By
Quit

Contract Review
Issue date
5/10/2024

Overview

The following is a review of Gondi V3, an NFTfi platform that allows lending and borrowing across various collections on Ethereum Mainnet. Loans are offered against a given asset, and can be refinanced (either fully or partially) while active, as long as the new loan is strictly better than the one it is replacing.

Users can also deposit to an ERC4626 style pool, which can be tapped for loans according to granular terms defined by the contract owner. While not in use, pooled funds are put to work accumulating interest via Aave or Lido.

Contracts in scope for this review include:

- `AddressManager.sol`
- `AuctionLoanLiquidator.sol`
- `AuctionWithBuyoutLoanLiquidator.sol`
- `InputChecker.sol`
- `LiquidationDistributor.sol`
- `LiquidationHandler.sol`
- `Multicall.sol`
- `UserVault.sol`
- `callbacks/CallbackHandler.sol`
- `callbacks/PurchaseBundler.sol`
- `loans/BaseLoan.sol`
- `loans/BaseLoanHelpers.sol`
- `loans/LoanManager.sol`
- `loans/LoanManagerRegistry.sol`
- `loans/MultiSourceLoan.sol`
- `loans/WithLoanManager.sol`
- `utils/Hash.sol`
- `utils/Interest.sol`
- `utils/TwoStepOwned.sol`
- `utils/ValidatorHelpers.sol`
- `utils/WithProtocolFee.sol`
- `validators/NftBitVectorValidator.sol`
- `validators/NftPackedListValidator.sol`
- `validators/RangeValidator.sol`

This review is based on SHA `ac51cc6102fcf5ab274f8812eb585539332431f4`, and aims to identify security vulnerabilities, opportunities for gas optimization, and general best practice recommendations with regards to the contracts in scope. The review should not be considered an endorsement of the project, nor is it a guarantee of security.

Findings

Incorrect @title specification
Severity: Informational
In `AaveUsdcBaseInterestAllocator.sol`, the contract's natspec specifies `EthBaseInterestAllocator` as the title. Recommend changing this to `AaveUsdcBaseInterestAllocator`.

Several TODOs exist throughout the codebase
Severity: Informational
There are several TODO comments that have not been reconciled throughout the codebase. TODO typically indicates unfinished code. Recommend completing these before going live. The following lines are affected:

- `AuctionWithBuyoutLoanLiquidator.sol L61`
- `LoanManager.sol L10`
- `AaveUsdcBaseInterestAllocator.sol L16, L90`
- `ValidatorHelpers.sol L4`
- `NftPackedListValidator L4`

View functions are not declared as `view` in interface
Severity: Informational
The following functions are declared as `view` in their implementations, but not in the underlying interface. Recommend marking their visibility as `view` in the interfaces as well.

- `IFeeManager.sol`
  - `getFees()`
  - `getPendingFees()`
  - `getPendingFeesSetTime()`
- `IPool`
  - `getMaxTotalWithdrawalQueues()`
  - `getMinTimeBetweenWithdrawalQueues()`
  - `getBaseInterestAllocator()`
  - `isActive()`
  - `getPendingBaseInterestAllocator()`
  - `getPendingBaseInterestAllocatorSetTime()`
- `IPoolOfferHandler`
  - `getMaxDuration()`

`getPendingBaseInterestAllocator` and `getPendingBaseInterestAllocatorSetTime` can be packed together into a struct

Severity: Gas Optimization

In `Pool.sol`, the base interest allocator can be set in a two step process that involves declaring the new interest allocator to be set, waiting for `UPDATE_WAIT_TIME`, and then confirming the new interest allocator. Only the contract owner can declare a new interest allocator, but anybody can confirm it after `UPDATE_WAIT_TIME` has passed. Two variables are set in both functions - `getPendingBaseInterestAllocatorSetTime` and `getPendingBaseInterestAllocator`. `getPendingBaseInterestAllocatorSetTime` can comfortable fit into a uint96 so that both variables can be packed into a struct and stored with a single `SSTORE`.

Arithmetic can be marked `unchecked` where safe

Severity: Gas Optimization

Where there is no concern of over/underflow, arithmetic throughout the contracts can be marked `unchecked` to save a small amount of gas. Some example lines that are affected include:

- `WithdrawalQueue.sol L67, L97-99, L124, L128, 143`
- `Pool.sol L152, L163, L323, L444, L461, L476, L504, L509-510, L525, L546-547, L649, L663, L672, L689-690, L741, L743, L749`

The savings are small, but recommend implementing unchecked math where you feel comfortable.

Recommend a push-pull mechanism for `TwoStepOwned`

`Severity: Low`

It is currently possible to mistakenly set an inaccessible address via `TwoStepOwned`. As an extra security step to ensure that the new owner is accessible, recommend altering the second step signature to remove the address input, leaving just `transferOwnership()` as the function signature. Then, ensure that `msg.sender == pendingOwner` before completing the handover.

Summary

Gondi contracts are generally well written, follow best practices, and make a best effort at avoiding centralization risk. The two step process with a minimum wait is appreciated, as it gives users time to react should the contract owner attempt to tweak settings in a malicious or unfavorable way.

Interactions with outside contracts are mostly considered a black box in the eyes of this review - so, it is essential to remain cautious when dealing with parts of the code that interact with Lido,

Aave, etc. NFTs with nonstandard behavior could also lead to unintended behavior within Gondi, so it is important to ensure compatibility before adding support for new NFT contracts.

Overall, Gondi continues to push the boundaries of NFTfi, providing new ways to ensure capital efficiency whether actively lending or not. As with any sufficiently complex protocol, users should exercise caution, especially in the days closer to launch. However, no issues of concern were uncovered as a result of this audit.

Note: findings related to the pool mechanism have been removed from this report, as the mechanism has been pushed to be included in a later version