Coupon Finance

Security Review

HickupHH3

4 October 2023

Contents

1 Introduction

The purpose of this audit is to review the contracts for Coupon Finance. Its aim is to eliminate term spread in traditional lending protocols by tokenizing interest rates.

- Financing perpetual borrowing with demand deposits exposes traditional lending protocols (Aave, Compound, etc) to term spread.
- Financing term loans with term deposits peer-to-peer removes term spread but breaks fungibility.
- Coupons, by tokenizing interest rates, match term lenders and borrowers in a peer-to-pool fashion.

As a result, lending/deposit spreads are tightened, yielding favorable rates for both depositors and borrowers.

1.1 Audit Scope

The scope consisted of the `coupon-finance` repository in the `main` branch at commit hash `6e917ad8f21b9b7c25469589dd733e1c1d92c87a`. The contracts found in the `src` folder that were included in scope were the following:

File
AaveTokenSubstitute.sol
AssetPool.sol
BondPositionManager.sol
BorrowController.sol
CouponManager.sol
CouponOracle.sol
DepositController.sol
LoanPositionManager.sol
OdosRepayAdapter.sol
libraries/* (excluding ReentrancyGuard.sol)

1.2 Audit Timeline

The audit was conducted from **23rd August** to **8th September**.

1.3 Fix Review

A review of the fixes was conducted subsequently from **8th September** to **4th October**.

1.4 Auditors Involved

HickupHH3

2 Risk Assessment Classification

There are 4 possible levels used to assess a vulnerability, with a separate section for gas optimizations.

High

Directly exploitable vulnerabilities with medium / high likelihood of loss of user funds, or contract functionality.

Resolving these issues are crucial to ensure the security and functionality of the contracts.

Medium

Vulnerabilities that relies on external dependencies / conditions to be met. Potentially leads to a loss of funds or functionality (eg. denial of service).

Resolving these issues are recommended to avoid undesired consequences.

Low

Issues arising from deviant behaviour than expected, but has no / little bearing from a security standpoint.

Informational

Issues that relate to security best practices recommendations, grammatical or styling errors, suggestions for variable/function name improvements etc. These issues are subjective and can be addressed based on the client's discretion.

While these issues may not directly affect the contract's functionality or security, addressing them can improve code readability, maintainability, and overall quality.

Gas Optimizations

Suggested changes to the codebase that will help reduce deployment or runtime gas costs, or to reduce the bytecode size should the limit be reached.

3 Codebase Impressions

Overall, while the codebase seems initially complex, its quality was high. There was very good abstraction such that the flow is rather similar for all user actions (lender deposits, withdrawals, borrowing, repayment and loan modifications).

It was accompanied with robust and comprehensive Foundry tests. This included integration testing with existing contracts on Arbitrum.

An area of improvement would be the addition of inline comments and documentation. The codebase took more time than expected to understand because of the lack of inline comments. This is especially so for assembly blocks (mostly used throughout the codebase for array length manipulation).

The documentation portal provides a high level overview and understanding of the protocol, but is in my opinion, inadequate for technical readers that will be diving deeply into the codebase.

3.1 User Flow

For clarity, here are all possible external actions:

3.1.1 Lenders

- Deposit: Loan out assets for a fixed period, where the loan interest is received upfront (hence a fixed rate deposit)
- Withdrawal: Withdrawing principal amount **before** loan maturity, where some interest may have to be paid back.
- Collection: Withdrawing principal amount **after** loan maturity.

3.1.2 Borrowers

- Borrow: Create an over-collateralised loan, where the loan interest is paid upfront; the user will receive less debt tokens than requested
- Lengthening / shortening of loan duration

- Collateral Addition / removal
- Repayment / borrowing more debt
- Repaying with collateral: swapping collateral for debt via an on-chain DEX (aggregator)

2 key things to note for borrowers:

1. Loans must be repaid or extended before expiry. The only recourse for expired positions is liquidation.
2. Full collateral withdrawal can be done before or after loan expiry, but only if debt has been fully repaid.

3.1.3 Liquidators

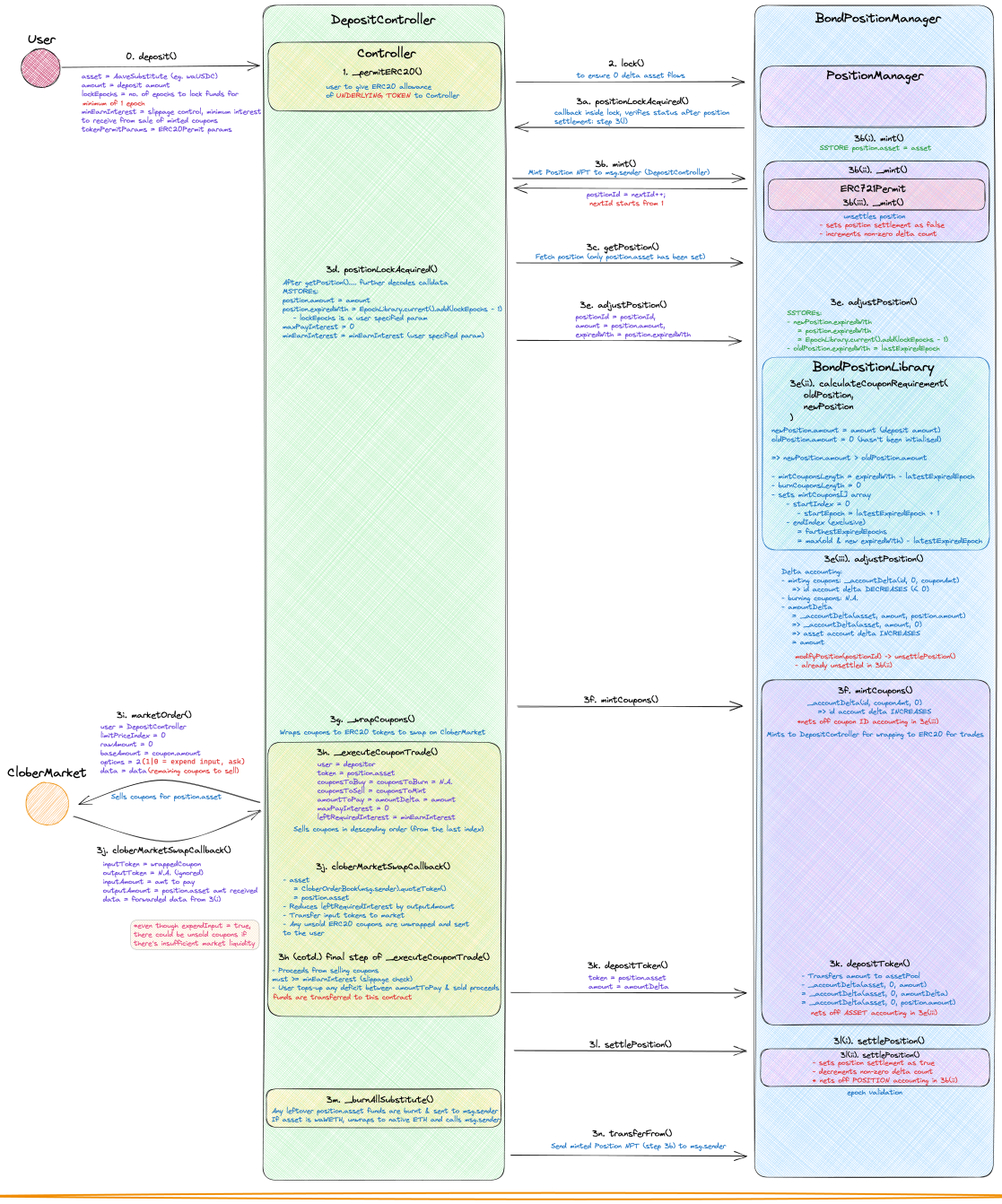
- Liquidation: Liquidating expired or under-collateralised unexpired positions. The LoanPositionLiquidateHelper contract would be a reference implementation

3.2 Flow Diagrams

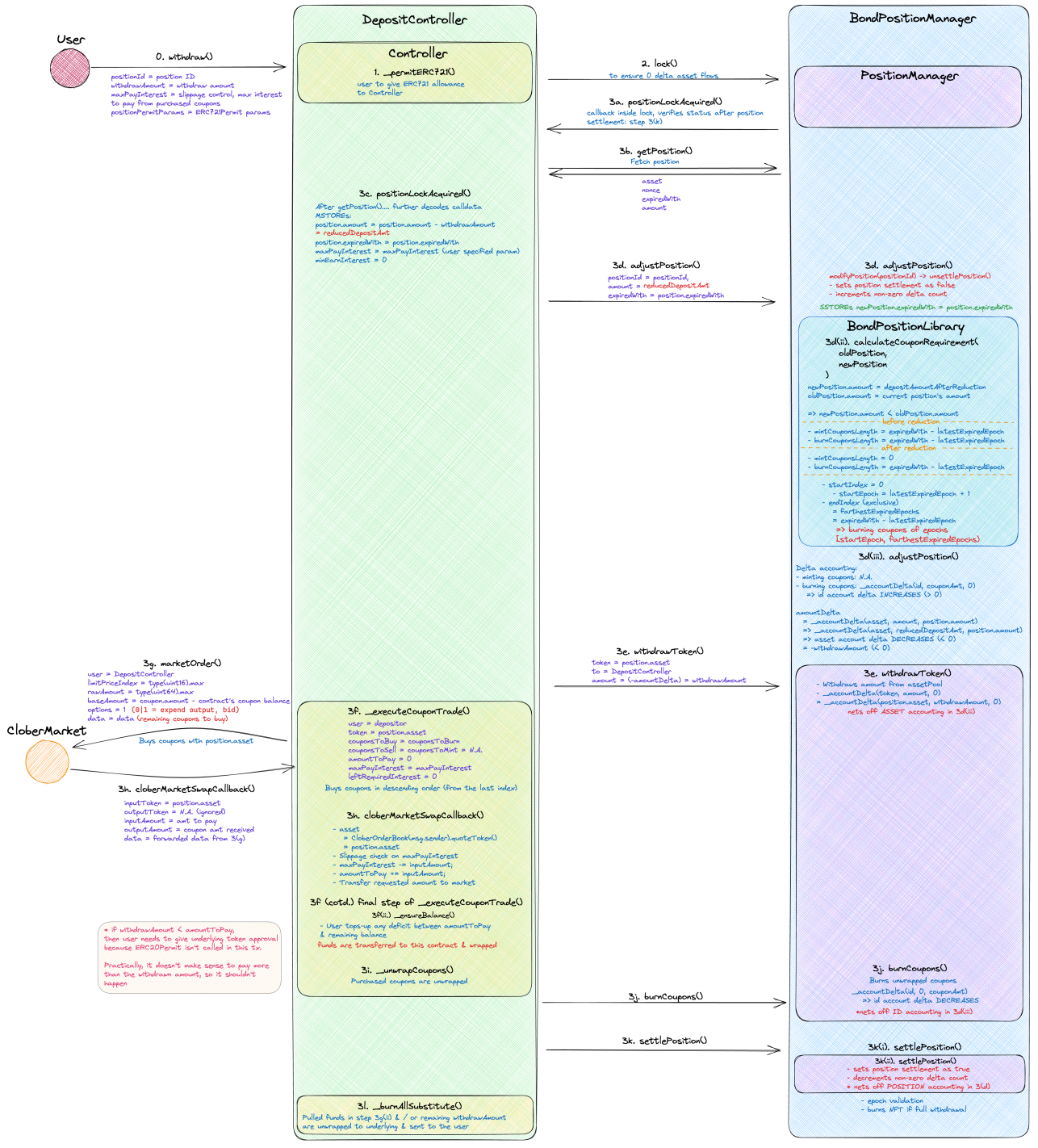
The flow for user actions are rather complex, and is daunting for new readers because of the numerous interactions and subcalls. To aid in understanding, I created state flow diagrams for some user actions, which can be found below.

One thing that becomes increasingly clear, is how the flow has been "standardized" for the different actions. It is simply a matter of understanding the flow of 1 action; the other actions are merely variants.

DEPOSIT FLOW

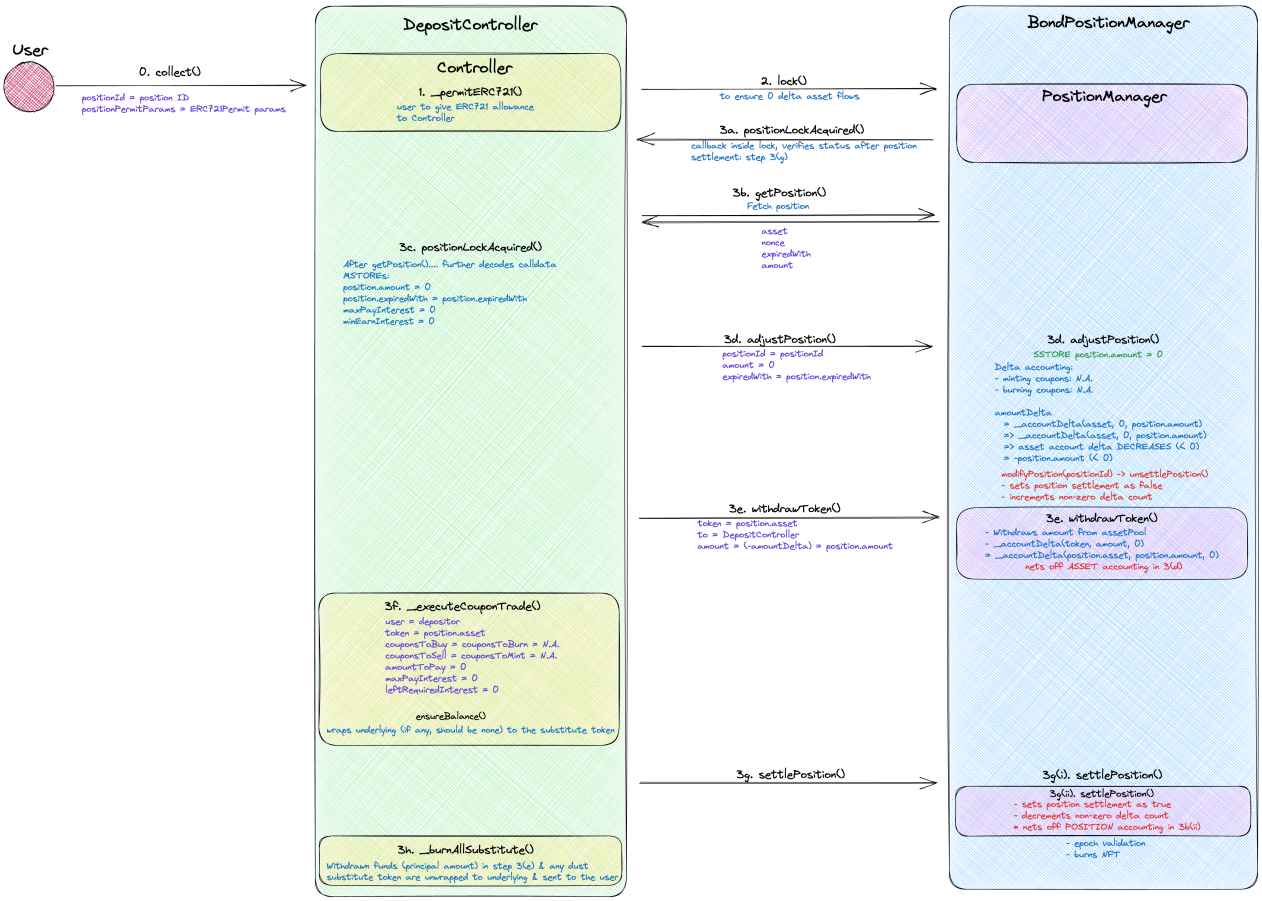


WITHDRAW FLOW

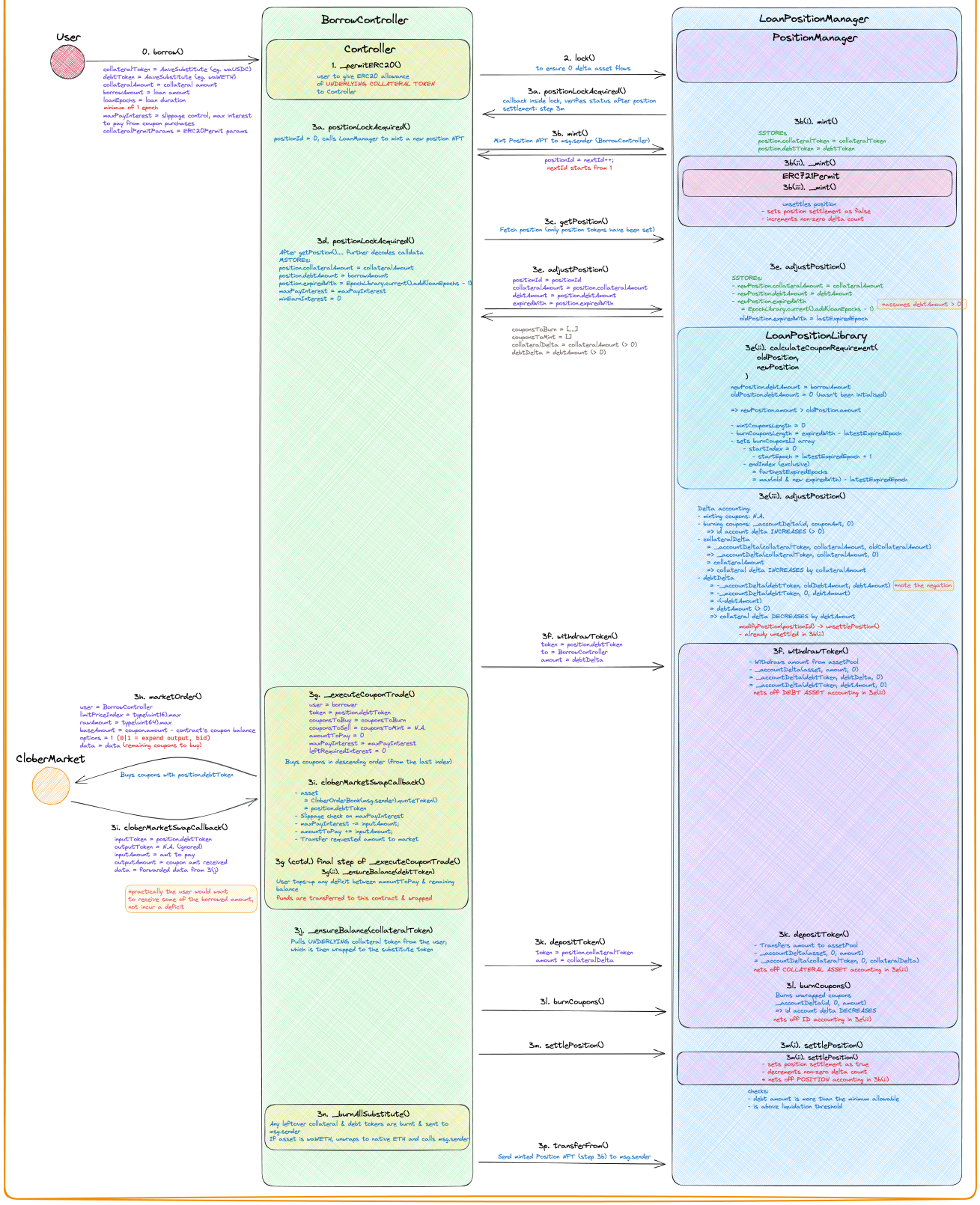


COLLECTION FLOW

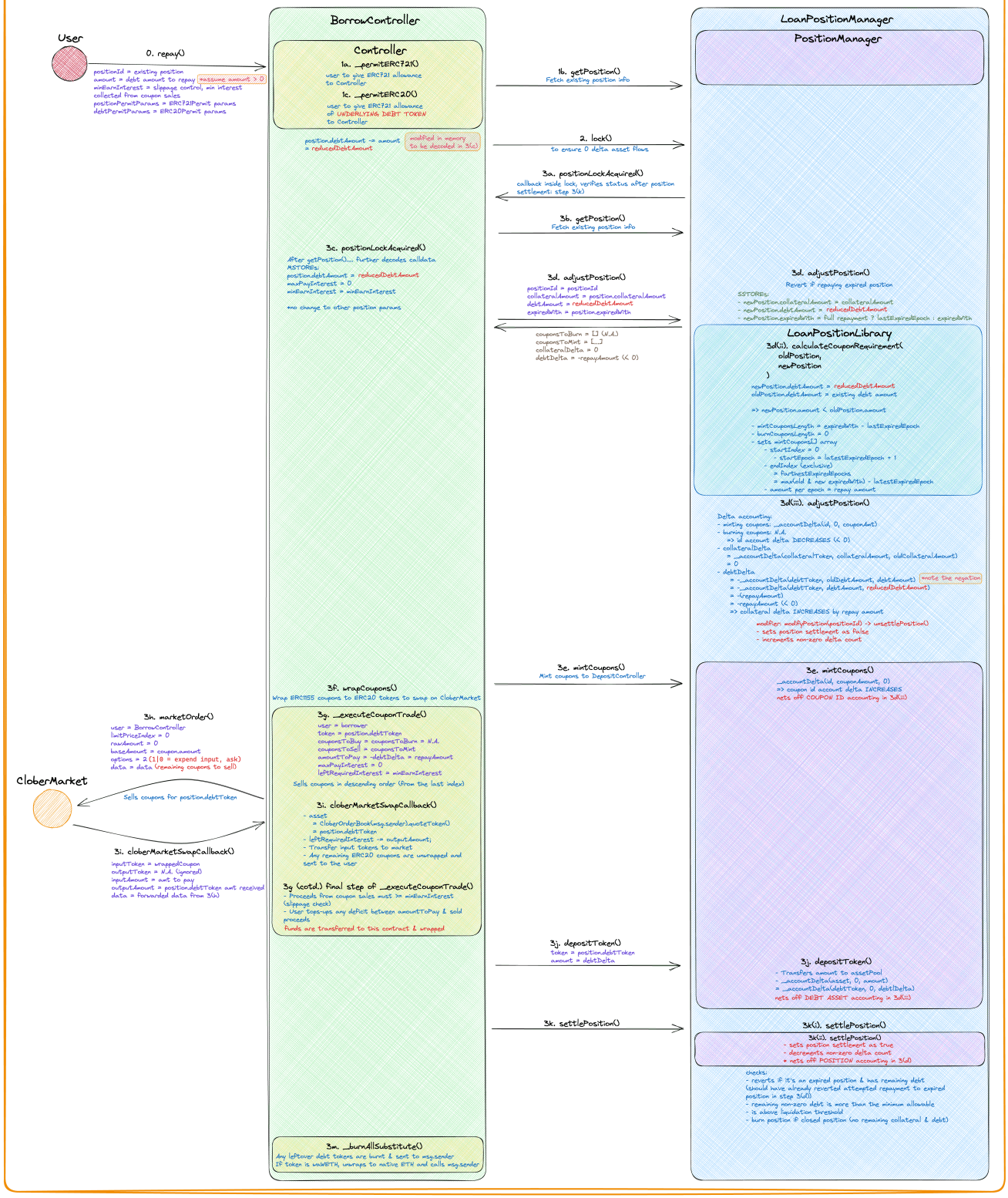
* Only considering the case of expired positions, as collection of non-expired positions would be similar to performing a full withdrawal



BORROW FLOW


User
0. borrow()
collateralToken = daveSubstitute (eg. waUSDC)
debtToken = daveSubstitute (eg. waWETH)
collateralAmount = collateral amount
borrowAmount = loan amount
loanEpochs = loan duration
minimum of 1 epoch
maxPayInterest = slippage control, max interest to pay from coupon purchases
collateralPermitParams = ERC20Permit params
BorrowController
Controller
1. _permitERC20()
user to give ERC20 allowance of UNDERLYING COLLATERAL TOKEN to Controller
3a. positionLockAcquired()
positionId = 0, calls LoanManager to mint a new position NFT
3d. positionLockAcquired()
After getPosition()... further decodes calldata
MSTORES:
position.collateralAmount = collateralAmount
position.debtAmount = borrowAmount
position.expiredWith = EpochLibrary.current().addNLoanEpochs - 1
maxPayInterest = maxPayInterest
minEarnInterest = 0
3h. marketOrder()
user = BorrowController
limitPriceIndex = type(uint16).max
rawAmount = type(uint64).max
baseAmount = coupon.amount - contract's coupon balance
options = 1 (0|1 = expend output, bid)
data = data (remaining coupons to buy)
CloberMarket
Buys coupons with position.debtToken
3i. cloberMarketSwapCallback()
inputToken = position.debtToken
outputToken = N.A. (ignored)
inputAmount = amt to pay
outputAmount = coupon amt received
data = forwarded data from 3(i)
practically the user would want to receive some of the borrowed amount, not incur a deficit
3g. _executeCouponTrade()
user = borrower
token = position.debtToken
couponsToBuy = couponsToBurn
couponsToSell = couponsToMint = N.A.
amountToPay = 0
maxPayInterest = maxPayInterest
leftRequiredInterest = 0
Buys coupons in descending order (from the last index)
3i. cloberMarketSwapCallback()
- asset
= CloberOrderBook(msg.sender).quoteToken()
= position.debtToken
- Slippage check on maxPayInterest
- maxPayInterest -= inputAmount;
- amountToPay += inputAmount;
- Transfer requested amount to market
3g (cotd.) final step of _executeCouponTrade()
3g(ii). _ensureBalance(debtToken)
User tops-up any deficit between amountToPay & remaining balance
Funds are transferred to this contract & wrapped
3j. _ensureBalance(collateralToken)
Pulls UNDERLYING collateral token from the user, which is then wrapped to the substitute token
3n. _burnAllSubstitute()
Any leftover collateral & debt tokens are burnt & sent to msg.sender
IF asset is waWETH, unwraps to native ETH and calls msg.sender
2. lock()
to ensure 0 delta asset flows
3a. positionLockAcquired()
callback inside lock, verifies status after position settlement: step 3a
3b. mint()
Mint Position NFT to msg.sender (BorrowController)
positionId = nextId++;
nextId starts from 1
3c. getPosition()
Fetch position (only position tokens have been set)
3e. adjustPosition()
positionId = positionId
collateralAmount = position.collateralAmount
debtAmount = position.debtAmount
expiredWith = position.expiredWith
couponsToBurn = [...]
couponsToMint = []
collateralDelta = collateralAmount (> 0)
debtDelta = debtAmount (> 0)
3f. withdrawToken()
token = position.debtToken
to = BorrowController
amount = debtDelta
3k. depositToken()
token = position.collateralToken
amount = collateralDelta
3l. burnCoupons()
3m. settlePosition()
3p. transferFrom()
Send minted Position NFT (step 3b) to msg.sender
LoanPositionManager
PositionManager
3b(i). mint()
SSTORES:
position.collateralToken = collateralToken
position.debtToken = debtToken
3b(ii). _mint()
ERC721Permit
3b(iii). _mint()
unsettles position
- sets position settlement as false
- increments non-zero delta count
3e. adjustPosition()
SSTORES:
- newPosition.collateralAmount = collateralAmount
- newPosition.debtAmount = debtAmount
- newPosition.expiredWith
= EpochLibrary.current().addNLoanEpochs - 1
assumes debtAmount > 0
oldPosition.expiredWith = lastExpiredEpoch
LoanPositionLibrary
3e(ii). calculateCouponRequirement(
oldPosition,
newPosition
)
newPosition.debtAmount = borrowAmount
oldPosition.debtAmount = 0 (hasn't been initialised)
=> newPosition.amount > oldPosition.amount
- mintCouponsLength = 0
- burnCouponsLength = expiredWith - latestExpiredEpoch
- sets burnCoupons[] array
- startIndex = 0
- startEpoch = latestExpiredEpoch + 1
- endIndex (exclusive)
= furthestExpiredEpochs
= max(old & new expiredWith) - latestExpiredEpoch
3e(iii). adjustPosition()
Delta accounting:
- minting coupons: N.A.
- burning coupons: _accountDelta(id, couponAmt, 0)
=> id account delta INCREASES (> 0)
- collateralDelta
= _accountDelta(collateralToken, collateralAmount, oldCollateralAmount)
=> _accountDelta(collateralToken, collateralAmount, 0)
= collateralAmount
=> collateral delta INCREASES by collateralAmount
- debtDelta
= -_accountDelta(debtToken, oldDebtAmount, debtAmount)
= -_accountDelta(debtToken, 0, debtAmount)
= -(-debtAmount)
= debtAmount (> 0)
=> collateral delta DECREASES by debtAmount
modifyPosition(positionId) -> unsettlePosition()
- already unsettled in 3b(ii)
3f. withdrawToken()
- Withdraws amount from assetPool
- _accountDelta(asset, amount, 0)
= _accountDelta(debtToken, debtDelta, 0)
= _accountDelta(debtToken, debtAmount, 0)
nets off DEBT ASSET accounting in 3g(ii)
3k. depositToken()
- Transfers amount to assetPool
- _accountDelta(asset, 0, amount)
= _accountDelta(collateralToken, 0, collateralDelta)
nets off COLLATERAL ASSET accounting in 3e(ii)
3l. burnCoupons()
Burns unwrapped coupons
_accountDelta(id, 0, amount)
=> id account delta DECREASES
nets off ID accounting in 3e(iii)
3m(i). settlePosition()
3m(ii). settlePosition()
- sets position settlement as true
- decrements non-zero delta count
* nets off POSITION accounting in 3b(ii)
checks:
- debt amount is more than the minimum allowable
- is above liquidation threshold

REPAY FLOW


User
0. repay()
positionId = existing position
amount = debt amount to repay
minEarnInterest = slippage control, min interest collected from coupon sales
positionPermitParams = ERC721Permit params
debtPermitParams = ERC20Permit params
BorrowController
Controller
1a. _permitERC721()
user to give ERC721 allowance to Controller
1c. _permitERC20()
user to give ERC721 allowance of UNDERLYING DEBT TOKEN to Controller
position.debtAmount -= amount = reducedDebtAmount
modified in memory to be decoded in 3(c)
3c. positionLockAcquired()
After getPosition()... further decodes calldata
MSTORES:
position.debtAmount = reducedDebtAmount
maxPayInterest = 0
minEarnInterest = minEarnInterest
+no change to other position params
3f. wrapCoupons()
Wrap ERC1155 coupons to ERC20 tokens to swap on CloberMarket
3g. _executeCouponTrade()
user = borrower
token = position.debtToken
couponsToBuy = couponsToBurn = N.A.
couponsToSell = couponsToMint
amountToPay = -debtDelta = repayAmount
maxPayInterest = 0
leftRequiredInterest = minEarnInterest
Sells coupons in descending order (from the last index)
3i. cloberMarketSwapCallback()
- asset
= CloberOrderBook(msg.sender).quoteToken()
= position.debtToken
- leftRequiredInterest -= outputAmount;
- Transfer input tokens to market
- Any remaining ERC20 coupons are unwrapped and sent to the user
3g (cont.) final step of _executeCouponTrade()
- Proceeds from coupon sales must >= minEarnInterest (slippage check)
- User tops-ups any deficit between amountToPay & sold proceeds
Funds are transferred to this contract & wrapped
3m. _burnAllSubstitute()
Any leftover debt tokens are burnt & sent to msg.sender
IF token is wnWETH, unwraps to native ETH and calls msg.sender
3h. marketOrder()
user = BorrowController
limitPriceIndex = 0
raw Amount = 0
baseAmount = coupon.amount
options = 2 (1|0 = expend input, ask)
data = data (remaining coupons to sell)
CloberMarket
Sells coupons for position.debtToken
3i. cloberMarketSwapCallback()
inputToken = wrappedCoupon
outputToken = N.A. (ignored)
inputAmount = amt to pay
outputAmount = position.debtToken amt received
data = forwarded data from 3(h)
1b. getPosition()
Fetch existing position info
2. lock()
to ensure 0 delta asset flows
3a. positionLockAcquired()
callback inside lock, verifies status after position settlement: step 3(k)
3b. getPosition()
Fetch existing position info
3d. adjustPosition()
positionId = positionId
collateralAmount = position.collateralAmount
debtAmount = reducedDebtAmount
expiredWith = position.expiredWith
couponsToBurn = [] (N.A.)
couponsToMint = [...]
collateralDelta = 0
debtDelta = -repayAmount (< 0)
3e. mintCoupons()
Mint coupons to DepositController
3j. depositToken()
token = position.debtToken
amount = debtDelta
3k. settlePosition()
LoanPositionManager
PositionManager
3d. adjustPosition()
Revert if repaying expired position
SSTORES:
- newPosition.collateralAmount = collateralAmount
- newPosition.debtAmount = reducedDebtAmount
- newPosition.expiredWith = full repayment ? lastExpiredEpoch : expiredWith
LoanPositionLibrary
3d(ii). calculateCouponRequirement(
oldPosition,
newPosition
)
newPosition.debtAmount = reducedDebtAmount
oldPosition.debtAmount = existing debt amount
=> newPosition.amount < oldPosition.amount
- mintCouponsLength = expiredWith - lastExpiredEpoch
- burnCouponsLength = 0
- sets mintCoupons[] array
- startIndex = 0
- startEpoch = latestExpiredEpoch + 1
- endIndex (exclusive)
= furthestExpiredEpochs
= maxValid & new expiredWith) - latestExpiredEpoch
- amount per epoch = repay amount
3d(iii). adjustPosition()
Delta accounting:
- minting coupons: _accountDelta(id, 0, couponAmt)
- burning coupons: N.A.
=> id account delta DECREASES (< 0)
- collateralDelta
= _accountDelta(collateralToken, collateralAmount, oldCollateralAmount)
= 0
- debtDelta
= -_accountDelta(debtToken, oldDebtAmount, debtAmount) note the negation
= -_accountDelta(debtToken, debtAmount, reducedDebtAmount)
= -(repayAmount)
= -repayAmount (< 0)
=> collateral delta INCREASES by repay amount
modifier modifyPosition(positionId) -> unsettlePosition()
- sets position settlement as false
- increments non-zero delta count
3e. mintCoupons()
_accountDelta(id, couponAmount, 0)
=> coupon id account delta INCREASES
nets off COUPON ID accounting in 3d(ii)
3j. depositToken()
- Transfers amount to assetPool
- _accountDelta(asset, 0, amount)
= _accountDelta(debtToken, 0, debtDelta)
nets off DEBT ASSET accounting in 3d(ii)
3k(). settlePosition()
3k(ii). settlePosition()
- sets position settlement as true
- decrements non-zero delta count
+ nets off POSITION accounting in 3(d)
checks:
- reverts if it's an expired position & has remaining debt (should have already reverted attempted repayment to expired position in step 3(d))
- remaining non-zero debt is more than the minimum allowable
- is above liquidation threshold
- burn position if closed position (no remaining collateral & debt)

3.3 Centralisation Risks

Considerable effort has been made to minimise owner privileges over the protocol:

- Assets can only be registered (added), but not removed
- `AssetPool` operators and `CouponManager` minters are immutable (set once in constructors)
- Loan pair configurations and asset price feeds can only be set once

The last point is a double-edged sword: there can be no tweaking of a loan pair's parameters after it's been set, meaning the ability to respond to a change of the risk nature of a collateral or debt token is limited.

Should the price oracle return 0 value, the protocol will attempt to fetch the price from a fallback oracle. The ability to set and update this fallback price oracle is therefore a notable centralisation risk, as it determines the solvency of loan positions.

4 Findings Summary

Severity	No. of issues
High	0
Medium	1
Low	6
Informational	7
Gas Optimizations	4
Total	18

4.1 [Medium] Lender withdrawals and collections can be bricked for Aave frozen markets

Context

Controller.sol#L189-L192

Details

Should an Aave market be frozen, supplying tokens into the market are disabled. Hence, should anyone accidentally or intentionally send some underlying tokens to the DepositController, lender withdrawals and collections will revert because of the attempt to supply the underlying token to the Aave market.

Impact

Lenders are unable to withdraw their principal amounts.

Mitigation

Wrap the Aave market supply in a try-catch statement. While it potentially opens up a new attack surface via gas limit manipulation where `supply()` reverts but the remaining tx execution is successful, the probability to exploit it seems low.

```
- _aaveV3Pool.supply(underlyingToken, amount, address(this), 0);
+ try _aaveV3Pool.supply(underlyingToken, amount, address(this), 0); {
+
+ } catch {}
```

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/76

Status

Fixed.

4.2 [Low] Lacking support for indirect USD price feeds

Context

CouponOracle.sol#L20-L28

Details

Some assets (eg. staking derivatives like rETH and wstETH) that can be supplied on AaveV3 Arbitrum do not have direct USD price feeds. The current implementation lacks support for such assets.

wstETH, for instance, is the 2nd largest supplied Aave asset on Arbitrum, but only the wstETH / ETH exchange rate is available. It would thus either require a composite price feed (eg. combining wstETH/ETH and ETH/USD prices) and complies with the `AggregatorV3Interface`, or rely on the fallback oracle.

Mitigation

Implement a composite price feed to support indirect asset/USD exchange rates.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/151.

Status

Fixed.

4.3 [Low] Lacking price freshness and Arbitrum sequencer checks

Context

CouponOracle.sol#L24-L25

Details

Chainlink recommends that all Optimistic L2 oracles consult the `SequencerUptimeFeed` to ensure that the sequencer is live before trusting the data returned by the oracle. This check "helps you prevent mass liquidations by providing a grace period to allow customers to react to such an event.", should the sequencer be offline.

In addition, there isn't a check for price staleness. This could cause liquidation status checks to be outdated, which could cause liquidatable / borderline solvent positions to remain solvent / liquidatable for longer / shorter than expected.

Mitigation

A reference implementation with sequencer checks has been provided here.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/100.

Status

Fixed.

4.4 [Low] Incorrect token used for `permit` in `extendLoanDuration()`

Context

BorrowController.sol#L159

Details

The token to give permit on has been incorrectly set as the `collateralToken`, so it'll very likely revert from a discrepancy in nonces.

Mitigation

Permission should be given on the `debtToken` instead.

```
- _permitERC20(position.collateralToken, maxPayInterest, debtPermitParams);
+ _permitERC20(position.debtToken, maxPayInterest, debtPermitParams);
```

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/81.

Status

Fixed.

4.5 [Low] Inconsistent treasury state between `AaveTokenSubstitute` and `LoanPositionManager`

Context

LoanPositionManager.sol#L80-L82
LoanPositionManager.sol#L30

Details

The `treasury` can be updated for `AaveTokenSubstitute`, but is immutable in `LoanPositionManager`.

Impact

Possible discrepancy in treasury addresses.

Mitigation

Keep the treasury state consistent in both contracts: either immutable or settable.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/132.

Status

Fixed. `LoanPositionManager` has its treasury mutable.

4.6 [Low] Lacking sanity checks when setting loan configuration

Context

LoanPositionManager.sol#L318-L339

Details

There are no sanity checks on the loan configuration params. Here are the implicit conditions:

- `0 < liquidationThreshold < RATE_PRECISION`

- `liquidationFee + liquidationTargetLtv < _RATE_PRECISION`
- `liquidationTargetLtv < liquidationThreshold`

Impact

Violation of the above condition(s) will cause `_getLiquidationAmount()` to revert, thus preventing liquidations.

Mitigation

It is recommended to add checks for these params, though these checks were presumably excluded as the `LoanPositionManager` is close to the bytecode size limit.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/130.

Status

Fixed.

```
if (
    liquidationThreshold >= _RATE_PRECISION || liquidationFee +
    liquidationTargetLtv >= _RATE_PRECISION
    || liquidationTargetLtv >= liquidationThreshold
) revert InvalidConfiguration();
```

4.7 [Low] `ERC721Permit` has redundant `payable` permit

Context

ERC721Permit.sol#L26
IERC721Permit.sol#L17
Controller.sol#L156

Details

The permit function is payable, but its invocation will not send native funds together with it. Hence, the payable keyword is redundant.

Impact

Loss of funds if users accidentally invoke `permit()` with non-zero `msg.value`.

Mitigation

Remove the `payable` keyword.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/131.

Status

Fixed.

4.8 [Info] `mintableAmount()` doesn't account for existing aToken supply or zero supply cap

Context

AaveTokenSubstitute.sol#L60-L64

Details

`mintableAmount()` fails to exclude the existing aToken supply, so it would return a larger amount than what's actually mintable.

Also, by default, the supply cap of an asset is 0, which signifies no cap. `mintableAmount()` returns 0 for this case.

Mitigation

Consider subtracting the existing aToken supply, and handle the case of zero supply caps.

Response

Fixed to `type(uint256).max` in https://github.com/clober-dex/coupon-finance/pu

Status

Acknowledged that `mintableAmount` returns a sanity max value. Noted and verified that in the same PR, the actual amount minted takes into account zero supply caps and handles the edge cases for frozen or paused Aave markets.

4.9 [Info] Clarify `endTime()` returned is exclusive

Context

Epoch.sol#L47-L49

Details

It isn't immediately clear whether the `endTime()` value returned is inclusive or exclusive. Based on its usage in `_getLiquidationAmount()`, one can infer that it is exclusive: `if (position.expiredWith.endTime() <= block.timestamp)` for ascertaining position expiry status.

However, the exclusivity of the `endTime` value breaks the inverse property: $f \circ f^{-1}(x) = x$. What we have is `_timestampToEpoch(_epochToTimestamp(epoch)) = epoch + 1`, which isn't intuitive.

Mitigation

Consider making `endTime()` inclusive instead, where `_epochToTimestamp()` returns a second lesser.

```
return (
    (months & 0xffff) + 365 * (year - 1970) + (year - 1969) / 4 - (year -
    1901) / 100 + (year - 1601) / 400
- ) * SECONDS_PER_DAY;
+ ) * SECONDS_PER_DAY - 1;
```

Otherwise, make it clear that `endTime()` is exclusive.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/133.

Status

Fixed. `endTime()`'s value has been modified to be inclusive.

4.10 [Info] Inconsistent return coupon order between `BondPositionLibrary` and `LoanPositionLibrary`

Context

BondPosition.sol#L32
LoanPosition.sol#L82

Details

The `BondPositionLibrary` and `LoanPositionLibrary` have reversed return parameters: the former returns `(mintCoupons, burnCoupons)` while the latter returns `(burnCoupons, mintCoupons)`.

This unnecessarily complicates readability when comparing the lender and borrower flows.

Mitigation

Keep the order consistent: recommend changing `LoanPositionLibrary` to match the order of `BondPositionLibrary`.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/134.

Status

Fixed.

4.11 [Info] Consider sending wrapped ERC20 coupons instead the unwrapped ERC1155 version

Context

LoanPositionManager.sol#L293-L297
OdosRepayAdapter.sol#L80
Controller.sol#L141-L144

Details

In a few cases, ERC1155 coupons are minted or unwrapped and sent to the user. The practicality and utility of their ERC20-wrapped counterparts can be argued to be better, especially considering how it's traded on Clober markets.

Furthermore, it's more secure as it avoids handing over flow control to the recipient by not invoking the `onERC1155BatchReceived()` when `mintBatch()` is called.

Mitigation

Send wrapped ERC20 coupons instead of the unwrapped ERC1155 version.

Response

Acknowledged.

Status

Acknowledged.

4.12 [Info] Rename `setCollateralAllowance()`

Context

BorrowController.sol#L205-L207
OdosRepayAdapter.sol#L121-L123

Details

The allowance given to the LoanPositionManager applies not just to collateral tokens, but debt tokens too.

Mitigation

Consider renaming to something more generic but purposeful, such as `giveLoanManagerAllowance(address token)`.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/138.

Status

Fixed.

4.13 [Info] Consider using `Ownable2Step` instead of `Ownable`

Details

OpenZeppelin has a new `Ownable2Step` contract that inherits `Ownable` which allows ownership transfers to be performed more securely in 2 steps.

Mitigation

Consider using `Ownable2Step` instead of `Ownable` for more secure ownership transfers.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/141.

Status

Fixed.

4.14 [Info] USDC fragmentation risk

Details

There is both bridged USDC (USDC.e) and native USDC that Circle introduced on Arbitrum. This complicates matters because it is likely for both to use the same price feed, but the bridged USDC could possibly de-peg from the native one due to bridge hacks, or other black swan events.

In addition, they share the same token symbol, so while they have separate addresses, the ERC20 wrapped coupon symbol and name would be the same too.

Mitigation

Something to bear in mind and consider as an edge case scenario to deal with.

Response

N.A.

Status

N.A.

4.15 [Gas] Replace `new bytes(0)` with `""`

Context

DepositController.sol#L59
BorrowController.sol#L63
OdosRepayAdapter.sol#L59

Details

It is more gas efficient to use `""` as an argument to provide empty calldata instead of `new bytes(0)`.

Mitigation

```
- _loanManager.mintCoupons(couponsToMint, address(this), new bytes(0));
+ _loanManager.mintCoupons(couponsToMint, address(this), "");

- _bondManager.mintCoupons(couponsToMint, address(this), new bytes(0));
+ _bondManager.mintCoupons(couponsToMint, address(this), "");
```

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/135.

Status

Fixed.

4.16 [Gas] Redundant appended "" param

Context

BorrowController.sol#L102

Details

There is a redundant "" param appended to the `lockData` encoding.

Mitigation

```
- collateralAmount, borrowAmount, EpochLibrary.current().add(loanEpochs - 1),
    maxPayInterest, 0, ""
+ collateralAmount, borrowAmount, EpochLibrary.current().add(loanEpochs - 1),
    maxPayInterest, 0,
```

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/77.

Status

Fixed.

4.17 [Gas] Redundant conditional check

Context

LoanPositionManager.sol#L227

Details

The former condition `repayAmount > newRepayAmount` is a subset of the latter condition `newRepayAmount < minDebtAmount + repayAmount`, and is therefore redundant.

Mitigation

```
- } else if (repayAmount > newRepayAmount || newRepayAmount < minDebtAmount +
    repayAmount) {
+ } else if (newRepayAmount < minDebtAmount + repayAmount) {
```

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/136.

Status

Fixed.

4.18 [Gas] `cloberMarketSwapCallback()` can be unchecked

Context

Controller.sol#L101-L136

Details

There are conditional checks prior to the additions and subtractions performed such that there will be no integer under/overflow issues.

Mitigation

Wrap the function in an unchecked block.

Response

Fixed in https://github.com/clober-dex/coupon-finance/pull/137.

Status

Fixed.

5 Disclaimer

The audit report provided reflects a thorough review conducted to the best of my ability. However, it is important to note that the time-boxing nature of the review and available resources may prevent the discovery of all potential security vulnerabilities. As such, this audit does not guarantee the absence of undiscovered vulnerabilities.

Furthermore, please be aware that the security review was conducted on a specific commit of the codebase, as indicated. Any subsequent modifications made to the code will necessitate a new security review to ensure comprehensive coverage.

Note that the contracts used in production and expected deployment values may defer significantly from what was reviewed.

To ensure a robust evaluation of the codebase, it is highly recommended to engage multiple auditors and firms, particularly for large and complex projects. The involvement of multiple perspectives can provide additional insights and potential missed vulnerabilities.

Please consider these factors when assessing the audit report and making decisions related to the security and reliability of the smart contracts. The security review is not an endorsement of the project or its team, and should not be treated as such.