Mendi Finance

Audit Report

Tue Oct 03 2023

contact@scalebit.xyz https://twitter.com/scalebit_

ScaleBit

Mendi Finance Audit Report

1 Executive Summary

1.1 Project Information

Description	A lending protocol build on Linea.
Type	Lending
Auditors	ScaleBit
Timeline	Mon Aug 21 2023 - Fri Sep 01 2023
Languages	Solidity
Platform	Linea
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/mendi-finance/lending-protocol https://github.com/mendi-finance/staking-protocol https://github.com/mendi-finance/mendi-token
Commits	https://github.com/mendi-finance/lending-protocol: c99b72930d478a47706026db57085892f6f1a300 https://github.com/mendi-finance/staking-protocol: c9e7cf63f13c8d28518dc8f77fc41bbaec268cba https://github.com/mendi-finance/mendi-token: cb4cf21dc56449353515dbb551f162dee6b9ae01

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
CE2	contracts/CErc20.sol	cc29cfee71b2d80b65e79dd65c6a960dc5d8e096
RDI	contracts/RewardDistributor.sol	2ea3e2fb6bc29c21ab1f8fc9752240f5f90b5fd3
CE2U	contracts/CErc20Upgradable.sol	5a621653a967d9190c15f2a5bcc0fcc3805ff02b
BLE	contracts/Lens/BasicLens.sol	0a39c8cb6725c2fd657943d3ce3dbba1ddca0b9e
EIP2NSI	contracts/EIP20NonStandardInterface.sol	b2cbe3f95e672d24661f7485ce3ec096d7ca9aca
CTI	contracts/CTokenInterfaces.sol	9d0e4c03e1056424ba9db662163d6024d066d2a4
ERE	contracts/ErrorReporter.sol	0fa8ca1cb6b7ac10f15349dc20cc661571c3585e
CTO	contracts/CToken.sol	b818824fe9bfd8825fd6b444d4d6c792b6355451
CIN	contracts/ComptrollerInterface.sol	62917396c9b7179f6a73fdd1e6cc230adbc5d58b
UNI	contracts/Unitroller.sol	720c40892523a3c6f58bdb3461a23963e7b969ad
CST	contracts/ComptrollerStorage.sol	9bd23be2f93eea76d8daf6193a07650ddb88f670

OWN	contracts/Ownership/Ownable.sol	c32aa009ab56ab632f1cff518448948d70678a62
SMA	contracts/SafeMath.sol	e3bc24993dcd56abc9ee26f4c8c3f2ab65dadfce
EIP2I	contracts/EIP20Interface.sol	2a97625760eed470ed68bff42adf303a95885b67
IRM	contracts/InterestRateModel.sol	0ff4dda8c2d430452caf0b62028ba93d55d9de15
JRMV4	contracts/JumpRateModelV4.sol	de7031804f839321e06c24db1ccff6df6865d11c
UPO	contracts/PriceOracle/UniswapPriceOracle.sol	91897e1db2e8e411f6fdcc646dd7ec8226af5ee6
SPO	contracts/PriceOracle/SimplePriceOracle.sol	11780ffc4b8a6b6f49aaa61ef5016885974252f0
CPO	contracts/PriceOracle/ChainlinkPriceOracle.sol	093e8fd0c2ef60f38980ff9034e6ca7abead7dd2
WPO	contracts/PriceOracle/WitnetPriceOracle.sol	0009b161955934821a464a0254357e2493bcec84
CE2I	contracts/CErc20Immutable.sol	66843e7b0d51bef4439a0f54a79d9956702e070c
POR	contracts/PriceOracle.sol	ca58bd9b259ee222a8842cf289fbbde396732888
COM	contracts/Comptroller.sol	10a41beeb807a410b39d83af9ba2c7da4d21d089
ENE	contracts/ExponentialNoError.sol	5b1bfa0f01c6642e2c850f45e47c97ecdc015e03

VCL	contracts/VesterCliff.sol	9ff64311ea9645f97a038e887746125208668bb2
VST	contracts/VesterStepped.sol	23bb3b7af8638bbd4a6bb438550cddce2314ec85
MAT	contracts/libraries/Math.sol	416f0b99850bc23ac7dfe514d3712136d75acb77
STO	contracts/libraries/SafeToken.sol	e50cdb6ad9219e29c5052f97aa7beeee9c349af9
SMA	contracts/libraries/SafeMath.sol	69ee499c3a7ed8aa5ef34a66c70646b72cd6a59d
MEN	contracts/Mendi.sol	e3557c0d067787df8df779b4f0bf6334b2cd3739
MUL	contracts/utils/Multicall.sol	f0d00268d14d0fa6a9bd60dae19626744c3fa264
MCL	contracts/test/MockClaimable.sol	9a60789a471c1a4fedf4d0b3140cbdb316ac56fd
MERC2T	contracts/test/MockERC20Token.sol	4f5963458662f21d25236eddd3b9369525e08df6
IVG	contracts/interfaces/IVelodromeGauge.sol	48a973087e90323750f8c706b2e9cf6823587f02
IDI	contracts/interfaces/IDistributor.sol	94616b45a43ba76423af244734b4c79c37c90bc3
IME	contracts/interfaces/IMendi.sol	18e397f473d8ed387a7edd40c09373b0f5ce1517
IVR	contracts/interfaces/IVelodromeRouter.sol	ebd7f63e24e6a7d4e195c163077a16bc36175d1f

ICL	contracts/interfaces/IClaimable.sol	77d25f851d1f4cb45052fdfc7a5ad594153be90c
IVE	contracts/interfaces/IVester.sol	be2786ff9732b17a6087c58ee70578cf564ecedf
IOD	contracts/interfaces/IOwnedDistributor.sol	549d89bb82199734df7e64911d95045a1650b840
ILG	contracts/interfaces/ILiquidityGenerator.sol	6d3963f885f6c50ff350f129a1e52ce565b589d5
IVVE	contracts/interfaces/IVelodromeVotingEscrow.sol	ca8f12251012da8e875374c51d491611d585b5f4
IVV	contracts/interfaces/IVelodromeVoter.sol	5bac146dd4cce17d758aa665a195583fe8e4ce51
IVPF	contracts/interfaces/IVelodromePairFactory.sol	58643b1bb589ab09f5f107c6a52bc87c0ba414be
IERC2	contracts/interfaces/IERC20.sol	8052e90d76ca1925013d28743861b83621802a70
ODI	contracts/OwnedDistributor.sol	d870b7cc7123d21195021d69e3fc47f0ccb88513
VSA	contracts/VesterSale.sol	57e4d651b7cbe918886efb282e31c1b06702a7a4
LGE	contracts/LiquidityGenerator.sol	b97a89dd9bf4cc0fe3e6f8b4ebf8eebe38f6a23b
VES	contracts/Vester.sol	f445f311b76b74c925b27b6ff66e16da678d821c
DIS	contracts/Distributor.sol	00131faa6c01583fba5050e7e1eadf7bff76962d

EIN	contracts/interfaces/EquilibreInterfaces.sol	7f340e62f5a384ef72517bd8c2283ad645c001e6
ICL	contracts/interfaces/IClaimable.sol	6aaf87beeb34d15f943f348efbab120ff8bc736b
RHO	contracts/RewardHolder.sol	7031721d6edb505509fce81bf8640108f36fae13
SDI	contracts/StakedDistributor.sol	c10b15666b8025d3f513eecf375ce0986f4a5363
DIS	contracts/Distributor.sol	435e5bbc495c8c79c021fd252b5e87722616dc53

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	4	0	4
Informational	1	0	1
Minor	1	0	1
Medium	1	0	1
Major	1	0	1
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Mendi Finance to identify any potential issues and vulnerabilities in the source code of the Mendi Finance smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 4 issues of varying severity, listed below.

ID	Title	Severity	Status
CTO-1	First `mintFresh` Design Flaws	Major	Acknowledged
DIS-1	No Null Checks For Input Addresses	Informational	Acknowledged
LGE-1	Missing Interface To Modify Admin	Medium	Acknowledged
VSA-1	Loss Of Precision	Minor	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the Mendi Finance Smart Contract:
Admin

- Admin can create a add reward tokens through `_whitelistToken()` .
- Admin can set reward holder where can be claimed through `_setClaimable()` .
- Admin can set the guy who can claim through `_setRecipient()` .
- Admin can set the `admin` , `reservesManager_` of `LiquidityGenerator` through `_setAdmin()` .

Recipient

- Recipient can claim reward coins through `RewardHolder.claim()` .

User

- User can update the reward shareIndex through `updateShareIndex()` .
- User can get their reward through `claim()` .
- User can get their stake coins through `mint()` .
- User can get their underlying coins through `burn()` .
- User can withdraw their underlying coins through `withdraw()` .
- User can deposit their token to get reward through `deposit()` .
- User can get Mendi Coin through `claim()` .

4 Findings

CTO-1 First `mintFresh` Design Flaws

Severity: Major

Status: Acknowledged

Code Location:

contracts/CToken.sol#499

Descriptions:

In the `mintFresh` contract, the `exchangeRateStoredInternal` function calculates the exchange rate by dividing the current pool assets (including balances, lent assets, and subtracting returned assets) by the total shares. If a hacker injects the smallest unit of a share when the pool is first created or emptied, and then transfer a certain amount of funds directly into the pool, this will result in an extremely large net value according to the above net worth algorithm. So if the exchange rate can be increased to a value greater than the user's deposit, so the user will always get 0 shares. Since only the attacker has shares of the pool, all the transferred funds will be taken by the attacker. The attacker utilizes the following steps:

1. the attacker can inject a share of the smallest unit when the pool is unfunded (just created, or when it is taken empty).
2. The attacker injects a certain amount of money into the pool, resulting in a very large net value of the pool.
3. Since the pool has a very large net value, the ordinary liquidity provider's funds are counted as 0 shares according to the previous formula, so the attacker keeps the full share after the market is created, when only the funds enter, but the user always gets 0 share.
4. The attacker redeems the shares and steals all the funds.

Suggestion:

The workaround to prevent this issue is to force lock in a non-withdrawable minimum deposit amount. This can be accomplished by minting a small number of CToken units to address 0x00 on the first deposit.

```
if (totalSupply == 0) {
    totalSupply = 1000;
    accountTokens[address(0)] = 1000;
    mintTokens -= 1000;
}
```

Resolution:

[**Mendi Team**]: Regarding the specific attack vector you highlighted, we acknowledge that an empty market, coupled with a non-zero collateral factor and a `totalSupply` of zero for the corresponding `meToken`, could lead to share price manipulation and potential draining of funds.

Our approach will involve a multi-step workflow that effectively prevents this attack vector without requiring an upgrade to the existing smart contracts. The steps we have implemented are as follows:

Addition of the market to the comptroller with a zero collateral factor: Before any borrowing or lending activity can take place, we add the market to the comptroller with a collateral factor of zero.

Minting and burning of a small amount of `meTokens`: To eliminate the possibility of an empty market, we mint a small amount of `meTokens` and subsequently burn them.

Setting the collateral factor for the market: Once the previous steps have been completed, we proceed to set an appropriate collateral factor for the market.

DIS-1 No Null Checks For Input Addresses

Severity: Informational

Status: Acknowledged

Code Location:

contracts/Distributor.sol#39

Descriptions:

Some functions lack parameter checking, e.g. `setAdmin` doesn't check if the incoming address is a null address.

Suggestion:

We propose to add check to null addresses.

LGE-1 Missing Interface To Modify Admin

Severity: Medium

Status: Acknowledged

Code Location:

contracts/LiquidityGenerator.sol

Descriptions:

For the `OwnedDistributor` contract, if the user wants to call the `editRecipient` function needs to be the admin of the `OwnedDistributor`, and at the same time in the LiquidityGenerator contract, for example, in the deposit function will be used to `editRecipient`, so the `LiquidityGenerator` contract is the administrator of the `OwnedDistributor`. `OwnedDistributor` contract exists setAdmin and `editRecipient` need admin address function, but is no corresponding modification to the `LiquidityGenerator` contract, so there is no address to call the setAdmin and `editRecipient` functions.

Suggestion:

Add an interface to the `LiquidityGenerator` contract that modifies the administrator's address accordingly.

Resolution:

[**Mendi Team**]: `LiquidityGenerator` has 2-step admin mechanism. It is also included in the deployed version. During the LGE, `OwnedDistributor`'s admin was `LiquidityGenerator` contract, `LiquidityGenerator` was able to edit recipients on `OwnedDistributor`. After that, we do not need and do not want to change the `OwnedDistributor`'s admin because locks the recipient shares on contracts.

VSA-1 Loss Of Precision

Severity: Minor

Status: Acknowledged

Code Location:

contracts/VesterSale.sol#31

Descriptions:

In the `VesterSale` contract, when calculating the number of unlocked tokens a user can obtain, the formula does not follow the principle of multiplication before division, so this may result in a loss of precision in the calculation of the result of the `amount`.

Suggestion:

It is recommended that all multiplication operations be placed before division operations.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. 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 own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.

SCALEBIT VERIFIED

SCALEBIT VERIFIED