AbraNFT contest

2022-07-18

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the AbraNFT smart contract system written in Solidity. The audit contest took place between April 27—May 1 2022.

Wardens

72 Wardens contributed reports to the AbraNFT contest:

1. gzeon
2. hyh
3. BowTiedWardens (BowTiedHeron, BowTiedPickle, m4rio_eth, Dravee, and BowTiedFirefox)
4. WatchPug (jtp and ming)
5. catchup
6. IllIllI
7. Ruhum
8. plotchy
9. GimelSec (rayn and sces60107)
10. kenzo
11. cccz
12. horsefacts
13. scaraven
14. berndartmueller
15. 0x1337
16. 0xf15ers (remora and twojoy)

17. antonttc
18. AuditsAreUS
19. Czar102
20. joestakey
21. bobi
22. reassor
23. defsec
24. robee
25. z3s
26. oyc_109
27. pauliax
28. simon135
29. delfin454000
30. CertoraInc (egjlmn1, OriDabush, ItayG, and shakedwinder)
31. Funen
32. samruna
33. kenta
34. 0x1f8b
35. MaratCerby
36. 0xkatana
37. bobirichman
38. sikorico
39. m9800
40. ilan
41. broccolirob
42. unforgiven
43. gs8nrv
44. jah
45. hubble (ksk2345 and shri4net)
46. kebabsec (okkothejawa and FlameHorizon)
47. throttle
48. mics
49. 0xDjango
50. Tomio
51. slywaters
52. 0xNazgul
53. Tadashi
54. NoamYakov
55. sorrynotsorry
56. TrungOre
57. Kulk0
58. fatherOfBlocks
59. Hawkeye (0xwags and 0xmint)

This contest was judged by 0xean.

Final report assembled by itsmetechjay and liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 6 unique vulnerabilities. Of these vulnerabilities, 5 received a risk rating in the category of HIGH severity and 1 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 45 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 33 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 AbraNFT contest repository, and is composed of 2 smart contracts written in the Solidity programming language and includes 1,333 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

High Risk Findings (5)

[H-01] Avoidance of Liquidation Via Malicious Oracle

Submitted by BowTiedWardens, also found by gzeon, and hyh

Issue: Arbitrary oracles are permitted on construction of loans, and there is no check that the lender agrees to the used oracle.

Consequences: A borrower who requests a loan with a malicious oracle can avoid legitimate liquidation.

Proof of Concept

- Borrower requests loan with an malicious oracle
- Lender accepts loan unknowingly
- Borrowers's bad oracle is set to never return a liquidating rate on `oracle.get` call.
- Lender cannot call `removeCollateral` to liquidate the NFT when it should be allowed, as it will fail the check on L288
- To liquidate the NFT, the lender would have to whitehat along the lines of H-01, by atomically updating to an honest oracle and calling `removeCollateral`.

Mitigations

- Add `require(params.oracle == accepted.oracle)` as a condition in `_lend`
- Consider only allowing whitelisted oracles, to avoid injection of malicious oracles at the initial loan request stage

cryptolyndon (AbraNFT) confirmed and commented: > Oracle not compared to lender agreed value: confirmed, and I think this is the first time I've seen this particular vulnerability pointed out. Not marking the entire issue as a duplicate for that reason. > > Oracle not checked on loan request: Not an issue, first reported in #62.

[H-02] The return value `success` of the get function of the INFTOracle interface is not checked

Submitted by cccz, also found by Ruhum, catchup, IllIllI, WatchPug, berndartmueller, plotchy, antonttc, hyh, and 0xf15ers

```
    function get(address pair, uint256 tokenId) external returns (bool success, uint256 rate
```

The get function of the INFTOracle interface returns two values, but the success value is not checked when used in the NFTPairWithOracle contract. When success is false, NFTOracle may return stale data.

Proof of Concept

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/interfaces/INFTOracle.sol#L10-L10

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L287-L287 https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L321-L321

Recommended Mitigation Steps

`(bool success, uint256 rate) = loanParams.oracle.get(address(this), tokenId); require(success);`

cryptolyndon (AbraNFT) confirmed and commented: > Agreed, and the first report of this issue.

0xean (judge) increased severity to High and commented: > I am upgrading this to High severity. > > This is a direct path to assets being lost. > > `> 3 - High: Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals). >`

[H-03] Critical Oracle Manipulation Risk by Lender

Submitted by 0x1337, also found by catchup, cccz, kenzo, GimelSec, BowTiedWardens, gzeon, horsefacts, and hyh

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L286-L288

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L200-L211

The intended use of the Oracle is to protect the lender from a drop in the borrower's collateral value. If the collateral value goes up significantly and higher than borrowed amount + interest, the lender should not be able to seize the collateral at the expense of the borrower. However, in the `NFTPairWithOracle` contract, the lender could change the Oracle once a loan is outstanding, and therefore seize the collateral at the expense of the borrower, if the actual value of the collateral has increased significantly. This is a critical risk because borrowers asset could be lost to malicious lenders.

Proof of Concept

In `NFTPairWithOracle`, the `params` are set by the `borrower` when they call `requestLoan()`, including the Oracle used. Once a lender agrees with the parameters and calls the `lend()` function, the `loan.status` changes to `LOAN_OUTSTANDING`.

Then, the lender can call the `updateLoanParams()` function and pass in its own `params` including the Oracle used. The `require` statement from line 205 to 211 does not check if `params.oracle` and `cur.oracle` are the same. A malicious lender could pass in his own `oracle` after the loan becomes outstanding, and the change would be reflected in line 221.

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L200-L211

In a situation where the actual value of the collateral has gone up by a lot, exceeding the amount the lender is owed (principal + interest), the lender would have an incentive to seize the collateral. If the Oracle is not tampered with, lender should not be able to do this, because line 288 should fail. But a lender could freely change Oracle once the loan is outstanding, then a tampered Oracle could produce a very low `rate` in line 287 such that line 288 would pass, allowing the lender to seize the collateral, hurting the borrower.

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L286-L288

Recommended Mitigation Steps

Once a loan is agreed to, the oracle used should not change. I'd recommend adding a check in the `require` statement in line 205 - 211 that `params.oracle == cur.oracle`

cryptolyndon (AbraNFT) confirmed and commented: > Confirmed, this is bad. First report of this particular exploit.

[H-04] Lender is able to seize the collateral by changing the loan parameters

Submitted by Ruhum, also found by IllIllI, WatchPug, BowTiedWardens, gzeon, plotchy, and scaraven

https://github.com/code-423n4/2022-04-abranft/blob/main/contracts/NFTPairWithOracle.sol#L198-L223

https://github.com/code-423n4/2022-04-abranft/blob/main/contracts/NFTPairWithOracle.sol#L200-L212

https://github.com/code-423n4/2022-04-abranft/blob/main/contracts/NFTPairWithOracle.sol#L288

The lender should only be able to seize the collateral if:

- the borrower didn't repay in time
- the collateral loses too much of its value

But, the lender is able to seize the collateral at any time by modifying the loan parameters.

Proof of Concept

The `updateLoanParams()` allows the lender to modify the parameters of an active loan in favor of the borrower. But, by setting the `ltvBPS` value to `0` they are able to seize the collateral.

If `ltvBPS` is `0` the following require statement in `removeCollateral()` will always be true:

https://github.com/code-423n4/2022-04-abranft/blob/main/contracts/NFTPairWithOracle.sol#L288

`rate * 0 / BPS < amount` is always `true`.

That allows the lender to seize the collateral although its value didn't decrease nor did the time to repay the loan come.

So the required steps are:

1. lend the funds to the borrower
2. call `updateLoanParams()` to set the `ltvBPS` value to `0`
3. call `removeCollateral()` to steal the collateral from the contract

Recommended Mitigation Steps

Don't allow `updateLoanParams()` to change the `ltvBPS` value.

cryptolyndon (AbraNFT) confirmed and commented: > Confirmed, and the first to report this particular exploit.

[H-05] Mistake while checking LTV to lender accepted LTV

Submitted by catchup, also found by WatchPug, gzeon, and hyh

It comments in the `\_lend()` function that lender accepted conditions must be at least as good as the borrower is asking for. The line which checks the accepted LTV (lender's LTV) against borrower asking LTV is: `params.ltvBPS >= accepted.ltvBPS`, This means lender should be offering a lower LTV, which must be the opposite way around. I think this may have the potential to strand the lender, if he enters a lower LTV. For example borrower asking LTV is 86%. However, lender enters his accepted LTV as 80%. lend() will execute with 86% LTV and punish the lender, whereas it should revert and acknowledge the lender that his bid is not good enough.

Proof of Concept

https://github.com/code-423n4/2022-04-abranft/blob/main/contracts/NFTPairWithOracle.sol#L316

Recommended Mitigation Steps

The condition should be changed as: `params.ltvBPS <= accepted.ltvBPS`,

cryptolyndon (AbraNFT) confirmed and commented: > Confirmed, and the first to note this particular issue.

Medium Risk Findings (1)

[M-01] Reentrancy at _requestLoan allows requesting a loan without supplying collateral

Submitted by kenzo, also found by WatchPug, GimelSec, Czar102, and AuditsAreUS

https://github.com/code-423n4/2022-04-abranft/blob/main/contracts/NFTPair.sol#L218

https://github.com/code-423n4/2022-04-abranft/blob/main/contracts/NFTPairWithOracle.sol#L238

`_requestLoan` makes an external call to the collateral contract before updating the NFTPair contract state.

Impact

If the ERC721 collateral has a afterTokenTransfer hook, The NFTPair contract can be reentered, and a loan can be requested without the borrower supplying the collateral. Someone can then lend for the loan while the collateral is missing from the contract. Therefore the malicious borrower received the loan without supplying collateral - so lender's funds can be stolen. The issue is present in both NFTPair and NFTPairWithOracle.

Proof of Concept

Assume the NFT contract has an afterTokenTransfer hook which calls back to the malicious borrower. POC in short: borrower will call `requestLoan` with `skim==false`, then during the hook will reenter `requestLoan` with `skim==true`, then call `removeCollateral` to get his collateral back, then the first `requestLoan` will resume and initiate the loan, although the collateral is not in the contract any more.

POC in long: the borrower will do the following:

- Call `requestLoan` with skim==false.
- `requestLoan` will call `collateral.transferFrom()`.

- The collateral will be transferred to the NFTPair. Afterwards, the ERC721 contract will execute the `afterTokenTransfer` hook, and hand control back to Malbo (malicious borrower).
- Malbo will call `requestLoan` again with `skim==true`.
- As the first request's details have not been updated yet, the tokenId status is still LOAN_INITIAL, and the require statement of the loan status will pass.
- The NFT has already been transfered to the contract, so the require statement of token ownership will pass.
- `requestLoan` (the second) will continue and set the loan details and status.
- After it finishes, still within the `afterTokenTransfer` hook, Malbo will call `removeCollateral`. His call will succeed as the loan is in status requested. So the collateral will get sent back to Malbo.
- Now the `afterTokenTransfer` hook finishes.
- The original `requestLoan` will resume operation at the point where all the loan details will be updated.
- Therefore, the contract will mark the loan is valid, although the collateral is not in the contract anymore. A lender might then lend funds against the loan without Malbo needing to pay back.

Recommended Mitigation Steps

Move the external call to the end of the function to conform with checks-effects-interaction pattern.

cryptolyndon (AbraNFT) disputed and commented: > Not a bug. We do not use `safeTransfer`, and if the collateral contract cannot be trusted, then all bets are off.

0xean (judge) downgraded to medium severity and commented: > I am downgrading this to medium severity and do believe it should be fixed by the sponsor. Re-entrancy has proved to be a problem in many ways in the space and while the sponsor says they are trusting the collateral contract, I dont think this is a defensible stance from what can be easily mitigated by either re-ordering code to conform to well established patterns or by adding a modifier. > > `> 2 - Med: 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 Risk and Non-Critical Issues

For this contest, 45 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **IllIllI** received the top score from the judge.

The following wardens also submitted reports: horsefacts, bobi, berndartmueller, reassor, joestakey, hyh, MaratCerby, defsec, pauliax, z3s, simon135, CertoraInc, delfin454000, 0xf15ers, BowTiedWardens, kenzo, Funen, antonttc, bobirichman, sikorico, samruna, catchup, oyc_109, m9800, gzeon, ilan, GimelSec, AuditsAreUS, broccolirob, cccz, kenta, unforgiven, robee, 0x1337, gs8nrv, jah, 0x1f8b, hubble, kebabsec, WatchPug, throttle, mics, Ruhum, and 0xDjango.

[L-01] Should ensure loan collateral is not immediately seizable

For the Oracle version there are checks to make sure that the current valuation is above the amount loaned. There should be a similar check that the loan duration is not zero. Zero is not useful for flash loans because of the origination fees.

```
File: contracts/NFTPairWithOracle.sol   #1

224          tokenLoanParams[tokenId] = params;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L224

```
File: contracts/NFTPairWithOracle.sol   #2

244          tokenLoanParams[tokenId] = params;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L244

[L-02] Pairs do not implement `ERC721TokenReceiver`

According to the `README.md`, `NFTPair`s specifically involve `ERC721` tokens. Therefore the contract should implement `ERC721TokenReceiver`, or customer transfers involving `safeTransferFrom()` calls will revert

```
File: contracts/NFTPair.sol   #1

59  contract NFTPair is BoringOwnable, Domain, IMasterContract {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L59

```
File: contracts/NFTPairWithOracle.sol   #2

69  contract NFTPairWithOracle is BoringOwnable, Domain, IMasterContract {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L69

[L-03] Incorrect comments

Skimming involves excess balances in the *bentobox*, not in the contract itself. This comment will lead to clients incorrectly passing tokens to the pair, rather than the bentobox. In addition, overall, there should be more comments devited to the interactions with the bentobox

```
File: contracts/NFTPair.sol   #1

320      /// @param skim True if the funds have been transfered to the contract
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L320

```
File: contracts/NFTPairWithOracle.sol   #2

355      /// @param skim True if the funds have been transfered to the contract
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L355

[L-04] Calls incorrectly allow non-zero `msg.value`

The comments below say that `msg.value` is "only applicable to" a subset of actions. All other actions should have a `require(!msg.value)`. Allowing it anyway is incorrect state handling

```
File: contracts/NFTPair.sol   #1

631      /// @param values A one-to-one mapped array to `actions`. ETH amounts to send along
632      /// Only applicable to `ACTION_CALL`, `ACTION_BENTO_DEPOSIT`.
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L631-L632

```
File: contracts/NFTPairWithOracle.sol   #2

664      /// @param values A one-to-one mapped array to `actions`. ETH amounts to send along
665      /// Only applicable to `ACTION_CALL`, `ACTION_BENTO_DEPOSIT`.
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L664-L665

[L-05] Missing checks for `address(0x0)` when assigning values to `address` state variables

```
File: contracts/NFTPair.sol   #1

729           feeTo = newFeeTo;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9 52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L729

```
File: contracts/NFTPairWithOracle.sol   #2

751           feeTo = newFeeTo;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9 52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L751

[L-06] `ecrecover` not checked for zero result

A return value of zero indicates an invalid signature, so this is both invalid state-handling and an incorrect message

```
File: contracts/NFTPair.sol   #1

383               require(ecrecover(_getDigest(dataHash), v, r, s) == lender, "NFTPair: signa
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9 52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L383

```
File: contracts/NFTPair.sol   #2

419           require(ecrecover(_getDigest(dataHash), v, r, s) == borrower, "NFTPair: signatu
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9 52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L419

```
File: contracts/NFTPairWithOracle.sol   #3

417               require(ecrecover(_getDigest(dataHash), signature.v, signature.r, signature
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9 52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L417

```
File: contracts/NFTPairWithOracle.sol   #4

452           require(ecrecover(_getDigest(dataHash), signature.v, signature.r, signature.s)
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9 52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L452

[N-01] Consider supporting CryptoPunks and EtherRocks

The project `README.md` says that `NFTPair`s are specifically ERC721 tokens, but not all NFTs are ERC721s. CryptoPunks and EtherRocks came before the standard and do not conform to it.

```
File: README.md   #1
```

```
58  - NFT Pair are a version of Cauldrons where the collateral isn't an ERC20 token but an 
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/README.md?plain=1#L58

[N-02] Contracts should be refactored to extend a base contract with the common functionality

```
$ fgrep -xf NFTPair.sol NFTPairWithOracle.sol | wc -l
686
$ wc -l NFTPair.sol NFTPairWithOracle.sol
732 NFTPair.sol
754 NFTPairWithOracle.sol
```

```
686 / 732 = 93.7%
686 / 754 = 91.0%
```

About 92% of the lines in each file are exactly the same as the lines in the other file. At the very least the shared constants, the common state variables, and the pure functions should be moved to a common base contract.

```
File: contracts/NFTPair.sol (various lines)   #1
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol

```
File: contracts/NFTPairWithOracle.sol (various lines)   #2
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol

[N-03] Some compilers can't handle two different contracts with the same name

Some compilers only compile the first one they encounter, ignoring the second one. If two contracts are different (e.g. different struct argument definitions) then they should have different names

```
File: contracts/NFTPair.sol   #1

37  interface ILendingClub {
38      // Per token settings.
39      function willLend(uint256 tokenId, TokenLoanParams memory params) external view ret
40
41      function lendingConditions(address nftPair, uint256 tokenId) external view returns 
42  }
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L37-L42

```
File: contracts/NFTPairWithOracle.sol   #2

47  interface ILendingClub {
48      // Per token settings.
49      function willLend(uint256 tokenId, TokenLoanParams memory params) external view retu
50
51      function lendingConditions(address nftPair, uint256 tokenId) external view returns (
52  }
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L47-L52

[N-04] Calls to `BoringMath.to128()` are redundant

All calls to `to128()` occur on the result of `calculateInterest()`, which itself already checks that the value fits into a `uint128`

```
File: contracts/NFTPairWithOracle.sol   #1

285                 ).to128();
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L285

```
File: contracts/NFTPairWithOracle.sol   #2

552         uint256 interest = calculateInterest(principal, uint64(block.timestamp - loan.s
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L552

```
File: contracts/NFTPair.sol   #3

519         uint256 interest = calculateInterest(principal, uint64(block.timestamp - loan.s
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L519

[N-05] `require()`/`revert()` statements should have descriptive reason strings

```
File: contracts/NFTPair.sol   #1

501             revert();
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L501

```
File: contracts/NFTPairWithOracle.sol   #2
```

```
534                 revert();
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L534

[N-06] `public` functions not called by the contract should be declared `external` instead

Contracts are allowed to override their parents' functions and change the visibility from `external` to `public`.

```
File: contracts/NFTPair.sol   #1

181       function updateLoanParams(uint256 tokenId, TokenLoanParams memory params) public
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L181

```
File: contracts/NFTPair.sol   #2

713       function withdrawFees() public {
714           address to = masterContract.feeTo();
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L713-L714

```
File: contracts/NFTPair.sol   #3

728       function setFeeTo(address newFeeTo) public onlyOwner {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L728

```
File: contracts/NFTPairWithOracle.sol   #4

198       function updateLoanParams(uint256 tokenId, TokenLoanParams memory params) public
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L198

```
File: contracts/NFTPairWithOracle.sol   #5

735       function withdrawFees() public {
736           address to = masterContract.feeTo();
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L735-L736

```
File: contracts/NFTPairWithOracle.sol   #6

750       function setFeeTo(address newFeeTo) public onlyOwner {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L750

[N-07] Interfaces should be moved to separate files

Most of the other interfaces in this project are in their own file in the interfaces directory. The interfaces below do not follow this pattern

```
File: contracts/NFTPairWithOracle.sol   #1

47  interface ILendingClub {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L47

```
File: contracts/NFTPairWithOracle.sol   #2

54  interface INFTPair {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L54

```
File: contracts/NFTPair.sol   #3

37  interface ILendingClub {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L37

```
File: contracts/NFTPair.sol   #4

44  interface INFTPair {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L44

[N-08] `2**<n> - 1` should be re-written as `type(uint<n>).max`

Earlier versions of solidity can use `uint<n>(-1)` instead. Expressions not including the `- 1` can often be re-written to accomodate the change (e.g. by using a `>` rather than a `>=`, which will also save some gas)

```
File: contracts/NFTPair.sol   #1

500           if (interest >= 2**128) {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L500

```
File: contracts/NFTPairWithOracle.sol   #2
```

```
533             if (interest >= 2**128) {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L533

[N-09] `constants` should be defined rather than using magic numbers

```
File: contracts/NFTPair.sol   #1

500             if (interest >= 2**128) {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L500

```
File: contracts/NFTPairWithOracle.sol   #2

533             if (interest >= 2**128) {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L533

[N-10] Numeric values having to do with time should use time units for readability

There are units for seconds, minutes, hours, days, and weeks

```
File: contracts/NFTPair.sol   #1

111       uint256 private constant YEAR_BPS = 3600 * 24 * 365 * 10_000;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L111

```
File: contracts/NFTPairWithOracle.sol   #2

128       uint256 private constant YEAR_BPS = 3600 * 24 * 365 * 10_000;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L128

[N-11] Use a more recent version of solidity

Use a solidity version of at least 0.8.4 to get `bytes.concat()` instead of `abi.encodePacked(<bytes>,<bytes>)` Use a solidity version of at least 0.8.12 to get `string.concat()` instead of `abi.encodePacked(<str>,<str>)`

```
File: contracts/NFTPair.sol   #1

20   pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L20

```
File: contracts/NFTPairWithOracle.sol   #2

20   pragma solidity 0.6.12;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L20

[N-12] Constant redefined elsewhere

Consider defining in only one contract so that values cannot become out of sync when only one location is updated. A cheap way to store constants in a single location is to create an `internal constant` in a `library`. If the variable is a local cache of another contract's value, consider making the cache variable internal or private, which will require external users to query the contract with the source of truth, so that callers don't get out of sync.

```
File: contracts/NFTPairWithOracle.sol   #1

93       IBentoBoxV1 public immutable bentoBox;
```

seen in contracts/NFTPair.sol https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L93

```
File: contracts/NFTPairWithOracle.sol   #2

94       NFTPairWithOracle public immutable masterContract;
```

seen in contracts/NFTPair.sol https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L94

[N-13] Fee mechanics should be better described

When reading through the code the first time, it wasn't clear exactly what `openFeeShare` was for and why it's being subtracted from `totalShare`. Add to this the fact that the `protocolFee` is based on the `openFeeShare` and it seems like this area needs more comments, specifically that `openFeeShare` is the fee paid to the lender by the borrower during loan initiation, for the privilege of being given a loan.

```
File: contracts/NFTPair.sol   #1

295          if (skim) {
296              require(
297                  bentoBox.balanceOf(asset, address(this)) >= (totalShare - openFeeShare
298                 "NFTPair: skim too much"
299             );
300         } else {
301             bentoBox.transfer(asset, lender, address(this), totalShare - openFeeShare -
302         }
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L295-L302

```
File: contracts/NFTPairWithOracle.sol   #2

330         if (skim) {
331             require(
332                 bentoBox.balanceOf(asset, address(this)) >= (totalShare - openFeeShare
333                 "NFTPair: skim too much"
334             );
335         } else {
336             bentoBox.transfer(asset, lender, address(this), totalShare - openFeeShare -
337         }
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L330-L337

[N-14] Typos

```
File: contracts/NFTPair.sol   #1

90       // Track assets we own. Used to allow skimming the excesss.
```

excesss https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c057
48e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L90

```
File: contracts/NFTPair.sol   #2

114       // `calculateIntest`.
```

calculateIntest https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc
3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L114

```
File: contracts/NFTPair.sol   #3

233       /// @param skim True if the token has already been transfered
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298
c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L233

```
File: contracts/NFTPair.sol   #4

320       /// @param skim True if the funds have been transfered to the contract
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L320

```
File: contracts/NFTPair.sol   #5

351       /// @param skimCollateral True if the collateral has already been transfered
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L351

```
File: contracts/NFTPair.sol   #6

389       /// @notice Take collateral from a pre-commited borrower and lend against it
```

commited https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L389

```
File: contracts/NFTPair.sol   #7

394       /// @param skimFunds True if the funds have been transfered to the contract
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L394

```
File: contracts/NFTPair.sol   #8

434       /// of the above inquality) fits in 128 bits, then the function is
```

inquality https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L434

```
File: contracts/NFTPair.sol   #9

446           // (NOTE: n is hardcoded as COMPOUND_INTEREST_TERMS)
```

hardcoded https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L446

```
File: contracts/NFTPairWithOracle.sol   #10

107       // Track assets we own. Used to allow skimming the excesss.
```

excesss https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L107

```
File: contracts/NFTPairWithOracle.sol   #11

131       // `calculateIntest`.
```

calculateIntest https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L131

```
File: contracts/NFTPairWithOracle.sol   #12

253       /// @param skim True if the token has already been transfered
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298
c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L253

```
File: contracts/NFTPairWithOracle.sol   #13

355       /// @param skim True if the funds have been transfered to the contract
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298
c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L355

```
File: contracts/NFTPairWithOracle.sol   #14

386       /// @param skimCollateral True if the collateral has already been transfered
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298
c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L386

```
File: contracts/NFTPairWithOracle.sol   #15

423       /// @notice Take collateral from a pre-commited borrower and lend against it
```

commited https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298
c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L423

```
File: contracts/NFTPairWithOracle.sol   #16

428       /// @param skimFunds True if the funds have been transfered to the contract
```

transfered https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298
c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L428

```
File: contracts/NFTPairWithOracle.sol   #17

467       /// of the above inquality) fits in 128 bits, then the function is
```

inquality https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c0
5748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L467

```
File: contracts/NFTPairWithOracle.sol   #18

479           // (NOTE: n is hardcoded as COMPOUND_INTEREST_TERMS)
```

hardcoded https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298
c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L479

[N-15] NatSpec is incomplete

```
File: contracts/NFTPair.sol   #1

346       /// @notice Caller provides collateral; loan can go to a different address.
347       /// @param tokenId ID of the token that will function as collateral
348       /// @param lender Lender, whose BentoBox balance the funds will come from
349       /// @param recipient Address to receive the loan.
350       /// @param params Loan parameters requested, and signed by the lender
351       /// @param skimCollateral True if the collateral has already been transfered
352       /// @param anyTokenId Set if lender agreed to any token. Must have tokenId 0 in si
353       function requestAndBorrow(
354           uint256 tokenId,
355           address lender,
356           address recipient,
357           TokenLoanParams memory params,
358           bool skimCollateral,
359           bool anyTokenId,
360           uint256 deadline,
361           uint8 v,
362           bytes32 r,
363           bytes32 s
```

Missing: `@param deadline` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L346-L363

```
File: contracts/NFTPair.sol   #2

346       /// @notice Caller provides collateral; loan can go to a different address.
347       /// @param tokenId ID of the token that will function as collateral
348       /// @param lender Lender, whose BentoBox balance the funds will come from
349       /// @param recipient Address to receive the loan.
350       /// @param params Loan parameters requested, and signed by the lender
351       /// @param skimCollateral True if the collateral has already been transfered
352       /// @param anyTokenId Set if lender agreed to any token. Must have tokenId 0 in si
353       function requestAndBorrow(
354           uint256 tokenId,
355           address lender,
356           address recipient,
357           TokenLoanParams memory params,
358           bool skimCollateral,
359           bool anyTokenId,
360           uint256 deadline,
361           uint8 v,
362           bytes32 r,
363           bytes32 s
```

Missing: `@param v` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L346-L363

```
File: contracts/NFTPair.sol   #3

346      /// @notice Caller provides collateral; loan can go to a different address.
347      /// @param tokenId ID of the token that will function as collateral
348      /// @param lender Lender, whose BentoBox balance the funds will come from
349      /// @param recipient Address to receive the loan.
350      /// @param params Loan parameters requested, and signed by the lender
351      /// @param skimCollateral True if the collateral has already been transfered
352      /// @param anyTokenId Set if lender agreed to any token. Must have tokenId 0 in si
353      function requestAndBorrow(
354          uint256 tokenId,
355          address lender,
356          address recipient,
357          TokenLoanParams memory params,
358          bool skimCollateral,
359          bool anyTokenId,
360          uint256 deadline,
361          uint8 v,
362          bytes32 r,
363          bytes32 s
```

Missing: `@param r` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L346-L363

```
File: contracts/NFTPair.sol   #4

346      /// @notice Caller provides collateral; loan can go to a different address.
347      /// @param tokenId ID of the token that will function as collateral
348      /// @param lender Lender, whose BentoBox balance the funds will come from
349      /// @param recipient Address to receive the loan.
350      /// @param params Loan parameters requested, and signed by the lender
351      /// @param skimCollateral True if the collateral has already been transfered
352      /// @param anyTokenId Set if lender agreed to any token. Must have tokenId 0 in si
353      function requestAndBorrow(
354          uint256 tokenId,
355          address lender,
356          address recipient,
357          TokenLoanParams memory params,
358          bool skimCollateral,
359          bool anyTokenId,
360          uint256 deadline,
361          uint8 v,
362          bytes32 r,
363          bytes32 s
```

Missing: `@param s` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L346-L363

```
File: contracts/NFTPair.sol   #5

389       /// @notice Take collateral from a pre-commited borrower and lend against it
390       /// @notice Collateral must come from the borrower, not a third party.
391       /// @param tokenId ID of the token that will function as collateral
392       /// @param borrower Address that provides collateral and receives the loan
393       /// @param params Loan terms offered, and signed by the borrower
394       /// @param skimFunds True if the funds have been transfered to the contract
395       function takeCollateralAndLend(
396           uint256 tokenId,
397           address borrower,
398           TokenLoanParams memory params,
399           bool skimFunds,
400           uint256 deadline,
401           uint8 v,
402           bytes32 r,
403           bytes32 s
```

Missing: `@param deadline` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L389-L403

```
File: contracts/NFTPair.sol   #6

389       /// @notice Take collateral from a pre-commited borrower and lend against it
390       /// @notice Collateral must come from the borrower, not a third party.
391       /// @param tokenId ID of the token that will function as collateral
392       /// @param borrower Address that provides collateral and receives the loan
393       /// @param params Loan terms offered, and signed by the borrower
394       /// @param skimFunds True if the funds have been transfered to the contract
395       function takeCollateralAndLend(
396           uint256 tokenId,
397           address borrower,
398           TokenLoanParams memory params,
399           bool skimFunds,
400           uint256 deadline,
401           uint8 v,
402           bytes32 r,
403           bytes32 s
```

Missing: `@param v` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L389-L403

```
File: contracts/NFTPair.sol   #7
```

```
389       /// @notice Take collateral from a pre-commited borrower and lend against it
390       /// @notice Collateral must come from the borrower, not a third party.
391       /// @param tokenId ID of the token that will function as collateral
392       /// @param borrower Address that provides collateral and receives the loan
393       /// @param params Loan terms offered, and signed by the borrower
394       /// @param skimFunds True if the funds have been transfered to the contract
395       function takeCollateralAndLend(
396           uint256 tokenId,
397           address borrower,
398           TokenLoanParams memory params,
399           bool skimFunds,
400           uint256 deadline,
401           uint8 v,
402           bytes32 r,
403           bytes32 s
```

Missing: `@param r` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L389-L403

```
File: contracts/NFTPair.sol   #8
```

```
389       /// @notice Take collateral from a pre-commited borrower and lend against it
390       /// @notice Collateral must come from the borrower, not a third party.
391       /// @param tokenId ID of the token that will function as collateral
392       /// @param borrower Address that provides collateral and receives the loan
393       /// @param params Loan terms offered, and signed by the borrower
394       /// @param skimFunds True if the funds have been transfered to the contract
395       function takeCollateralAndLend(
396           uint256 tokenId,
397           address borrower,
398           TokenLoanParams memory params,
399           bool skimFunds,
400           uint256 deadline,
401           uint8 v,
402           bytes32 r,
403           bytes32 s
```

Missing: `@param s` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L389-L403

```
File: contracts/NFTPairWithOracle.sol   #9
```

```
381       /// @notice Caller provides collateral; loan can go to a different address.
382       /// @param tokenId ID of the token that will function as collateral
383       /// @param lender Lender, whose BentoBox balance the funds will come from
384       /// @param recipient Address to receive the loan.
385       /// @param params Loan parameters requested, and signed by the lender
386       /// @param skimCollateral True if the collateral has already been transfered
```

```
387       /// @param anyTokenId Set if lender agreed to any token. Must have tokenId 0 in si
388       function requestAndBorrow(
389           uint256 tokenId,
390           address lender,
391           address recipient,
392           TokenLoanParams memory params,
393           bool skimCollateral,
394           bool anyTokenId,
395           SignatureParams memory signature
```

Missing: `@param signature` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L381-L395

```
File: contracts/NFTPairWithOracle.sol   #10

423       /// @notice Take collateral from a pre-commited borrower and lend against it
424       /// @notice Collateral must come from the borrower, not a third party.
425       /// @param tokenId ID of the token that will function as collateral
426       /// @param borrower Address that provides collateral and receives the loan
427       /// @param params Loan terms offered, and signed by the borrower
428       /// @param skimFunds True if the funds have been transfered to the contract
429       function takeCollateralAndLend(
430           uint256 tokenId,
431           address borrower,
432           TokenLoanParams memory params,
433           bool skimFunds,
434           SignatureParams memory signature
```

Missing: `@param signature` https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L423-L434

[N-16] Event is missing `indexed` fields

Each `event` should use three `indexed` fields if there are three or more fields

```
File: contracts/NFTPair.sol   #1

65       event LogRequestLoan(address indexed borrower, uint256 indexed tokenId, uint128 val
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L65

```
File: contracts/NFTPair.sol   #2

66       event LogUpdateLoanParams(uint256 indexed tokenId, uint128 valuation, uint64 durati
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L66

```
File: contracts/NFTPair.sol   #3

68       event LogRemoveCollateral(uint256 indexed tokenId, address recipient);
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L68

```
File: contracts/NFTPair.sol   #4

73       event LogWithdrawFees(address indexed feeTo, uint256 feeShare);
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L73

```
File: contracts/NFTPairWithOracle.sol   #5

75       event LogRequestLoan(
76           address indexed borrower,
77           uint256 indexed tokenId,
78           uint128 valuation,
79           uint64 duration,
80           uint16 annualInterestBPS,
81           uint16 ltvBPS
82       );
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L75-L82

```
File: contracts/NFTPairWithOracle.sol   #6

83       event LogUpdateLoanParams(uint256 indexed tokenId, uint128 valuation, uint64 durati
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L83

```
File: contracts/NFTPairWithOracle.sol   #7

85       event LogRemoveCollateral(uint256 indexed tokenId, address recipient);
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L85

```
File: contracts/NFTPairWithOracle.sol   #8

90       event LogWithdrawFees(address indexed feeTo, uint256 feeShare);
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L90

[N-17] A best practice is to check for signature malleability

```
File: contracts/NFTPair.sol   #1

383              require(ecrecover(_getDigest(dataHash), v, r, s) == lender, "NFTPair: signa
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L383

```
File: contracts/NFTPair.sol   #2

419          require(ecrecover(_getDigest(dataHash), v, r, s) == borrower, "NFTPair: signatu
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L419

```
File: contracts/NFTPairWithOracle.sol   #3

417              require(ecrecover(_getDigest(dataHash), signature.v, signature.r, signature
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L417

```
File: contracts/NFTPairWithOracle.sol   #4

452          require(ecrecover(_getDigest(dataHash), signature.v, signature.r, signature.s)
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L452

[N-18] Consider making contract `Pausable` to have some protection against ongoing exploits

```
File: contracts/NFTPair.sol   #1

59  contract NFTPair is BoringOwnable, Domain, IMasterContract {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L59

```
File: contracts/NFTPairWithOracle.sol   #2

69  contract NFTPairWithOracle is BoringOwnable, Domain, IMasterContract {
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L69

[N-19] States/flags should use `Enums` rather than separate constants

```
File: contracts/NFTPair.sol   #1

96      uint8 private constant LOAN_INITIAL = 0;
97      uint8 private constant LOAN_REQUESTED = 1;
98      uint8 private constant LOAN_OUTSTANDING = 2;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L96-L98

```
File: contracts/NFTPairWithOracle.sol   #2

113      uint8 private constant LOAN_INITIAL = 0;
114      uint8 private constant LOAN_REQUESTED = 1;
115      uint8 private constant LOAN_OUTSTANDING = 2;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L113-L115

```
File: contracts/NFTPair.sol   #3

545      uint8 internal constant ACTION_REPAY = 2;
546      uint8 internal constant ACTION_REMOVE_COLLATERAL = 4;
547
548      uint8 internal constant ACTION_REQUEST_LOAN = 12;
549      uint8 internal constant ACTION_LEND = 13;
550
551      // Function on BentoBox
552      uint8 internal constant ACTION_BENTO_DEPOSIT = 20;
553      uint8 internal constant ACTION_BENTO_WITHDRAW = 21;
554      uint8 internal constant ACTION_BENTO_TRANSFER = 22;
555      uint8 internal constant ACTION_BENTO_TRANSFER_MULTIPLE = 23;
556      uint8 internal constant ACTION_BENTO_SETAPPROVAL = 24;
557
558      // Any external call (except to BentoBox)
559      uint8 internal constant ACTION_CALL = 30;
560
561      // Signed requests
562      uint8 internal constant ACTION_REQUEST_AND_BORROW = 40;
563      uint8 internal constant ACTION_TAKE_COLLATERAL_AND_LEND = 41;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e952f8a8c93e42f930a78c2/contracts/NFTPair.sol#L545-L563

```
File: contracts/NFTPairWithOracle.sol   #4

578      uint8 internal constant ACTION_REPAY = 2;
```

```
579      uint8 internal constant ACTION_REMOVE_COLLATERAL = 4;
580
581      uint8 internal constant ACTION_REQUEST_LOAN = 12;
582      uint8 internal constant ACTION_LEND = 13;
583
584      // Function on BentoBox
585      uint8 internal constant ACTION_BENTO_DEPOSIT = 20;
586      uint8 internal constant ACTION_BENTO_WITHDRAW = 21;
587      uint8 internal constant ACTION_BENTO_TRANSFER = 22;
588      uint8 internal constant ACTION_BENTO_TRANSFER_MULTIPLE = 23;
589      uint8 internal constant ACTION_BENTO_SETAPPROVAL = 24;
590
591      // Any external call (except to BentoBox)
592      uint8 internal constant ACTION_CALL = 30;
593
594      // Signed requests
595      uint8 internal constant ACTION_REQUEST_AND_BORROW = 40;
596      uint8 internal constant ACTION_TAKE_COLLATERAL_AND_LEND = 41;
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L578-L596

[N-20] Non-exploitable re-entrancies

Code should follow the best-practice of check-effects-interaction

See original submission for details.

[N-21] Comments should be enforced by `require()`s

The comments below should be enforced by `require(block.timestamp < uint64(-1))`

```
File: contracts/NFTPair.sol   #1

311          loan.startTime = uint64(block.timestamp); // Do not use in 12e10 years..
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPair.sol#L311

```
File: contracts/NFTPairWithOracle.sol   #2

346          loan.startTime = uint64(block.timestamp); // Do not use in 12e10 years..
```

https://github.com/code-423n4/2022-04-abranft/blob/5cd4edc3298c05748e9
52f8a8c93e42f930a78c2/contracts/NFTPairWithOracle.sol#L346

cryptolyndon (AbraNFT) commented: > **Low-risk issues:** > > [**L-03**] Agreed; this does suggest ERC-20 transfers. > > [**L-04**] Simply requiring msg.value to be zero would break things when some, but not all, actions use it. > > [**L-05**] The zero address is pretty much the ONLY wrong address we could enter where actual loss of funds is not possible.

> **Non-critical issues:**
>
> [**N-01**] Nonstandard NFT types that are popular enough to use warrant their own contract type.
>
> [**N-03**] This is not some example project intended to be forked and used with a wide range of different compiler setups.
>
> [**N-11**] As time ticks on 0.8.x becomes increasingly safe to use, but the suggested reason here does not even apply to our contract.
>
> [**N-12**] `bentoBox` is not a constant that will necessarily be invariable across different master contracts. Clones already work as suggested.
>
> [**N-13**] The contract is not meant to serve as sole documentation of our fee schedule.
>
> [**N-17**] We use nonces to prevent replay attacks, rather than storing used signatures. A different, equally valid, signature of the same data would be of no use to an attacker.
>
> [**N-21**] If you think this is going to be an issue, then think of all the gas wasted until then by even that single check! Time enough to write a V2.

0xean (judge) commented: > I believe this to be the most complete QA report.

Gas Optimizations

For this contest, 33 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **BowTiedWardens** received the top score from the judge.

The following wardens also submitted reports: joestakey, horsefacts, IllIllI, 0xkatana, robee, defsec, oyc_109, reassor, Tomio, z3s, slywaters, catchup, 0xNazgul, delfin454000, Tadashi, NoamYakov, simon135, gzeon, Funen, sorrynotsorry, CertoraInc, pauliax, 0xf15ers, antonttc, kenta, TrungOre, Kulk0, fatherOfBlocks, 0x1f8b, samruna, GimelSec, and Hawkeye.

Table of Contents

See original submission.

[G-01] `NFTPair.init` and `NFTPairWithOracle.init`: Storage usage optimization

I suggest replacing:

```
175:     function init(bytes calldata data) public payable override {
176:         require(address(collateral) == address(0), "NFTPair: already initialized");
177:         (collateral, asset) = abi.decode(data, (IERC721, IERC20));
178:         require(address(collateral) != address(0), "NFTPair: bad pair"); //@audit coul
179:     }
```

with:

```
function init(bytes calldata data) public payable override {
    require(address(collateral) == address(0), "NFTPair: already initialized");
    (address _collateral, address _asset) = abi.decode(data, (IERC721, IERC20));
    require(address(_collateral) != address(0), "NFTPair: bad pair");
    (collateral, asset) = (_collateral, _asset);
}
```

Here, we're saving 1 SLOAD at the cost of 2 MSTOREs and 3 MLOADs => around 85 gas. Furthermore, in case of revert, a lot more gas would be refunded, as the 2 SSTORE operations are done after the `require` statements.

[G-02] Caching storage values in memory

The code can be optimized by minimising the number of SLOADs. SLOADs are expensive (100 gas) compared to MLOADs/MSTOREs (3 gas). Here, storage values should get cached in memory (see the `@audit` tags for further details):

```
contracts/NFTPair.sol:
  290:         uint256 totalShare = bentoBox.toShare(asset, params.valuation, false); //@au
  297:                 bentoBox.balanceOf(asset, address(this)) >= (totalShare - openFeeSha
  301:             bentoBox.transfer(asset, lender, address(this), totalShare - openFeeShar
  305:         bentoBox.transfer(asset, address(this), loan.borrower, borrowerShare);  //@a
  523:         uint256 totalShare = bentoBox.toShare(asset, amount, false); //@audit gas: a
  524:         uint256 feeShare = bentoBox.toShare(asset, fee, false); //@audit gas: asset
  528:             require(bentoBox.balanceOf(asset, address(this)) >= (totalShare + feesEa
  532:             bentoBox.transfer(asset, msg.sender, address(this), feeShare); //@audit
  539:         bentoBox.transfer(asset, from, loan.lender, totalShare - feeShare); //@audit

contracts/NFTPairWithOracle.sol:
  325:         uint256 totalShare = bentoBox.toShare(asset, params.valuation, false); //@au
  332:                 bentoBox.balanceOf(asset, address(this)) >= (totalShare - openFeeSha
  336:             bentoBox.transfer(asset, lender, address(this), totalShare - openFeeShar
  340:         bentoBox.transfer(asset, address(this), loan.borrower, borrowerShare); //@au
  556:         uint256 totalShare = bentoBox.toShare(asset, amount, false); //@audit gas: a
  557:         uint256 feeShare = bentoBox.toShare(asset, fee, false); //@audit gas: asset
  561:            require(bentoBox.balanceOf(asset, address(this)) >= (totalShare + feesEar
  565:            bentoBox.transfer(asset, msg.sender, address(this), feeShare); //@audit g
  572:        bentoBox.transfer(asset, from, loan.lender, totalShare - feeShare); //@audit
```

[G-03] Splitting `require()` statements that use `&&` saves gas

Instead of using the `&&` operator in a single require statement to check multiple conditions, I suggest using multiple require statements with 1 condition per require statement (saving 3 gas per `&`):

```
contracts/NFTPair.sol:
  622:        require(callee != address(bentoBox) && callee != address(collateral) && calle

contracts/NFTPairWithOracle.sol:
  655:        require(callee != address(bentoBox) && callee != address(collateral) && calle
```

[G-04] An array's length should be cached to save gas in for-loops

Reading array length at each iteration of the loop takes 6 gas (3 for mload and 3 to place memory_offset) in the stack.

Caching the array length in the stack saves around 3 gas per iteration.

Here, I suggest storing the array's length in a variable before the for-loop, and use it instead:

```
NFTPair.sol:641:        for (uint256 i = 0; i < actions.length; i++) {
NFTPairWithOracle.sol:674:        for (uint256 i = 0; i < actions.length; i++) {
```

[G-05] `++i` costs less gas compared to `i++` or `i += 1`

`++i` costs less gas compared to `i++` or `i += 1` for unsigned integer, as pre-increment is cheaper (about 5 gas per iteration). This statement is true even with the optimizer enabled.

`i++` increments `i` and returns the initial value of `i`. Which means:

```
uint i = 1;
i++; // == 1 but i == 2
```

But `++i` returns the actual incremented value:

```
uint i = 1;
++i; // == 2 and i == 2 too, so no need for a temporary variable
```

In the first case, the compiler has to create a temporary variable (when used) for returning `1` instead of `2`

Instances include:

```
NFTPair.sol:494:        for (uint256 k = 2; k <= COMPOUND_INTEREST_TERMS; k++) {
NFTPair.sol:641:        for (uint256 i = 0; i < actions.length; i++) {
NFTPairWithOracle.sol:527:        for (uint256 k = 2; k <= COMPOUND_INTEREST_TERMS; k++) {
NFTPairWithOracle.sol:674:        for (uint256 i = 0; i < actions.length; i++) {
```

I suggest using `++i` instead of `i++` to increment the value of an uint variable.

[G-06] Public functions to external

The following functions could be set external to save gas and improve code quality.

```
    NFTPair.init(bytes) (contracts/NFTPair.sol#175-179)
    NFTPairWithOracle.init(bytes) (contracts/NFTPairWithOracle.sol#192-196)
    NFTPair.updateLoanParams(uint256,TokenLoanParams) (contracts/NFTPair.sol#181-203)
    NFTPair.withdrawFees() (contracts/NFTPair.sol#713-723)
    NFTPair.setFeeTo(address) (contracts/NFTPair.sol#728-731)
    NFTPairWithOracle.updateLoanParams(uint256,TokenLoanParams) (contracts/NFTPairWithOracle
    NFTPairWithOracle.withdrawFees() (contracts/NFTPairWithOracle.sol#735-745)
    NFTPairWithOracle.setFeeTo(address) (contracts/NFTPairWithOracle.sol#750-753)
```

[G-07] `updateLoanParams()`: Replace `memory` with `calldata` and `public` with `external`

This is valid in both files `NFTPair.sol` and `NFTPairWithOracle.sol`. As mentioned above, `updateLoanParams()` should be external. Furthermore, `TokenLoanParams memory params` should be `TokenLoanParams calldata params`. Therefore, we'd go from:

```
function updateLoanParams(uint256 tokenId, TokenLoanParams memory params) public
```

to

```
function updateLoanParams(uint256 tokenId, TokenLoanParams calldata params) external
```

[G-08] `updateLoanParams()`: Copying in memory can be more expensive than using the `storage` keyword

This is valid in both files `NFTPair.sol` and `NFTPairWithOracle.sol`. In this particular case here, I suggest using the `storage` keyword instead of the `memory` one, as the copy in memory is wasting some MSTOREs and MLOADs. See the `@audit` tags for more details:

```
    function updateLoanParams(uint256 tokenId, TokenLoanParams memory params) public {
        TokenLoan memory loan = tokenLoan[tokenId]; //@audit gas: use the storage keyword i
        if (loan.status == LOAN_OUTSTANDING) {
            // The lender can change terms so long as the changes are strictly
            // the same or better for the borrower:
            require(msg.sender == loan.lender, "NFTPair: not the lender");
            TokenLoanParams memory cur = tokenLoanParams[tokenId];  //@audit gas: copying ir
            require(
                params.duration >= cur.duration &&
                    params.valuation <= cur.valuation &&
                    params.annualInterestBPS <= cur.annualInterestBPS &&
                    params.ltvBPS <= cur.ltvBPS,
                "NFTPair: worse params"
            );
        } else if (loan.status == LOAN_REQUESTED) {
            // The borrower has already deposited the collateral and can
            // change whatever they like
            require(msg.sender == loan.borrower, "NFTPair: not the borrower");
        } else {
          (...)
```

I suggest:

- Using `TokenLoan storage loan = tokenLoan[tokenId];`
- Only caching `loan.status` in memory as it can be evaluated twice (in the if/else statement)
- Using `TokenLoanParams storage cur = tokenLoanParams[tokenId];`

[G-09] `_lend()`: Copying in memory can be more expensive than using the `storage` keyword

In this function, I suggest replacing `TokenLoan memory loan = tokenLoan[tokenId];` with `TokenLoan storage loan = tokenLoan[tokenId];`. Only 2 SLOADs are made (`loan.status` and `loan.borrower`) and the function is writing in memory (`loan` variable) before writing in storage. These steps are superfluous and there's no value from a copy in memory here.

[G-10] No need to explicitly initialize variables with default values

If a variable is not set/initialized, it is assumed to have the default value (`0` for `uint`, `false` for `bool`, `address(0)` for address…). Explicitly initializing it with its default value is an anti-pattern and wastes gas.

As an example: `for (uint256 i = 0; i < numIterations; ++i) {` should be replaced with `for (uint256 i; i < numIterations; ++i) {`

Instances include:

```
NFTPair.sol:96:    uint8 private constant LOAN_INITIAL = 0;
NFTPair.sol:641:        for (uint256 i = 0; i < actions.length; i++) {
NFTPairWithOracle.sol:113:    uint8 private constant LOAN_INITIAL = 0;
NFTPairWithOracle.sol:674:        for (uint256 i = 0; i < actions.length; i++) {
```

I suggest removing explicit initializations for default values.

[G-11] Reduce the size of error messages (Long revert Strings)

Shortening revert strings to fit in 32 bytes will decrease deployment time gas and will decrease runtime gas when the revert condition is met.

Revert strings that are longer than 32 bytes require at least one additional mstore, along with additional overhead for computing memory offset, etc.

Revert strings > 32 bytes:

```
NFTPair.sol:366:            require(ILendingClub(lender).willLend(tokenId, params), "NFTPair
NFTPairWithOracle.sol:398:            require(ILendingClub(lender).willLend(tokenId, params)
```

I suggest shortening the revert strings to fit in 32 bytes, or using custom errors as described next.

cryptolyndon (AbraNFT) commented: > Good report, thank you. Detailed, specific to the actual contract / project, more in depth than the usual drive-by checklists.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.