Gondi Security Assessment

Nov 3, 2023

Prepared for

Gondi

Prepared by:

0xfoobar

Table of Contents

Summary

Overview

Gondi is an NFT lending platform supporting borrowing, lending, and automatic refinancing.

Project Scope

We reviewed the core smart contracts and test suite contained within commit hash \`ce97fe6841013303f6df58098b9fd7ec958b02fe\` on branch \`develop\` at https://github.com/pixeldaogg/florida-contracts. Fixes were reviewed at commit hash 867d7da52d7a93f88cee75beeaacbb0e2df65366 on branch \`develop\` at https://github.com/pixeldaogg/florida-contracts. Code in src/lib/validators was not reviewed at team's request. We also noted concurrent findings from other auditors such as Quantstamp and do not double-list issues from their report that have been remediated in the latest commit hash.

Severity Classification

High - Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).
Medium - Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.
Low - Assets are not at risk: state handling, function incorrect as to spec, issues with comments.
Informational - Code style, clarity, syntax, versioning, off-chain monitoring (events, etc)

Summary of Findings

Severity	Findings	Resolved
High	**0**	**-**
Medium	**0**	**-**
Low	**3**	**2**
Informational	**7**	**4**

Disclaimer

This security assessment should not be used as investment advice.

We do not provide any guarantees on eliminating all possible security issues. foostudio recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

We did not assess the following areas that were outside the scope of this engagement:

- Website frontend components
- Offchain order management infrastructure
- Multisig or EOA private key custody
- Metadata generation

Key Findings and Recommendations

1. Using address(0) to denote ETH can lead to invalid storage writes

Severity: **Low**
Files: UserVault.sol

Description

ETH is not an ERC20, and so the contract uses a magic constant value in place of a token address. However using address(0) as the default has the failure mode of being the default value after storage data is cleared, and also the default value for an empty input for some contract interaction interfaces.

Impact

Storage corruption or accidental user inputs.

Recommendation

Follow the pattern adopted in EIP-7528, EIP-7535 and other top DeFi protocols such as Yearn, Curve, and Uniswap of using the magic value `` `0xEeeeeEeeeEeEeeEeEeeEEEeeeeEeeeeeeeEEeE` `` to denote ETH.

Response

Fixed.

2. Stop untrusted external callbacks by replacing safeTransfer with transfer

Severity: **Low**
Files: IWrappedPunk.sol, Leverage.sol, MultiSourceLoan.sol

Description

The `safeTransferFrom` method is unfortunately overloaded by ecosystem convention. Its first usage in ERC20 SafeTransferLib is useful and desirable to handle nonstandard ERC20s such as USDT. However its usage in ERC721 to trigger unsafe external callbacks on receiving contracts opens up unnecessary novel attack surface to the protocol. Since users can only grief themselves, there is no serious threat of NFTs being lost up by a malicious attacker.

Impact

Heavy reentrancy guards become necessary, and far more difficult to reason about behavior when safeTransferFrom callbacks violate checks-effects-interactions.

Recommendation

Replace ERC721 safeTransferFrom calls with transferFrom calls wherever applicable.

Response

Partially fixed.

3. Possible precision loss on interest rate calculation

Severity: **Low**
Files: Interest.sol

Description

Premature rounding in Interest.sol::_getInterest(). This leads to precision loss and potentially compounding mathematical errors.

Impact

Precision loss and potentially compounding mathematical errors.

Recommendation

For math functions that will not overflow, perform all multiplications in the numerator and denominator separately before doing the division as the final step. This is more accurate than smaller fractions being multiplied together. The new logic would be `return _amount.mulDivUp(_aprBps*_duration, _PRECISION*_SECONDS_PER_YEAR);`

Response

Not fixed.

4. Avoid msg.sender usage in internal functions

Severity: **Informational**
Files: UserVault.sol

Description

Not a vulnerability at present, but generally good to avoid using msg.sender or msg.value variables in internal functions and pass those as function parameters instead, only reading them in the outermost external func. Makes the internal funcs more stateless and less likely to misuse.

Impact

Potential bugs introduced later.

Recommendation

Pass `msg.sender` as a `from` parameter when going from external functions to internal.

Response

Not implemented.

5. AddressManager has stale unused functions

Severity: **Informational**
Files: AddressManager.sol

Description

`addressToIndex` and `indexToAddress` are not used in the V2 code. These can be removed while still preserving the singleton whitelist contract for easy verifiability and reuse.

Impact

Increased deployment size and gas costs.

Recommendation

Remove unused functions and state.

Response

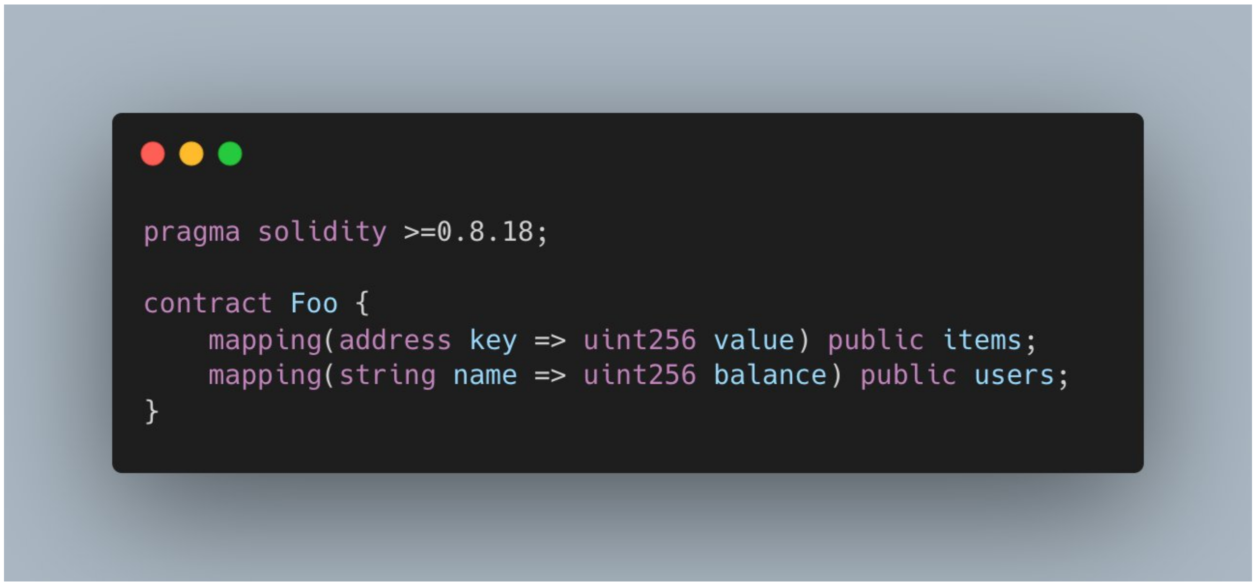
Not implemented.

6. Use named mappings

Severity: **Informational**
Files: *.sol

Description

Solidity 0.8.18 and later support a language feature called named mappings, where keys and values can be named to give devs better documentation of what's expected in each.



```
pragma solidity >=0.8.18;

contract Foo {
    mapping(address key => uint256 value) public items;
    mapping(string name => uint256 balance) public users;
}
```

Impact

Extraneous comments that could be inlined.

Recommendation

Experiment with named mappings where the storage layout otherwise requires comment explanations.

Response

Not implemented.

7. Remove unused imports and use relative imports over absolute

Severity: **Informational**
Files: *.sol

Description

Cleanup was pushed in this PR: https://github.com/pixeldaogg/florida-contracts/pull/280

8. Fix Natspec

Severity: **Informational**
Files: *.sol

Description

The first word of Natspec return comments should be the name of the returned variable, instead of the first word of the description of it.

Impact

Cleaner autogenerated docs.

Recommendation

Update Natspec.

Response

Fixed.

9. Add multicall

Severity: **Informational**
Files: UserVault.sol

Description

Currently you have single and plural methods for several actions - depositNft & depositNfts, withdrawNft & withdrawNfts, withdrawToken & withdrawTokens, etc

It makes sense to use a generalizable multicall here instead, like so:

```
function multicall(bytes[] calldata data) external payable override returns (bytes[] memory
results) {
        results = new bytes[](data.length);
        bool success;
        unchecked {
                for (uint256 i = 0; i < data.length; ++i) {
                        //slither-disable-next-line calls-loop,delegatecall-loop
                        (success, results[i]) = address(this).delegatecall(data[i]);
                        if (!success) revert MulticallFailed();
                }
        }
}
```

Impact

Cleaner UX for frontend batching implementation, less deployment codesize.

Response

Implemented.

10. Replace private mappings + public getters with public mappings

Severity: **Informational**
Files: *.sol

Description

Solidity automatically populates getter functions for public mappings. So it’s cleaner to replace private mappings that expose one-to-one getter functions with a public mapping.

Impact

Code bloat.

Recommendation

Simplify these into public mappings.

Response

Implemented.