minterest.

SMART CONTRACT AUDIT

zokyo

March 14th 2023 | v. 3.0

Security Audit Score

PASS

Zokyo Security has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

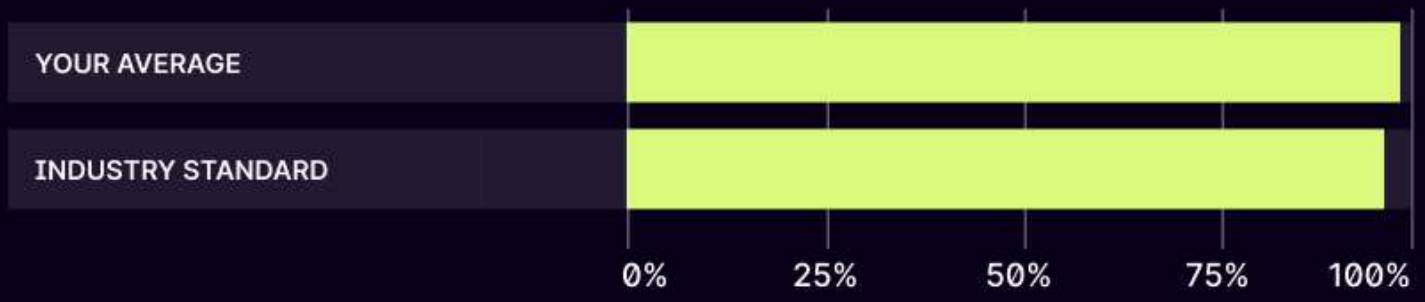
TECHNICAL SUMMARY

This document outlines the overall security of the Minterest smart contracts evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Minterest smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



98% of the code is testable, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Minterest team put in place a bug bounty program to encourage further active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Minterest repository for the second audit iteration:
https://github.com/minterest-finance/protocol

Intial commit: develop branch, 0f9388d80cb1f89477f38bbcf1d801af35ded693
Final commit: develop branch, 55ba929627a2b42083249051fe6913bbcc3844c8

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- governance/Mnt.sol
- governance/MntErrorCodes.sol
- governance/MntGovernor.sol
- governance/MntTimelock.sol
- governance/MntVotes.sol
- libraries/ErrorCodes.sol
- libraries/PauseControl.sol
- oracles/ChainlinkPriceOracle.sol
- oracles/FallbackPriceFeed.sol
- All contracts within ./contracts/*.sol

The source code of the smart contract was taken from the Minterest repository for the **second** and **third** audit iteration:
https://github.com/minterest-finance/protocol

Branch: develop
Initial commit: 43bad236ef94655daf0d7dcf195bb69b4da2e876
Final commit: a4e2e055a602d405d86ca3dfcbc1bdd233c7b08f

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- contracts\Buyback.sol
- contracts\DeadDrop.sol
- contracts\Liquidation.sol
- contracts\MToken.sol
- contracts\MinterestNFT.sol
- contracts\RewardsHub.sol
- contracts\Supervisor.sol
- contracts\WeightAggregator.sol
- contracts\libraries\ErrorCodes.sol
- contracts\oracles\FallbackPriceFeed.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most resent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Minterest smart contracts. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Hardhat testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contracts by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

Zokyo Security received the whole set of contracts provided by the Minterest team. The protocol represents a lending and borrowing protocol, which also includes the implementation of flashloans and an improved rewards distribution system.

The goal of the audit was to verify that the contracts function correctly and with a known level of security and check the code line by line. Auditors also checked the code of the contracts against their own checklist of vulnerabilities, validated the bussiness logic of the contracts, and made sure that best practises in terms of gas spendings are applied.

There were two high-severity and several low-severity and informational issues found during the manual part of the audit. One of the high-severity issues was connected to the unsafe initializations of the contracts, while the other one was connected to the absence of swap parameters validation. The first issue was successfully fixed by the Minterest team, while the second one was not addressed during the current audit iteration. According to the Minterest team, the issue is well-known and will be fixed later. All other issues were successfully fixed or verified by the Minterest team.

Another part of the audit process was to check the native tests prepared by the Minterest team and prepare a set of custom test scenarios. It should be mentioned that the Minterest team had provided a solid set of tests, which cover all the core logic of the contracts. Nevertheless, Zokyo Security prepared their own set of unit tests. The complete set of unit tests can be seen in the Code coverage and Test Result sections.

The total security of the contracts is high enough. The contracts are well-written and tested. The basic borrow and lending functionality and additional logic written by the Minterest team was carefully audited. Minterest has also provided a detailed documentation on implemented functionalities. The link to the documentation is https://minterest.gitbook.io/sec-audit-doc/user-flow/emission-rewards.

During **the second audit iteration**, the Zokyo Security team audited all changes since the last audit iteration. The Minterest team added several features to the smart contracts. Therefore the goal of the second audit iteration was to audit all the changed code, check its correspondence to the business logic of the protocol, and check that code is optimized in terms of gas spending.

During the manual audit, the Zokyo Security team audited all the smart contract changes. The Minterest team successfully verified all found issues. Auditors have also raised a high-severity issue found in the first audit iteration. The issue was connected to the absence of the validation of swap data and was verified by the Minterest team as well. According to the team, in order not to increase the cost of the transaction and due to the complexity of the fix, the code will stay as is. Since only accounts with the GateKeeper role can perform swap, the issue is difficult to exploit by the attackers and can only be exploited in case of the leak of private keys.

Zokyo Security has also prepared a set of additional tests to validate the new code. All tests written in the first audit iteration were also fixed and run. Thus, the Minterest protocol has passed all the security tests. The overall security of the protocol, particularly the new code, is high enough.

For the **third audit iteration**, the Minterest team has updated the implementation of several smart contracts. Most of the changes were connected to the Buyback.sol, Liquidation.sol, and Vesting.sol. Though the changes weren't major, the Zokyo Security team has carefully audited and tested them.

The Zokyo Security team hasn't detected any issues in the updated code. There were two informational issues described in the third iteration: one of them was about the absence of validation, and the other one was connected to logic clarification in Buyback.sol. The Minterest team has verified both of the issues.

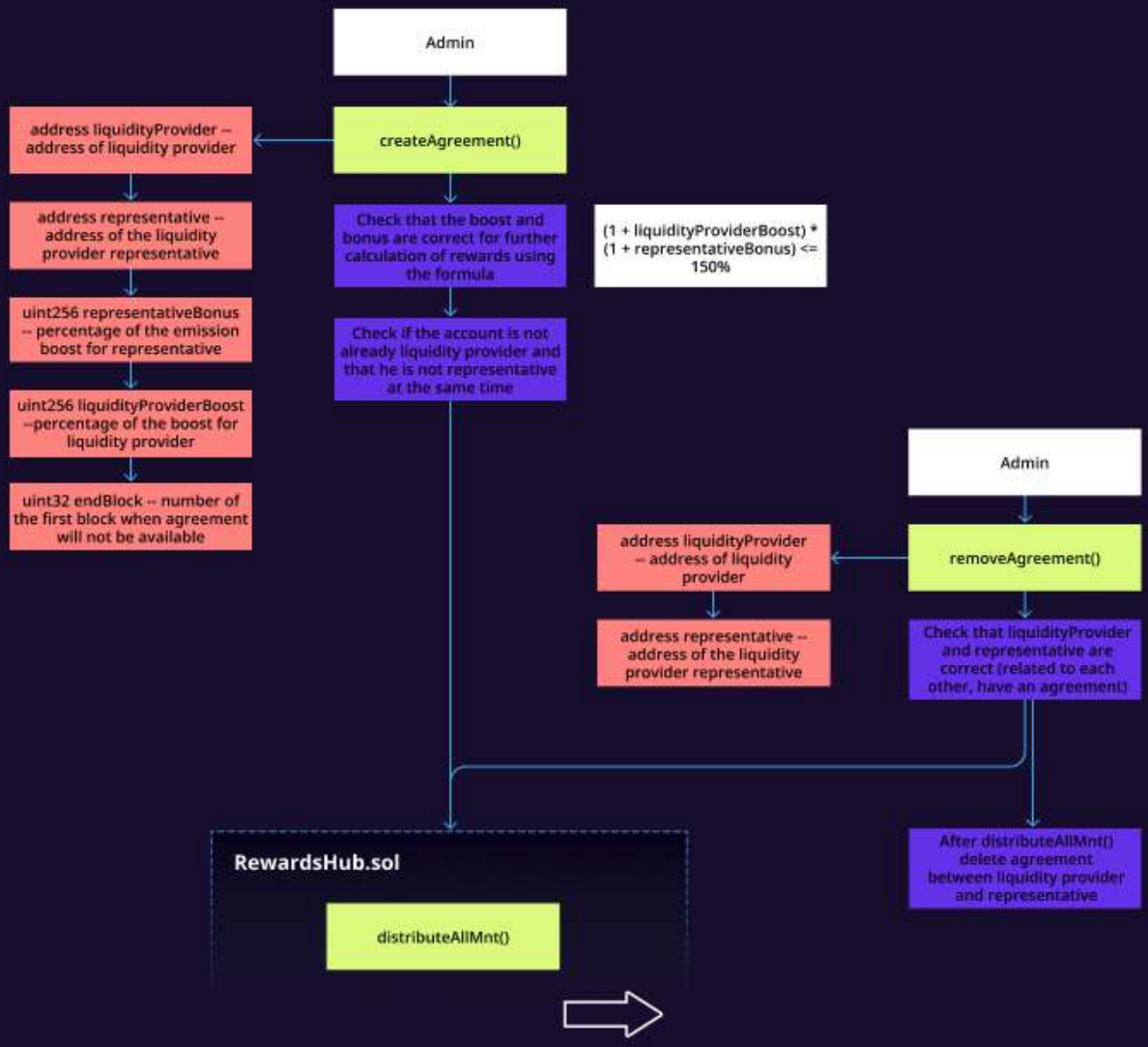
Since most of the changes were added to already existing functions, the Zokyo Security team has updated their set of unit tests prepared during previous audit iterations to validate new logic. Additional test-cases were prepared for the vesting flow.

The security of updated smart contracts is high enough. The code is well-written, and contains sufficient documentation. The Minterest team has also prepared a solid set of unit tests for the whole protocol. Nevertheless, the Zokyo Security team has also tested all the changes within the current audit iteration.

Protocol Overview

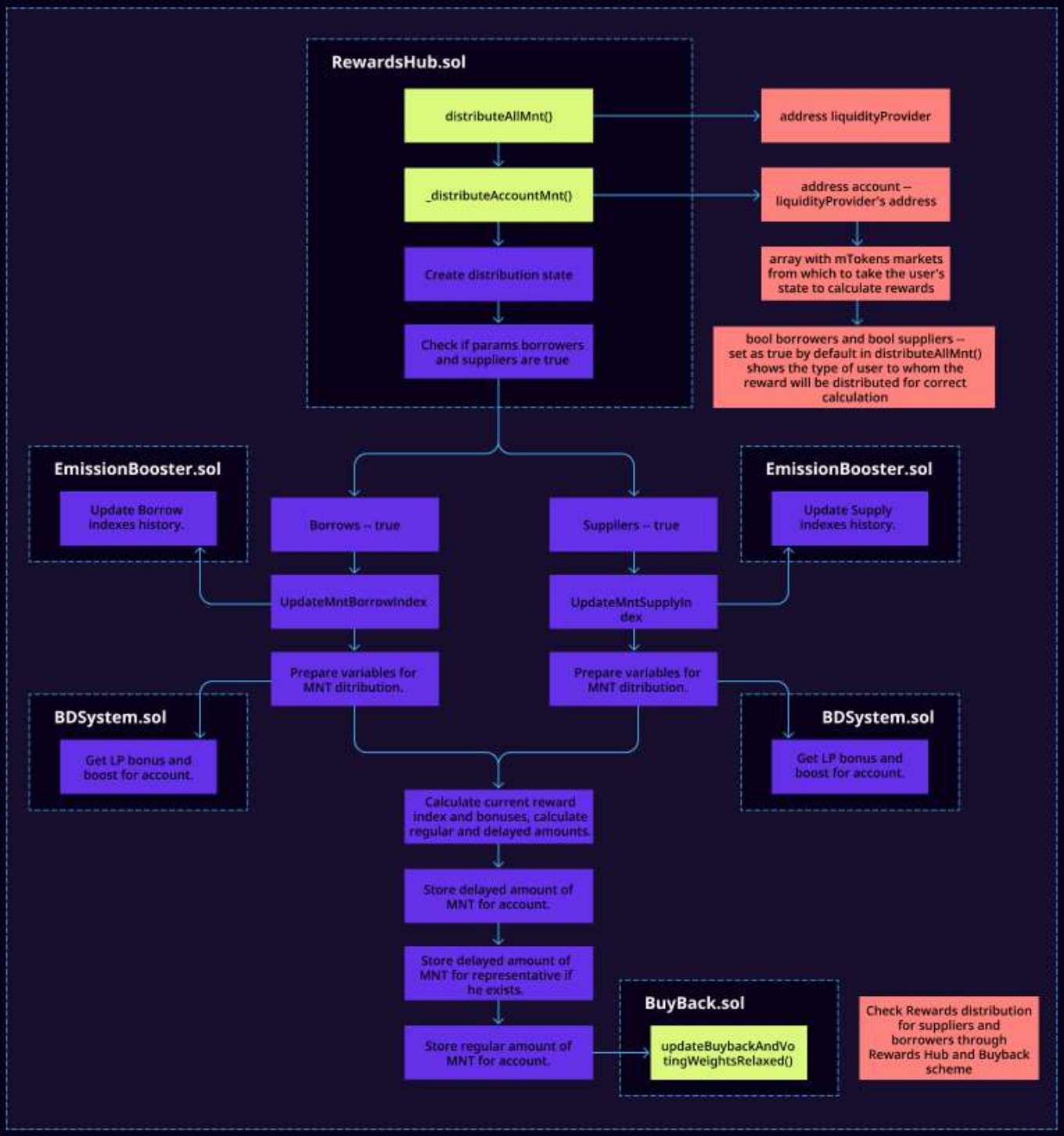
Agreements operation through BDSystem.sol

A lending and borrowing protocol protocol that also has additional functionality such as auto-liquidations, flash loans, and expanded reward system.
There are:
1) Standard rewards that the user receives for supply and borrow.
2) NFT rewards that are received by Minterest NFT holders who supply and borrow.
3) Governance rewards received by users who use governance staking of MNT.
Also, the protocol has a referral program contract (BDSystem on the scheme) designed for whales and BD representatives.



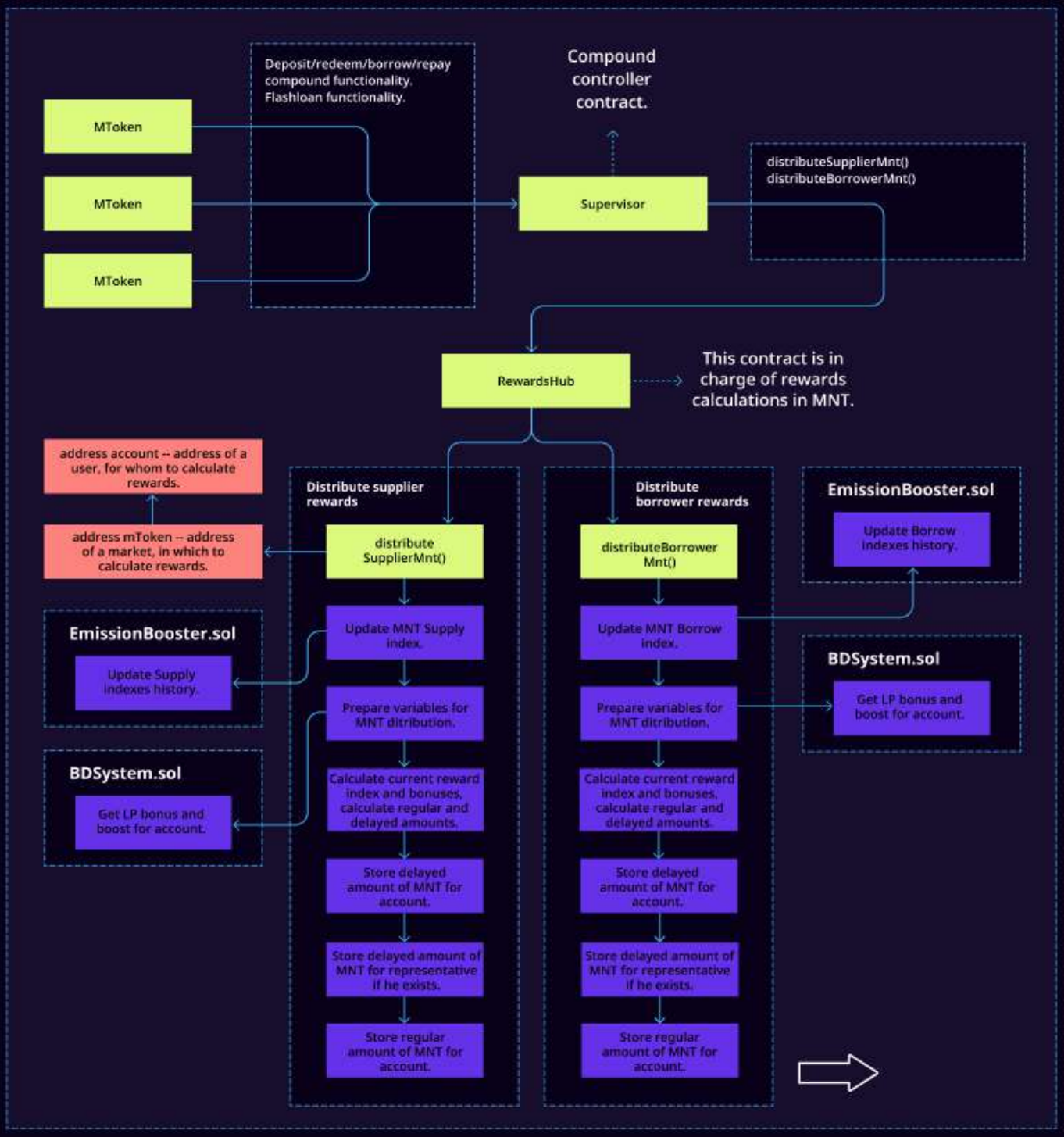
Protocol Overview

Agreements operation through BDSystem.sol



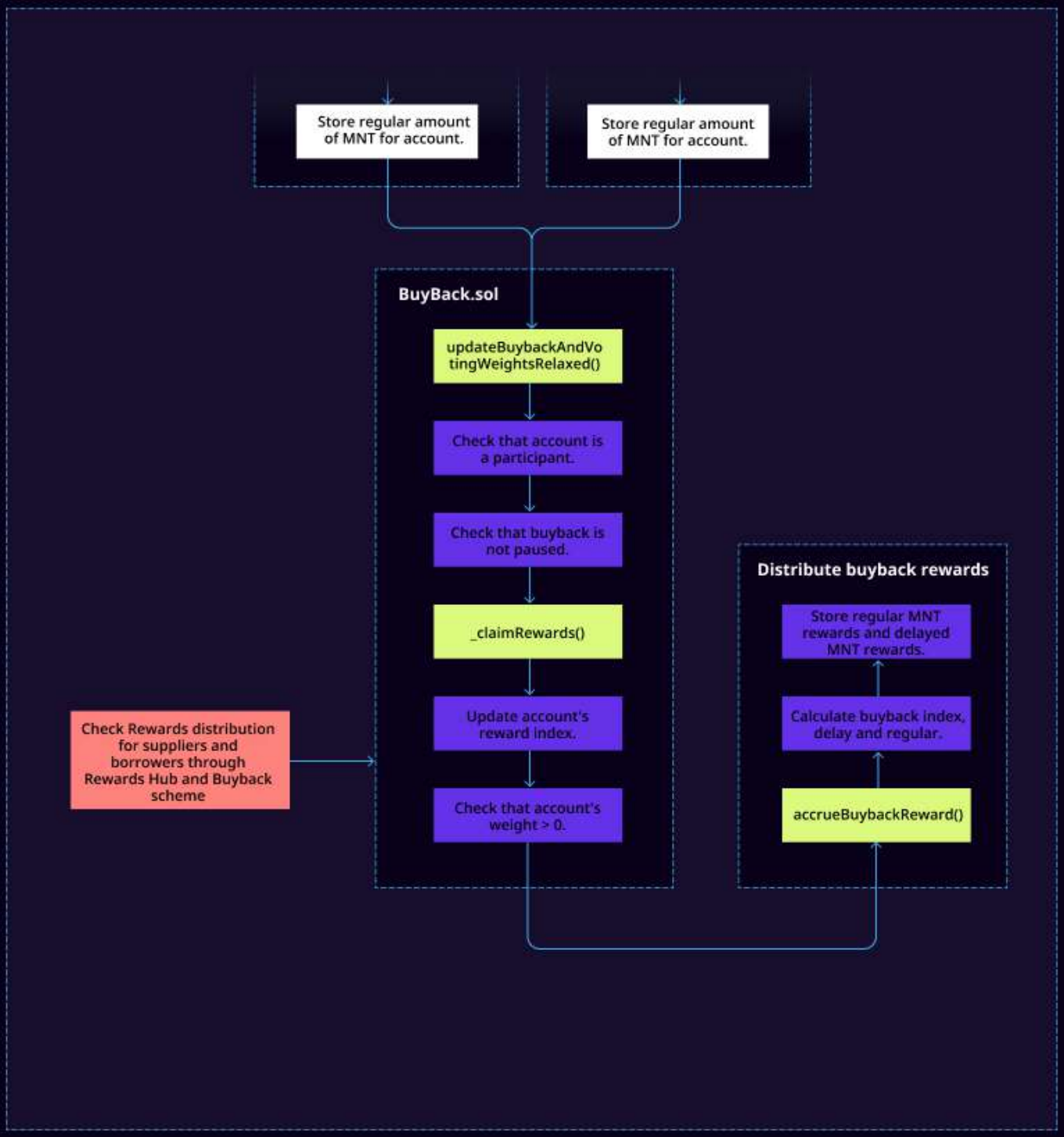
Protocol Overview

Rewards distribution for suppliers and borrowers through the Rewards Hub and Buyback.



Protocol Overview

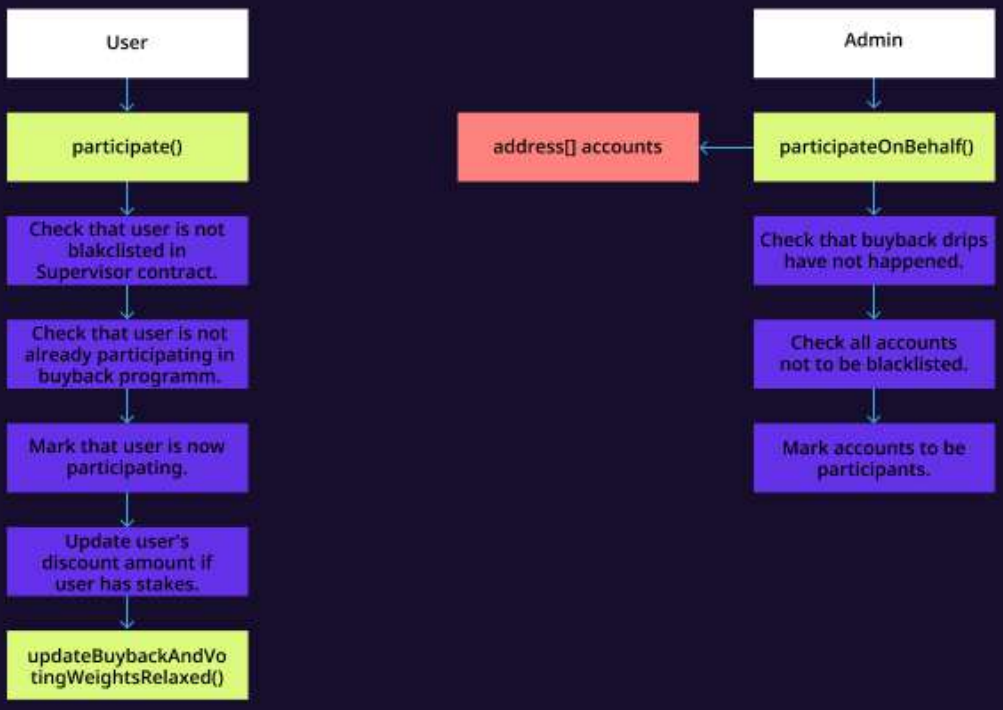
Rewards distribution for suppliers and borrowers through the Rewards Hub and Buyback.



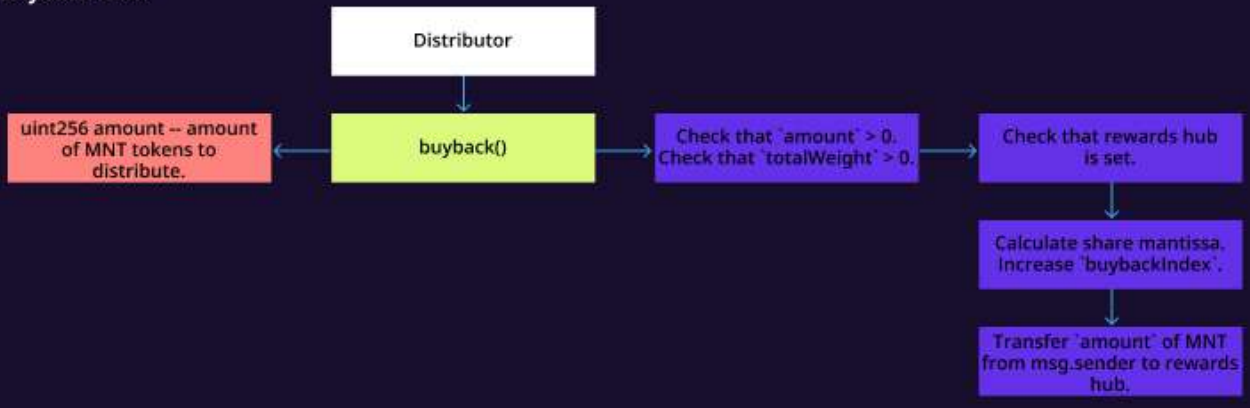
Protocol Overview

Buyback.sol

Participation flow.

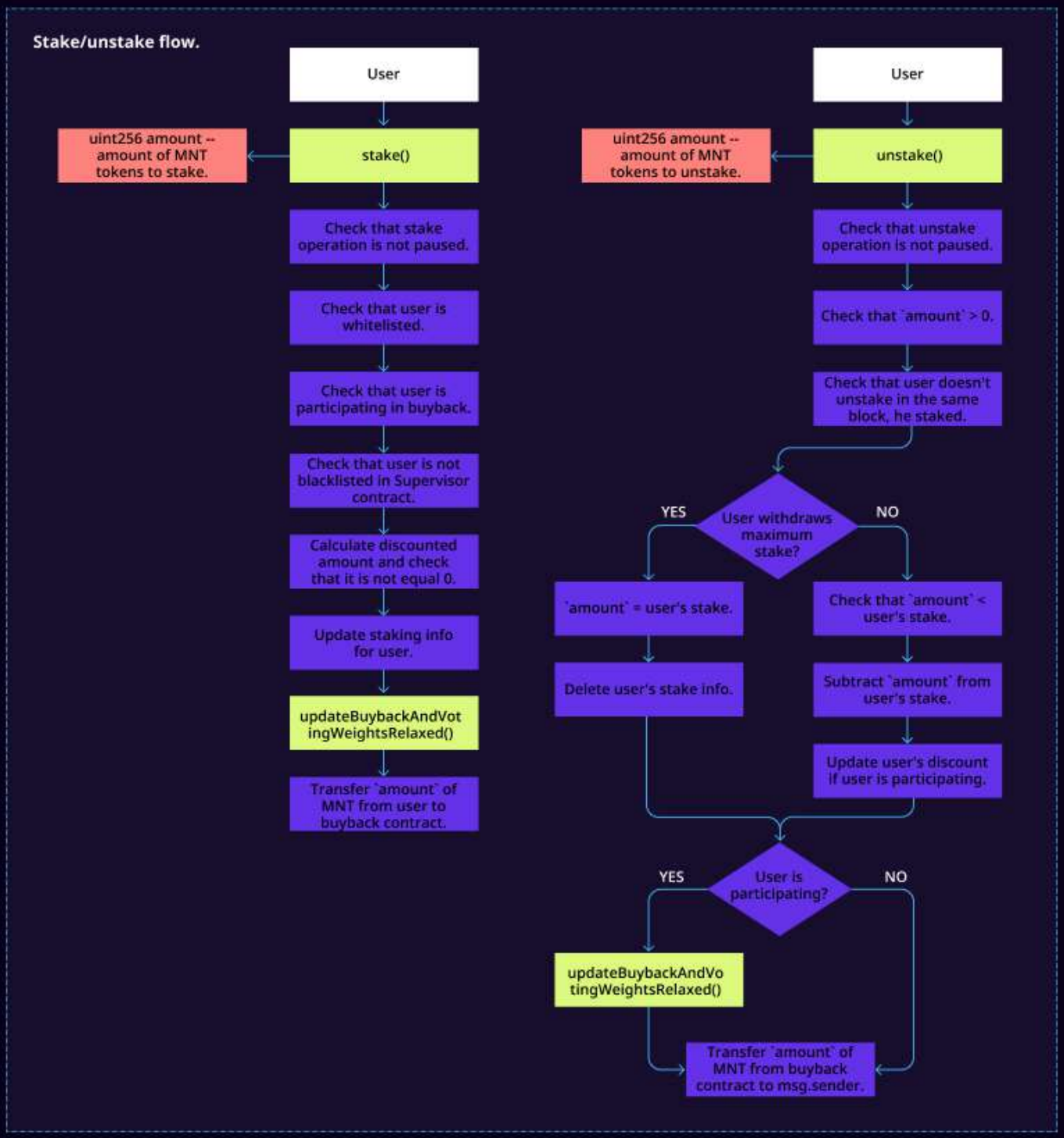


Buyback flow.



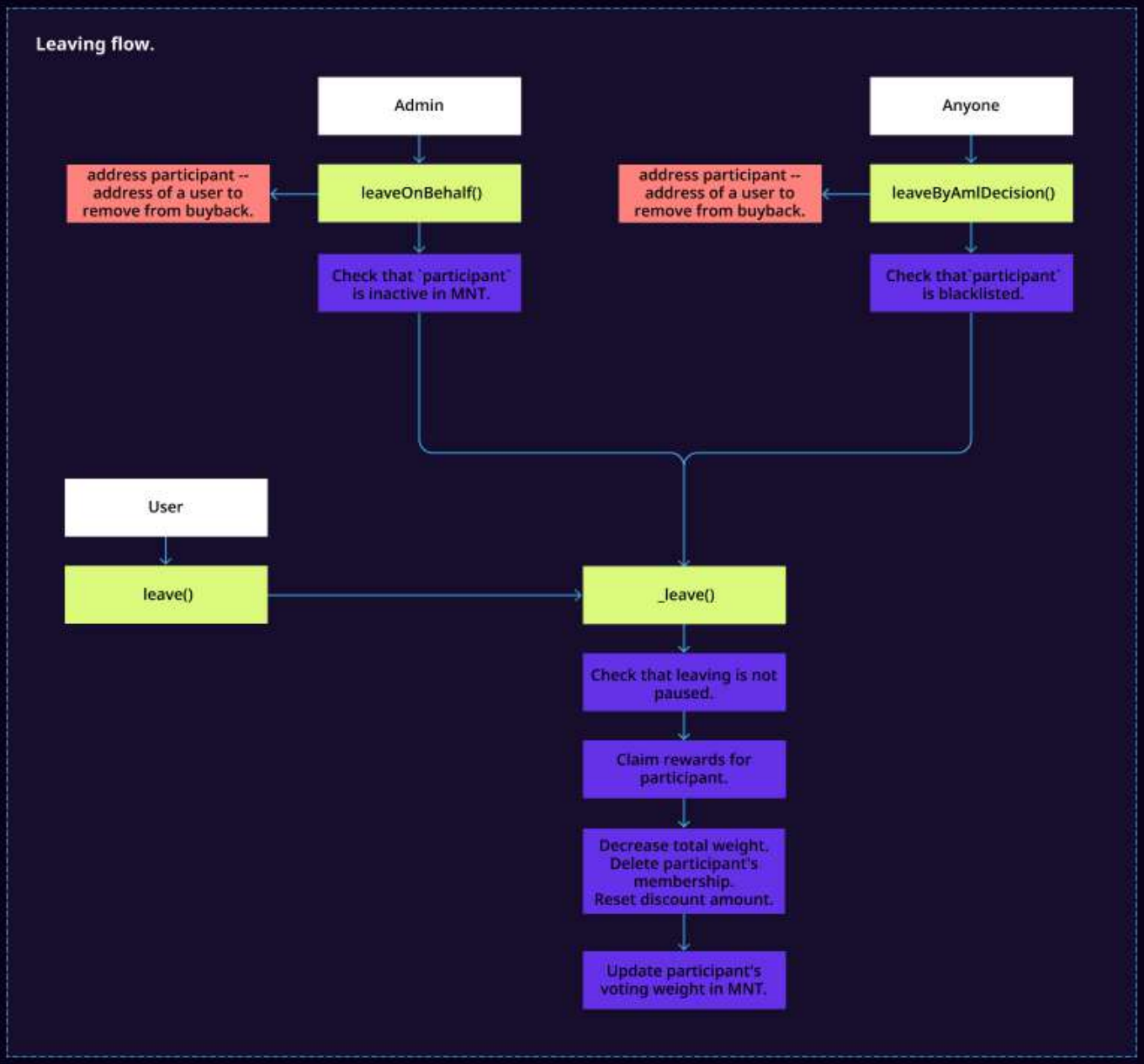
Protocol Overview

Buyback.sol



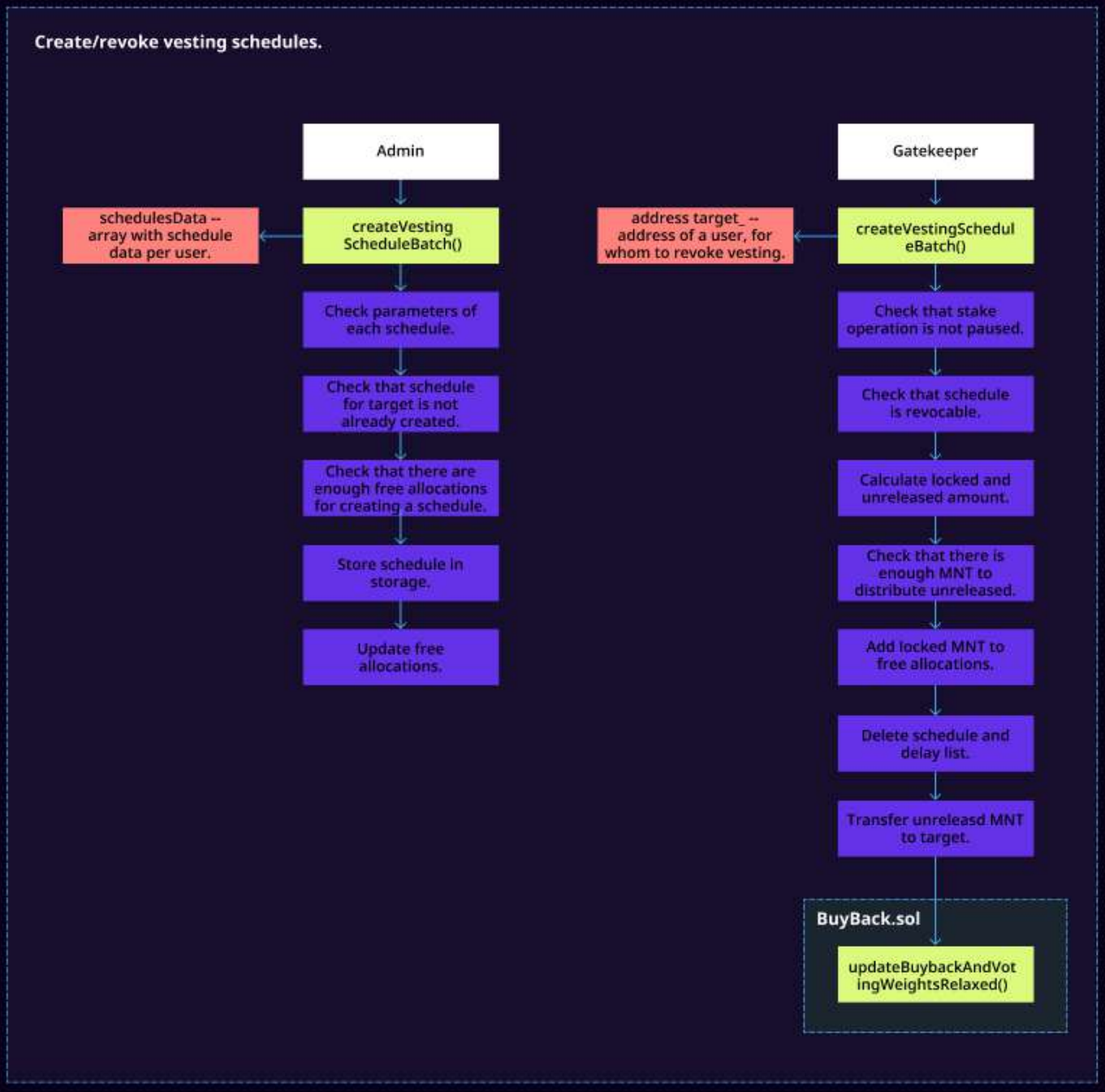
Protocol Overview

Buyback.sol



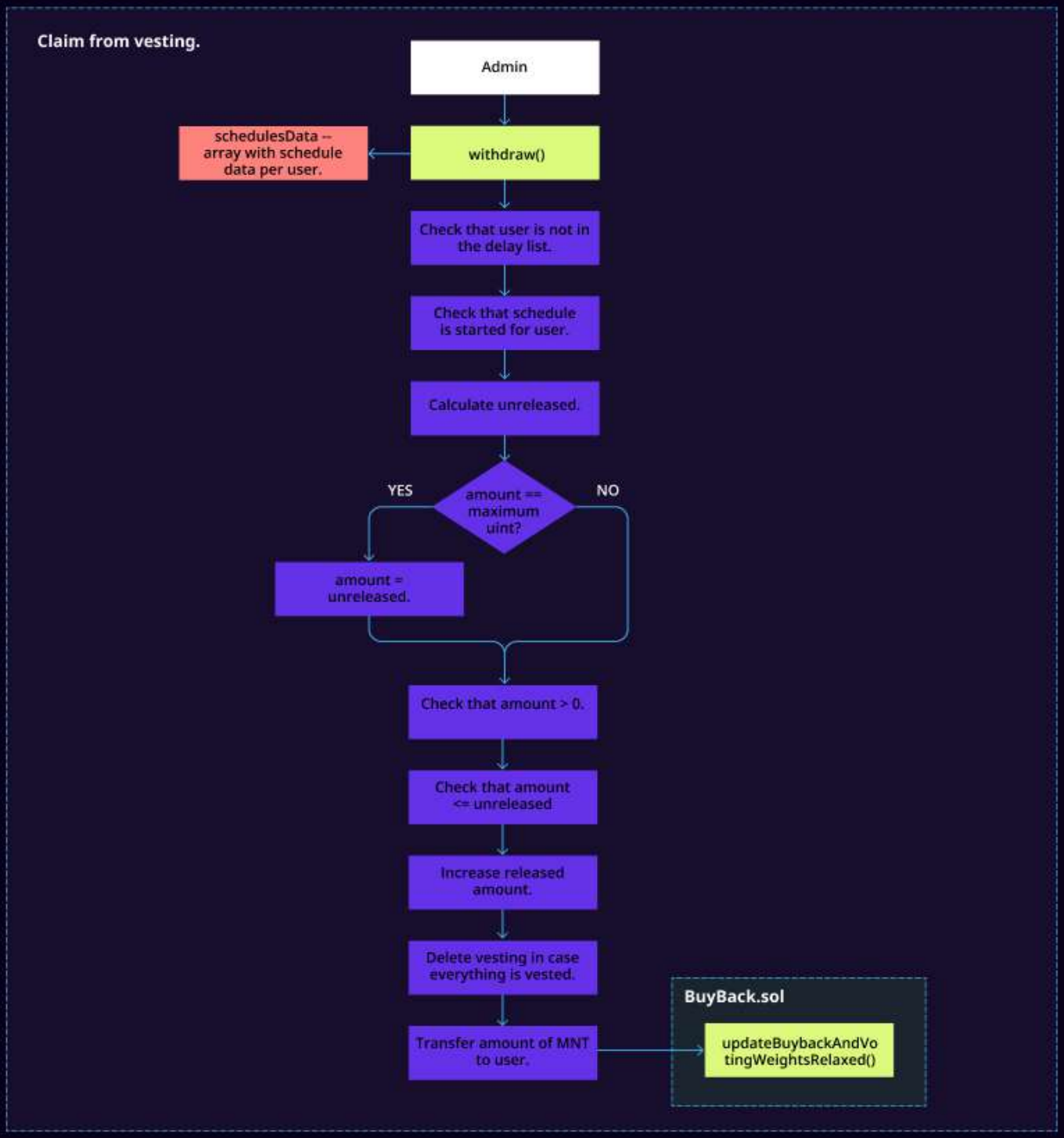
Protocol Overview

Vesting.sol



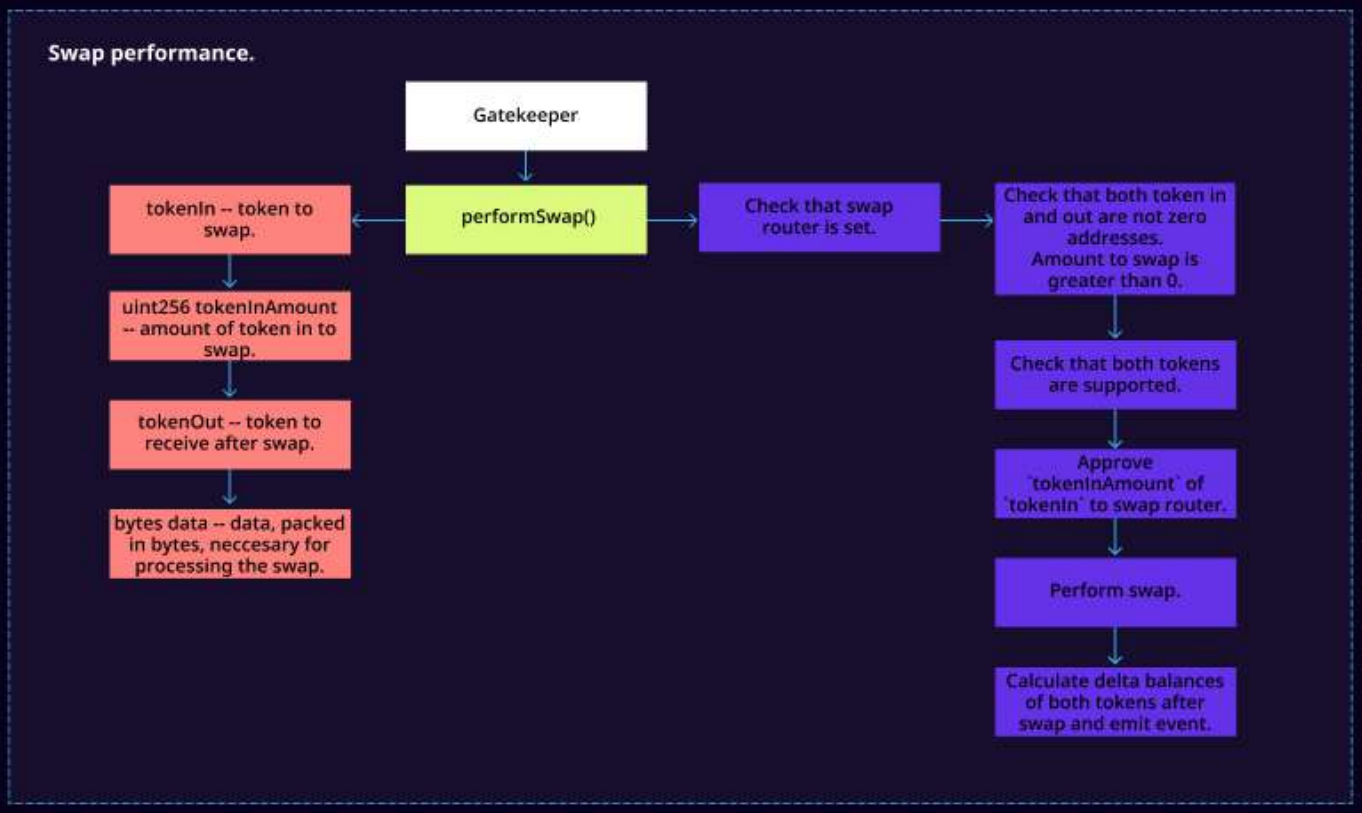
Protocol Overview

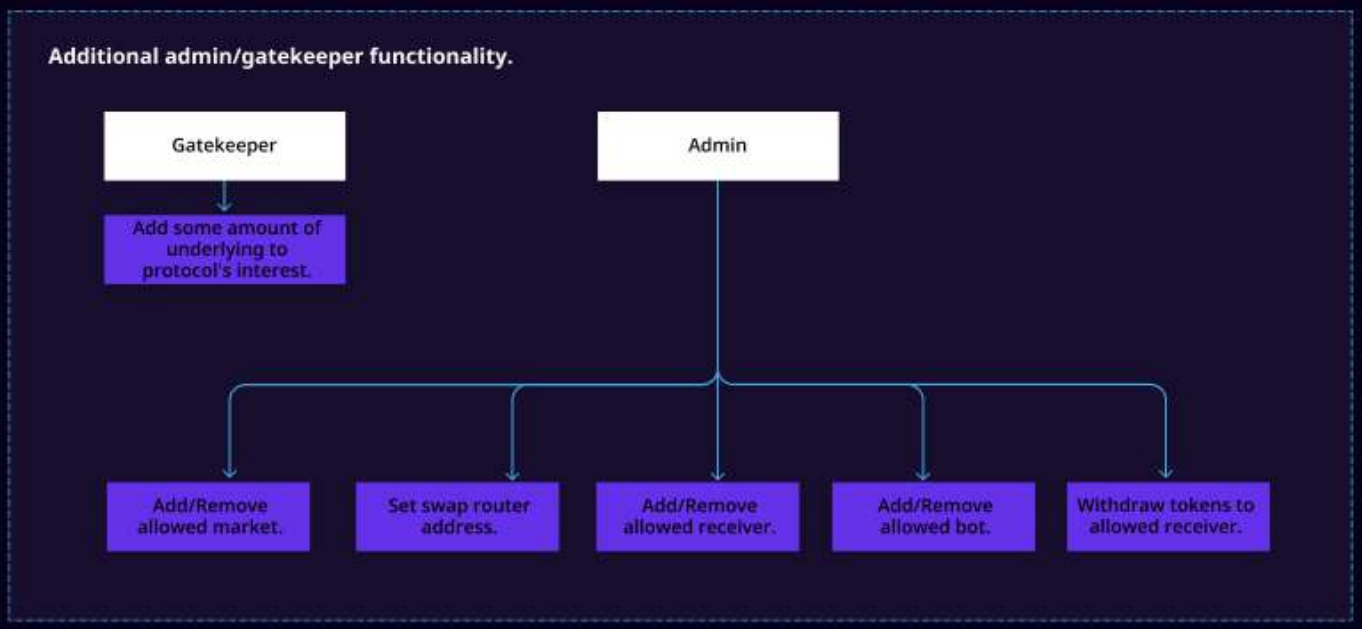
Vesting.sol



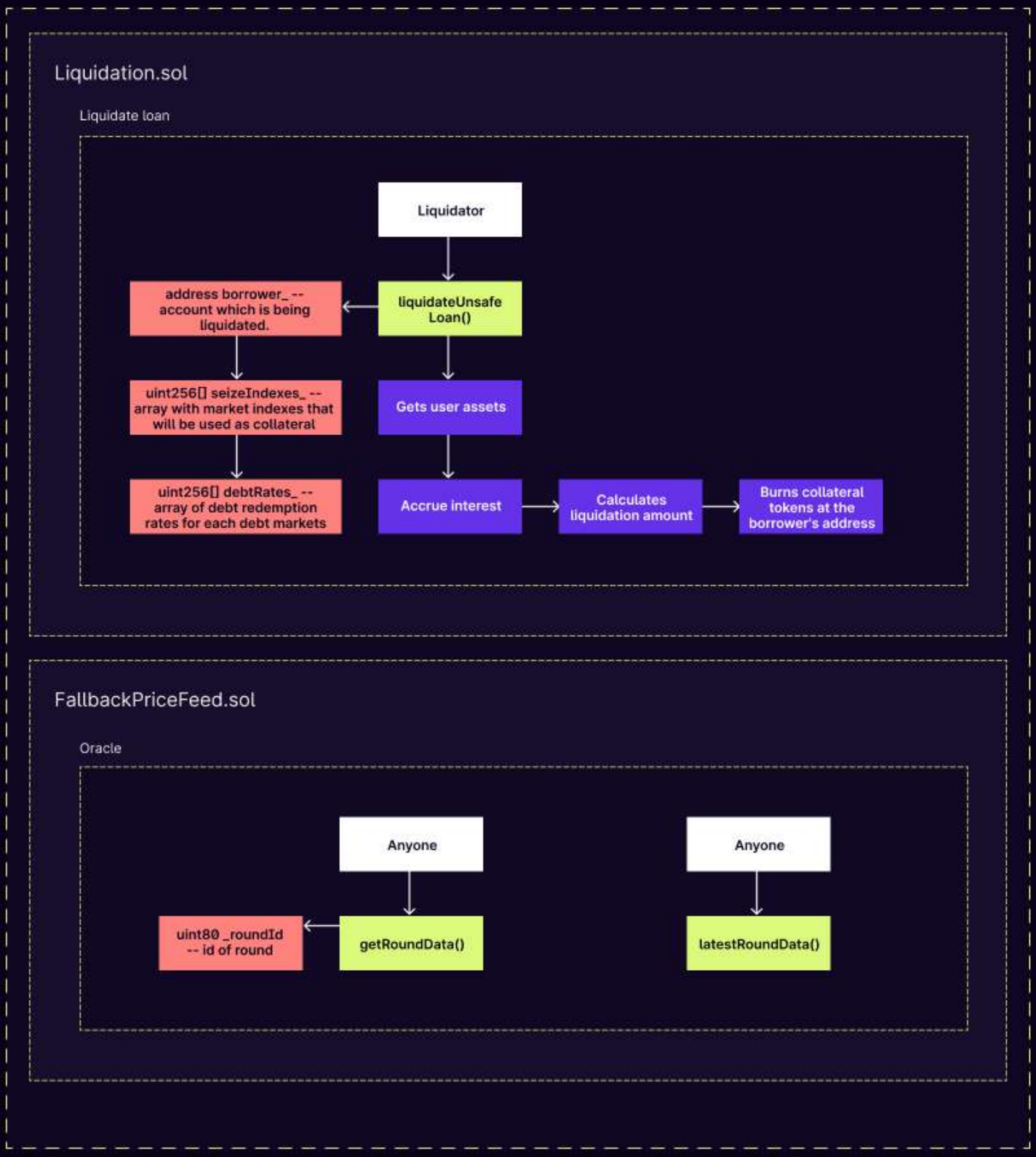
Protocol Overview

DeadDrop.sol

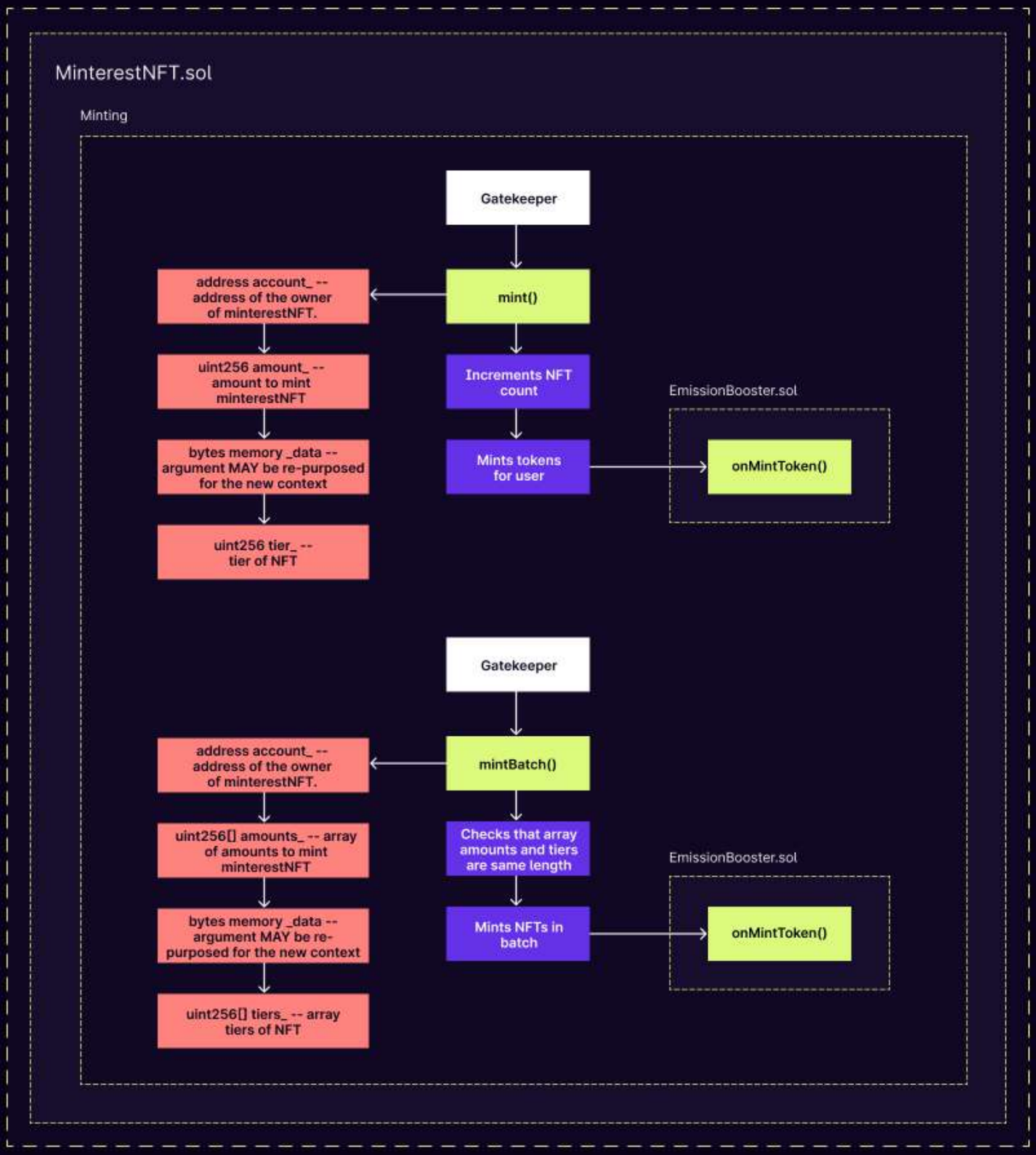




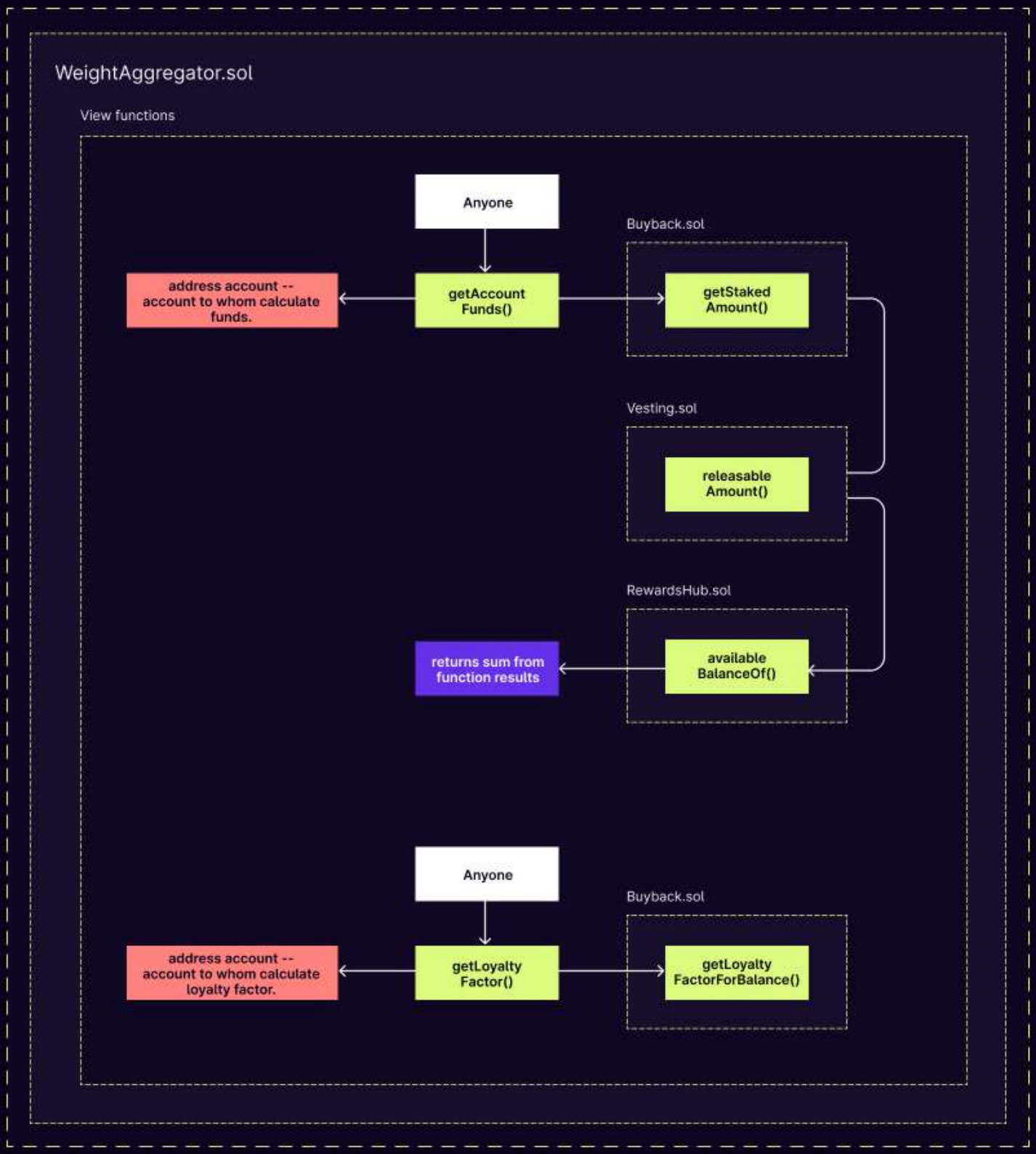
Protocol Overview



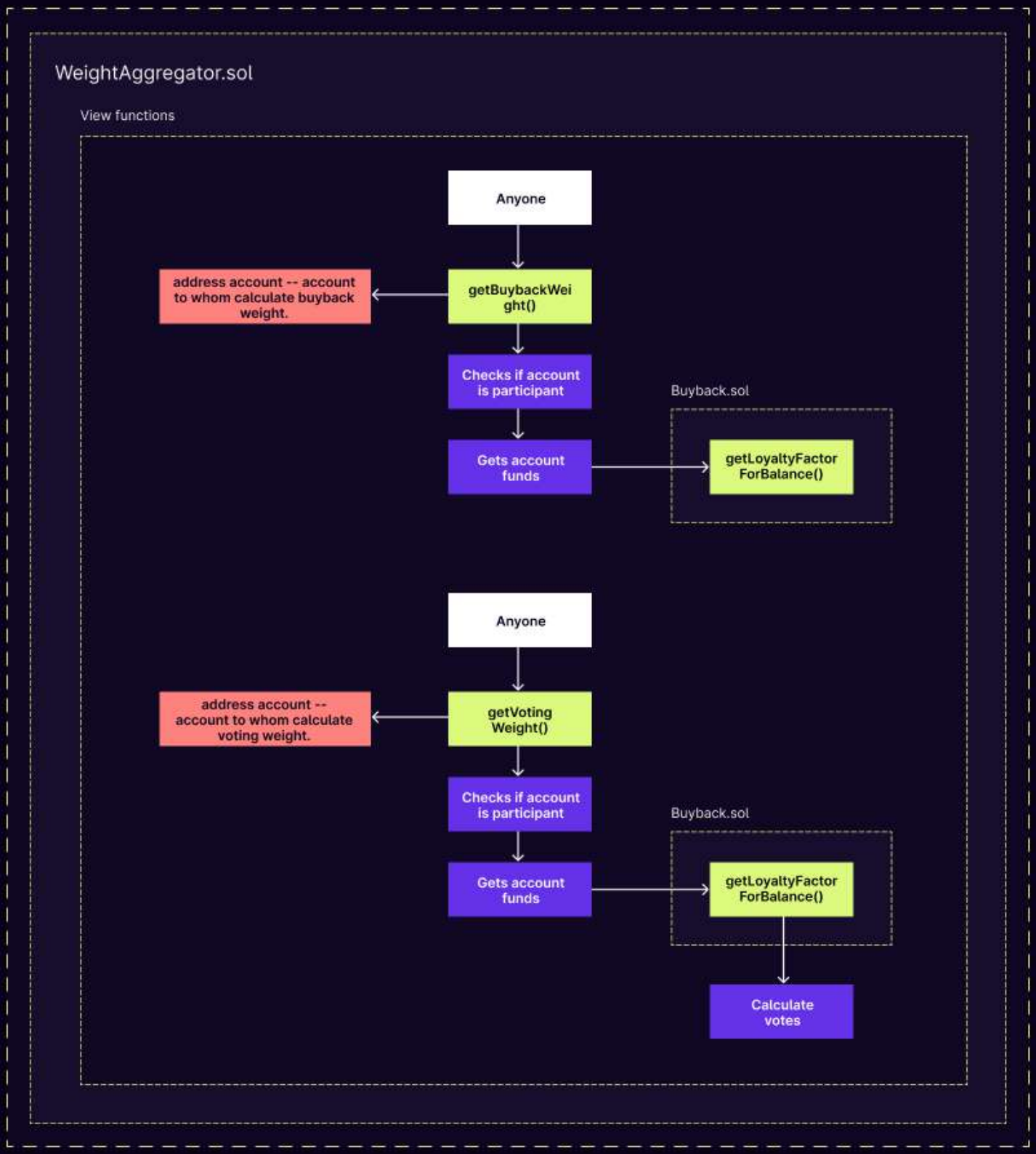
Protocol Overview



Protocol Overview



Minterest scheme



STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as "Resolved" or "Unresolved" depending on whether they have been fixed or addressed. The issues that are tagged as "Verified" contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn't significantly hinder its behavior.

Low

The issue has minimal impact on the contract's ability to operate.

Informational

The issue has no impact on the contract's ability to operate.

COMPLETE ANALYSIS
1ST AUDIT ITERATION COMPLETE ANALYSIS

HIGH-1 | RESOLVED

Possible re-initialization of the contracts.

Some of the contracts do not utilize the `initializer` modifier from standard OpenZeppelin contracts. Instead, a storage variable is checked. For example, in the RewardsHub.sol contract, the `delayStart` variable is validated not to be equal to `0`. Yet, it is possible to set it as `0` or upgrade the contract and add a setter for this variable. This fact and the fact that the initialize function has no admin restrictions might lead to possible exploits, which is why the issue is marked as high.

Recommendation:
Use the standard `initializer` modifier from OpenZeppelin in all contracts to esure that the contract is initialized only once.

Post-audit.
Initializer modifier is used now.

HIGH-2 | ACKNOWLEDGED

Swap parameters are not validated.

DeadDrop.sol: function performSwap().
Those parameters that are packed in the `data` bytes parameter are not validated. This can lead to some of the possible attacks:

1) Tokens from `data` are not equal to the `tokenIn` and `tokenOut` tokens.
2) The amount of tokens to actually swap is not equal to `tokenInAmount`.
3) The recipient is an arbitrary address (not the address of the DeadDrop contract).
4) The `minAmountOut` parameter, which is present in all decentralized exchanges to prevent users from frontrun attacks, is set to `0`. This can lead to the possible loss of funds in case of a frontrun attack.

Though the function can only be called by the keeper, there are no additional scripts in the scope explaining how variables are validated before packing into bytes.

Recommendation:
Validate swap parameters. Validate the correctness of tokens, recipient, and the `minAmountOut` parameter.

From the client.
According to the Minterest team, this is a known issue. The fix will be implemented till the next audit revision. Till then, the contract will not be used.

Post-audit:
The issue was raised during the second audit iteration. Minterest team has verified that the current implementation will be left as is. Since only Gate Keepers can execute swap transactions, the risk of exploit is low. Since the fix requires decoding the data parameter on chain, which increases the gas cost of a liquidation. This leads to an increase of minimal viable loan position the protocol may process with profit. Nevertheless it is recommended to validate parameters off chain before performing swap.

LOW-1 | RESOLVED

Pause on the Buyback might block protocol actions.

Buyback.sol: function updateBuybackAndVotingWeights(), line 232. This function is called every time by RewardsHub.sol (in distributeSupplierMnt() and distributeBorrowerMnt()), which are called every time the user performs actiona in MToken, such as deposit/borrow/repayBorrow, etc. Thus, in case Buyback.sol in set on pause, all the actions on the platform will be frozen as any transaction will revert. It should be verified that a pause on the Buyback contract should stop platform actions, especially prevent users from withdrawing/repaying debts.

Recommendation:
Verify that protocol actions are not blocked due to the pause on Buyback.sol.

Post-audit.
A “relaxed” version of function was added, which doesn’t block the protocol due to pause.

LOW-2 | RESOLVED

Validation is missing.

1) Liquidation.sol: function constructor().
During the contract creation, the supervisor address variable can be passed as address `0`, and further calls to this address in this contract will fail.

2) KinkMultiplierModel.sol: function constructor().
During the contract creation, variables initialRatePerYear, interestRateMultiplierPerYear, kinkCurveMultiplierPerYear, kinkPoint can be passed as `0`, and next calculations such as division and multiplication with these variables will also be zero.

3) MNTSource.sol: function constructor().
During the contract creation, the dripRate variable can be passed as `0`, and next calculations such as division and multiplication with this variable will also be zero. Also, the admin can be passed as zero address.

4) BuyBack.sol: function initialize().
During the contract creation, the supervisor, mnt, rewardsHub, and admin addresses variables can be passed as address `0`, and further calls on these contracts will fail.

Recommendation:
Add necessary validation.

Post-audit.
Point 4 was not fixed.

INFO-1 | VERIFIED

Unclear way of how rewards are sent to the contract's balance.

RewardsHub.sol
There is no mandatory transfer of the rewards present in contract when the emission rate is set. Thus, there might be cases when there are not enough MNT tokens to pay rewards. The issue is marked as informational since the lack of rewards doesn't prevent actions on the platform (such as deposit or borrow) since rewards are collected in a separate function.

Recommendation:
Verify how reward tokens are sent to the contract's balance.

From the client.
According to the team, such functionality is developed for the Governance to decide the distribution rate and the amount of tokens for each market.

INFO-2 | VERIFIED

Accounts are not validated and a discount is not updated when participating on behalf.

Buyback.sol: function participateOnBehalf().
Unlike in the participate() function, there is no validation that `accounts[i]` is not already a participant and the discount amount is not updated for new participants. Though this is not a security issue, it should be verified by the team if this is an intended logic.

Recommendation:
Validate that `accounts[i]` is not already a participant and update the discount or verify that it is unnecessary in this function.

From the client.
According to the team, such validation is unnecessary since the function will be called only once during the protocol launch.

INFO-3 | RESOLVED

Lack of rewards balance.

Vesting.sol: function createVestingScheduleBatch(), line 113.
Since there is no mandatory transfer of rewards to the balance of vesting, validating that `mntRemaining` >= `_freeAllocation` might be inaccurate since `mntRemaining` might contain the balance that is already being distributed. As a result, there might be a lack of rewards necessary to fulfill all the schedules.

Recommendation:
Validate how rewards are transferred to vesting or add a mandatory transfer of rewards (for example, in constructor).

Post-audit.
A function for transferring rewards was added.

INFO-4 | VERIFIED

Order of operations mismatch.

RewardsHub.sol: function distributeAccountMnt(), distributeSupplierMnt(), distributeBorrowerMnt(). In the distributeAccountMnt() function, the distribution account state is first created, and then the MNT index is updated, depending on whether the parameter account address belongs to the borrower or supplier. In special functions, distributeSupplierMnt() and distributeBorrowerMnt(), where these operations are performed separately for the supplier and for the borrower, first the index is updated and then the state is created.

Recommendation:
Verify whether the order of operations is important.

From the client.
According to the team, DistributionState doesn't depend on market indexes in any way, so the order of it's creation in relation to index updates doesn't matter.

INFO-5 | RESOLVED

Unnecessary gas spendings.

1) MToken.sol: function exchangeRateStoredInternal(), line 251.
There is a check that variable uint256 totalTokenSupply <= 0. Since such type of variable cannot be less than zero by default, more gas is spent for an unnecessary check.
Supervisor.sol: function beforeRedeemInternal(), line 220. Similar issue to the MToken contract, but with the shortfall variable.

2) Liquidation.sol: function verifyExternalData(), line 156.
seizeIndexes_[i] <= accountAssetsLength - 1 → seizeIndexes_[i] < accountAssetsLength.

Recommendation:
Change the validation so that unnecessary gas is not spent.

INFO-6 | VERIFIED

Missing time check.

MToken.sol: function reduceProtocolInterest(), line 765.
The Compound protocol has a similar function and a check of the current block.
https://github.com/compound-finance/compound-protocol/blob/a3214f67b73310d547e00fc578e8355911c9d376/contracts/CToken.sol#L1043

Recommendation:
Verify why the time check was removed.

From the client.
The accrueInterest call ensures that the market index and block are up-to-date, so this check was considered as an unnecessary gas spending.

INFO-7 | VERIFIED

Users can be blacklisted.

Supervisor.sol: function isNotBlacklisted().
At the moment, the function always returns true, although contracts check whether the user is blacklisted. If you have done this as a reserve for the future, please note that at the moment, the logic does not limit users in any way, but in the future the contract admin can update the contract as they please and deprive users of their funds.

Recommendation:
Verify the described logic.

From the client.
The blacklist system is designed to prevent accounts from bringing tainted assets into the protocol. This is a part of the upcoming AML function and is also subject to Governance. That is why, it does not prevent users from withdrawing their assets and rewards.

INFO-8 | VERIFIED

Unused code.

MntGovernor.sol: modifier onlyProposer().
This modifier is not called anywhere in the contract and outside of the contract.

Recommendation:
Remove unused code.

INFO-9 | VERIFIED

Use of the tool does not match the documentation.

MntTimelock.sol: function initialize().
The initialize function of this contract has two modifiers: initializer and onlyInitializing. According to the OpenZeppelin documentation, the initializer modifier can only be called once, even when using inheritance. But the onlyInitializing modifier is recommended to be used by parent contracts only in private initialization functions.
https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable

Recommendation:
Use the necessary functionality for its intended purpose and remove unnecessary parts.

INFO-10 | RESOLVED

The code does not represent comments in the contract.

BDSystem.sol: modifier createAgreement(), comment line 58.
Comment says that one user can not be a liquidity provider and a representative at the same time, but there is no check on arguments in the function.

Recommendation:
Add additional checks or correct the comments.

INFO-11 | RESOLVED

Variable is initialized outside of functions in the upgradable contract.

BDSystem.sol: variable initializedVersion, line 28. Assigning a value to a non-constant variable will not work outside of the initializer. As a result, the value will not be assigned. Also, the value of the variable in Solidity is `0` by default, and you don't need to assign `0` again. https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#avoid-initial-values-in-field-declarations

Recommendation:
Move the assignment to the initializer or remove the assignment.

MEDIUM-1 | VERIFIED

Unstaking might be blocked.

Buyback.sol: unstake(), line 233.
If the user is a participant, unstake process includes the updateBuybackAndVotingWeights() function call. And if the update action is paused, the function will revert and block unstaking for users. Even if the user tries to call function leave() before unstaking to stop being a participant, leaving action might also be paused, preventing the user from leaving participation. Thus users might lose access to their funds due to pauses.

Recommendation:
Call function updateBuybackAndVotingWeightsRelaxed() instead.

From client:
The relaxed version of the update function can not be used in this context, as every MNT balance decrease must go through weight recalculation. As a result, the pause of weight updates indeed blocks the withdrawal of funds and the Buyback leaving simultaneously. But both operations have their pause flags `(LEAVE_OP and UNSTAKE_OP)` and can be triggered by the same admin separately. This is part of our Gatekeeping approach, where a gatekeeper can initiate a pause of almost any user operation for the sake of security in case of hacks. The call under discussion is indeed redundant and may be removed some time in future .

INFO-1 | ACKNOWLEDGED

Custom errors should be used.

Starting from the 0.8.4 version of Solidity it is recommended to use custom errors instead of storing error message strings in storage and use “require” statements. Using custom errors is more efficient in terms of gas spending and increases code readability.

Recommendation:
Use custom errors.

From client:
Noted, but we have no time and resources to move to custom errors, as it requires to modify lots of contracts.

INFO-2 | RESOLVED

Unknown implementation of ProxyRegistry.

MinterestNFT.sol
Implementation of this contract is unknown and out of scope. Though it is set only once in constructor, this contract might be upgradable.
Since it affects the approval system, it is a crucial part of the system and its safety should be validated.

Recommendation:

Provide the implementation of ProxyRegistry which will be used during deployment so that its safety can be verified.

Post-audit:
OpenSea ProxyRegistry will be used.

INFO-3 | VERIFIED

Function is not restricted.

Buyback.sol: leaveByAmlDecision().
The function looks like it should be invoked only by Admin or by users with any other specific role, but anyone can invoke it. Though currently, the function will always revert due to validation on line 271, it still should be verified whether it should be restricted.

Recommendation:
Provide information on function usage and who can invoke function.

Response:
Current AML-related code is a placeholder for future work. We plan to plug in an AML info provider and allow anyone to exclude any blacklisted by this AML oracle user from the protocol. It makes the flow faster than working through admins or DAO. At the same time, the source of trust is shifted from the initiating user to the AML oracle.

INFO-4 | UNRESOLVED

Lack of event.

FallbackPriceFeed.sol: setPrice().
In order to keep track of historical changes of storage variables, it is recommended to emit events on every change in the setter.

Recommendation:

Emit event in setter.

From client:
According to the Minterest team, FallbackPriceFeed is a temporary contract. Thus applying any fixes in it is unnecessary.

3RD AUDIT ITERATION COMPLETE ANALYSIS

INFO-1 | ACKNOWLEDGED

Parameters should be validated.

MntGovernance.sol: initialize().
In case of __GovernorVotes_init it does not check if token is zero address. To be sure that contract will deploy properly, it is better to check if _token is not zero address.

Recommendation:

Validate function arguments.

From client:
Issue is noted, though the contracts are already live, so the code won't be changed.

INFO-2 | VERIFIED

Amount is not validated to be greater than base threshold.

Buyback.sol: stake(), _findGroupByBalance().
When the necessary group is searched for during staking, it is not validated if amount > loyaltyGroupThresholds[0] since iteration for searching is started from element with index 1. It is stated in the commentary section that amount is assumed to be greater than zero threshold. Though it should be validated if protocol allows staking amounts which are less than a base threshold.

Recommendation:

Add a validation that amount >= loyaltyGroupThresholds[0] OR verify that staking amount can be below base threshold.

From client:
Minterest team has verified that such functionality is by design. If a user is below the threshold, no loyalty reward is generated. The team has also prepared a test which covers this scenario: https://github.com/minterest-finance/protocol/blob/develop/test/buyback.test.js#L1174

	Mnt.sol	MntErrorCodes.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	MntGovernor.sol	MntTimelock.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	MntVotes.sol	ErrorCodes.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	PauseControl.sol	ChainlinkPriceOracle.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	FallbackPriceFeed.sol	Buyback.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	DeadDrop.sol	Liquidation.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	MToken.sol	MinterestNFT.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	RewardsHub.sol	Supervisor.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	WeightAggregator.sol	libraries\ErrorCodes.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	oracles\FallbackPriceFeed.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES FOR THE 1ST AUDIT REVISION

Tests written by Zokyo Security

As a part of our work assisting Minterest in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Hardhat testing framework.

The tests were based on the functionality of the code, as well as a review of the Minterest contract requirements for details about issuance amounts and how the system handles these.

Buyback contract

weights

✓ should return buyback weight 0 if not participating account

Buyback Dripper

started

✓ shouldn't refill with 0 amount (132ms)

ChainlinkPriceOracle contract

✓ shouldn't set price if caller is not admin (866ms)

DeadDrop contract

performSwap function tests

✓ performSwap should revert if caller is not gate keeper (142ms)

Liquidation contract

✓ setInsignificantLoanThreshold() shouldn't work if caller is not timelock (132ms)

Mnt

updateVoteTimestamp

✓ should revert updateTotalWeightCheckpoint if caller is not governor (65ms)

Proxy

✓ update should revert if caller is not admin (42ms)

MNTSource scen tests

✓ Refill should revert if amount = 0

✓ Refill should revert if caller is not token provider (85ms)

✓ should not sweep if param amount = 0 (55ms)

Vesting contract

✓ Withdraw should fail if amount is zero (59ms)

BDSystem

Warning: All subsequent Upgrades warnings will be silenced.

Make sure you have manually checked all uses of unsafe flags.

Main

✓ Create agreement (38ms)
✓ Remove agreement (58ms)

Revert

✓ when try to initialize contract second time
✓ when try to create agreement if sender is not admin (45ms)
✓ when try to create agreement if boost is invalid
✓ when try to create agreement if user is already provider
✓ when try to create agreement if user is already representative
✓ when try to remove agreement if sender is not admin (40ms)
✓ when try to remove agreement if user is not provider
✓ when try to check if agreement is expired

Buyback

Initialize

✓ initialized correctly

Main

✓ Get discount parameters
✓ Participate in buyback (51ms)
✓ Add users to participation in buyback
✓ Leave participation in buyback (61ms)
✓ Make user leave by admin (63ms)
✓ Make user leave by aml (59ms)
✓ Stake mnt tokens (117ms)
✓ Unstake mnt tokens (215ms)
✓ Buyback mnt tokens (125ms)
✓ Stake, buyback and unstake (235ms)

Revert

✓ when try to initialize contract second time
✓ when try to stake if user is not in whitelist
✓ when try to stake if user is not participant
✓ when try to stake if user is in blacklist (57ms)
✓ when try to stake if amount = 0 (58ms)
✓ when try to unstake if amount = 0
✓ when try to unstake if user has not staked tokens
✓ when try to stake if stake operation is paused
✓ when try to unstake if unstake operation is paused
✓ when try to buyback if sender is not distributor (39ms)
✓ when buyback amount = 0
✓ when not enough users to buyback
✓ when try to participate if user is in blacklist
✓ when try to add participant if user is in blacklist
✓ when try to add participant if buyback is happened (125ms)

- ✓ when try to add participant if sender is not admin (39ms)
- ✓ when user is already participate (55ms)
- ✓ when try to leave if user is not participant
- ✓ when try to leave by aml if user is not in blacklist
- ✓ when try to make user leave if sender is not admin
- ✓ when try to leave if leave operation is paused
- ✓ when try to validate pause is not gatekeeper role
- ✓ when try to validate unpause is not admin

BuybackDripper

Main

- ✓ set period duration
- ✓ set period rate
- ✓ refill contract
- ✓ drip

Revert

- ✓ when try to set period duration to zero
- ✓ when try to set period rate too high
- ✓ when try to refill for zero tokens
- ✓ when try to drip after drip
- ✓ when try to set period duration if sender is not timelock role
- ✓ when try to set period rate if sender is not timelock role
- ✓ when try to refill if sender is not token provider

Compound -- MToken

Compound differ

- ✓ Transfer tokens (56ms)
- ✓ Balance of underlying calculates correctly
- ✓ Return account snapshot
- ✓ Lend works correctly (43ms)
- ✓ Redeem works correctly (80ms)
- ✓ Borrow works correctly (82ms)
- ✓ Repay works correctly (124ms)
- ✓ Transfer tokens that is not used by protocol
- ✓ Deposit interest to protocol

MNTSource

Deploy

- ✓ deployed correctly

Main

- ✓ Drip
- ✓ Refill
- ✓ Sweep

Revert

✓ when refill amount is zero
✓ when sweep amount is zero
✓ when sweep amount is more than contract balance
✓ when try to sweep if user is not admin
✓ when try to refill if user is not token provider

MToken

Initialize

✓ initialized correctly

Main Token functions

✓ totalSupply (43ms)
✓ approve (56ms)
✓ transfer (46ms)
✓ transferFrom (65ms)
✓ balanceOf Underlying token
✓ transfer tokens that is not connected to protocol

Lending

✓ lend tokens and receive MToken (47ms)

Redeeming

✓ redeem tokens and receive underlying tokens (90ms)
✓ redeem tokens by aml (88ms)
✓ redeem underlying tokens (89ms)

Borrowing

✓ borrow tokens (146ms)

Repaying

✓ repay borrowed tokens (158ms)
✓ repay borrowed tokens for user (174ms)
✓ auto liquidate borrowed tokens for user (163ms)
✓ auto liquidate borrowed tokens for user (181ms)

Protocol interest

✓ add protocol interest (38ms)

Admin functionality

✓ set protocol interest factor
✓ reduce protocol interest
✓ set interestRate model
✓ set flash loan max shares
✓ set flash loan fee

Revert

✓ when try to initialize contract second time
✓ when try to transfer to same address as sender
✓ when try to reduce protocol interest if reduce amount is too big
✓ when try to reduce protocol interest if reduce amount is more that total interest
✓ when try to set protocol interest factor if user is not timelock role

✓ when try to set protocol interest factor if user is not timelock role
✓ when try to reduce protocol interest if user is not admin
✓ when try to set interest rate model if user is not admin (241ms)
✓ when try to set max flash loan share if user is not admin (444ms)
✓ when try to sweep tokens if user is not admin (308ms)
✓ when try to set flash loan fee if user is not timelock role (85ms)
✓ when try to sweep tokens if token is underlying
✓ when try to set max flash loan share if new share is too big
✓ when try to set flash loan fee if new fee is too big
✓ when try to set protocol interest factor if new factor is less that old
✓ when try to auto repay borrow if protocol interest is low (63ms)

RewardsHub

Initialize

✓ initialized correctly

Main

✓ Grant user mnt tokens
✓ Set mnt emission rate (50ms)
✓ Distribute mnt tokens (612ms)

Revert

✓ when try to initialize contract second time (290ms)
✓ when try to initialize market is not supervisor
✓ if try to distribute mnt when token is not on listed on market
✓ when try to accrue buyback reward is not buyback contract
✓ when try to withdraw more than contract balance
✓ when try to withdraw if withdraw operation is paused
✓ when not admin try to grant user (64ms)
✓ when grant amount is 0
✓ when try to grant if contract token amount is 0
✓ if try set mnt emission rate when token is not on listed on market
✓ when not timelock role try set mnt emission rate (45ms)
✓ when try to validate pause is not gatekeeper role
✓ when try to validate unpause is not admin

Supervisor

Initialize

✓ initialized correctly

Main

✓ Enable mToken to market
✓ Enable/Disable token as collateral (118ms)

Revert

✓ when try to initialize contract second time
✓ when not admin try to add token to market
✓ when not timelock role try to set utilisation factor

- ✓ when not timelock role try to set liquidation fee
- ✓ when not gatekeeper role try to set market borrow caps
- ✓ when token is already listed in market
- ✓ when try to enable as collateral if token is not listed in market
- ✓ when try to disable collateral if user has borrows
- ✓ when token is not listed in market before lend
- ✓ when token is not listed in market before redeem
- ✓ when token is not listed in market before borrow
- ✓ when token is not listed in market before repay borrow
- ✓ when token is not listed in market before flash loan
- ✓ when token is not listed in market when try to set utilisation factor
- ✓ when token is not listed in market when try to set liquidation fee
- ✓ when check for redeem verify if redeem tokens == 0 or redeem amount != 0
- ✓ when try to enable as collateral if user is in blacklist
- ✓ when user is in blacklist before lend
- ✓ when user is in blacklist before borrow
- ✓ when user is in blacklist before transfer
- ✓ when user is in blacklist before flash loan
- ✓ when user is not in whitelist before lend
- ✓ when user is not in whitelist before redeem
- ✓ when user is not in whitelist before borrow
- ✓ when user is not in whitelist before repay borrow
- ✓ when try to set utilisation factor more that max value
- ✓ when new liquidation fee = 0
- ✓ when try to validate pause is not gatekeeper role
- ✓ when try to validate pause if token is not listed in market
- ✓ when try to validate unpause is not admin
- ✓ when try to validate unpause is not admin
- ✓ when try to check before lend if lend operation is paused
- ✓ when try to check before borrow if borrow operation is paused
- ✓ when try to check before flash loan if flash loan operation is paused
- ✓ when try to check before transfer if transfer operation is paused

Whitelist

Main

- ✓ Add member
- ✓ Remove member
- ✓ Set maximum members in whitelist

Revert

- ✓ when try to add member that is in whitelist already
- ✓ when try to add member if list is full
- ✓ when try to remove not existed user
- ✓ when try to set maximum members if users is more that new max

Mnt

Initialize

✓ initialized correctly

Main

✓ Transfer tokens
✓ Update voting weight
✓ Set governor
✓ Update total weight checkpoint
✓ Set maximal non voting period
✓ Get user last activity
✓ Check supported interface

Revert

✓ when try to initialize contract second time
✓ when try to transfer tokens if user is in blacklist
✓ when try to update votes on zero address
✓ when try to update weight if user is not governor
✓ when try to update vote timestamp if user is not governor
✓ when try to set governor if user is not admin
✓ when try to set governor to zero address
✓ when try to set governor second time
✓ when try to set voting period if user is not admin
✓ when try to set voting period in range < 90 days
✓ when try to set voting period in range > 365 days * 2
✓ when try to set same voting period

Mnt Governor

Main

✓ Create proposal, vote and execute (128ms)
✓ Cancel proposal (52ms)
✓ Interface support

Revert

✓ when try to initialize contract second time
✓ when try to propose if user is not proposer role
✓ when try to cancel propose if user is not cancel role

Scenario -- Multiple users use governance

Governance process

✓ Setup (1321ms)
✓ Create proposal
✓ Wait 100 blocks to start vote
✓ Voting process (Defeat proposal) (177ms)
✓ Wait 100 blocks to end proposal

Scenario -- Multiple users use protocol

Users lend/borrow/repay

- ✓ Transfer tokens (88ms)
- ✓ Approve tokens (53ms)
- ✓ Lend tokens (508ms)
- ✓ Borrow tokens (723ms)
- ✓ Repay tokens (532ms)
- ✓ Redeem tokens (698ms)

221 passing (27s)

FILE	% STMTS	% BRANCH	% FUNCS
BDSystem.sol	100	100	100
BuyBack.sol	100	95.83	100
BuyBackDripper.sol	100	95	100
DeadDrop.sol	80	98.33	100
EmissionBooster.sol	100	95.89	100
Interconnector.sol	100	100	100
InterconnectorLeaf.sol	100	100	100
KinkMultiplierModel.sol	100	100	100
Liquidation.sol	100	98.48	100
MEther.sol	100	100	100
MinterestNFT.sol	100	100	100
MintProxy.sol	100	100	100
MNTSource.sol	100	100	100
MToken.sol	100	84.09	100

FILE	% STMTS	% BRANCH	% FUNCS
MTokenStorage.sol	100	100	100
RewardsHub.sol	100	97.46	97.37
Supervisor.sol	99.17	92.42	94.14
Vesting.sol	100	92.31	100
WeightAggregator.sol	100	100	100
Whitelist.sol	100	96.15	100
governance			
.../Mnt.sol	100	100	100
.../MntGovernor.sol	92.59	85.71	94.12
.../MntTimelock.sol	100	75	100
.../MntVotes.sol	100	94.12	100
	98.15	88.71	98.53
libraries			
.../ErrorCodes.sol	100	100	100
.../PauseControl.sol	100	75	100
.../ProtocolLinkage.sol	100	100	100
	100	91.67	100
oracles			
ChainlinkPriceOracle.sol	100	100	100
FallbackPriceFeed.sol	100	100	100
	100	100	100
All files	**98.97**	**96.74**	**99.4**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES FOR THE 2ND AUDIT REVISION (BY ZOKYO TEAM)

FallbackPriceFeed

- ✓ Admin should set new price
- ✓ Not admin should not set new price (109ms)
- ✓ User should view round data by round ID
- ✓ User should view latest round data

MinterestNFT

- ✓ Should not deployed if proxyRegistry address is AddressZero (123ms)
- ✓ Should not deployed if admin address is AddressZero (117ms)
- ✓ Gatekeeper should mint (with tier = 0)
- ✓ Gatekeeper should mint (with tier > 0) (42ms)
- ✓ Not Gatekeeper should not mint (76ms)
- ✓ Gatekeeper should mint batch (with tier = 0) (46ms)
- ✓ Not Gatekeeper should not mint batch (79ms)
- ✓ Gatekeeper should not mint batch if input data arrays have different length
- ✓ User should safe transfer (50ms)
- ✓ User should safe transfer batch (64ms)
- ✓ Admin should set URI (for existent token)
- ✓ Admin should set URI (for non-existent token)
- ✓ Not Admin should not set URI (56ms)
- ✓ User should view 'ApprovedForAll' status
- ✓ User should view next id to be minted
- ✓ User should view supports interface

WeightAggregator

- ✓ User should view account funds
- ✓ User should view loyalty factor
- ✓ User should view buyback weight if user is not participating
- ✓ User should view buyback weight if user is participating
- ✓ User should view voting weight if user is participating (59ms)

.getVotingWeight():

- ✓ User should view voting weight if user is not participating (weight = 0)
- ✓ User should view voting weight if user is member (weight = 0) (56ms)
- ✓ User should view voting weight if user is member (weight != 0) (335ms)

Buyback

Main

- ✓ Stake mnt tokens
- ✓ Stake mnt tokens second time
- ✓ Stake mnt tokens second time and appear in the new group
- ✓ Unstake mnt tokens
- ✓ Stake, buyback and unstake
- ✓ Stake, buyback, leave and unstake

Setters

✓ Set new loyalty parameters
✓ Set new loyalty strata
✓ Set new loyalty groups

Revert

✓ when used wrong arguments for initializer
✓ when rewards hub is address(0)
✓ when paused
✓ when use wrong arguments for setters
✓ when try to set parameters if sender is not admin
✓ when try to initialize contract second time

RewardsHub

✓ Distribute mnt tokens for one token
✓ Distribute mnt tokens for all tokens
✓ Withdraw
✓ Emit event RepresentativeRewardAccrued
✓ Set new emission rate

Revert

✓ when try to initialize contract second time
✓ when try to accrue buyback reward is not buyback contract

Supervisor

✓ Check blacklist if requested
✓ borrow with borrow cup
✓ calculate account liquidity

Revert

✓ when in blacklist

DeadDrop

Setters

✓ add market
✓ add router
✓ set liquidation
✓ set receiver
✓ set bot

Removers

✓ remove market
✓ remove receiver
✓ remove bot

Logic

✓ withdraw

Bot functions

✓ perform swap (49ms)
✓ withdraw to protocol interest (39ms)

Liquidations

✓ initialize and finalize liquidation

Revert

- ✓ when not Gatekeeper try to perform swap
- ✓ when not Gatekeeper try to withdraw to protocol interest (39ms)
- ✓ when not Gatekeeper try to update processing state (38ms)
- ✓ when not Gatekeeper try to finalize liquidation (39ms)
- ✓ when not Admin try to withdraw
- ✓ when not Admin try to add market
- ✓ when not Admin try to add router
- ✓ when not Admin try to set liquidation address
- ✓ when not Admin try to add receiver
- ✓ when not Admin try to add bot
- ✓ when not Admin try to remove market
- ✓ when not Admin try to remove receiver
- ✓ when not Admin try to remove bot
- ✓ when swap router is not set
- ✓ when tokenIn is zero
- ✓ when tokenOut is zero
- ✓ when amount is zero
- ✓ when token is not allowed
- ✓ when try to swap if preconditions not met
- ✓ when market is not set
- ✓ when try to withdraw to protocol interest if preconditions not met
- ✓ when sender is not liquidation
- ✓ when try to finalize liquidation if preconditions not met
- ✓ when try to withdraw if sender is wrong receiver
- ✓ when try to withdraw if not enough tokens
- ✓ when try to add market as zero address
- ✓ when try to add router as zero address
- ✓ when try to set same router
- ✓ when try to add liquidation as zero address
- ✓ when try to set same liquidation
- ✓ when try to add receiver as zero address
- ✓ when try to set same receiver
- ✓ when try to add bot as zero address
- ✓ when try to set same bot
- ✓ when try to remove unknown market
- ✓ when try to remove receiver if it is not receiver
- ✓ when try to remove bot if it is not bot

Liquidation

Setters

- ✓ sets healthy factor limit
- ✓ sets DeadDrop address
- ✓ sets loan threshold
- ✓ sets processing state

Liquidations

- ✓ calculate liquidation amounts
- ✓ liquidate unsafe loan (166ms)
- ✓ accrue (109ms)
- ✓ seize (187ms)
- ✓ repay (203ms)
- ✓ approveBorrowerHealthyFactor (131ms)

Revert

- ✓ when not liquidator try to liquidate unsafe loan
- ✓ when sets same healthy factor limit
- ✓ when try to set DeadDrop address as zero
- ✓ when sets same processing state
- ✓ when not Timelock tries to set healthy factor
- ✓ when not admin try to set DeadDrop address
- ✓ when not Timelock tries to set loan threshold
- ✓ when not admin tries to set processing state

MToken

- ✓ check flash loans variables
- ✓ check supported interfaces
- ✓ when try to check flash fee if token is not underlying

Buyback Scenario -- check Loyalty functionality

Loyalty

- ✓ Get into loyalty group (273ms)
- ✓ Check loyalty info
- ✓ Unstake to check if loyalty will change (67ms)

FILE	% STMTS	% BRANCH	% FUNCS
BuyBack.sol	98.21	96.27	100
DeadDrop.sol	95	98.75	100
Liquidation.sol	91.26	73.68	100
MinterestNFT.sol	93.75	90	100
MToken.sol	88.89	65.91	96.36
RewardsHub.sol	98.54	83.85	97.67
Supervisor.sol	96.67	82.58	88.89
WeightAggregator.sol	100	100	100
FallbackPriceFeed.sol	100	100	100
ErrorCodes.sol	100	100	100
All files	**96.23**	**89.1**	**98.29**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES FOR THE 3RD AUDIT REVISION (BY ZOKYO TEAM)

```
  Main
    ✓ createVestingScheduleBatch (60ms)
    ✓ revokeVestingSchedule (110ms)
    ✓ addToDelayList (40ms)
    ✓ removeFromDelayList (41ms)
    ✓ getReleasableWithoutCliff
    ✓ getReleasableWithoutCliff
    ✓ vestedAmount
    ✓ cannot refill vesting with zero amount
    ✓ only token provider can refill
    ✓ sweep (47ms)
    ✓ canot sweep zero
  withdraw
    ✓ withdraw half after half time (45ms)
    ✓ cannot withdraw when user on delay list (39ms)
    ✓ cannot withdraw with no vesting schedule
    ✓ withdraw all tokens after time has passed (44ms)
    ✓ cannot withdraw zero amount (44ms)
    ✓ cannot withdraw more than unreleased
  view
    ✓ endOfVesting
    ✓ lockedAmount

  19 passing (2s)
```

For the third audit iteration, Zokyo Security team has updated previously written tests for Buyback.sol, Liquidation.sol, Vesting.sol, DeadDrop.sol, as well as added more test cases to validate the flow of vesting. Updated tests contain all the necessary checks for the new logic of smart contracts.

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by the Minterest team

As a part of our work assisting Minterest in verifying the correctness of their contract code, our team has checked the complete set of tests prepared by the Minterest team.

We need to mention that the original code has a significant original coverage with testing scenarios provided by the Minterest team. All of them were also carefully checked by the team of auditors.

Account liquidity

liquidity

✓ fails if a price has not been set (191ms)
✓ allows a borrow up to utilisationFactor, but not more (678ms)
✓ allows enabling as collateral 3 markets, supplying to 2 and borrowing up to utilisationFactor in the 3rd (1478ms)

getAccountLiquidity

✓ returns 0 if not 'in' any markets

getHypotheticalAccountLiquidity

✓ returns 0 if not 'in' any markets
✓ returns utilisation factor times dollar amount of tokens lended in a single market (238ms)

assetListTest

enableAsCollateral

✓ properly emits events (151ms)
✓ adds to the asset list only once (531ms)
✓ the market must be listed for add to succeed (211ms)
✓ returns a list of codes mapping to account's ultimate membership in given addresses (153ms)

disableAsCollateral

✓ doesn't let you disable if you have a borrow balance (167ms)
✓ rejects unless redeem allowed (250ms)
✓ accepts when you're not in the market already (159ms)
✓ properly removes when there's only one asset (192ms)
✓ properly removes when there's only two assets, removing the first (205ms)
✓ properly removes when there's only two assets, removing the second (205ms)
✓ properly removes when there's only three assets, removing the first (215ms)
✓ properly removes when there's only three assets, removing the second (204ms)
✓ properly removes when there's only three assets, removing the third (206ms)

enabling from beforeBorrow

✓ enable when called by a mtoken (174ms)
✓ reverts when called by not a mtoken (108ms)

✓ adds to the asset list only once (301ms)

Business development system tests

✓ Should initialize with correct args

✓ Should revert if initialized twice

createAgreement

✓ Should revert if condition (1 + providerBoost) * (1 + representativeBoost) <= 150% is not met

✓ Should revert if condition (1 + providerBoost) * (1 + representativeBoost) <= 150% is not met

✓ Should revert if condition (1 + providerBoost) * (1 + representativeBoost) <= 150% is not met

✓ Should revert if condition (1 + providerBoost) * (1 + representativeBoost) <= 150% is not met

✓ One account at one time can be a liquidity provider once (83ms)

✓ One account can't be a liquidity provider and a representative at the same time (94ms)

✓ Should revert if caller does not have DEFAULT_ADMIN_ROLE permission (57ms)

✓ Should call distributeAllMnt() from rewardsHub (155ms)

✓ Representative can have multiple agreements (183ms)

✓ Should create Agreement between provider1 and representative1 and emit AgreementAdded event with correct args (98ms)

✓ Shouldn't give emission boost for LP if agreement was created in this block (171ms)

removeAgreement

✓ Should revert if `provider1` and `representative1` doesn't have agreement

✓ Should revert if caller does not have DEFAULT_ADMIN_ROLE permission (55ms)

✓ Should call distributeAllMnt function from rewardsHub (207ms)

✓ Should remove Agreement between provider1 and representative1 and emit AgreementEnded event with correct args (73ms)

isAgreementExpired()

✓ should revert if the account is not liquidity provider

✓ should return false if the agreement hasn't expired

✓ should return true if the agreement has expired

Buyback contract

Initialize

✓ Should initialize correctly (45ms)

✓ Should failed if called second time

staked funds

✓ Should revert if account tries to unstake in the same block when he staked (131ms)

✓ should stake with discount (278ms)

✓ should revert if account isn't whitelisted (130ms)

✓ should revert if account is not participating in buyback

✓ should revert when no balance (130ms)

✓ should revert when stake function not enabled

✓ should unstake and recalculate discounted if account is participating in buyback (340ms)

✓ should unstake all staked funds if MaxUint256 is passed to the method unstake() (211ms)

✓ should unstake correct if amount_ == staked.amount (224ms)

✓ should unstake and not recalculate discounted if account is not participating in buyback (261ms)

✓ unstake should claim reward if account is participating in buyback (280ms)
✓ should not unstake if operation is paused (184ms)
✓ should not unstake more than staked (134ms)
✓ should revert if amount is equal to zero (127ms)

discount

✓ discount parameters
✓ same stake amount gets more discounted with time (42ms)
✓ in far future discount should not be zero (40ms)
✓ older stakes should add less weight (144ms)

weights

✓ stake should increase weight sum by the same weight (151ms)
✓ unstake should decrease weight sum (325ms)
✓ unstake should decrease weight by discounted amount (335ms)
✓ unstake should not leave weight greater than staked amount (319ms)
✓ more staked - more weight (242ms)
✓ weights must sum (265ms)
✓ vesting revokeVestingSchedule should update buyback weight (315ms)

buyback

✓ should increase buyback index (105ms)
✓ should not buyback more than balance
✓ should claim reward (222ms)
✓ should not update if operation is paused (53ms)
✓ should not leave if operation is paused (57ms)
✓ new members should not receive reward for previous buyback (261ms)
✓ should fail if caller has no distributor role (110ms)
✓ should fail if nothing to distribute
✓ should revert if there is no participants (190ms)

vesting

✓ should add flat weight w/o staked value (88ms)
✓ should increase weight with time (227ms)
✓ vesting withdraw should reduce weight (247ms)
✓ vesting revokeVestingSchedule should reduce weight (245ms)
✓ should work alongside with staked funds (261ms)
✓ vesting withdraw without participation in buyback (87ms)

weight from MNT distribution

✓ withdraw resets weight (323ms)
✓ updateBuybackAndVotingWeights() shouldn't distribute MNT tokens (68ms)

participating in buyback

✓ participate() should fail if account already participating
✓ leave() should fail if account is not participating in buyback
✓ leaveOnBehalf() should fail if called by non-admin account (68ms)
✓ leaveOnBehalf() should fail if participant is not active (47ms)

✓ leaveOnBehalf() shouldn't fail (58ms)
✓ leaveByAmlDecision() should fail if AML system is disabled
✓ leaveByAmlDecision() should fail if AML system is enabled and participant is not in AML black list
✓ leaveByAmlDecision() shouldn't fail if AML system is enabled and participant is in AML black list
✓ leaveByAmlDecision() shouldn't fail if participant is active in BB system
✓ should not stake when account is not participating in buyback
✓ leaving buyback participation before buyback event should not fail (49ms)
✓ leaving buyback participation should reduce weights (566ms)
✓ participate buyback emit event (117ms)
✓ participate adds discounted amount for sender if he has staked funds (1077ms)
✓ participate doesn't add discounted amount for sender if he doesn't have staked funds (140ms)
✓ leaving buyback participation with zero stake should not emit events other than leave (55ms)
✓ leaving buyback participation with positive stake should emit leave event (180ms)

participateOnBehalf

✓ participateOnBehalf should fail when buyback drips already happened (193ms)
✓ participateOnBehalf can call only admin (73ms)
✓ participateOnBehalf should work (48ms)

discounting curve logging and validation

- get curve points from day zero up to two points after the kink point, daily
- get all points around the kink point, two minutes
- plot polynomial vs e^-kt daily
- verify that buyback will not accumulate dust weight over time

discounting curve model

✓ polynomial should match e^-kx with precision up to 0.002 during first 4 years, fortnightly (1144ms)
✓ buyback with small discount should affect bigger weight properly (640ms)
✓ should revert when stake is discounted to zero (259ms)
✓ staking dust gives full buyback if no one else is there (459ms)
✓ sequential buyback restakes should decrease total weight properly (4130ms)
✓ verify that account gets buyback reward even with miniscule share (667ms)

Voting power

✓ restake() emits new VotesUpdated event (119ms)
✓ leave() emits new VotesUpdated event and changes voting power (212ms)
✓ leaveOnBehalf() emits new VotesUpdated event and changes voting power (201ms)
✓ leaveByAmlDecision() emits new VotesUpdated event and changes voting power (184ms)
✓ updateBuybackAndVotingWeights() should update voting weight (256ms)
✓ vesting revokeVestingSchedule should update voting weight (196ms)

Anti-Money-Laundering tests

✓ blacklisted users can't participate (88ms)
✓ blacklisted users can't participateOnBehalf (39ms)
✓ blacklisted users can't stake
✓ blacklisted users can unstake (98ms)
✓ blacklisted users can updateBuybackAndVotingWeights (56ms)

✓ blacklisted users can leave
✓ blacklisted users can leaveOnBehalf (40ms)

Buyback Dripper

✓ deploy should allow buyback to transferFrom
✓ first drip should set initial values (106ms)

started

✓ setPeriodRate should change nextPeriodRate
✓ setPeriodRate should be in range (38ms)
✓ setPeriodRate can only be used by an TIMELOCK (84ms)
✓ drip calls buyback (250ms)
✓ can't drip in same time slot as start (137ms)
✓ drip maximum once per hour (142ms)
✓ drip for multiple time slots (126ms)
✓ new period starts after 730 hours (143ms)
✓ reset period if there was no drips for period duration (441ms)
✓ period duration can be changed (215ms)
✓ setPeriodDuration can only be used by an TIMELOCK (70ms)
✓ setPeriodDuration reverts if period is set to zero
✓ drip only for current period (597ms)
✓ unauthorized entity can't refill (65ms)

drip per hour changes

✓ from refill (47ms)
✓ from period rate
✓ from duration
✓ and does not change from balance

ChainlinkPriceOracle contract

✓ should get price from FallbackPriceFeed (724ms)

CahinlinkFeedMock checks

✓ reportNewRound() should work
✓ latestRoundData() should work (45ms)
✓ getRoundData() should work (48ms)
✓ decimals() should work
✓ description() should work
✓ version() should work

CahinlinkPriceOracle positive tests

✓ should return price value and decimals for DAI (scaled by 1e18) (60ms)
✓ should return price value and decimals for USDC (scaled by 1e30) (58ms)
✓ should return price value and decimals for REP (scaled by 1e18) (51ms)
✓ should set timestampThreshold (97ms)

CahinlinkPriceOracle negative tests

✓ getRoundData() for incorrect roundId should be reverted
✓ setTokenConfig() with 0 mToken address

- ✓ setTokenConfig() with 0 feed address
- ✓ setTokenConfig() with 0 decimals
- ✓ setTokenConfig() with 0 reporterMultiplier
- ✓ setTokenConfig() with decimals > 26
- ✓ setTokenConfig() with reporterMultiplier > 26
- ✓ setTokenConfig() with 0 timestampThreshold
- ✓ get price of unknown token
- ✓ get underlying price for 0 mToken address
- ✓ get price for 0 asset address
- ✓ oracle return negative price (60ms)
- ✓ oracle return zero price (56ms)
- ✓ oracle return wrong timestamp (expired) (59ms)
- ✓ oracle return wrong timestamp (from the future) (50ms)
- ✓ attempt to set TokenConfig by non-admin should be reverted (71ms)
- ✓ oracle return wrong round ID (83ms)
- ✓ lookup test (947ms)

DeadDrop contract

Admin functions tests

- ✓ withdraw should work (55ms)
- ✓ withdraw should revert if called by not-admin (64ms)
- ✓ withdraw should revert if receiver is not in the whitelist
- ✓ withdraw should revert if receiver there is not enough liquidity (47ms)
- ✓ addAllowedMarket should work
- ✓ addAllowedMarket should revert if called by not-admin (59ms)
- ✓ addAllowedMarket should revert if called with zero address arg
- ✓ setRouterAddress should work
- ✓ setRouterAddress should revert if called by not-admin (58ms)
- ✓ setRouterAddress should revert if called with zero address arg
- ✓ setRouterAddress should revert if router already added (43ms)
- ✓ addAllowedReceiver should work
- ✓ addAllowedReceiver should revert if called by not-admin (54ms)
- ✓ addAllowedReceiver should revert if called with zero address args
- ✓ addAllowedReceiver should revert if receiver already added (42ms)
- ✓ addAllowedBot should work
- ✓ addAllowedBot should revert if called by not-admin (54ms)
- ✓ addAllowedBot should revert if called with zero address args
- ✓ addAllowedBot should revert if receiver already added (45ms)
- ✓ removeAllowedMarket should work (48ms)
- ✓ removeAllowedMarket should revert if called by not-admin (56ms)
- ✓ removeAllowedMarket should revert if called with incorrect underlying
- ✓ removeAllowedReceiver should work (59ms)
- ✓ removeAllowedReceiver should revert if called by not-admin (57ms)

✓ removeAllowedReceiver should revert if called with incorrect underlying
✓ removeAllowedBot should work
✓ removeAllowedBot should revert if called by not-admin (52ms)
✓ removeAllowedBot should revert if called with incorrect underlying

Withdraw to market functions tests

✓ withdrawToProtocolInterest should work (96ms)
✓ withdrawToProtocolInterest should work if called by bot (139ms)
✓ withdrawToProtocolInterest should revert if called neither by admin nor by the bot (58ms)
✓ withdrawToProtocolInterest should revert if receiver there is not enough liquidity (109ms)
✓ withdrawToProtocolInterest should revert for incorrect underlying token

performSwap function tests

✓ performSwap should revert if swapRouter was not set
✓ performSwap should revert if tokenOut address is zero
✓ performSwap should revert if tokenIn address is zero
✓ performSwap should revert tokenIn amount <= 0
✓ performSwap should revert if tokens are unsupported (57ms)

TestProtocol deploy

✓ Supported markets should be correctly set (1335ms)
✓ MNT initial balances
✓ PriceOracle should set correct prices (1411ms)
✓ Supervisor setters should set correct params (1518ms)
✓ All contract setters should set correct params
✓ MNT contract address should be set

EmissionBoosts

✓ Should initialize contract with correct args
✓ Should revert if initialized twice
✓ getCurrentAccountBoost (141ms)

onMintToken

✓ reverts when called by a not token
✓ revert if tier does not exist
✓ revert if tier was activated (161ms)
✓ NFT mint should change tier bitmap (71ms)
✓ should call distribute MNT only if accounts tiers change (296ms)

onTransferToken

✓ check test conditions
✓ reverts when called by a not token
✓ should skip zero amount (66ms)
✓ should move tiers (397ms)
✓ should call distribute MNT only if accounts tiers change (295ms)

enableEmissionBoosting()

✓ Should revert if caller is not whitelist contract
✓ Should revert if whitelist address is zero
✓ Should enable emission boosting and emits the necessary events when caller is admin (172ms)

createTiers()
✓ fails if not called by admin (60ms)
✓ fails if emissionBoost is zero
✓ fails if emissionBoost greater than 0.5
✓ fails if the lengths of the arrays don't match
✓ fails if end boost block is less than the current block number (45ms)
✓ fails if tiers limit reached (155ms)
✓ succeeds, creates new Tiers and emits an event (326ms)
enableTiers()
✓ fails if not called by admin (57ms)
✓ fails if try to enable zero Tier
✓ fails if tier doesn't exist
✓ fails if end boost block is less than the current block number
✓ fails if tier was already enabled (231ms)
✓ succeeds, enables Tiers and emits an event (531ms)
isTierActive()
✓ should work correctly (239ms)
isAccountHaveTiers()
✓ Should work correct (120ms)
getMarketSpecificData()
✓ should return correct values (52ms)
calculateEmissionBoost()
✓ should fail if user has NFT and passed marketIndex == 0
✓ should fail if passed user last updated index that less than initial index
✓ should fail if user doesn't have NFT token and marketIndex < userLastUpdatedIndex
✓ should return zero if user doesn't have NFT token
✓ should return zero if emission boosting is disabled
✓ should return correct value (blocks: 1200 - 1800) (81ms)
✓ should return correct value (blocks: 2000 - 3200) (120ms)
✓ should return correct value (blocks: 500 - 5000) (173ms)
Methods for calculating and updating indexes in markets
✓ Should return first tier to expire if no indexes really updated for all 15 tiers (2315ms)
✓ Should return correctly if there are 15 tiers to verify and index updated after each expiration (3075ms)
✓ Should return correctly if there are 15 tiers but all of them are not enabled till the last expiration date (836ms)
✓ Should return correctly if more than one tier skipped (2422ms)
✓ Should return tier update order correctly if more than one tier skipped and indexes updated (2119ms)
✓ Should return tier correctly if some random tier index already updated (2431ms)
✓ create a new Tier, which has the closest number of the end emission boost block, should return this new tier (3341ms)

- calculate gas cost for 15 tiers
✓ updateSupplyIndexesHistory() should fail if passed index less than initial (49ms)
✓ updateSupplyIndexesHistory() should fail if caller isn't the Supervisor
✓ updateSupplyIndexesHistory() should return correctly across all boost periods (8446ms)
✓ updateSupplyIndexesHistory() should emit events (526ms)
✓ updateBorrowIndexesHistory() should fail if passed index less than initial (43ms)
✓ updateBorrowIndexesHistory() should fail if caller isn't the Supervisor
✓ updateBorrowIndexesHistory() should emit events (526ms)

Uninitialized NFT tiers

✓ getNextTier Should return zero if only zero tier available

EmissionBooster constructor

✓ initialize reverts when token address is zero
✓ initialize reverts when admin address is zero

Emission system tests

setMntEmissionRates()

✓ should correctly set different emission rates and emit necessary events (56ms)
✓ should update market index correct when newEmissionRate != current (138ms)
✓ should correctly drop market from emission system if called by admin (74ms)
✓ should not drop market from emission system if not called by admin (97ms)
✓ should fail if market is not listed (38ms)

calculateUpdatedMarketState()

✓ shouldn't update index if emissionRate is equal to zero (44ms)
✓ shouldn't update state if deltaBlocks is equal to zero
✓ should update state correctly (39ms)

getUpdatedMntSupplyIndex() and getUpdatedMntBorrowIndex()

✓ getUpdatedMntSupplyIndex should return correct state (99ms)
✓ getUpdatedMntBorrowIndex should return correct state (85ms)
✓ getUpdatedMntBorrowIndex should fail if marketBorrowIndex == 0 (87ms)

updateMntSupplyIndex()

✓ should not call updateSupplyIndexesHistory() if minterestNFT contract is not connected
✓ should not call updateSupplyIndexesHistory() if emission boosting is turned off
✓ should call updateSupplyIndexesHistory() (76ms)
✓ should calculate and update mntSupplyState if minterest NFT is installed (112ms)
✓ should calculate mnt supply index correctly (98ms)
✓ should not update index if no blocks passed since last accrual (83ms)
✓ should not matter if the index is updated multiple times (1332ms)

updateMntBorrowIndex()

✓ should not call updateBorrowIndexesHistory() if minterestNFT contract is not connected
✓ should not call updateBorrowIndexesHistory() if emission boosting is turned off
✓ should call updateBorrowIndexesHistory() (122ms)
✓ should calculate and update mntBorrowState if minterest NFT is installed (112ms)
✓ should calculate mnt borrower index correctly (125ms)

✓ should not update index if no blocks passed since last accrual (63ms)
✓ should not update index if mnt emissionRate is 0 (148ms)

distributeSupplierMnt()

✓ shouldn't give NFT boost or representative bonus if boosts are not enabled (108ms)
✓ should give NFT boost if user doesn't have LP boost (111ms)
✓ should calculate NFT boost correctly (110ms)
✓ should give only representative bonus (without NFT) and calculate correctly, also emit event RepresentativeRewardAccrued (173ms)
✓ should transfer MNT and update supply state correctly for first time account (104ms)
✓ should update mnt accrued and supply index for repeat account (119ms)
✓ should split nft boost correctly (123ms)
✓ should split bdr boost correctly (171ms)

distributeBorrowerMnt()

✓ shouldn't give NFT boost or representative bonus if boosts are not enabled (115ms)
✓ should give NFT boost if user doesn't have LP boost (112ms)
✓ should calculate NFT boost correctly (111ms)
✓ should give only representative bonus (without NFT) and calculate correctly, also emit event RepresentativeRewardAccrued (153ms)
✓ should update borrower state checkpoint but not mntAccrued for first time account (97ms)
✓ should update borrower state checkpoint correctly for repeat time account (149ms)
✓ should split nft boost correctly (121ms)
✓ should split bdr boost correctly (160ms)

distributeAllMnt()

✓ should update MNT supply index only for markets with non-zero userSupply balance (248ms)
✓ should update MNT borrow index only for markets with non-zero userBorrow balance (263ms)
✓ should distribute MNT tokens only for markets with non-zero user balances (328ms)

distributeMnt()

✓ should accrue MNT correctly (519ms)
✓ should accrue MNT for multiple suppliers only (1354ms)
✓ should accrue MNT for multiple borrowers only, primes uninitiated (892ms)
✓ should accrue the expected amount when holders and mTokens are duplicated (1770ms)
✓ should revert when a market is not listed

updateAndGetMntIndexes()

✓ should work correctly (153ms)

Flash Loans

maxFlashLoan

✓ returns share of balance (52ms)
✓ returns zero on other tokens

flashFee

✓ returns share of amount
✓ wrong token arg reverts

change params

✓ setFlashLoanMaxShare (41ms)
✓ setFlashLoanFeeShare (40ms)
✓ should revert if args are greater than 1e18 (66ms)
✓ should be timelock only (77ms)
✓ should be admin only (76ms)

flashLoan

✓ should call callback (129ms)
✓ should transfer tokens (134ms)
✓ there should be tokens inside callback (361ms)
✓ should increase protocol interest by fee (118ms)
✓ should return true (117ms)
✓ should accrue market interest (167ms)
✓ should revert if is paused in supervisor (85ms)
✓ should revert if not enough fee (124ms)
✓ should revert if not enough allowance (104ms)
✓ should revert if token is not underlying
✓ should revert if asks more than max (40ms)
✓ should revert if callback returns not success (92ms)

Interconnector

Interconnector logic

✓ Deploy Interconnector should fail if owner address iz zero (54ms)
✓ switchLinkageRoot should fail if during interconnection internal errors occur (577ms)

InterconnectorLeaf logic

✓ For a not initialized leaf switchLinkageRoot() should accept any root from any sender (38ms)
✓ For the initialized leaves only old root could initiate new root switching (93ms)
✓ switchLinkageRoot() do not emit if newRoot == oldRoot (52ms)
✓ switchLinkageRoot() should fail if zero root address was provided

Leaf contracts and Interconnector interaction

✓ Initial root connection should work (171ms)
✓ Initial root could be connected by everyone (168ms)
✓ interconnect should emit LinkageRootInterconnected event (102ms)
✓ SwitchLinkageRoot should work correctly (322ms)
✓ Second Leaf connection is possible only by the owner and only through the current Interconnector (342ms)
✓ The root address for an initialized Leaf contract cannot be changed directly through itself (121ms)
✓ The root address for a not initialized Leaf contract can be changed directly through itself (41ms)
✓ Deprecated Interconnector can not connect itself to core contracts (130ms)
✓ Different delegate call behaviour in switchLinkageRoot() with immutable storage variables (3283ms)

InterestRateModel

KinkMultiplierModel borrowRate and supplyRate tests

✓ curve: initialPoint - 0.025, slope - 0.20 (652ms)
✓ curve: initialPoint - 0.05, slope - 0.45 (641ms)
✓ curve: initialPoint - 0.1, slope - 0.45 (686ms)
✓ curve: initialPoint - 0.10, slope - 0.45 (644ms)
✓ should fail if cash + borrows overflows
✓ should fail if exp of borrows / cash + borrows overflows
✓ should fail if utilisationRate * multiplier overflows (506ms)
✓ should fail if (utilisationRate * multiplier + initialRate) overflows (502ms)

for jump=1.0, kinkPoint=0.9, initialRate=0.1, slope=0.2

✓ utilisation=0.0%: getBorrowRate() should calculate correct. (525ms)
✓ utilisation=0.0%: getSupplyRate() should calculate correct. (515ms)
✓ utilisation=10.0%: getBorrowRate() should calculate correct. (499ms)
✓ utilisation=10.0%: getSupplyRate() should calculate correct. (522ms)
✓ utilisation=89.0%: getBorrowRate() should calculate correct. (501ms)
✓ utilisation=89.0%: getSupplyRate() should calculate correct. (534ms)
✓ utilisation=90.0%: getBorrowRate() should calculate correct. (498ms)
✓ utilisation=90.0%: getSupplyRate() should calculate correct. (524ms)
✓ utilisation=91.0%: getBorrowRate() should calculate correct. (504ms)
✓ utilisation=91.0%: getSupplyRate() should calculate correct. (523ms)
✓ utilisation=100.0%: getBorrowRate() should calculate correct. (501ms)
✓ utilisation=100.0%: getSupplyRate() should calculate correct. (512ms)

for jump=0.2, kinkPoint=0.9, initialRate=0.1, slope=0.2

✓ utilisation=0.0%: getBorrowRate() should calculate correct. (505ms)
✓ utilisation=0.0%: getSupplyRate() should calculate correct. (505ms)
✓ utilisation=10.0%: getBorrowRate() should calculate correct. (504ms)
✓ utilisation=10.0%: getSupplyRate() should calculate correct. (537ms)
✓ utilisation=100.0%: getBorrowRate() should calculate correct. (519ms)
✓ utilisation=100.0%: getSupplyRate() should calculate correct. (518ms)

for jump=0.0, kinkPoint=0.9, initialRate=0.1, slope=0.2

✓ utilisation=0.0%: getBorrowRate() should calculate correct. (523ms)
✓ utilisation=0.0%: getSupplyRate() should calculate correct. (501ms)
✓ utilisation=10.0%: getBorrowRate() should calculate correct. (501ms)
✓ utilisation=10.0%: getSupplyRate() should calculate correct. (517ms)
✓ utilisation=100.0%: getBorrowRate() should calculate correct. (512ms)
✓ utilisation=100.0%: getSupplyRate() should calculate correct. (524ms)

for jump=0.0, kinkPoint=1.1, initialRate=0.1, slope=0.2

✓ utilisation=0.0%: getBorrowRate() should calculate correct. (525ms)
✓ utilisation=0.0%: getSupplyRate() should calculate correct. (544ms)
✓ utilisation=10.0%: getBorrowRate() should calculate correct. (536ms)
✓ utilisation=10.0%: getSupplyRate() should calculate correct. (509ms)

✓ utilisation=100.0%: getBorrowRate() should calculate correct. (499ms)
✓ utilisation=100.0%: getSupplyRate() should calculate correct. (534ms)
✓ utilisation=120.0%: getBorrowRate() should calculate correct. (527ms)
✓ utilisation=120.0%: getSupplyRate() should calculate correct. (535ms)

for jump=20.0, kinkPoint=0.0, initialRate=0.1, slope=0.2

✓ utilisation=0.0%: getBorrowRate() should calculate correct. (529ms)
✓ utilisation=0.0%: getSupplyRate() should calculate correct. (525ms)
✓ utilisation=10.0%: getBorrowRate() should calculate correct. (539ms)
✓ utilisation=10.0%: getSupplyRate() should calculate correct. (528ms)
✓ utilisation=100.0%: getBorrowRate() should calculate correct (531ms)
✓ utilisation=100.0%: getSupplyRate() should calculate correct. (584ms)

Liquidation contract

✓ Check initial params
✓ Should revert: treasury address can`t be address zero (302ms)
✓ setHealthyFactorLimit() should work
✓ setHealthyFactorLimit() should fail (91ms)
✓ _setTDeadDrop() should work
✓ setDeadDrop() should fail (66ms)
✓ setDeadDrop() should revert
✓ setInsignificantLoanThreshold() should work
✓ includes() should work correctly
✓ oracle() should work correctly (1412ms)

seize()

✓ Auto mode: should work correctly (111ms)
✓ Manual mode: should work correctly (98ms)

approveBorrowerHealthyFactor()

✓ reverts if oraclePrice is equal 0 (38ms)
✓ reverts if borrowUnderlying * oraclePrice causes overflow (54ms)
✓ reverts if supplyWrap * oraclePrice causes overflow
✓ reverts if supplyWrap * exchangeRateMantissa causes overflow
✓ reverts if accountTotalBorrow is equal to 0 (52ms)
✓ should return true if currentHealthyFactor is correct (41ms)
✓ should return false if currentHealthyFactor is incorrect (52ms)

accrue() should work correctly

✓ Should work correctly (68ms)
✓ Should accrue based only on debtRates if seizeIndexes is empty (44ms)
✓ Should fail if debtRates is LT accountAssets
✓ Should not fail if debtRates is GT accountAssets Array (45ms)
✓ Should work if seizeIndexes has index out of the range of accountAssets array (46ms)
✓ Should work if seizeIndexes is GT than accountAssets and debtMarkets (88ms)

calculateLiquidationAmounts() should work correctly

✓ Fails if asset price is 0 (45ms)
✓ Fails if asset price causes overflow (47ms)
✓ Fails if the account borrow underlying causes overflow (61ms)
✓ Fails if getAccountSnapshot() causes overflow (58ms)
✓ Fails if the redemptionRate causes overflow (71ms)
✓ Fail if debtRates arr length is less than accountAssets arr (87ms)
✓ Fail if selected collateral markets don't cover required seizeAmount (108ms)
✓ Fail if seizeIndexes has index out of accountAssets array (102ms)
✓ Returns correct values (106ms)
✓ Doesn't fail if totalSupply in selected collateral markets is equal to seizeAmount (92ms)
✓ Doesn't fail if unnecessary extra collateral markets have been passed in seizeIndexes (97ms)
✓ Doesn't fail if seizeIndexes arr contains collateral markets with zero supply (94ms)

verifyExternalData() should work correctly

✓ Should not revert if passed correct data
✓ Should revert if one of the debtRates is greater than 1e18
✓ Should revert if seizeIndexes has duplicate values
✓ Should revert if any seizeIndex is out of the range
✓ Should revert if seizeIndexes is empty
✓ Should revert if seizeIndexes.length > accountAssets.length
✓ Should revert if accountAssets arr is empty
✓ Should revert if debtRates.length is not equal to accountAssets.length

LiquidateUnsafeLoan() should work correctly

✓ Have to be reverted with access control error (56ms)
✓ Have to be reverted with Insufficient shortfall (68ms)
✓ Have to be reverted with Healthy factor not in range (324ms)

repay()

✓ Auto Mode: should work correctly (338ms)
✓ Manual Mode: should work correctly (476ms)

MinterestNFT

✓ nextIdToBeMinted() should return correct ID (106ms)
✓ Should initialize contract with correct args
✓ Should revert if proxyRegistry address is 0 (303ms)

Minting nft

✓ Should revert if the caller is not admin (59ms)
✓ Should revert if the caller is not gateKeeper (66ms)
✓ Should mint new minterestNFT (51ms)
✓ Should revert batchMint if args does not have equal length
✓ Should revert mintBatch function call if the caller is not admin (64ms)
✓ Should revert mintBatch function call if the caller is not minter (84ms)
✓ Should mintBatch new minterestNFTs (67ms)
✓ should call only onMintToken callback (327ms)

transfer
✓ should skip zero amount (46ms)
✓ safeTransfer partial (129ms)
✓ safeTransfer full (66ms)
✓ safeBatchTransfer one tier partial (68ms)
✓ safeBatchTransfer multiple tiers partial (78ms)
✓ safeBatchTransfer multiple tiers full (59ms)
✓ should call only onTransferToken callback (271ms)
uri functions
✓ fails if not called by admin (54ms)
✓ accepts new base URI and emits a NewBaseUri event
✓ Should return correct URI (72ms)
isApprovedForAll
✓ Should work (82ms)
supportsInterface
✓ Should return true
✓ Should return false

Mnt

could be initialized only once
✓ Mnt second initialization should fail
metadata
✓ has given name
✓ has given symbol
weightAggregator
✓ weightAggregator() should work correctly (791ms)
balanceOf
✓ grants to initial account
delegate
✓ should update lastActivityTimestamp for delegator, when delegatee votes after delegation (201ms)
delegateBySig
✓ reverts if the signatory is invalid
✓ reverts if the nonce is bad (63ms)
✓ reverts if the signature has expired
✓ delegates on behalf of the signatory (84ms)
validate checkpoints behavior for account votes and total votes
✓ numCheckpoints returns the number of checkpoints for a delegate after voting power update (347ms)
✓ checkpoints not created on transfer and delegate (125ms)
✓ updateVotingWeight does not add more than one checkpoint in a block (548ms)
✓ updateVotingWeight emits VotesUpdated (153ms)
✓ updateVotingWeight reverts if target address is zero
✓ updateTotalVotes emits TotalVotesUpdated (197ms)

✓ updateTotalVotes updates votes if total amount decreased (960ms)

getPastVotes

✓ reverts if block number >= current block
✓ returns 0 if there are no checkpoints
✓ returns the latest block if >= last checkpoint block (140ms)
✓ returns zero if < first checkpoint block (141ms)
✓ generally returns the voting balance at the appropriate checkpoint (555ms)

isParticipantActive()

✓ should return true if account voted for the last `maxNonVotingPeriod` blocks
✓ should return false if account didn't vote for the last `maxNonVotingPeriod` blocks

lastActivityTimestamp()

✓ must be zero timestamps at start
✓ alice delegates bob, should return zero timestamp for alice and bob (106ms)
✓ alice delegates to bob, bob casts, should return correct timestamps (184ms)
✓ alice casts, alice delegates to bob, should return correct timestamps (185ms)
✓ alice casts, alice delegates to bob, bob casts, should return correct timestamps (209ms)
✓ alice delegates to bob, bob casts, alice delegates to charlie, should return correct timestamps (197ms)
✓ alice delegates to bob, bob casts, alice delegates to alice, should return correct timestamps (188ms)
✓ alice delegates to bob, bob casts, bob delegates to charlie, charlie casts should return correct timestamps (208ms)

setMaxNonVotingPeriod()

✓ should revert if caller is not an admin (83ms)
✓ should revert if passed identical value
✓ should revert if passed period less than 90 days and greater than 2 years (43ms)
✓ accepts a valid value and emits a MaxNonVotingPeriodChanged event

updateVoteTimestamp

✓ should fail if governor was not set
✓ should fail if called by non-governor account
✓ should set correct timestamp for account

setGovernor()

✓ should revert when called by not an admin (83ms)
✓ should revert when called with zero address
✓ should revert on second initialization
✓ should accept new governor and emit NewGovernor event

getPastTotalSupply

✓ reverts if block number == current block
✓ reverts if block number > current block (43ms)
✓ returns 0 if there are no checkpoints (42ms)
✓ returns zero if < first checkpoint block (167ms)

supportInterface()

✓ Should return true

✓ Should return false

MntGovernor

Governor and Timelock could be initialized only once

✓ Governor second initialization should fail

✓ Timelock second initialization should fail

Transaction should pass if:

✓ We cast vote, get quorum and execute proposal (618ms)

✓ We cast vote, get quorum and cancel proposal after that (545ms)

We must revert if:

✓ There does not exist a proposal with matching proposal id where the current block number is between the proposal's start block (exclusive) and end block (inclusive)

✓ Such proposal already has an entry in its voters set matching the sender (232ms)

✓ Cancel called from unauthorized account

✓ Execute for successful proposal called directly from the governor (1086ms)

propose()

✓ Should create new proposal from any account if zero address is set in timelock for proposer role (230ms)

✓ Should revert call from unauthorized account if zero address is not set in timelock for proposer role (120ms)

supportsInterface()

✓ Should return true

✓ Should return false

_castVote()

✓ should update lastActivityTimestamp for account (107ms)

_executor()

✓ executor should be the timelock

multicall feature tests

Multicall test

✓ multicall works with proxy contract (792ms)

✓ callStatic and multicall should work with calculateLiquidationAmounts (910ms)

Proxy

✓ update (57ms)

✓ upgradeToAndCall() should work (56ms)

✓ upgradeToAndCall() should fail if called by non-admin and there's no fallback function

✓ change admin (224ms)

RewardsHub

index splitting

✓ should increase regular part linearly with time (94ms)

✓ should delay 75% at a period from start to half year (52ms)

✓ should delay 50% at a period from start to the end (51ms)

✓ should delay 25% at a period from half year to the end (88ms)
✓ should delay 50% at a period from 1/4 year to 3/4 of year (76ms)
✓ should partial split on after delay end (84ms)
✓ should treat delta as fully regular delta after delay period ends (59ms)
✓ should split boost correctly if accruedRegular != accruedTotal
✓ should split boost correctly if accruedRegular == accruedTotal
✓ splitBoost should fail if accruedTotal is zero

accrueBuybackReward

✓ should accrue some MNT (103ms)
✓ should delay 25% at a period from half year to the end (168ms)
✓ should delay nothing after final delay share (111ms)
✓ should fail when called not from Buyback
✓ should not accrue if user has zero weight (61ms)

market initialization

✓ should be called only from supervisor (61ms)
✓ should create initial index (45ms)
✓ should revert when initialized twice (74ms)

emission distribution

✓ should revert when updating not initialized market (163ms)
✓ should accrue supply emission rewards (119ms)
✓ should accrue borrow emission rewards (105ms)
✓ should accrue all emission rewards (182ms)
✓ should accrue emission rewards from specified markets (183ms)
✓ should accrue none of emission rewards (52ms)

claim and withdraw

✓ should transfer out all available MNT of an account (136ms)
✓ should be able to claim partially during claim period (206ms)
✓ should be able to claim and withdraw multiple times (447ms)
✓ should unlock 0% of delayed before claim period
✓ should unlock 50% of delayed after 1.5 years
✓ should unlock 100% of delayed after 2 years
✓ should unlock linearly with time (63ms)
✓ should revert when withdraw amount is too high (69ms)
✓ should revert when not enough MNT to payout (78ms)

grant

✓ should transfer MNT out (118ms)
✓ should revert when called by non-admin (100ms)
✓ should revert when trying to grant zero or too much (67ms)

pauses

✓ should revert when (un)pause called not by gatekeeper (51ms)
✓ should pause distribution (347ms)
✓ should pause and unpause withdraw (152ms)

Borrow scen tests

- ✓ Borrow some BAT and enable BAT if BAT not enabled as collateral (665ms)
- ✓ Borrow some BAT fails and account not enabled as collateral (693ms)
- ✓ Borrow some BAT fails when no BAT available (633ms)
- ✓ Borrow fails if market not listed (528ms)
- ✓ Borrow some BAT from Excess Cash (668ms)
- ✓ Borrow some BAT reverts if borrow is paused or price is equal to zero (846ms)

BorrowBalance scen tests

- ✓ Borrow Balance after 300000 blocks (832ms)
- ✓ Borrow Balance after 300000 blocks and then 600000 blocks (899ms)
- ✓ Borrow Balance after accrual then changed interest rate (1182ms)
- ✓ Total Borrow Balance with Two Borrowers (2781ms)

BorrowCap scen tests

- ✓ Attempt to borrow over set cap ERC20 (729ms)
- ✓ Attempt to borrow at set cap ERC20 (822ms)
- ✓ Borrow some Eth enable Eth and succeeds when Eth not enabled as collateral. At under borrow cap (892ms)
- ✓ Borrow cap keeper can set borrow caps (165ms)
- ✓ Setting borrow cap restricted to admin (56ms)
- ✓ fails if passed arrays with different length
- ✓ SetBorrowCaps works correctly too (850ms)

BorrowWBTC scen tests

- ✓ Borrow some WBTC enable WBTC and succeeds when not enabled as collateral (539ms)
- ✓ Borrow some WBTC fails when no WBTC available (563ms)
- ✓ Borrow some WBTC fails when WBTC paused (570ms)
- ✓ Borrow some WBTC from Excess Cash (562ms)

Business Development System scen tests

- ✓ NFT emission boost doesn't work with LP emission boost at the same time (NFT boost < LP boost) in public mode (4138ms)
- ✓ NFT emission boost doesn't work with LP emission boost at the same time (NFT boost < LP boost) in private mode (3462ms)
- ✓ NFT emission boost doesn't work with LP emission boost at the same time (NFT boost > LP boost) (3241ms)
- ✓ NFT emission boost works with representative bonus at the same time (3550ms)
- ✓ The hierarchical system in the agreements is prohibited (301ms)

Protocol user can become a liquidity provider and get an emission system boost in:

- ✓ whitelistMode (3553ms)
- ✓ publicMode (3631ms)

Protocol user can become a business development representative and get an emission system boost

- ✓ whitelistMode (4144ms)
- ✓ publicMode (6159ms)

Admin can remove working agreement, remove transaction distributes MNT tokens

✓ whitelistMode (1958ms)

✓ publicMode (2032ms)

Agreement can be removed if LP and representative already redeemed all their wrap tokens

✓ whitelistMode (2146ms)

✓ publicMode (2169ms)

Emission system scen tests

✓ Market supply and borrow states properly initialized (8362ms)

✓ Accrue supplier MNT during a lend (2196ms)

✓ Accrue borrower MNT during a borrow (1896ms)

✓ Accrue supplier MNT during redeem (1729ms)

✓ Accrue borrower MNT during a repay (2058ms)

✓ Accrue borrower MNT during a repayBorrowBehalf of 0 (1811ms)

✓ Don't accrue borrower MNT during a transfer (1788ms)

✓ Accrue supplier MNT during a transfer (1785ms)

✓ Accrues correctly when MNT rewards are added (after market activation), removed, then added again (1417ms)

✓ MNT is not withdrawn automatically (1697ms)

✓ New MNT emission rates apply to both prior borrowers+suppliers and later borrowers+suppliers correctly (6863ms)

✓ New MNT emission rates apply to both prior borrowers+suppliers and later borrowers+suppliers correctly with uninitialized prior borrower/supplier state indices (7540ms)

✓ Zero MNT emission rate markets don't accrue rewards with uninitialized borrower/supplier state indices (7093ms)

✓ Accrue supplier and borrower MNT during partial liquidation (2190ms)

Emission system integration tests

✓ Check total emission of MNT (supervisor, NFT, vesting, buyback) (8222ms)

Emission rate scen tests

✓ MNT supply emission rate can be set (566ms)

✓ MNT supply emission rate can be set then unset (795ms)

✓ MNT supply emission rate can be set then set again (1200ms)

✓ MNT supply emission rate can be set with borrow emission rate (352ms)

✓ MNT supply emission rate can be set then unset with borrow emission rate (661ms)

✓ MNT supply emission rate can be set then set, unset, and set again with borrow emission rate (1143ms)

✓ MNT borrow emission rate can be set (323ms)

✓ MNT borrow emission rate can be set then unset (690ms)

✓ MNT borrow emission rate can be set then set again (995ms)

✓ MNT borrow emission rate can be set with supply emission rate (383ms)

✓ MNT borrow emission rate can be set then unset with supply emission rate (771ms)

✓ MNT borrow emission rate can be set then set, unset, and set again with supply emission rate (1121ms)

✓ Many different MNT supply emission rates can be set (745ms)
✓ Many different MNT supply emission rates can be set then unset (1367ms)
✓ Many different MNT supply emission rates can be set, unset, and set again (2109ms)
✓ Many different MNT supply emission rates can be set with borrow emission rates (841ms)
✓ Many different MNT supply emission rates can be set then unset with borrow emission rates (1488ms)
✓ Many different MNT supply emission rates can be set, unset, and set again w/ borrow emission rates (4058ms)
✓ Many different MNT borrow emission rates can be set (1177ms)
✓ Many different MNT borrow emission rates can be set then unset (1727ms)
✓ Many different MNT borrow emission rates can be set, unset, and set again (2094ms)
✓ Many different MNT borrow emission rates can be set with supply emission rates (810ms)
✓ Many different MNT borrow emission rates can be set then unset with supply emission rates (1648ms)
✓ Many different MNT borrow emission rates can be set, unset, and set again with supply emission rates (2858ms)
✓ should accrue MNT correctly with different supply and borrow emission rates (1797ms)

MNTSource scen tests

✓ MNTSource is initialized correctly
✓ MNTSource properly drips first drip (181ms)
✓ MNTSource properly drips second drip (182ms)
✓ MNTSource properly drips zero sequentially (273ms)
✓ MNTSource handles not having enough to drip (266ms)
✓ Revert on dripTotal overflow (70ms)
✓ MNTSource does not use improperly transfered tokens (215ms)
✓ Admin can sweep improperly transfered tokens (192ms)
✓ non-Admin can't sweep (104ms)
✓ Admin can't sweep more than available (81ms)

Delay logic integration tests

Buyback delay logic

✓ When there was no any buyback - subsequent buyback rewards should be delayed from delayStart timestamp (627ms)
✓ When there was some buyback action - subsequent buyback rewards should be delayed from the last user action (943ms)
✓ When final buybackIndex has not been finalized - all buyback rewards should be delayed (939ms)
✓ When final buybackIndex has been finalized - only rewards during the delay period should be delayed (926ms)
✓ Should withdraw correct MNT amount after passing delay && claim periods, with several participants and leaving in the middle of a delay period (1355ms)
✓ Preparation of the buyback mechanism before the start + LBP participants (740ms)

Boost delay logic

✓ Should split boost correctly when delayEnd index was not finalized (990ms)
✓ Should split boost correctly when delayEndIndex was finalized (1750ms)
✓ Should split boost correctly after the delay period (1716ms)

Fee scen tests

✓ Transfer fee goes to admin (55ms)
✓ Lend should work and not change exchange rate (434ms)
✓ Should be able to redeem a fee MToken, exchange Rate should not change (544ms)
✓ Order of redeems should not matter if no interest accrued (1036ms)
✓ Repay borrow should work and not change exchange rate (22233ms)

Governance scenario tests

Admin

✓ set voting delay (1097ms)
✓ set voting period (2095ms)
✓ set proposal threshold (1136ms)
✓ check onlyGovernance access for voting settings (61ms)
✓ changeAdmin() can be called only by admin (79ms)

Cancel

✓ cancel a pending proposal before the start of voting period (490ms)
✓ cancel a pending proposal after start of voting period (846ms)
✓ cancel a defeated (unvoted) proposal (1069ms)
✓ cancel an active proposal with some votes (1063ms)
✓ cancel a succeeded proposal (1080ms)
✓ cancel a queued proposal (1380ms)
✓ revert when trying to cancel executed proposal (1742ms)
✓ revert when non-canceler is trying to cancel (1200ms)

Defeat

✓ defeat when For votes do not reach quorum (826ms)
✓ defeat when there are more Against than For votes (1413ms)
✓ defeat when there are only abstain votes (1513ms)
✓ not defeat when there are any For votes and abstain votes provide quorum (1831ms)
✓ not defeat when vote is ongoing (1594ms)
✓ not defeat when Fors pass quorum and Nays (2668ms)

Execute

✓ execute a simple queued proposal with value (2338ms)
✓ execute a complex queued proposal with value (1345ms)
✓ revert when trying to execute a succeded but unqueued proposal (1016ms)
✓ revert when executing proposal that reverts, proposal remains queued (1871ms)
✓ assert execution order (1122ms)
✓ cannot execute cancelled proposal (1253ms)

Propose

✓ propose with 1 action (712ms)
✓ propose with two actions (1434ms)

✓ fails when insufficient sender votes (429ms)
✓ fails when no actions given (422ms)
✓ fails when actions mismatch length (581ms)
✓ propose when proposer has active proposal (1033ms)
✓ revert when proposed on invalid timelock (3372ms)

Queue

✓ queue simple action (1602ms)
✓ queue 2 actions with multiple params (1308ms)
✓ accept repeated proposal items (1697ms)
✓ revert if proposal is Pending (1259ms)
✓ revert if proposal is Active (1631ms)
✓ revert if proposal is Defeated (1051ms)
✓ revert if proposal is Queued (1115ms)
✓ revert if queuing an already executed proposal (1380ms)
✓ don't queue when canceled (880ms)

Vote

✓ cast For vote (863ms)
✓ cast Against vote (1359ms)
✓ cast Abstain vote (1369ms)
✓ cast zero vote (776ms)
✓ can't cast vote twice (942ms)
✓ can't get more votes by voting with delegatee and personally (1801ms)
✓ can get more votes by staking between propose and votes start if voting delay is not set to zero (1112ms)
✓ can vote if voting delay is set to zero (917ms)
✓ can't vote before start (670ms)
✓ can't vote after end (765ms)
✓ can't vote on canceled vote (2275ms)
✓ can't vote on succeeded, queued and executed proposals (1815ms)
✓ amount of votes does not depend on MNT balance (1035ms)
✓ cast vote with reason (710ms)
✓ cast vote by signature (766ms)

Votes close to equal share

✓ succeed when For is 1 wei above Against votes (1086ms)
✓ defeat when Against equals For votes (1080ms)
✓ defeat when Against is 1 wei more than For votes (1214ms)

Check voting power

✓ Check voting power delegation (954ms)

mnt scen tests

✓ Check Name
✓ Check Symbol
✓ Check Decimals
✓ Check Total Supply

✓ Check account receives Total Supply after deploy
✓ Check approve sets correct approval and emits Approval event
✓ Check transfer updates balances correctly, emits Transfer event, and returns true (126ms)
✓ Check self-transfer updates balances correctly, emits Transfer event, and returns true (183ms)
✓ Check transferFrom with max allowance updates balances correctly, emits Transfer event, and returns true (140ms)
✓ Check transferFrom with allowance updates balances correctly, emits Transfer event, and returns true (89ms)
✓ Check transferFrom reverts with not sufficient allowance
✓ Check transfer reverts when transferring too much
✓ Check transfer reverts when transferring to address 0
✓ Delegate with zero balance doesn't change votes checkpoints (60ms)
✓ Delegatee changes from address(0) to account with zero checkpoints (149ms)
✓ Delegatee changes from address(0) to account with existing checkpoints (268ms)
✓ Delegate to address(0) (329ms)
✓ Delegate from one account to another account with zero checkpoints (335ms)
✓ Delegate from one account to another account with multiple checkpoints (456ms)
✓ Vote checkpoints don't change on transfer (1159ms)
✓ One checkpoint created when delegators change voting power for the same amount in the same block (429ms)
✓ Only one checkpoint is added per block for multiple vote updates (499ms)
- Total votes checkpoint updated only on demand
✓ Should revert if burn called (57ms)

Lend scen tests

✓ Lend 1 mZRX (384ms)
✓ Lend with insufficient allowance (135ms)
✓ Lend with insufficient balance (132ms)
✓ Lend two ZRX after lending two ZRX, and then I lend two more (657ms)
✓ Two accounts Lend (737ms)
✓ Lend accrues no interest without borrows (396ms)
✓ Lend transfer in fails (380ms)
✓ Denied by supervisor because unlisted (367ms)
✓ Lend reverts if lend is paused (390ms)

LendWBTC scen tests

✓ Lend 1 mWBTC (507ms)
✓ Lend WBTC with insufficient allowance (134ms)
✓ Lend WBTC with insufficient balance (153ms)
✓ Lend two WBTC after lending two WBTC, and then I lend two more (807ms)
✓ Two accounts Lend WBTC (921ms)
✓ Lend WBTC accrues no interest without borrows (418ms)
✓ Lend WBTC transfer in fails due to paused (377ms)
✓ Denied by supervisor because WBTC unlisted (309ms)

Liquidation scenario tests

Liquidation scenario tests

✓ Separate markets: liquidation should work correctly (7354ms)
✓ Cross markets: liquidation should work correctly (3951ms)
✓ Partial liquidation should work correctly if accountTotalBorrow <= insignificantLoanThreshold (9625ms)
✓ Only safe 'debtRedemptionRate' allowed (9273ms)
✓ When liquidation allowed due to BTC is borrowed and is paused (3043ms)
✓ When liquidation not allowed due to BTC is taken as collateral and is paused (4558ms)
✓ When liquidation not allowed due to asset price is zero (8471ms)
✓ Liquidation should revert if TPI is insufficient in one market (4838ms)
✓ Partial liquidation: huge supply - small utilisationFactor, tiny supply - normal utilisationFactor (4772ms)
✓ Partial liquidation: huge supply - max utilisationFactor, tiny supply - normal utilisationFactor (5047ms)
✓ Partial liquidation: collateral with zero utilisationFactor (4876ms)

Liquidation of unhealthy debts

✓ Separate markets: liquidation should work correctly (7059ms)

Manual liquidation scenario tests

✓ Manual mode: Partial liquidation with sufficient balance in liquidator's address (6028ms)
✓ Manual mode: Partial liquidation should fail due to insufficient balance in liquidator's address (5181ms)
✓ Manual mode: Partial liquidation of insignificant debt (6406ms)
✓ Trusted liquidator can not liquidate manually even if he has enough funds to repay (6646ms)

HypotheticalAccountLiquidity scen tests

✓ Calculates hypothetical account liquidity (42538ms)

AddProtocolInterest scen tests

✓ Add protocol interest and verify effects (2835ms)
✓ Add protocol interest behalf and verify effects (2463ms)
✓ Remove and re add protocol interest and remove again (2583ms)
✓ Add protocol interest to empty contract (107ms)
✓ Add protocol interest failures (94ms)
✓ Add protocol interest WBTC when paused (2514ms)
✓ Add protocol interest behalf should fail if called not by Liquidation contract (68ms)

EnableDisableMarkets scen tests

✓ Enable Markets Idempotent (180ms)
✓ Enabled Market Must Be Supported (213ms)
✓ Disable single market (141ms)
✓ Disable non-enabled market (76ms)
✓ Disable one of two market from the front (184ms)
✓ Disable one of two market from the back (220ms)
✓ Disable multiple markets (817ms)

✓ Realistic Market Scenario (3312ms)
✓ do not join market on first lend (452ms)

ExchangeRate scen tests

✓ Initial Exchange Rate
✓ Initial Exchange Rate with Lend (920ms)
✓ ZRX: Exch. Rate:2e9, Cash(51e18) + Borrows(2.0e18) - Protocol Interest(0.5e18) / Tokens(2.5e10) (526ms)
✓ USDC: Exch. Rate:2e-3, Cash(51e18) + Borrows(2.0e18) - Protocol Interest(0.5e18) / Tokens(2.5e10) (478ms)

ReduceProtocolInterest scen tests

✓ Reduce all protocol interest and verify effects (3386ms)
✓ Reduce partial protocol interest and verify effects (3490ms)
✓ Redeem all and then reduce all protocol interest (5443ms)
✓ Reduce protocol interest WBTC when paused (3365ms)

SweepToken scen tests

✓ fails if called by non-admin (4854ms)
✓ Attempt to sweep underlying token (3885ms)
✓ Successfully Sweep standard non-underlying token from MToken (1022ms)
✓ Successfully Sweep non-standard non-underlying token from MToken (837ms)

TokenTransfer scen tests

✓ Simple mToken Transfer (500ms)
✓ Simple mToken Transfer When Underlying Paused (502ms)
✓ Simple mToken Transfer 1:1 Rate (4448ms)
✓ Simple mToken Transfer Not Allowed by Supervisor (4485ms)
✓ Simple mToken Transfer From (560ms)
✓ mToken Transfer From Not Allowed (570ms)
✓ mToken Transfer paused (724ms)

MinterestNFT scenario tests

✓ NFT holders can interact with the protocol in private mode (8571ms)
✓ Checking mode switching from private to public (8902ms)
✓ Receiving and sending NFT tokens by the user, distribution should be calculated correctly (4327ms)
✓ Adding new market to existing NFTs during public mode does not break emission logic and accrues MNT (11658ms)
✓ Adding new market to existing NFTs during private mode does not break emission logic and accrues MNT (11640ms)
✓ Moving activated NFTs from one user to another keeps MNT distribution calculation (11682ms)
✓ Moving two activated NFTs with different emission rates changes MNT distribution for a new market (11198ms)
✓ Minting NFTs with different emission rates and same end block does not break calculations (10922ms)

✓ Checking the minting of new NFT tokens (2255ms)
✓ Checking the creation of a new tier and NFT tokens for it (2285ms)
✓ Minting NFTs with same emission rates and different durations does not break calculations (11509ms)

Proxy scen tests

✓ Should upgrade Supervisor with correct storage (1980ms)

Redeem scen tests

✓ Lend then Redeem All (1364ms)
✓ Lend, Enable and then Redeem All (1338ms)
✓ Lend then Redeem Part. It should also fail if the market isn't listed (1519ms)
✓ Lend then Redeem Too Much (1072ms)
✓ Lend then Redeem Zero (1063ms)
✓ Lend then redeem with interest - no protocol interest (3403ms)
✓ Lend then redeem part with interest - no protocol interest (3392ms)
✓ Lend then redeem with protocol interest and interest (3534ms)
✓ Two accounts Lend, one redeems (906ms)
✓ Redeem transfer out fails (520ms)
✓ Lend, Enable, then Redeem Too Much (utilisation factor: constants.Zero) (527ms)
✓ Lend, Enable, then Redeem Too Much (utilisation factor: 0.1) (528ms)

Redeem scen tests

AML system is enabled. Checks for a wallets that is outside the AML system and sanctioned by the AML system

✓ Lend then redeemByAmlDecision (1048ms)
✓ Lend then redeemByAmlDecision. It should also fail if the market isn't listed (1100ms)
✓ Lend then redeemByAmlDecision with interest - no protocol interest (3643ms)
✓ Lend then redeemByAmlDecision with protocol interest and interest (3654ms)
✓ Two accounts Lend, one redeems (1001ms)

RedeemUnderlying scen tests

✓ Lend then Redeem All (1354ms)
✓ Lend then Redeem Part (1229ms)
✓ Lend then Redeem Too Much (1121ms)
✓ Lend then Redeem Zero (1105ms)
✓ Lend then redeem with interest - no protocol interest (3464ms)
✓ Lend then redeem part with interest - no protocol interest (3325ms)
✓ Lend then redeem with protocol interest and interest (3271ms)
✓ Two accounts Lend, one redeems (914ms)
✓ Lend then Redeem 1 wei of underlying (777ms)

RedeemUnderlyingWBTC scen tests

✓ Lend then Redeem All (1002ms)
✓ Lend then Redeem Part (1237ms)
✓ Lend then Redeem Too Much (1264ms)
✓ Lend then Redeem Zero (779ms)
✓ Lend then redeem with interest - no protocol interest (3265ms)

✓ Lend then redeem part with interest - no protocol interest (3225ms)
✓ Lend then redeem with protocol interest and interest (3277ms)
✓ Two accounts Lend, one redeems (899ms)
✓ Lend then Redeem 1 wei of underlying is allowed for 1:1 assets (1070ms)

RedeemWBTC scen tests

✓ Lend WBTC then Redeem All (987ms)
✓ Lend WBTC, Enable and then Redeem All (1102ms)
✓ Lend WBTC then Redeem Part (1047ms)
✓ Lend WBTC then Redeem Too Much (1009ms)
✓ Lend WBTC then Redeem Zero (1053ms)
✓ Lend WBTC then redeem with interest - no protocol interest (4022ms)
✓ Lend WBTC then redeem part with interest - no protocol interest (3215ms)
✓ Lend WBTC then redeem with protocol interest and interest (2824ms)
✓ Two accounts Lend WBTC, one redeems (874ms)
✓ Redeem WBTC transfer out fails (790ms)
✓ Lend WBTC, Enable, then Redeem Too Much (utilisation factor: constants.Zero) (537ms)
✓ Lend WBTC, Enable, then Redeem Too Much (utilisation factor: 0.1) (527ms)

RepayBorrow scen tests

✓ Borrow, hold a few blocks, and repay part (241ms)
✓ Borrow, hold a few blocks, and repay full (266ms)
✓ Borrow, hold a few blocks, and repay all (263ms)
✓ Repay all several times (7167ms)
✓ Borrow, hold a few blocks, and repay too much (261ms)
✓ Borrow, and get a negative total cash situation (201ms)
✓ Borrow, hold a few blocks, and repay behalf part (539ms)
✓ Prohibit repay by supervisor rejection due to mock unlist (448ms)
✓ Repay fails with insufficient allowance (200ms)
✓ Repay fails with insufficient balance (164ms)

RepayBorrowWbtc scen tests

✓ Borrow WBTC, hold a few blocks, and repay part (370ms)
✓ Borrow, hold a few blocks, and repay full (294ms)
✓ Borrow, hold a few blocks, and repay too much (503ms)
✓ Borrow, and get a negative total cash situation (228ms)
✓ Borrow, hold a few blocks, and repay behalf part (566ms)
✓ Prohibit repay by supervisor rejection due to mock unlist (407ms)
✓ Borrow WBTC, can't repay when paused (206ms)
✓ Repay fails with insufficient allowance (220ms)
✓ Repay fails with insufficient balance (217ms)

TestProtocol scenarios

✓ Lend in the first market and borrow another market should work (49540ms)

Spinarama

#lendLend

✓ should succeed (212ms)
✓ should partial succeed (325ms)

#lendRedeem

✓ should succeed (253ms)

#redeemLend

✓ should succeed (311ms)

#repayRepay

✓ should succeed (621ms)

Supervisor contract

✓ beforeAutoLiquidationRepay should fail if incorrect liquidator was passed
✓ beforeAutoLiquidationRepay should call emission system if required (201ms)
✓ change admin (295ms)
✓ isLiquidator() should work correctly
✓ Initialize should be called once (43ms)

redeemVerify

✓ should allow you to redeem 0 underlying for 0 tokens
✓ should allow you to redeem 5 underlying for 5 tokens
✓ should not allow you to redeem 5 underlying for 0 tokens

beforeAutoLiquidationSeize

✓ reverts if called by an unreliable liquidator (44ms)
✓ should call updateMntSupplyIndex() and distributeSupplierMnt() (216ms)

setLiquidationFee

✓ fails if called by non-timelock (106ms)
✓ fails if asset is not listed
✓ fails if fee is equal to zero
✓ accepts a valid fee and emits a NewLiquidationFee event (45ms)

setUtilisationFactor

✓ fails if not called by timelock (71ms)
✓ fails if asset is not listed
✓ fails if factor is set without an underlying price (38ms)
✓ fails if factor is too high
✓ succeeds and sets market (55ms)

supportMarket

✓ fails if not called by admin (72ms)
✓ succeeds and sets market (49ms)
✓ cannot list a market a second time
✓ can list two different markets (55ms)

PauseControl

✓ should let admin pause and unpause (57ms)
✓ should let GateKeeper pause but not unpause

✓ should not let strangers pause and unpause (46ms)
✓ should pause lend on one market (89ms)
✓ should pause borrow on one market (64ms)
✓ should pause flash loan on one market
✓ should pause transfer globally

Whitelist mode for allowed methods should work

✓ beforeLend revert if whitelist mode is on and user is not in whitelist (154ms)
✓ beforeRedeem revert if whitelist mode is on and user is not in whitelist (152ms)
✓ beforeBorrow revert if whitelist mode is on and user is not in whitelist (172ms)
✓ beforeRepayBorrow revert if whitelist mode is on and user is not in whitelist (149ms)

The AML system should allow tokens to be redeemed only for bad accounts

✓ AML process, redeemer is blacklisted (73ms)
✓ AML process, redeemer is not blacklisted
✓ Not AML process, redeemer is blacklisted (88ms)
✓ Not AML process, redeemer is not blacklisted (72ms)

The AML system should block some of the tokens operations

✓ enableAsCollateral should be blocked for blacklisted users
✓ beforeLend should be blocked for blacklisted users
✓ beforeBorrow should be blocked for blacklisted users
✓ beforeTransfer should be blocked for blacklisted users if at least one of them blacklisted (132ms)
✓ beforeFlashLoan should be blocked for blacklisted users
✓ mnt transfer should be blocked if sender is blacklisted
✓ mnt transfer should be blocked if recipient is blacklisted

The AML system should not affect some of the tokens operations

✓ blacklisted users can redeem (51ms)
✓ blacklisted users can disableAsCollateral
✓ blacklisted users can repay (57ms)
✓ blacklisted users can distributeAllMnt (51ms)
✓ blacklisted users can distributeMnt (88ms)
✓ blacklisted users can updateAndGetMntIndexes (56ms)

BorrowAndRepay

borrow

✓ fails if supervisor tells it to (190ms)
✓ proceeds if supervisor tells it to
✓ continues if fresh
✓ fails if error if protocol has less than borrowAmount of underlying (52ms)
✓ fails if borrowBalanceStored fails (due to non-zero stored principal with zero account index) (98ms)
✓ fails if calculating account new total borrow balance overflows (95ms)
✓ fails if calculation of new total borrow balance overflows (59ms)
✓ reverts if transfer out fails (67ms)
✓ transfers the underlying cash, tokens, and emits Transfer, Borrow events (97ms)

✓ stores new borrow principal and interest index (134ms)
✓ emits a borrow failure if interest accrual fails
✓ reverts from borrow without emitting any extra logs (39ms)
✓ returns success from borrowFresh and transfers the correct amount (92ms)

repayBorrowFresh

benefactor paying

✓ fails if repay is not allowed
✓ fails if block number ≠ current block number
✓ fails if insufficient approval (49ms)
✓ fails if insufficient balance
✓ reverts if calculating account new account borrow balance fails (92ms)
✓ reverts if calculation of new total borrow balance fails (49ms)
✓ reverts if doTransferIn fails
✓ transfers the underlying cash, and emits Transfer, RepayBorrow events (76ms)
✓ stores new borrow principal and interest index (97ms)

borrower paying

✓ fails if repay is not allowed
✓ fails if block number ≠ current block number (39ms)
✓ fails if insufficient approval (48ms)
✓ fails if insufficient balance (48ms)
✓ reverts if calculating account new account borrow balance fails (80ms)
✓ reverts if calculation of new total borrow balance fails (101ms)
✓ reverts if doTransferIn fails (40ms)
✓ transfers the underlying cash, and emits Transfer, RepayBorrow events (110ms)
✓ stores new borrow principal and interest index (83ms)

repayBorrow

✓ emits a repay borrow failure if interest accrual fails (58ms)
✓ reverts error from repayBorrowFresh without emitting any extra logs (57ms)
✓ returns success from repayBorrowFresh and repays the right amount (103ms)
✓ repays the full amount owed if payer has enough (82ms)
✓ reverts if payer does not have enough (62ms)

repayBorrowBehalf

✓ emits a repay borrow failure if interest accrual fails (49ms)
✓ reverts from repayBorrowFresh without emitting any extra logs (73ms)
✓ returns success from repayBorrowFresh and repays the right amount (154ms)

autoLiquidationRepayBorrow

✓ emits a repay borrow failure if market is not fresh (74ms)
✓ reverts from autoLiquidationRepayBorrowFresh if TPI doesn't cover repay amount (74ms)
✓ returns success from autoLiquidationRepayBorrow and repays the right amount (121ms)

LendAndRedeem

lendFresh

✓ fails if supervisor tells it to
✓ proceeds if supervisor tells it to (48ms)

✓ fails if not fresh
✓ continues if fresh (40ms)
✓ fails if insufficient approval
✓ fails if insufficient balance
✓ fails if exchange calculation fails (46ms)
✓ fails if transferring in fails
✓ transfers the underlying cash, tokens, and emits Lend, Transfer events (691ms)

lend

✓ emits a lend failure if interest accrual fails (324ms)
✓ returns error from lendFresh without emitting any extra logs
✓ returns success from lendFresh and lends the correct number of tokens (371ms)
✓ emits an AccrueInterest event (386ms)

redeemFreshTokens

✓ fails if supervisor tells it to
✓ fails if not fresh
✓ continues if fresh (38ms)
✓ fails if insufficient protocol cash to transfer out (39ms)
✓ fails if exchange calculation fails
✓ fails if transferring out fails (44ms)
✓ fails if total supply < redemption amount
✓ reverts if new account balance underflows
✓ transfers the underlying cash, tokens, and emits Redeem, Transfer events (679ms)

redeemFreshAmount

✓ fails if supervisor tells it to
✓ fails if not fresh
✓ continues if fresh (42ms)
✓ fails if insufficient protocol cash to transfer out
✓ fails if exchange calculation fails
✓ fails if transferring out fails (115ms)
✓ fails if total supply < redemption amount
✓ reverts if new account balance underflows (38ms)
✓ transfers the underlying cash, tokens, and emits Redeem, Transfer events (654ms)

redeem

✓ should fail if interest accrual fails (56ms)
✓ should fail in redeemFresh without emitting any extra logs (70ms)
✓ returns success from redeemFresh and redeems the right amount (97ms)
✓ returns success from redeemFresh and redeems the right amount of underlying (94ms)
✓ emits an AccrueInterest event (370ms)

MEther

✓ mEther basic operations (3124ms)
✓ borrowETH should not transfer ETH for uncollateralized user (288ms)

MToken

mToken constructor should work

✓ fails when 0 initial exchange rate (3713ms)
✓ succeeds with erc-20 underlying and non-zero exchange rate (3593ms)

name, symbol, decimals

✓ should return correct name
✓ should return correct symbol
✓ should return correct decimals

balanceOfUnderlying

✓ has an underlying balance (42ms)

borrowRatePerBlock

✓ has a borrow rate

supplyRatePerBlock

✓ returns 0 if there's no supply
✓ has a supply rate (79ms)

borrowBalanceCurrent

✓ reverts if interest accrual fails (42ms)
✓ returns successful result from borrowBalanceStored with no interest (72ms)
✓ returns successful result from borrowBalanceCurrent with no interest (95ms)

borrowBalanceStored

✓ returns 0 for account with no borrows
✓ returns stored principal when account and market indexes are the same (55ms)
✓ returns calculated balance when market index is higher than account index (51ms)
✓ has undefined behavior when market index is lower than account index
✓ reverts on overflow of principal (52ms)
✓ reverts on non-zero stored principal with zero account index (55ms)

exchangeRateStored

✓ returns initial exchange rate with zero mTokenSupply
✓ calculates with single mTokenSupply and single total borrow
✓ calculates with mTokenSupply and total borrows
✓ calculates with cash and mTokenSupply (56ms)
✓ calculates with cash, borrows, protocol interest and mTokenSupply (54ms)

getCash

✓ gets the cash

supportsInterface

✓ Should return true
✓ Should return false

Protocol Interest

setProtocolInterestFactor

✓ rejects change by non-timelock (76ms)
✓ rejects new protocol interest factor that descales to 1
✓ accepts new protocol interest factor in valid range and emits log

✓ accepts a change back to zero (39ms)
✓ emits a protocol interest factor failure if interest accrual fails (50ms)
✓ returns error from setProtocolInterestFactor without emitting any extra logs
✓ returns success from setProtocolInterestFactor (58ms)

reduceProtocolInterest

✓ fails if called by non-admin (66ms)
✓ fails if amount exceeds protocol interest
✓ fails if amount exceeds available cash
✓ increases admin balance and reduces protocol interest on success (44ms)
✓ emits an event on success
✓ emits a protocol interest reduction failure if interest accrual fails (46ms)
✓ returns error from reduceProtocolInterest without emitting any extra logs
✓ returns success code from reduceProtocolInterest and reduces the correct amount (57ms)

mToken contract

autoLiquidationSeize

✓ reverts if autoLiquidationSeize is not allowed (44ms)
✓ reverts if exchangeRateStoredInternal failed
✓ reverts if accountTokens minus borrowerSeizeTokens causes overflow
✓ reverts if doTransferOut failed (48ms)
✓ Significant loan: should change storage correct and emit events (74ms)
✓ Insignificant loan: should change storage correct and emit events (72ms)
✓ Infinity seizeUnderlyingAmount_ initiates full seize (56ms)

admin functions should work

setInterestRateModelFresh

✓ fails if market not fresh
✓ reverts if passed a contract that doesn't implement isInterestRateModel
✓ emits expected log when accepting a new valid interest rate model

setInterestRateModel

✓ fails if called by non-admin (63ms)
✓ emits a set market interest rate model failure if interest accrual fails (46ms)
✓ returns error from setInterestRateModelFresh without emitting any extra logs (52ms)
✓ reports success when setInterestRateModelFresh succeeds

mTokens accrueInterest should work

✓ reverts if the interest rate is absurdly high (78ms)
✓ fails if new borrow rate calculation fails (73ms)
✓ fails if simple interest factor calculation fails (74ms)
✓ fails if new borrow index calculation fails (77ms)
✓ fails if new borrow interest index calculation fails (73ms)
✓ fails if interest accumulated calculation fails (74ms)
✓ fails if new total borrows calculation fails (83ms)
✓ fails if interest accumulated for protocolInterests calculation fails (104ms)
✓ fails if new total protocolInterests calculation fails (110ms)

✓ succeeds and saves updated values in storage on success (134ms)

Transfer

✓ cannot transfer from a zero balance (107ms)
✓ transfers 50 tokens (146ms)
✓ doesn't transfer when src == dst
✓ rejects transfer when not allowed (62ms)

Vesting contract

✓ End of the vesting should be calculated correctly
✓ Locked and vested amount should be calculated correctly (165ms)
✓ Withdraw should fail if amount is zero (54ms)
✓ Withdraw should fail if amount is greater than unreleased tokens
✓ Withdraw should transfer all unreleased tokens (111ms)
✓ Withdraw should work (602ms)
✓ Only admin may call createVestingScheduleBatch (56ms)
✓ CreateVestingScheduleBatch should fail if target is zero address
✓ CreateVestingScheduleBatch should fail if freeAllocation is insufficient
✓ CreateVestingScheduleBatch should fail if vestingAmount == 0 (49ms)
✓ CreateVestingScheduleBatch should fail if the schedule for this user already exists (43ms)
✓ CreateVestingScheduleBatch should fail if insufficient token balance (51ms)
✓ CreateVestingScheduleBatch should work (119ms)
✓ CreateVestingScheduleBatch after revoke for the same address (114ms)
✓ Only gateKeeper may call revokeVestingSchedule (50ms)
✓ RevokeVestingSchedule should fail if schedule is not revocable
✓ RevokeVestingSchedule should fail if there is no vesting schedule at the address
✓ RevokeVestingSchedule should work (200ms)
✓ withdraw for user in delayList should work correctly (185ms)
✓ Revoking schedule from the account while he is in delay list (145ms)

constructor

✓ Should set mnt address correctly

addToDelayList

✓ addToDelayList should work
✓ addToDelayList should fail if account schedule isn't revocable
✓ addToDelayList should fail if called by not a gatekeeper (50ms)

removeFromDelayList

✓ removeFromDelayList should work
✓ removeFromDelayList should fail if user isn't in delay list
✓ removeFromDelayList should fail if called by not a gatekeeper (48ms)

Whitelist contract

Constructor

✓ should fail due to membership limit reached
✓ should ignore duplicate address (74ms)
✓ should return correct initial values
✓ should emit 'MemberAdded' for every address (70ms)

addMember()
✓ Should work when called by admin account
✓ Should work when called by gateKeeper account (50ms)
✓ should fail when called by non-admin or non-gateKeeper account (109ms)
removeMember()
✓ Should work when called by admin account
✓ Should work when called by gateKeeper account (84ms)
✓ should fail when called by non-admin or non-gateKeeper account (56ms)
turnOffWhitelistMode()
✓ Should fail if caller is not admin (66ms)
✓ turnOffWhitelistMode should work (61ms)
setMaxMembers()
✓ Should work when called by admin account
✓ Should work when called by gateKeeper account (60ms)
✓ should fail when called by non-admin or non-gateKeeper account (99ms)
isWhitelisted()
✓ whitelistMode disabled, should work correct (62ms)
✓ whitelistMode enabled, should work correct

1167 passing (40m)
6 pending

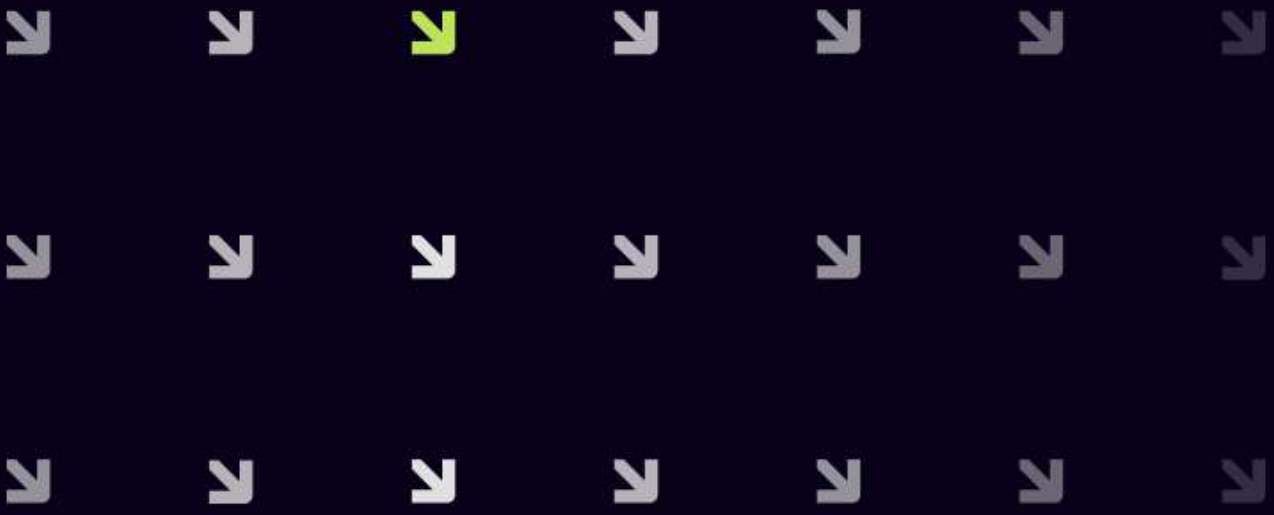
FILE	% STMTS	% BRANCH	% FUNCS
BDSystem.sol	100	100	100
BuyBack.sol	98.88	94.44	96.15
BuyBackDripper.sol	100	90	100
DeadDrop.sol	80	96.67	100
EmissionBooster.sol	100	95.89	100
Interconnector.sol	100	100	100
InterconnectorLeaf.sol	100	100	100
KinkMultiplierModel.sol	100	100	100
Liquidation.sol	100	96.97	100

FILE	% STMTS	% BRANCH	% FUNCS
MEther.sol	100	100	100
MinterestNFT.sol	100	100	100
MintProxy.sol	100	87.5	100
MNTSource.sol	100	66.67	100
MToken.sol	100	84.09	100
MTokenStorage.sol	100	100	100
RewardsHub.sol	100	97.46	97.37
Supervisor.sol	99.17	92.42	94.14
Vesting.sol	100	92.31	100
WeightAggregator.sol	83.33	75	100
Whitelist.sol	100	96.15	100
		governance	
.../Mnt.sol	100	97.22	100
.../MntGovernor.sol	92.59	85.71	94.12
.../MntTimelock.sol	100	75	100
.../MntVotes.sol	100	94.12	100
	98.15	88.01	98.53
		libraries	
.../ErrorCodes.sol	100	100	100
.../PauseControl.sol	100	75	100
.../ProtocolLinkage.sol	100	100	100
	100	91,67	100
		oracles	
ChainlinkPriceOracle.sol	100	100	100
FallbackPriceFeed.sol	100	50	100
	100	75	100
All files	**98.24**	**92.19**	**99.4**

We are grateful for the opportunity to work with the Minterest team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Minterest team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.



zokyo