. Governance

- GovernorAlpha.sol

Compiler Errors: Critical

Source and Description:

Line 11:

`function quorumVotes() public pure returns (uint) { return MdxToken.totalSupply() / 25; } // 400,000 = 4% of MDX`

Line 14:

`function proposalThreshold() public pure returns (uint) { return MdxToken.totalSupply() / 100; }`

Compilation couldn't pass.

Recommendation:

Consider using `mdx` instead of `MdxToken` and changing the property of both `quorumVotes()` and `proposalThreshold()` to `view`. The recommended change is as follows:

Line 11:

`function quorumVotes() public view returns (uint) { return mdx.totalSupply() / 25; }`

Line 14:

`function proposalThreshold() public view returns (uint) { return mdx.totalSupply() / 100; }`

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended change.

Type Mismatch: Critical

Source and Description:

Line 268 - 278:

```
uint96 votes = mdx.getPriorVotes(voter, proposal.startBlock);

if (support) {
    proposal.forVotes = add256(proposal.forVotes, votes);
} else {
    proposal.againstVotes = add256(proposal.againstVotes, votes);
}

receipt.hasVoted = true;
receipt.support = support;
receipt.votes = votes;
```

In line `uint96 votes = mdx.getPriorVotes(voter, proposal.startBlock);`, the type of `mdx.getPriorVotes(voter, proposal.startBlock)` is `uint256`, while the type of `votes` is `uint96`.

These two types don't match. It is a type mismatch error and it was found in

`proposal.forVotes = add256(proposal.forVotes, votes);` and

`proposal.againstVotes = add256(proposal.againstVotes, votes);` as well.

Compilation couldn't pass.

Recommendation:

Consider changing

```
/// @notice The number of votes the voter had, which were cast
uint96 votes;
```

to:

```
/// @notice The number of votes the voter had, which were cast
uint256 votes;
```

and changing

`uint96 votes = mdx.getPriorVotes(voter, proposal.startBlock);`

to

`uint256 votes = mdx.getPriorVotes(voter, proposal.startBlock);`

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended changes.

Deprecated Usage: Low

Source and Description:

Line 199:

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

The usage of `.value(...)` is deprecated.

Compiler warnings were generated.

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: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended change.

Inappropriate Assumption: Low

Source and Description:

Line 23:

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

This function implementation assumes blocks on Ethereum are generated once every 3 seconds. However this assumption doesn't always hold ture. If the rate of block generation is much lower than that, execution of this code may 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 delopyed in a non-Ethereum blockchain.

- Timelock.sol

Compiler Version Mismatch: Critical

Source and Description:

The compiler version defined in this contract is `pragma solidity ^0.5.16;` .However the compiler version defined in `SafeMath` imported by `import "@openzeppelin/contracts/math/SafeMath.sol";` is `pragma solidity >=0.6.0 <0.8.0;`

The two versions don't match. This is a compiler version mismatch error. Compilation couldn't pass.

Recommendation:

Consider changing the compiler version defined in the `Timelock.sol` contract to

`pragma solidity ^0.6.0;` and changing line 34

`function() external payable { }`

to

`receive() external payable { }`

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735*.

Deprecated Usage: Low

Source and Description:

Line 99:

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

The usage of `call.value(value)(callData)` is 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: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended change.

B. Heco

- Factory.sol

Incorrect Use of Price Feeds: Critical

Source and Description:

In Line 311 the function `price` takes instant price feeds as the prices for trading two tokens. However the prices can 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 price feeds. It will not cause risks.

Inappropriately Setting Variable: Low

Source and Description:

Code that sets `initCodeHash`

Two functions are used to retrieve the value and set the value of `initCodeHash` respectively. And only the `feeToSetter` is allowed to set the value and only allowed to set it once. If the value is inappropriately set there is 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:

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: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended changes.

- SwapMining.sol

Redefinition Error: Critical

Source and Description:

Redefinition of IERC20 Interfaces.

Compilation couldn't pass.

Recommendation:

Consider making changes in the interface\IMdx.sol contract. The specific changes can be made in the interface\IMdx.sol contract by

changing

`import "@openzeppelin/contracts/token/ERC20/IERC20.sol";`

to

`import {IERC20 as SIERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";`

and changing

`interface IMdx is IERC20`

to

`interface IMdx is SIERC20`

The recommended changes are made in the interface\IMdx.sol contract as follows:

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

import {IERC20 as SIERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IMdx is SIERC20 {
    function mint(address to, uint256 amount) external returns (bool);
}
```

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended changes.

Incorrect Use of Price Feeds: Critical

Source and Description:

Line 326:

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

And

Line 333 - 334:

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

This error is introduced by the incorrect use of price feeds in the function `price` of the `Factory.sol` contract. This could cause the prices to be manipulated by an attacked.

Recommendation:

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

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team renaming the function `getPrice` to `getQuantity` and reimplementing its logic.

Number of Input Parameters Mismatch: Critical

Source and Description:

Line 242:

`address pair = IMdexFactory(factory).pairFor(address(factory), input, output);`

The function `pairFor()` takes three input parameters.

Compilation couldn't pass.

Recommendation:

Consider removing `address(factory)`. A recommended change is as follows:

Changing

```
address pair = IMdexFactory(factory).pairFor(address(factory), input, output);
```

to

```
address pair = IMdexFactory(factory).pairFor(input, output);
```

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended change.

Incorrect Calculation: Critical

Source and Description:

Line 254:

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

Recommendation:

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

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

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team reimplementing the function `swap`.

Unexpected User Experience Brought by Inappropriate Setting: High

Source and Description:

In line 151 the function `setTargetToken` changes the target token that is used as the unit of token quantity. When this function changes the target token it may change the existing `quantity` of a token that uses this target token as the unit. Affected lines include 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: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team removing the function `setTargetToken`

Missing "Require" Check: Medium

Source and Description:

In line 77 the implementation of the function `addPair` needs a "require" check for the `_pair` input parameter. If `_pair` is set to `address(0)` by the caller of the function `addPair`, execution of the function `massMintPools` will fail.

Recommendation:

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

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

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting the recommended change.

Inappropriate Naming of Variable: Low

Source and Description:

In line 319, the variables `base` , `baseDecimal` and `decimal` are inappropriately named in the function `getPrice` . The code is as follows:

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

These variables are named in a way that doesn'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 are used accordingly.

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team renaming the function `getPrice` to `getQuantity` and reimplementing the function.

Unnecessary Functionality: Low

Source and Description:

Line 146 the implementation of the function `setFactory` .

The `factory` contract has state variables(data) and therefore it is very unlikely the contract will be replaced by calling the function `setFactory` . And this functionality is unnecessary.

Recommendation:

Consider removing the function `setFactory` .

Update: Fixed in [90ab23fbd268610ddfca9a706fc6615d4a1f6735] by the team removing the function `setFactory`.

Inappropriate Naming of Function: Low

Source and Description:

Line 305 the function `getPoolList`.

It is named in a way that doesn't describe its behavior.

Recommendation:

Consider renaming it to `getPoolDetail`.

Update: Fixed in [90ab23fbd268610ddfca9a706fc6615d4a1f6735] by the team renaming `getPoolList` to `getPoolInfo`.

- HecoPool.sol

Undefined Variable: Critical

Source and Description:

Line 77:

```
devAddress = _devAddress;
```

Variable `devAddress` is undefined. Compilation couldn't pass.

Recommendation:

Consider removing this variable and commenting out lines 72 and 77.

Update: Fixed in [90ab23fbd268610ddfca9a706fc6615d4a1f6735] by the team removing lines 72 and 77.

Missing "Require" Check: Medium

Source and Description:

In line 149 the implementation of the function `add` needs a "require" check for the `_lpToken` input parameter. If `_lpToken` is set to `address(0)` by the caller of the function `add`, execution of the function `massUpdatePools` will fail.

Recommendation:

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

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

Update: Fixed in [90ab23fbd268610ddfca9a706fc6615d4a1f6735] by the team adding a require check for `address(_lpToken) != address(0)`.

Inappropriate Naming of Variable: Low

Source and Description:

In line 474 the function modifier `notPause()`.

The variable `pause` is named in a way that doesn't describe its behavior.

In addition, `notPause` literally means "not paused". However what it does is to set `pause == true` which means "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: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adopting partial recommended changes.

Repeated Use of Number Literal: Low

Source and Description:

The number literal `1e12` is 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: Acknowledged by the mdex team. And the team prefers to keep it for now and may make a change later.

C. Mainnet

- CoinChef.sol

Edge Case Error: Critical

Source and Description:

In line 139, implementation of the function `updatePool`.

When `block.number > endBlock`, execution of the function `updatePool` will set `pool.lastRewardBlock` to the current block's `block.number` in line 165

`pool.lastRewardBlock = block.number;`

and this will 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 the `updatePool` fucntion, 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: Fixed in 90ab23fbd268610ddfca9a706fc6615d4a1f6735 by the team adopting the recommended change.

Missing "Require" Check: Medium

Source and Description:

In line 98 the implementation of the function `add` needs a "require" check for the `_lpToken` input parameter. If `_lpToken` is set to `address(0)` by the caller of the function `add`, execution of the function `massUpdatePools` will fail.

Recommendation:

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

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

Update: Fixed in *90ab23fbd268610ddfca9a706fc6615d4a1f6735* by the team adding a require check for `address(_lpToken) != address(0)`.

Repeated Use of Number Literal: Low

Source and Description:

The number literal `1e12` is 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: Acknowledged by the mdex team. The team prefers to keep it for now and may make a change later.

- MdxToken.sol

Inappropriately Setting Minter: Critical

Source and Description:

The function `setMinter` has a modifier `onlyOwner` which only allows the owner to call this function. Since the `MdxToken.sol` contract is `Ownable`, the owner himself/hersef or an attacker who compromises 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: Fixed in 90ab23fbd268610ddfca9a706fc6615d4a1f6735 by the team adopting the recommended change.