By
Quit

Contract Review
Issue date
10/28/2023

Overview

The following is a review of Gondi, an NFT lending platform that is launching their V2 set of contracts. Gondi allows buyers and sellers to participate in loan agreements, with the key changes from V1 revolving around the addition of BNPL.

The contracts are generally well written. Natspec comments exist through each file, custom errors provide accurate descriptors of reverts, and best practices are followed throughout. As is the case with any lending/borrowing platform, there are many complex interactions to consider, especially given interaction with outside contracts such as marketplaces and the internal use of multicall as a generalized batching mechanism.

This review included searching opportunities for gas optimization, general best practice recommendations, and a security assessment. When assessing the security, the focus was in looking for:

- Opportunities to leverage reentrancy to wrongfully extract assets from the protocol
- Opportunities to lock assets in the protocol
- Avenues for exploitation that could include:
    - Causing unintended loan behavior
    - Loss of assets for a borrower or lender
    - Potential trust requirements that result in an aspect of centralization

Users should exhibit caution when dealing with any newly deployed contracts, especially those as complex as Gondi. However, no critical vulnerabilities were found and the protocol logic appears to be sound.

Findings

Use preincrement operator on storage variables during assignment of memory variables
Severity: None (nit) Status: Addressed
There are several places within the protocol that a storage value is incremented and cached into a local variable, and then reassigned back to the storage variable. This can be simplified by into a single line with `localVar = ++storageVar`, as seen below for `mint()` (same exists in `_getAndSetNewLoanId()`):

```
89      /// @inheritdoc IUserVault
90      function mint() external nonReentrant returns (uint256) {
91          uint256 vaultId = ++_totalSupply;
92
93          _mint(msg.sender, vaultId);
94          return vaultId;
95      }
```

Use tx.origin over msg.sender when initializing `Owned`

Severity: Low Status: Addressed

Using tx.origin here will allow for counterfactual deployments (for instance, through the KeylessCreate2Factory by 0age) without having to hardcode an ownership address. Using msg.sender would fail here due to the contract intermediary. Counterfactual deployments offer many advantages over the standard `create`, and standard `create` is unaffected by switching to tx.origin here.

Use unchecked blocks where practical

Severity: Low Status: Addressed

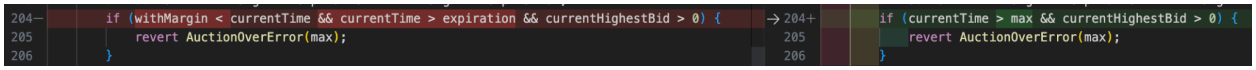
There are several functions that could be optimized very slightly by adding an unchecked block. For instance, `mint` increments _totalSupply by 1 unit at a time. As a uint256, there is not enough ETH in existence to overflow _totalSupply. Recommend adding unchecked blocks to save a bit of gas.

```
89      /// @inheritdoc IUserVault
90      function mint() external nonReentrant returns (uint256) {
91          uint256 vaultId;
92          unchecked {
93              vaultId = ++_totalSupply;
94          }
95
96          _mint(msg.sender, vaultId);
97          return vaultId;
98      }
```

Consolidate time-based checks

Severity: Low Status: Addressed

AuctionLoanLiquidator L204: While placing a bid, the contract assigns the greater of `withMargin` and `expiration` to be the current auction end time. It then compares the `currentTime` (`block.timestamp`) to both, and if `currentTime` is greater than both the auction is deemed to be over and an error is thrown. This check can be simplified by comparing `currentTime` to `max` directly, instead of comparing to both `withMargin` and `expiration`.



Contract Owner can cause unexpected behavior by manipulating _triggerFee
Severity: Low Status: Addressed
_triggerFee can be changed while an auction is ongoing. It is possible for the contract owner to raise the fee (up to the max of 500 bps) or lower the fee (as low as zero) arbitrarily, potentially even frontrunning the settlement transaction. As an alternative, the current _triggerFee could be saved as part of the auction struct at creation.

settleAuction being publicly executable effectively makes _triggerFee a validator bribe
Severity: Low Status: Acknowledged. The team has chosen not to change the behavior here.
Upon settlement of an auction, half of the trigger fee is distributed to the auction originator. The other half is sent to the address that triggered the settlement. Bots will arbitrage this, and competition will drive miner tips to approach parity with _triggerFee (race to the bottom in terms of profits for settlers). Marked as low because if the goal is simply to encourage auction settlement, this will still accomplish that goal. But, could consider a mechanism to avoid excessive validator bribes and drive more profit to the originator.

Delegations are not cleared upon loan finalization
Severity: Medium Status: Addressed
A borrower is able to call the delegate function to assign any number of delegates for a borrowed ERC721 token. If the loan is liquidated or otherwise finalized, these delegations are not cleared. A future borrower using the same token is able to clear these delegations, but will have to both be aware of the delegations, and pay the gas to revoke each one at a time. Existing delegations should be explicitly stated so that borrowers are aware of all possible scenarios before depositing an NFT.

Conclusion

The contract will function as stands, but may have some low severity unexpected behaviors. There are tweaks that can be made to improve effectiveness and lower overall gas consumption. Centralization risk is low and the protocol seems to have been architected with trustlessness in mind. The addition of BNPL along with Delegatexyz support make the Gondi a powerful tool in the NFTfi space.