Nibbl contest
Findings & Analysis Report

2022-08-01

Table of contents

- Overview
  - About C4
  - Wardens
- Summary
- Scope
- Severity Criteria
- Medium Risk Findings (12)
  - [M-01] Buyout cannot be rejected when paused
  - [M-02] `Twav.sol#_getTwav()` will revert when timestamp > 4294967296
  - [M-03] User Could Change The State Of The System While In `Pause` Mode
  - [M-04] Ineffective TWAV Implementation
  - [M-05] Lack of sanity check on _initialTokenSupply and _initialTokenPrice can lead to a seller losing his NFT
  - [M-06] NibblVault: In the buy function, users can avoid paying fees
  - [M-07] `_updateTwav()` and `_getTwav()` will revert when cumulativePrice overflows

- [M-08] [PNM-004] Calculation of `_secondaryReserveRatio` can be overflowed
- [M-09] NibblVault buyout duration longer than update timelock
- [M-10] Reentrancy bug in Basket's withdraw multiple tokens function which gives attacker ability to transfer basket ownership and spend it but withdraw all the tokens out of basket
- [M-11] Twav.*getTwav() will return a wrong result when twavObservations[TWAV*BLOCK_NUMBERS - 1].timestamp = 0.
- [M-12] Basket NFT have no name and symbol

- Low Risk and Non-Critical Issues

  - Low Risk Issues
  - L-01 Buyouts that occur during the timestamp wrap will have valuation errors
  - L-02 `ecrecover()` not checked for signer address of zero
  - L-03 Return values of `transfer()` / `transferFrom()` not checked
  - L-04 Input array lengths may differ
  - L-05 `_safeMint()` should be used rather than `_mint()` wherever possible
  - L-06 Missing checks for `address(0x0)` when assigning values to `address` state variables
  - L-07 Vulnerable to cross-chain replay attacks due to static `DOMAIN_SEPARATOR` / `domainSeparator`
  - L-08 Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions
  - L-09 Incorrect comments
  - Non-Critical Issues
  - N-01 Consider addings checks for signature malleability
  - N-02 Misleading variable name
  - N-03 Inconsistent version of English being used
  - N-04 Missing `initializer` modifier on constructor

- N-05 Contract implements interface without extending the interface
- N-06 `require()` / `revert()` statements should have descriptive reason strings
- N-07 `public` functions not called by the contract should be declared `external` instead
- N-08 Non-assembly method available
- N-09 `2**<n> - 1` should be re-written as `type(uint<n>).max`
- N-10 `constant` s should be defined rather than using magic numbers
- N-11 Cast is more restrictive than the type of the variable being assigned
- N-12 Missing event and or timelock for critical parameter change
- N-13 Expressions for constant values such as a call to `keccak256()` , should use `immutable` rather than `constant`
- N-14 Inconsistent spacing in comments
- N-15 Lines are too long
- N-16 Non-library/interface files should use fixed compiler versions, not floating ones
- N-17 Typos
- N-18 File is missing NatSpec
- N-19 NatSpec is incomplete
- N-20 Event is missing `indexed` fields

- Gas Optimizations

  - 1 Setting `DEFAULT_ADMIN_ROLE` as the role admin is redundant
  - 2 Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas
  - 3 Using `storage` instead of `memory` for structs/arrays saves gas
  - 4 State variables should be cached in stack variables rather than re-reading them from storage
  - 5 `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables
  - 6 `internal` functions only called once can be inlined to save gas

- 7 Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if` -statement
- 8 `<array>.length` should not be looked up in every loop of a `for` -loop
- 9 `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops
- 10 `require()` / `revert()` strings longer than 32 bytes cost extra gas
- 11 Using `bool` s for storage incurs overhead
- 12 Use a more recent version of solidity
- 13 `>=` costs less gas than `>`
- 14 It costs more gas to initialize non- `constant` /non- `immutable` variables to zero than to let the default of zero be applied
- 15 `++i` costs less gas than `i++` , especially when it's used in `for` -loops (`--i` / `i--` too)
- 16 Splitting `require()` statements that use `&&` saves gas
- 17 Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead
- 18 Using `private` rather than `public` for constants, saves gas
- 19 Don't use `SafeMath` once the solidity version is 0.8.0 or greater
- 20 Duplicated `require()` / `revert()` checks should be refactored to a modifier or function
- 21 Empty blocks should be removed or emit something
- 22 Use custom errors rather than `revert()` / `require()` strings to save gas
- 23 Functions guaranteed to revert when called by normal users can be marked `payable`

- Disclosures

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 Nibbl smart contract system written in Solidity. The audit contest took place between June 21—June 24 2022.

Wardens

107 Wardens contributed reports to the Nibbl contest:

1. xiaoming90
2. hansfriese
3. reassor
4. unforgiven
5. itsmeSTYJ
6. WatchPug (jtp and ming)
7. Lambda
8. cccz
9. Picodes
10. SmartSek (0xDjango and hake)
11. ych18
12. hyh
13. PwnedNoMore (izhuer, ItsNio and papr1ka2)
14. peritoflores
15. IllIllI
16. zzzitron

17. BowTiedWardens (BowTiedHeron, BowTiedPickle, m4rio_eth, Dravee and BowTiedFirefox)
18. kenzo
19. joestakey
20. 0x1f8b
21. _Adam
22. 0x29A (0x4non and rotcivegaf)
23. codexploder
24. 0xkatana
25. Chom
26. defsec
27. MiloTruck
28. c3phas
29. catchup
30. ellahi
31. minhquanym
32. 0xNazgul
33. robee
34. UnusualTurtle
35. cryptphi
36. TomJ
37. 0xf15ers (remora and twojoy)
38. sashik_eth
39. pashov
40. delfin454000
41. StErMi
42. Tomio
43. oyc_109
44. slywaters

45. ElKu
46. kenta
47. saian
48. JC
49. sach1r0
50. Noah3o6
51. simon135
52. TerrierLover
53. rfa
54. Waze
55. fatherOfBlocks
56. zuhaibmohd
57. Randyyy
58. kebabsec (okkothejawa and FlameHorizon)
59. Nyamcil
60. Funen
61. exd0tpy
62. Limbooo
63. m_Rassska
64. berndartmueller
65. 0xNineDec
66. cloudjunky
67. 0xKitsune
68. shenwilly
69. sorrynotsorry
70. JohnSmith
71. Tadashi
72. sseefried
73. MadWookie

74. JMukesh
75. asutorufos
76. dipp
77. naps62
78. apostle0x01
79. Wayne
80. Alex the Entreprenerd
81. Varun_Verma
82. 0xc0ffEE
83. Treasure-Seeker
84. masterchief
85. 0x52
86. Nethermind
87. RoiEvenHaim
88. 8olidity
89. ajtra
90. ACai
91. cRat1st0s
92. Chandr
93. 0v3rf10w
94. ynnad
95. IgnacioB
96. Fitraldys

This contest was judged by HardlyDifficult.

Final report assembled by itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 12 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 12 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 82 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 63 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 Nibbl contest repository](), and is composed of 8 smart contracts written in the Solidity programming language and includes 539 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]().

Medium Risk Findings (12)

[M-01] Buyout cannot be rejected when paused

Submitted by Lambda, also found by SmartSek

https://github.com/code-423n4/2022-06-nibbl/blob/71b639f977c0351c9928dd3b78eaa4bebb738bc1/contracts/NibblVault.sol#L300

https://github.com/code-423n4/2022-06-nibbl/blob/71b639f977c0351c9928dd3b78eaa4bebb738bc1/contracts/NibblVault.sol#L362

https://github.com/code-423n4/2022-06-nibbl/blob/71b639f977c0351c9928dd3b78eaa4bebb738bc1/contracts/NibblVault.sol#L464

https://github.com/code-423n4/2022-06-nibbl/blob/71b639f977c0351c9928dd3b78eaa4bebb738bc1/contracts/NibblVault.sol#L495

Impact

While `buy()` and `sell()` are only callable when the system is not paused, `redeem()` and `withdrawERC721()` are also callable when it is not. This means that the `BUYOUT_DURATION` is ignored in such cases and it is possible that users are not able to reject certain buyouts.

Proof of Concept

A user initiates a buyout via `initiateBuyout()`. Just afterwards, the system is stopped. The token holders now cannot buy new tokens to increase the value. However, after two days, the `bidder` can still withdraw the NFT, i.e. there was no way for the users to reject this buyout.

Recommended Mitigation Steps

It should be possible to reset the `buyoutEndTime` (to the current `block.timestamp`) when the system is paused such that the token holders always have the possibility to reject a buyout.

[mundhrakeshav (Nibbl) disputed and commented]():

> Expected. When paused no operations should be available.

fatherGoose1 (warden) commented:

> Strongly disagree with the sponsor's comment. Given that redeem() and withdrawERC721() DO NOT contain the `whenNotPaused` modifier, this ensures that pauses that occur during a buyout process will ensure the success of the buyout. The buyout success occurs by time passing a certain block.timestamp and the functionality to claim the NFT and retrieve the underlying are left open even during the pause.

> Similar to issue #261

> I would agree with the sponsor if all of the `withdraw()/redeem()` functions contained the `whenNotPaused` modifier so that truly all functions were locked during a pause.

mundhrakeshav (Nibbl) commented:

> Hmmm. Makes sense. We should pause redeem and Withdraw too.

HardlyDifficult (judge) commented:

> The readme does include "Out of scope: Admin can pause and change certain parameters of the contract." however this report is not strictly about the ability to pause.

> It should be possible to reset the buyoutEndTime

> In this scenario, an end time has already been defined. If `pause` is used at that time the window shortens or closes so when resumed the opportunity may have been missed already. The warden's recommendation here, or some variation of it, would provide a way to effectively allow the system to resume from where it left off when originally paused.

> I suspect the alternative of also pausing redeem / withdraw is not sufficient, as the window to buy/sell will still potentially be passed by the time the system resumes.

> I agree with the submitted Med risk for this issue since the “function of the protocol or its availability could be impacted”.

[M-02] `Twav.sol#_getTwav()` will revert when timestamp > 4294967296

Submitted by WatchPug, also found by hansfriese and IllIllI

```
    function _getTwav() internal view returns(uint256 _twav){
        if (twavObservations[TWAV_BLOCK_NUMBERS - 1].timestamp != 0)
            uint8 _index = ((twavObservationsIndex + TWAV_BLOCK_NUME
            TwavObservation memory _twavObservationCurrent = twavObs
            TwavObservation memory _twavObservationPrev = twavObserv
            _twav = (_twavObservationCurrent.cumulativeValuation - _
        }
    }
```

Since `_blockTimestamp` is `uint32`, subtraction underflow is desired at `_twavObservationCurrent.timestamp - _twavObservationPrev.timestamp`.

See: https://github.com/Uniswap/v2-periphery/blob/master/contracts/examples/ExampleOracleSimple.sol#L43

```
    function update() external {
        (uint price0Cumulative, uint price1Cumulative, uint32 blockT
            UniswapV2OracleLibrary.currentCumulativePrices(address(p
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; //
```

Because the solidity version used by the current implementation is `0.8.10`, and there are some breaking changes in Solidity v0.8.0:

> Arithmetic operations revert on underflow and overflow.

Ref: https://docs.soliditylang.org/en/v0.8.13/080-breaking-changes.html#silent-changes-of-the-semantics

The timestamp subtraction may revert due to underflow.

Impact

Since `_getTwav()` is used in `NibblVault.sol#_rejectBuyout()`, if it reverts and there is a `buyout`, an essential feature of the `NibblVault` contract will be unavailable, causing users' funds to be frozen in the contract.

Recommendation

Change to:

```
    function _updateTWAV(uint256 _valuation, uint32 _blockTimestamp)
        uint32 _timeElapsed;
        unchecked {
            _timeElapsed = _blockTimestamp - lastBlockTimeStamp;
        }

        uint256 _prevCumulativeValuation = twavObservations[((twavOb
        unchecked {
            twavObservations[twavObservationsIndex] = TwavObservatic
        }

        twavObservationsIndex = (twavObservationsIndex + 1) % TWAV_E
        lastBlockTimeStamp = _blockTimestamp;
    }



    function _getTwav() internal view returns(uint256 _twav){
        if (twavObservations[TWAV_BLOCK_NUMBERS - 1].timestamp != 0)
            uint8 _index = ((twavObservationsIndex + TWAV_BLOCK_NUME
            TwavObservation memory _twavObservationCurrent = twavObs
            TwavObservation memory _twavObservationPrev = twavObserv
            unchecked {
                _twav = (_twavObservationCurrent.cumulativeValuatior
            }
        }
    }
```

mundhrakeshav (Nibbl) acknowledged, but disagreed with severity

KenzoAgada (warden) commented:

> If I'm not mistaken, timestamp 4294967296 is 2106, I wouldn't call the contract breaking in 84 years a high severity issue. Plus the contract is truncating the timestamp on purpose. Seems to me more like a design choice and less of a bug.

[mingwatch (warden) commented]:

> If I'm not mistaken, timestamp 4294967296 is 2106, I wouldn't call the contract breaking in 84 years a high severity issue. Plus the contract is truncating the timestamp on purpose. Seems to me more like a design choice and less of a bug.

> [https://github.com/code-423n4/2022-06-nibbl/blob/main/contracts/Twav/Twav.sol#L23-L25]

```
    function _updateTWAV(uint256 _valuation, uint32 _blockTimestamp)
            uint32 _timeElapsed;
            unchecked {
                _timeElapsed = _blockTimestamp - lastBlockTimeStamp;
            }
```

> According to the above code, is obviously not a design choice.

[KenzoAgada (warden) commented]:

> According to the above code, is obviously not a design choice.

> Ah, I think I understand what you mean, it is not handled consistently.

[HardlyDifficult (judge) decreased severity to Medium and commented]:

> it is not handled consistently

> Thanks for the clarifications!

> `_getTwav` would overflow once timestamps overflow uint32, but only when the current observation has overflowed while the previous observation did not.

> The window for this vulnerability is very small, just at the time timestamp starts to overflow in 2106 - vaults active before or after that time should work as expected.

> This appears to be a Medium risk finding. There's potentially a case to be made for high here but it's hard to make that call without a more complete POC included. The vault is an upgradeable contract so they have 84 years to sort this out — but it does seem like an issue that should be fixed.

[M-03] User Could Change The State Of The System While In `Pause` Mode

Submitted by xiaoming90

Calling `NibblVault.updateTWAP` function will change the state of the system. It will cause the TWAP to be updated and buyout to be rejected in certain condition.

When the system is in `Pause` mode, the system state should be frozen. However, it was possible for someone to call the `NibblVault.updateTWAP` function during the `Pause` mode, thus making changes to the system state.

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L443

```
    /// @notice Updates the TWAV when in buyout
    /// @dev TWAV can be updated only in buyout state
    function updateTWAV() external override {
        require(status == Status.buyout, "NibblVault: Status!=Buyout
        uint32 _blockTimestamp = uint32(block.timestamp % 2**32);
        if (_blockTimestamp != lastBlockTimeStamp) {
            _updateTWAV(getCurrentValuation(), _blockTimestamp);
            _rejectBuyout(); //For the case when TWAV goes up when u
        }
    }
```

Recommended Mitigation Steps

Ensure that the `NibblVault.updateVault` function cannot be called when the system is in `Pause` mode.

Add the `whenNotPaused` modifier to the function.

```
    /// @notice Updates the TWAV when in buyout
    /// @dev TWAV can be updated only in buyout state
    function updateTWAV() external override whenNotPaused {
        require(status == Status.buyout, "NibblVault: Status!=Buyout
        uint32 _blockTimestamp = uint32(block.timestamp % 2**32);
        if (_blockTimestamp != lastBlockTimeStamp) {
            _updateTWAV(getCurrentValuation(), _blockTimestamp);
            _rejectBuyout(); //For the case when TWAV goes up when ι
        }
    }
```

mundhrakeshav (Nibbl) marked as duplicate and commented:

> Duplicate of #56

HardlyDifficult (judge) commented:

> ## 56

> It's not clear to me how this is a dupe of #56

> This is a valid concern and potentially a change worth making.

> It will cause the TWAP to be updated and buyout to be rejected

> This makes me think Medium risk is correct here. In this scenario a buyout could be rejected without allowing other users to challenge that — seemingly breaking one of the benefits behind using Twap for this logic.

[M-04] Ineffective TWAV Implementation

Submitted by xiaoming90, also found by hyh

The current TWAV implementation consists of an array of 4 observations/valuations called `twavObservations`. Whenever, the new valuation is updated, the new cumulative valuation will be appended to the `twavObservations` array and the oldest observation/valuation will be removed from the `twavObservations` array.

Description of current TWAV implementation can be found at
https://github.com/NibblNFT/nibbl-smartcontracts#twavsol

- Time-weighted average valuation
- Uses an array of length 4 which stores cumulative valuation and timestamp.
- TWAV is calculated between the most and least recent observations recorded in the array.
- TWAV array is updated only when the system is in buyout state. In case of buyout rejection, the array is reset.

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L11

```
/// @notice current index of twavObservations index
uint8 public twavObservationsIndex;
uint8 private constant TWAV_BLOCK_NUMBERS = 4; //TWAV of last 4
uint32 public lastBlockTimeStamp;

/// @notice record of TWAV
TwavObservation[TWAV_BLOCK_NUMBERS] public twavObservations;

/// @notice updates twavObservations array
/// @param _blockTimestamp timestamp of the block
/// @param _valuation current valuation
function _updateTWAV(uint256 _valuation, uint32 _blockTimestamp)
    uint32 _timeElapsed;
    unchecked {
        _timeElapsed = _blockTimestamp - lastBlockTimeStamp;
    }

    uint256 _prevCumulativeValuation = twavObservations[((twavOb
    twavObservations[twavObservationsIndex] = TwavObservation(_b
    twavObservationsIndex = (twavObservationsIndex + 1) % TWAV_B
    lastBlockTimeStamp = _blockTimestamp;
}
```

Within the `NibblVault` contract, the `_updateTWAV` function will be called whenever the following events happen during the buyout period:

1. `NibbleVault.buy()` and `NibbleVault.Sell()` functions are called
2. `NibbleVault.initiateBuyout` function is called
3. `NibbleVault.updateTWAV` function is called

Per the code and comment of `_getTwav()` function, the function will return the TWAV of the last four (4) blocks. This function can be called by anyone.

```
/// @notice returns the TWAV of the last 4 blocks
/// @return _twav TWAV of the last 4 blocks
function _getTwav() internal view returns(uint256 _twav){
    if (twavObservations[TWAV_BLOCK_NUMBERS - 1].timestamp != 0)
        uint8 _index = ((twavObservationsIndex + TWAV_BLOCK_NUME
        TwavObservation memory _twavObservationCurrent = twavObs
        TwavObservation memory _twavObservationPrev = twavObserv
        _twav = (_twavObservationCurrent.cumulativeValuation - _
    }
}
```

Time weighted average valuation (TWAV) is supposed to be the average value of a security over a specified time (e.g. 15 minutes, 1 hour, 24 hours). However, based on the above implementation of the `_getTwav` function, it is not the average value of a security over a specific time.

A user could call the `updateTWAV` function to add the new valuation/observation to the `twavObservations` array each time a new Ethereum block is mined. As such, the current implementation becomes the average value of a security over a specific number of observations (in this case 4 observations), thus it can be considered as Observation weighted average valuation (OWAV).

There is a fundamental difference between TWAV and OWAV.

Proof of Concept

In Ethereum, the average block time is around 15 seconds, so the time to take to mine 4 blocks will be 1 minute. As such, in term of TWAV, the current implementation only have a period of 1 minute, which is too short to prevent price manipulation.

The following shows an example where it is possible to buy tokens→ increase the valuation above the rejection valuation→ reject the buyout→ dump the tokens within 1 minute:

Assume that a buyer has triggered a buyout on the vault/NFT, and the buyout rejection price is 120 ETH and the current valuation is 100 ETH. Further assume that all elements in the `twavObservations` array have already been populated.

Note: Fees are ignored to illustrate the issue.

1. Block 100 at Time 0 - Attacker called `buy` function to increase the current valuation to 120 ETH attempting to reject the buyout.
2. Block 101 at Time 15 - Attacker called `updateTWAV` function. The current valuation (120 ETH) will be replaced the first element in `twavObservations` array.
3. Block 102 at Time 30 - Attacker called `updateTWAV` function. The current valuation (120 ETH) will be replaced the second element in `twavObservations` array.
4. Block 103 at Time 45 - Attacker called `updateTWAV` function. The current valuation (120 ETH) will be replaced the third element in `twavObservations` array.
5. Block 104 at Time 60 - Attacker called `sell` function to sell/dump all his shares. Within the `sell` function, `_updateTWAV` will be first called, thus the current valuation (120 ETH) will be replaced the fourth element in `twavObservations` array. Then, the `_rejectBuyout()` will be called, and the `_getTwav` function will be triggered. At this point, the TWAV valuation is finally 120 ETH, thus the buyout is rejected. Subseqently, attacker's shares are burned, and attacker get back his funds.

Since attacker could perform the above attack within 1 minute, it is very unlikely that the attackers will lose money to arbitrageurs as it takes some time for the arbitrageurs to notice such an opportunity.

Attacker could also front-run or set a higher gas fee to ensure that their transaction get mined in the next block to minimize the attack window period.

Impact

Buyout can be easily rejected by attackers

Recommended Mitigation Steps

Implement a proper TWAV that provides the average value of a security over a specified time. The time period/windows of the TWAV must be explicitly defined (e.g. 15 minutes, 1 hour, 24 hours) in the contract.

There are trade offs when choosing the length of the period of time to calculate a TWAP. Longer periods are better to protect against price manipulation, but come at the expense of a slower, and potentially less accurate, price. Thus, the team should determine the optimal period.

Consider referencing the popular Uniswap V2 TWAP design (https://docs.uniswap.org/protocol/V2/concepts/core-concepts/oracles)

mundhrakeshav (Nibbl) disagreed with severity

sseefried (warden) commented:

> I had a Low Risk (#142) associated with `_getTWAV` too. I'm not sure it even averages over 4 observations.

Picodes (warden) commented:

> Some comments, as I personally don't think it's an high security issue:
>
> - the primary goal of the TWAV here is to avoid manipulations within the same block. As long as it's over multiple blocks, the attackers takes a risk as there could be arbitrages, so the attack risk is mitigated.
> - the main assumption of the issue is: "it takes some time for the arbitrageurs to notice such an opportunity, and the 4 block window is too short." which seems false when you check on chain data: arbitrageurs are frequently super quick to react as it's their primary job: the first to check an opportunity takes it.

HardlyDifficult (judge) decreased severity to Medium and commented:

> Great summary @Picodes , I agree with both points.

> Lowering this to a Medium risk. I may be incorrect, but it seems a secondary goal of Twap is price smoothing to avoid scenarios like what was outlined here. If that’s correct then this impacts the function of the protocol and the recommendation is a good consideration.

dmitriia (warden) commented:

> Same here, TWAP is essential to the protocol, while the ability to manipulate the price during last minute breaks the core logic of price discovery by greatly reducing the number of participants. The rationale that 4 blocks are enough and arbitrage is generally quick is sufficient for mainstream cases only, all other trading is at risk of direct manipulation, which is existential risk for the protocol. Can’t see why the team pressing for medium here.

HardlyDifficult (judge) commented:

> Same here, TWAP is essential to the protocol, while the ability to manipulate the price during last minute breaks the core logic of price discovery by greatly reducing the number of participants. The rationale that 4 blocks are enough and arbitrage is generally quick is sufficient for mainstream cases only, all other trading is at risk of direct manipulation, which is existential risk for the protocol. Can’t see why the team pressing for medium here.

> The stated goal of using Twap in their documentation is the same as above, to prevent same-block attacks. It seems the concern is the implicit behavior expected from using a “time weighted” variable. Personally I agree this seems like an area they may want to revisit. However the system behaves correctly and there is a tiny window for bots to respond.

> @mundhrakeshav @Picodes would you mind elaborating here as well?

mundhrakeshav (Nibbl) commented:

> Yeah, makes sense. We do plan to increase the array length.

Picodes (warden) commented:

> I fully agree with both of you: it’d be indeed better to increase the array length to increase the robustness, but I still feel this is a medium issue as the system works as intended

dmitriia (warden) commented:

> The system formally/technically works as intended, but being a price discovery engine, where participants having incentives to try to determine the NFT price, it will not be used if the window for such a discovery is mere 4 blocks. Formally the bots will have space to react. Realistically it will happen in the most mainstream cases only, when price discovery isn't much needed. I.e. exactly when the system can bring in something new, adding a mechanics for 'slow' usual users to add price information, it will not work as the set of participants who can add the information to the metric (react to move the price) is shrunk by a tiny reaction window. Who will take part in a price discovery game knowing that last minute bots are always can surface? Quite not everyone. This reduces the traction, and so the amount of information brought in by the system, as this is the users who bring in the additional signal, and ultimately it will be a piece of well known NFTs synched with main market with bots, adding no value to it. I.e. the system will work technically, but economically it will not make much sense, so will not be widely used by market actors and can end up provide little value to broad market. This is what I mean by existential risk, which is, of course high.

> I just feel that here and in general in C4 economic issues are tend to be underestimated, while having one usually tend to be a game changing aspect for a project.

IllIllI000 (warden) commented:

> @HardlyDifficult If the sponsor had used the term OWAV rather than TWAV, would this still be a medium-severity issue? It seems as though they knew the behavior they wanted (at least four user interactions where the average price is above the threshold) and just used the wrong term to describe it. I didn't file this issue because it seemed that way. The screenshot in this issue shows that they're interested in interactions, not duration of time https://github.com/code-423n4/2022-06-nibbl-findings/issues/144. It's possible they confirmed the issue because they weren't aware that comment vs code consistency issues are usually categorized as low risk

HardlyDifficult (judge) commented:

> Great points @dmitriia and @IllIllI000 ! Both are compelling. This is certainly a grey area.

> Given how significantly this impacts how users would potentially view and interact with the system, I'm inclined to leave this a Medium risk instead of downgrading to Low, falling under "the function of the protocol or its availability could be impacted".

> And since this was intentional design and there is a window for bots to respond, I don't feel that High risk is justified.

> I'm happy to continue here or on Discord, and would love more input if others want to chime in on the severity here. I don't feel strongly but do think that Medium is the best fit here.

[M-05] Lack of sanity check on _initialTokenSupply and _initialTokenPrice can lead to a seller losing his NFT

Submitted by itsmeSTYJ

There is no sanity check to ensure that `(primaryReserveRatio * _initialTokenSupply * _initialTokenPrice)` is ≥ `(SCALE * 1e18)`. As a result, `_primaryReserveBalance` is given a value of 0 since divisions in solidity are rounded down. This also means that `primaryReserveBalance` and `fictitiousPrimaryReserveBalance` have a value of 0.

When this happens, the `buy` function doesn't work because `_chargeFee` will revert on line 222. You can't `sell` either since `totalSupply == initialSupply`. The only way to recover the NFT is for the owner of the NFT to call `initiateBuyout` but there is always the possibility that someone else also spotted this mistake and will attempt to also call `initiateBuyout` once the `minBuyoutTime` is reached. If the owner loses this gas war, the owner has effectively lost his NFT.

Recommended Mitigation Steps

Add some sanity checks to ensure a sane expected value for `_initialTokenSupply` and `_initialTokenPrice`. There were multiple instances when a user tried to interact with a contract but entered a wrong value because they are not aware they needed to include decimals. A recent example of this is https://cointelegraph.com/news/1-million-rock-nft-sells-for-a-penny-in-all-ore-nothing-error.

Note: Normally, I would categorise issues like this as medium severity since it is a loss predicated on having met certain conditions but because there is also a lack of sanity check on `_minBuyoutTime`, it is entirely possible for a seller to lose his NFT immediately once the vault is created. There are many monsters waiting in the dark forest, all it takes is one mistake. That said, I will defer the final judgement to the judges & sponsors.

mundhrakeshav (Nibbl) disputed, disagreed with severity and commented:

> Case when that would happen is _initialTokenSupply or _initialTokenPrice is 0. But then it would revert here https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L183

HardlyDifficult (judge) commented:

> Case when that would happen is _initialTokenSupply or _initialTokenPrice is 0.

> primaryReserveRatio = 200*000 and SCALE = 1000*000 — so it seems this applies anytime _initialTokenSupply * _initialTokenPrice < 1000*000 * 1e18 / 200*000, not just when one of those values is 0. Please correct me if I got that wrong.

HardlyDifficult (judge) decreased severity to Medium and commented:

> This is a good report. I agree this seems like something that should be addressed. Assuming my math is right, if the initialTokenPrice was 1 wei then the initialTokenSupply must be >= 5e18. They are using the default of 18 decimals (which is also industry standard) so that's 5 tokens. Given this is not very large window and these values impact the curve — without a clear POC High would not be warranted, downgrading to Medium risk.

[M-06] NibblVault: In the buy function, users can avoid paying fees

Submitted by cccz, also found by kenzo, Lambda, WatchPug, xiaoming90, and zzzitron

In the buy function of the NibblVault contract, when msg.value > _lowerCurveDiff, the fee for SecondaryCurve part is not charged.

```
        if (_lowerCurveDiff >= msg.value) {
            _purchaseReturn = _buySecondaryCurve(msg.value,
        } else {
            //Gas Optimization
            _purchaseReturn = _initialTokenSupply - _totalSu
            secondaryReserveBalance += _lowerCurveDiff;
            // _purchaseReturn = _buySecondaryCurve(_to, _lc
            _purchaseReturn += _buyPrimaryCurve(msg.value -
        }
```

Proof of Concept

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L314-L323

Recommended Mitigation Steps

The fee for the SecondaryCurve part is complex to charge. I implemented the _caculateFeeSecondaryCurve and _reverseFeeSecondaryCurve functions to do the relevant calculations.

The _caculateFeeSecondaryCurve function is used to calculate the value after the fee is charged, but not to actually charge the fee. The _reverseFeeSecondaryCurve function is used to calculate the value before the fee is charged.

```
        if (_lowerCurveDiff >= _caculateFeeSecondaryCurve(ms
            _purchaseReturn = _buySecondaryCurve(msg.value,
        } else {
            //Gas Optimization
            _purchaseReturn = _initialTokenSupply - _totalSu
            secondaryReserveBalance += _lowerCurveDiff;
            uint256 _amount = _reverseFeeSecondaryCurve(_low
            _chargeFeeSecondaryCurve(_amount);
            // _purchaseReturn = _buySecondaryCurve(_to, _lc
            _purchaseReturn += _buyPrimaryCurve(msg.value -
        }
```

```
    function _caculateFeeSecondaryCurve(uint256 _amount) private
       address payable _factory = factory;
        uint256 _adminFeeAmt = NibblVaultFactory(_factory).feeAc
        uint256 _feeAdmin = (_amount * _adminFeeAmt) / SCALE ;
        uint256 _feeCurator = (_amount * curatorFee) / SCALE ;
        return _amount - (_feeAdmin + _feeCurator);
    }
    function _reverseFeeSecondaryCurve(uint256 _amount) private
       address payable _factory = factory;
        uint256 _adminFeeAmt = NibblVaultFactory(_factory).feeAc
        return _amount * SCALE / (SCALE - (_adminFeeAmt + curato
    }
```

mundhrakeshav (Nibbl) acknowledged, but disagreed with severity

HardlyDifficult (judge) commented:

> Failing to collect fees is a form of leaking value for the curator and therefore falls into the Medium risk definition — keeping the severity as reported.

[M-07] `_updateTwav()` and `_getTwav()` will revert when cumulativePrice overflows

Submitted by peritoflores, also found by WatchPug

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L28

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L40

Impact

Contract will break when `cumulativeValuation` overflows.

Proof of Concept

Cumulative prices are designed to work with overflows/underflows because in the end the difference is important.

In `_updateTwav()` when `_prevCumulativeValuation + (_valuation *_timeElapsed)` overflows the contract will not work anymore.

```
    twavObservations[twavObservationsIndex] = TwavObservation(_block
```

Same problem in `_getTwav()`

```
      _twav = (_twavObservationCurrent.cumulativeValuation - _twavOb
    }
```

Similar issues

https://github.com/code-423n4/2022-04-phuture-findings/issues/62

Recommended Mitigation

Add unchecked keyword in every line you add / subtract cumulative prices.

mundhrakeshav (Nibbl) acknowledged

HardlyDifficult (judge) decreased severity to Medium and commented:

> Without a better POC of the issue occurring it's hard to justify this is High risk. e.g. maybe it could be forced by spamming `updateTWAV`, but it's not clear if that would require extremely large values or an unrealistic number of transactions.

> Related to https://github.com/code-423n4/2022-06-nibbl-findings/issues/178, that one includes unchecking the price in the recommendation but the rest of the description focuses on timestamp overflows while this one looks at price overflows.

[M-08] [PNM-004] Calculation of `_secondaryReserveRatio` can be overflowed

Submitted by PwnedNoMore, also found by ych18

```
uint32 _secondaryReserveRatio = uint32((msg.value * SCALE * 1e18) /
(_initialTokenSupply * _initialTokenPrice));
```

`_secondaryReserveRatio` can be overflowed by setting a relatively small `_initialTokenSupply` and `_initialTokenPrice`. The result will be truncated by `uint32`, causing an overflow.

This overflow can bypass all the checks in function `initialize`. Any following functionality will be impacted since the `_secondaryReserveRatio` is incorrect.

Proof of Concept / Attack Scenario

- The user provide `_initialTokenSupply` and `_initialTokenPrice`, which meets `SCALE * 1e18 == _initialTokenSupply * _initialTokenPrice`
- The `msg.value` is set as `2 ** 32 + X`, where `MIN_SECONDARY_RESERVE_RATIO <= X <= primaryReserveRatio`. Note that `msg.value` is in Wei, so the deposited fund is not huge.

Suggested Fix

Add overflow checks.

mundhrakeshav (Nibbl) acknowledged

HardlyDifficult (judge) commented:

> OpenZeppelin has safe cast helpers that could be leveraged here.

> It is concerning that due to the truncation here, the configuration would not work how the user expects given the input parameters. And at this point the NFT has been escrowed into the vault. Because of this it seems Medium risk is a fair assessment.

[M-09] NibblVault buyout duration longer than update timelock

Submitted by reassor

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/NibblVaultFactoryData.sol#L6

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L158-L169

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L37

Impact

User can buy out NFT by initiating the process through `initiateBuyout`, then he has to wait `BUYOUT_DURATION` which is 5 days and if the buyout will not get rejected he can claim the NFT. During that period bidder cannot cancel the process. The issue is that since `NibblVault` is used through proxy it is possible to change its implementation through administrative functionality in `NibblVaultFactory` and the timelock for update'ing implementation is only 2 days.

Attack Scenario:

1. Bidder initiates buyout through `initiateBuyout`
2. Administrator of the protocol updates the `vaultImplementation` through `proposeNewVaultImplementation`
3. Bidder really does not like new implementation but cannot cancel buyout process
4. Administrator waits 2 days (the `UPDATE_TIME`) uses `updateVaultImplementation` and changes the implementation
5. Bidder loses funds/fait in the protocol

Proof of Concept

- https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/NibblVaultFactoryData.sol#L6
- https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L158-L169
- https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L37

Tools Used

Manual Review / VSCode

Recommended Mitigation Steps

It is recommended to either implement functionality for bidder to cancel the bid or increase/decrease the `UPDATE_TIME` / `BUYOUT_DURATION` so the invariant `BUYOUT_DURATION < UPDATE_TIME` holds.

mundhrakeshav (Nibbl) acknowledged, but disagreed with severity

HardlyDifficult (judge) decreased severity to Medium and commented:

> It seems the bidder could be left in a bad state, and updating the thresholds here may be a nice way to maintain expectations. Since this scenario is based on the admin making an undesirable change, this is a Medium risk report.

[M-10] Reentrancy bug in Basket’s withdraw multiple tokens function which gives attacker ability to transfer basket ownership and spend it but withdraw all the tokens out of basket

Submitted by unforgiven

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L41-L47

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L68-L75

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L91-L97

Impact

`Basket` is used for keep multiple tokens in contract and mint one `NFT` token to represent their ownership. `Basket` only allows for owner of `NFT(id=0)` to withdraw tokens from `Basket` address. users can deposit multiple tokens in one `Basket` and then create a `NibbVault` based on that `Basket NFT`. but due to reentrancy vulnerability in `Basket` it's possible to call the multiple-token-withdraw functions (`withdrawMultipleERC721()`, `withdrawMultipleERC1155()`, `withdrawMultipleERC721()` and `withdrawMultipleERC20()`) and in the middle their external calls, spend `Basket NFT` (transfer ownership of `id=0` to other contract, for example `createVault()`) and receive some fund from other, then in the rest of the multiple-token-withdraw function withdraw all the basket tokens. `Basket` shouldn't allow transferring ownership of `id=0` in the middle of multiple token withdraws.

Proof of Concept

This is `withdrawMultipleERC721()` code:

```
    function withdrawMultipleERC721(address[] memory _tokens, ui
        require(_isApprovedOrOwner(msg.sender, 0), "withdraw:not
        for (uint256 i = 0; i < _tokens.length; i++) {
            IERC721(_tokens[i]).safeTransferFrom(address(this),
            emit WithdrawERC721(_tokens[i], _tokenId[i], _to);
        }
    }
```

As you can see, contract only checks the ownership of `id=0` in the beginning of the function to see that user allowed to perform this action or not. then it iterates through user specified addresses and call `safeTransferFrom()` function in those address by user specified values. the bug is that in the middle of the external calls attacker can spend `Basket NFT id=0` (give ownership of that basket to other contracts and receive fund from them, for example attacker can call `createVault` in `NibblVaultFactory` and create a vault and call other contracts to invest in that vault) then in the rest of the iterations in `withdrawMultipleERC721()` attacker can withdraw `Basket` tokens. so even so the ownership of the `Basket` has been transferred and attacker received funds for it, attacker withdraw `Basket` tokens too.

This is the steps attacker would perform:

1. Create a `Basket` with well known `NFT` token list. let's assume the `Basket` name is `Basket_M`
2. Give approve permission to `NibblVaultFactory` for `Basket_M id=0` token.
3. Call `Basket_M.withdrawMultipleERC721(address[] memory _tokens, uint256[] memory _tokenId, address _to)` with list of all the tokens in basket to withdraw all of them, but the first address in the `_tokens` list is the address that attacker controls.
4. `Basket_M` would check that attacker is owner of the basket (owner of the `id = 0`) and in first iteration of the `for` it would call attacker controlled address which is a contract that attacker wrote its code.
5. Attacker contract would call `NibblVaultFactory.createVault()` with `Basket_M` address and `id=0` to create a vault which then transfer the ownership of `Basket_M id=0` to the vault address. let's assume it's `Vault_M`.
6. Attacker contract would buy some fraction of `Vault_M` by calling `buy()` function.
7. Let's assume there are other contracts(call it `Invest_Contract`) that would want to buy fraction of the well known `NFT`s in the basket and `Invest_Contract` invest some fund in vault having those `NFT` in vault's address or vault's basket just by calling `Invest_Contract`. attacker contract would call `Invest_Contract` to invest in `Vault_M` and `Invest_Contract`

would check that well known `NFT` is in `Basket_M id=0` which belongs to `Vault_M` to it would invest money on it by calling `initiateBuyout()`

8. Attacker contract then withdraw his money from `Vault_M` .
9. The rest of `Basket_M.withdrawMultipleERC721()` for iterations performs and all the `NFT` tokens of the `Basket_M` would be send to attacker and `Basket_M` would have nothing.

Steps 5 to 8 can be other things, the point is in those steps attacker would spent `Basket_M` and receive some fund from other contract while those other contracts checks that they are owner of the `Basket_M` which has well known `NFT` tokens, but in fact attacker withdraw those well known `NFT` tokens from `Basket_M` after spending it in the rest of the `withdrawMultipleERC721()` iterations. (those above step 5-8 is just a sample case)

So `Basket` shouldn't allow ownership transfer in the middle of the `Basket_M.withdrawMultipleERC721()` and similar multiple-token-withdraw functions or it should check the ownership in every iteration.

Tools Used

VIM

Recommended Mitigation Steps

Check ownership of `id=0` in every iteration or don't allow ownership transfer in the multiple-token-transfer functions.

mundhrakeshav (Nibbl) disputed

Alex the Entreprenerd (warden) commented:

> Contract checks if you own it as owner of Basket has bought it, and as such is entitled to underlying tokens.

KenzoAgada (warden) commented:

> The attack is contingent on a regular user, creating a smart contract, which allows anybody to call it, which checks that a parameter-supplied Nibbl vault contains a

> Nibbl basket which contains a specific NFT, and then proceeds to buyout/buy shares of that vault.

> Honestly it seems like the vector of attack is possible but quite far fetched.

[HardlyDifficult (judge) decreased severity to Medium and commented]:

> Creative thinking, this is what I'm here for!

> If I'm following the flow correctly.. you kick off a bulk withdraw from the basket, the first NFT in the list is a malicious contract which then creates a vault for that basket (so the contract needs to be the basket owner, which is okay). Now the vault is fully created for that basket which still has valuable NFTs in it but you're mid-tx. Your malicious contract pings other contracts which can be prompted to ape in via on-chain logic — their logic confirms all looks well and buys. But then control returns to the original batch withdrawal and the basket is drained.

> Honestly it seems like the vector of attack is possible but quite far fetched.

> I think I'd agree. To put it in terms of risk, this is not High: "valid attack path that does not have hand-wavy hypotheticals" — this sounds a bit hand-wavy. Namely because it assumes `Invest_Contract` allows any address to trigger a purchase using other users funds which seems risky. And the victim here is `Invest_Contract`, not regular users of the protocol. Lowering to Medium. Great stuff though.

[Alex the Entreprenerd (warden) commented]:

> The finding in essence is claiming that you can setup an empty `Basket` and sell it to external contracts, and those contracts would lose funds.

> If that were the case the vulnerability would be in the "sniping / buying" contracts and not in the Basket nor the Vault.

> The only thing the warden has shown is that they can create a Basket with a malicious token and through that they can call the Factory to create a Vault which after the tx will be empty.

> This is logically equivalent to selling an empty vault, or selling a vault of `BryptoPunks` (typo on purpose, it's a scam token).

[HardlyDifficult (judge) commented]:

> @Alex the Entreprenerd - I agree in terms of normal usage. The key here is a 3rd party contract uses on-chain logic in order to authorize a purchase. If that were the case, while in the middle of the attack as described all checks that contract may perform would confirm assets were included and terms look good — it would not be able to determine that the basket was in the middle of a batch withdraw request. Let me know if I'm overlooking something.

> The basket itself does not need to hold a malicious token — the withdraw request takes an array of addresses, so the malicious contract only need to appear there.

[Alex the Entreprenerd (warden) commented]:

> If that were the case, while in the middle of the attack as described all checks that contract may perform would confirm assets were included and terms look good — it would not be able to determine that the basket was in the middle of a batch withdraw request. Let me know if I'm overlooking something.

> The 3rd party contract would need to check that the Basket is properly set via `ownerOf(Basket) == Vault` (where Vault is an address contained in the list of `nibbledVaults` from factory).

> That would allow to determine if the contract is properly setup.

> The basket itself does not need to hold a malicious token,

> That is correct as you can setup any contract to accept the `safeTransferFrom` call.

> My statement is that an automated 3rd party contract can get rekt, but that's not the contract under audit.

[HardlyDifficult (judge) commented]:

> Agree that it's the 3rd party contract that suffers a loss.

> An `ownerOf(Basket) == Vault && ownerOf(NFT) == Basket` check is insufficient here because if it's in the middle of this scenario then the owner checks will appear legit but by the end of the tx they won't be. That's the part that I'm still hung up on. Part of the Medium definition is "the function of the protocol or its availability could be impacted" — although not an explicit goal, is it not implicit that protocols can be built upon with other contracts. The concern here seems to limit that ability, one could not build a contract that decided to participate based on on-chain state alone w/ or w/o an allow list of NFTs — it would require a trusted actor to allow list specific vaults or to perform the action itself.

> This is certainly grey though. Very hypothetical, e.g. it's not clear that a 3rd party contract would ever be interested in a capability like this. This is an interesting discussion! I'll sleep on it, but please chime in if you have more to add - I appreciate the feedback.

[M-11] Twav.*getTwav() will return a wrong result when twavObservations[TWAV*BLOCK_NUMBERS - 1].timestamp = 0.

Submitted by hansfriese

The "if" condition of Twav._getTwav() is missing some edge cases.

In this case, this function will return 0 which is different from the correct value and it will affect the main functions like NibblVault.buy() and NibblVault.sell().

Proof of Concept

I think this condition is to confirm at least 4 values were saved for twav calculation.

Btw this timestamp would be zero even though there are more than 4 values properly as it's modularized by 2**32.

In this case, the if condition will be false and this function will return 0.

Tools Used

Solidity Visual Developer of VSCode

Recommended Mitigation Steps

I see "cumulativeValuation" is increasing all the time and recommend replacing "timestamp" with "cumulativeValuation".

```
    if (twavObservations[TWAV_BLOCK_NUMBERS - 1].cumulativeValuatior
```

mundhrakeshav (Nibbl) confirmed and resolved

HardlyDifficult (judge) commented:

> Interesting catch. This is related to #178 but presents a distinct issue.

[M-12] Basket NFT have no name and symbol

Submitted by Picodes, also found by cccz

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L13

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L6

Impact

The `Basket` contract is intended to be used behind a proxy. But the `ERC721` implementation used is not upgradeable, and its constructor is called at deployment time on the implementation. So all proxies will have a void name and symbol, breaking all potential integrations and listings.

Proof of Concept

`ERC721("NFT Basket", "NFTB")` is called at deployment time, and sets private variable at the implementation level. Therefore when loading the code during `delegateCall`, these variables will not be initialized.

Recommended Mitigation Steps

The easiest mitigation would be to pass this variable as immutable so they are hardcoded in the implementation byte code.

mundhrakeshav (Nibbl) confirmed

Alex the Entreprenerd (warden) commented:

> Finding is valid, impact is the name of the tokens.

HardlyDifficult (judge) decreased severity to QA and commented:

> Confirmed this is an issue.

> Assets are not at risk, and the function of the protocol is not impacted. All baskets created will have an empty name/symbol but generally there is no requirement that these values are populated. It's mostly for a better experience on frontends including etherscan. Downgrading and merging with the warden's QA report #314.

Picodes (warden) commented:

> @HardlyDifficult Indeed it does not break the protocol's logic and funds are not at risk, but the name and the symbol of the NFTs are not the ones chosen by the sponsor, and as it's the core of EIP721Metadata we could argue that the function of the protocol are impacted. Also the experience on frontends (etherscan, opensea, etc) would have been significantly degraded. It could easily be considered a medium issue to me - especially considering the previous comments / reactions and the label "confirmed" added by the sponsor while it was high.

HardlyDifficult (judge) increased severity to Medium and commented:

> Thanks @Picodes! I can get onboard with that line of thinking. Given how significant these fields are for 3rd party integrators such as Etherscan and Opensea this can be considered to fall under that definition of Medium risk. I'll upgrade this report and the dupes to Medium.

Low Risk and Non-Critical Issues

For this contest, 82 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: BowTiedWardens, joestakey, 0x1f8b, reassor, 0x29A, codexploder, Chom, hyh, berndartmueller, xiaoming90, defsec, ellahi, 0xNineDec, cloudjunky, catchup, cryptphi, cccz, c3phas, 0xNazgul, 0xf15ers, unforgiven, shenwilly, StErMi, oyc_109, Lambda, kenta, sorrynotsorry, JohnSmith, UnusualTurtle, Tadashi, sseefried, simon135, pashov, zzzitron, saian, robee, minhquanym, MadWookie, JMukesh, asutorufos, Picodes, rfa, TerrierLover, dipp, MiloTruck, SmartSek, naps62, TomJ, apostle0x01, zuhaibmohd, Wayne, Waze, kebabsec, PwnedNoMore, exd0tpy, Alex the Entreprenerd, Tomio, Varun_Verma, 0xc0ffEE, hansfriese, Treasure-Seeker, delfin454000, Limbooo, JC, 0xkatana, masterchief, fatherOfBlocks, slywaters, peritoflores, sashik_eth, _Adam, Funen, Nyamcil, sach1r0, Randyyy, 0x52, ElKu, Nethermind, RoiEvenHaim, ych18, *and* Noah3o6.

Low Risk Issues

	Issue	Instances
1	Buyouts that occur during the timestamp wrap will have valuation errors	1
2	`ecrecover()` not checked for signer address of zero	1
3	Return values of `transfer()` / `transferFrom()` not checked	4
4	Input array lengths may differ	4
5	`_safeMint()` should be used rather than `_mint()` wherever possible	1
6	Missing checks for `address(0x0)` when assigning values to `address` state variables	6
7	Vulnerable to cross-chain replay attacks due to static `DOMAIN_SEPARATOR` / `domainSeparator`	1
8	Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions	1
9	Incorrect comments	3

Total: 22 instances over 9 issues

[L-01] Buyouts that occur during the timestamp wrap will have valuation errors

The `_blockTimestamp` has a modulo applied, so at some point, there will be a timestamp with a value close to 2^32, followed by a timestamp close to zero. The `_updateTWAV` function does an unchecked subtraction of the two timestamps, so this will lead to an underflow, making the valuation based on a long time period rather than the actual one. Until more TWAV entries are added, valuations will be wrong

There is 1 instance of this issue:

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

303              uint32 _blockTimestamp = uint32(block.timestam
304              if (_blockTimestamp != lastBlockTimeStamp) {
305:                 _updateTWAV(getCurrentValuation(), _blockTi
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L303-L305

[L-02] `ecrecover()` not checked for signer address of zero

The `ecrecover()` function returns an address of zero when the signature does not match. This can cause problems if address zero is ever the owner of assets, and someone uses the permit function on address zero. If that happens, any invalid signature will pass the checks, and the assets will be stealable. In this case, the asset of concern is the vault's ERC20 token, and fortunately OpenZeppelin's implementation does a good job of making sure that address zero is never able to have a positive balance. If this contract ever changes to another ERC20 implementation that is laxer in its checks in favor of saving gas, this code may become a problem.

There is 1 instance of this issue:

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

563:         address signer = ecrecover(toTypedMessageHash(struc
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L563

[L-03] Return values of `transfer()` / `transferFrom()` not checked

Not all `IERC20` implementations `revert()` when there's a failure in `transfer()` / `transferFrom()` . The function signature has a `boolean` return value and they indicate errors that way instead. By not checking the return value, operations that should have marked as failed, may potentially go through without actually making a payment

There are 4 instances of this issue:

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

87:           IERC20(_token).transfer(msg.sender, IERC20(_token)
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L87

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

94:               IERC20(_tokens[i]).transfer(msg.sender, IERC2(
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L94

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

517:           IERC20(_asset).transfer(_to, IERC20(_asset).balanc
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L517

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

526:               IERC20(_assets[i]).transfer(_to, IERC20(_asset
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L526

[L-04] Input array lengths may differ

If the caller makes a copy-paste error, the lengths may be mismatchd and an operation believed to have been completed may not in fact have been completed

There are 4 instances of this issue:

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

41:       function withdrawMultipleERC721(address[] memory _toker
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L41

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

68:       function withdrawMultipleERC1155(address[] memory _toke
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L68

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

545:     function withdrawMultipleERC1155(address[] memory _asse
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L545

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

504:     function withdrawMultipleERC721(address[] memory _asset
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L504

[L-05] `_safeMint()` should be used rather than `_mint()` wherever possible

`_mint()` is discouraged in favor of `_safeMint()` which ensures that the recipient is either an EOA or implements `IERC721Receiver`. Both OpenZeppelin and solmate have versions of this function

There is 1 instance of this issue:

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

24:             _mint(_curator, 0);
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol

#L24

[L-06] Missing checks for `address(0x0)` when assigning values to `address` state variables

There are 6 instances of this issue:

```
File: contracts/NibblVault.sol

191:            assetAddress = _assetAddress;

193:            curator = _curator;

487:            curator = _newCurator;
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L191

```
File: contracts/NibblVaultFactory.sol

100:          pendingBasketImplementation = _newBasketImplementat

124:          pendingFeeTo = _newFeeAddress;

159:          pendingVaultImplementation = _newVaultImplementatic
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L100

[L-07] Vulnerable to cross-chain replay attacks due to static `DOMAIN_SEPARATOR` / `domainSeparator`

See this issue from a prior contest for details

There is 1 instance of this issue:

```
File: contracts/Utilities/EIP712Base.sol   #1

15        function INIT_EIP712(string memory name, string memory
16            domainSeperator = keccak256(
17                abi.encode(
18                    EIP712_DOMAIN_TYPEHASH,
19                    keccak256(bytes(name)),
20                    keccak256(bytes(version)),
21                    getChainID(),
22                    address(this)
23                )
24            );
25:       }
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/EIP712Base.sol#L15-L25

[L-08] Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions

See this link for a description of this storage variable. While some contracts may not currently be sub-classed, adding the variable now protects against forgetting to add it in the future.

There is 1 instance of this issue:

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

20:   contract NibblVault is INibblVault, BancorFormula, ERC20U
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L20

[L-09] Incorrect comments

There are 3 instances of this issue:

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

/// @audit ERC1155, not ERC721
58:      /// @notice withdraw an ERC721 token from this contract
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L58

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

/// @audit or zero if there have been fewer than four blocks
34:      /// @return _twav TWAV of the last 4 blocks
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L34

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

/// @audit of the last four updates, not necessarily of the last
34:      /// @return _twav TWAV of the last 4 blocks
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L34

Non-Critical Issues

	Issue	Instances
1	Consider addings checks for signature malleability	1
2	Misleading variable name	1
3	Inconsistent version of English being used	2
4	Missing `initializer` modifier on constructor	1

	Issue	Instances
5	Contract implements interface without extending the interface	1
6	`require()` / `revert()` statements should have descriptive reason strings	1
7	`public` functions not called by the contract should be declared `external` instead	3
8	Non-assembly method available	1
9	`2**<n> - 1` should be re-written as `type(uint<n>).max`	4
10	`constant` s should be defined rather than using magic numbers	10
11	Cast is more restrictive than the type of the variable being assigned	1
12	Missing event and or timelock for critical parameter change	4
13	Expressions for constant values such as a call to `keccak256()`, should use `immutable` rather than `constant`	5
14	Inconsistent spacing in comments	27
15	Lines are too long	14
16	Non-library/interface files should use fixed compiler versions, not floating ones	1
17	Typos	14
18	File is missing NatSpec	1
19	NatSpec is incomplete	12
20	Event is missing `indexed` fields	5

Total: 109 instances over 20 issues

[N-01] Consider addings checks for signature malleability

Use OpenZeppelin's `ECDSA` contract rather than calling `ecrecover()` directly

There is 1 instance of this issue:

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

563:         address signer = ecrecover(toTypedMessageHash(struc
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L563

[N-02] Misleading variable name

`_twavObservationPrev` is not the previous observation - it's more like the trailing, or next-to-expire TWAV observation

There is 1 instance of this issue:

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

39:             TwavObservation memory _twavObservationPrev = t
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L39

[N-03] Inconsistent version of English being used

Some functions use American English, whereas others use British English. A single project should use only one of the two

There are 2 instances of this issue:

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

173:     function initialize(
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L173

```
File: contracts/Interfaces/IBasket.sol   #2
```

```
10:       function initialise(address _curator) external;
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Interfaces/IBasket.sol#L10

[N-04] Missing `initializer` modifier on constructor

OpenZeppelin [recommends]() that the `initializer` modifier be applied to constructors in order to avoid potential griefs, [social engineering](), or exploits. Ensure that the modifier is applied to the implementation contract. If the default constructor is currently being used, it should be changed to be an explicit one with the modifier applied.

There is 1 instance of this issue:

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

13:   contract Basket is IBasket, ERC721("NFT Basket", "NFTB"),
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L13

[N-05] Contract implements interface without extending the interface

Not extending the interface may lead to the wrong function signature being used, leading to unexpected behavior. If the interface is in fact being implemented, use the `override` keyword to indicate that fact

There is 1 instance of this issue:

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

/// @audit onERC721Received(), onERC1155Received()
```

```
20:   contract NibblVault is INibblVault, BancorFormula, ERC20Up
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L20

[N-06] `require()` / `revert()` statements should have descriptive reason strings

There is 1 instance of this issue:

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

114:            require(_success);
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L114

[N-07] `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`.

There are 3 instances of this issue:

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

64          function getVaultAddress(
65              address _curator,
66              address _assetAddress,
67              uint256 _assetTokenID,
68              uint256 _initialSupply,
69:             uint256 _initialTokenPrice) public view returns(ac
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L64-L69

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

76:       function getVaults() public view returns(ProxyVault[]
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L76

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

44:       function getTwavObservations() public view returns(Twa
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L44

[N-08] Non-assembly method available

`assembly{ id := chainid() }` => `uint256 id = block.chainid`, `assembly { size := extcodesize() }` => `uint256 size = address().code.length`

There is 1 instance of this issue:

```
File: contracts/Utilities/EIP712Base.sol   #1

29:                 id := chainid()
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/EIP712Base.sol#L29

[N-09] `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)

There are 4 instances of this issue:

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

303:                uint32 _blockTimestamp = uint32(block.timestan
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L303

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

365:                uint32 _blockTimestamp = uint32(block.timestan
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L365

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

413:            _updateTWAV(_currentValuation, uint32(block.timest
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L413

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

```
445:            uint32 _blockTimestamp = uint32(block.timestamp %
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L445

[N-10] `constant`s should be defined rather than using magic numbers

Even [assembly](https://github.com/code-423n4/2022-05-opensea-seaport/blob/9d7ce4d08bf3c3010304a0476a785c70c0e90ae7/contracts/lib/TokenTransferrer.sol#L35-L39) can benefit from using readable constants instead of hex/numeric literals

There are 10 instances of this issue:

```
File: contracts/NibblVaultFactory.sol

/// @audit 0xff
72:             bytes32 _hash = keccak256(abi.encodePacked(bytes1(

/// @audit 0xff
91:             bytes32 hash = keccak256(abi.encodePacked(bytes1((
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L72

```
File: contracts/NibblVault.sol

/// @audit 1e18
183:            uint32 _secondaryReserveRatio = uint32((msg.value

/// @audit 1e18
195:            uint _primaryReserveBalance = (primaryReserveRatic

/// @audit 1e18
226:            secondaryReserveRatio = uint32((secondaryReserveBa

/// @audit 1e18
253:                return ((secondaryReserveRatio * initialTokenS
/// @audit 32
303:                uint32 _blockTimestamp = uint32(block.timestan

/// @audit 32
365:                uint32 _blockTimestamp = uint32(block.timestan

/// @audit 32
413:            _updateTWAV(_currentValuation, uint32(block.timest

/// @audit 32
445:            uint32 _blockTimestamp = uint32(block.timestamp %
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L183

[N-11] Cast is more restrictive than the type of the variable being assigned

If `address foo` is being used in an expression such as `IERC20 token = FooToken(foo)`, then the more specific cast to `FooToken` is a waste because the only thing the compiler will check for is that `FooToken` extends `IERC20` - it won't check any of the function signatures. Therefore, it makes more sense to do `IERC20 token = IERC20(token)` or better yet `FooToken token = FooToken(foo)`. The former may allow the file in which it's used to remove the import for `FooToken`

There is 1 instance of this issue:

```
File: contracts/Proxy/ProxyBasket.sol    #1

/// @audit payable vs address
20:            implementation = payable(_implementation);
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Proxy/ProxyBasket.sol#L20

[N-12] Missing event and or timelock for critical parameter change

Events help non-contract tools to track changes, and events prevent users from being surprised by changes

There are 4 instances of this issue:

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

485      function updateCurator(address _newCurator) external c
486          require(msg.sender == curator,"NibblVault: Only Cu
487          curator = _newCurator;
488:     }
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L485-L488

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

100:         pendingBasketImplementation = _newBasketImplementat
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L100

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

124:         pendingFeeTo = _newFeeAddress;
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L124

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

```
159:        pendingVaultImplementation = _newVaultImplementatic
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L159

[N-13] Expressions for constant values such as a call to `keccak256()`, should use `immutable` rather than `constant`

There are 5 instances of this issue:

```
File: contracts/NibblVault.sol

51:        bytes32 private constant PERMIT_TYPEHASH = keccak256('
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L51

```
File: contracts/Utilities/AccessControlMechanism.sol

12:        bytes32 public constant FEE_ROLE = keccak256("FEE_ROLE

13:        bytes32 public constant PAUSER_ROLE = keccak256("PAUSE

14:        bytes32 public constant IMPLEMENTER_ROLE = keccak256('
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L12

```
File: contracts/Utilities/EIP712Base.sol

7          bytes32 internal constant EIP712_DOMAIN_TYPEHASH = kec
8              bytes(
9                  "EIP712Domain(string name,string version,uint2
10             )
11:         );
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/EIP712Base.sol#L7-L11

[N-14] Inconsistent spacing in comments

Some lines use `// x` and some use `//x`. The instances below point out the usages that don't follow the majority, within each file

There are 27 instances of this issue:

```
File: contracts/NibblVault.sol

28:         uint32 private constant primaryReserveRatio = 200_000;

34:         uint256 private constant REJECTION_PREMIUM = 150_000;

46:         uint256 private constant MIN_CURATOR_FEE = 5_000; //0.

122:        ///@notice current status of vault

125:        ///@notice reenterancy guard

200:            //curator fee is proportional to the secondary res

201:            curatorFee = (((_secondaryReserveRatio - MIN_SECON

220:            //_maxSecondaryBalanceIncrease: is the max amount

221:            //_maxSecondaryBalanceIncrease cannot be more than

226:            secondaryReserveRatio = uint32((secondaryReserveBa

228:                safeTransferETH(_factory, _feeAdmin); //Transf

244:                safeTransferETH(_factory, _feeAdmin); //Transf

301:            //Make update on the first tx of the block

318:                    //Gas Optimization
```

```
363:            //Make update on the first tx of the block

368:                    _rejectBuyout(); //For the case when TWAV

377:                    //Gas Optimization

389:            safeTransferETH(_to, _saleReturn); //send _saleRet

402:            //_buyoutBid: Bid User has made

448:                _rejectBuyout(); //For the case when TWAV goes

500:        ///@notice withdraw multiple ERC721s
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L28

```
File: contracts/Proxy/ProxyBasket.sol

28:        //solhint-disable-next-line no-complex-fallback

31:            //solhint-disable-next-line no-inline-assembly
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Proxy/ProxyBasket.sol#L28

```
File: contracts/Proxy/ProxyVault.sol

28:        //solhint-disable-next-line no-complex-fallback

31:            //solhint-disable-next-line no-inline-assembly
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Proxy/ProxyVault.sol#L28

```
File: contracts/Twav/Twav.sol

12:       uint8 private constant TWAV_BLOCK_NUMBERS = 4; //TWAV

28:           twavObservations[twavObservationsIndex] = TwavObse
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L12

[N-15] Lines are too long

Usually lines in source code are limited to 80 characters. Today's screens are much larger so it's reasonable to stretch this in some cases. Since the files will most likely reside in GitHub, and GitHub starts using a scroll bar in all cases when the length is over 164 characters, the lines below should be split when they reach that length

There are 14 instances of this issue:

```
File: contracts/Basket.sol

109:      function onERC1155BatchReceived(address, address from,
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L109

```
File: contracts/NibblVaultFactory.sol

50:           _proxyVault = payable(new ProxyVault{salt: keccak
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L50

```
File: contracts/NibblVault.sol
```

```
19:   /// @dev The secondary curve is dynamic and has a variable

79:       /// @dev This variable also defines the amount of rese

201:          curatorFee = (((_secondaryReserveRatio - MIN_SECON

224:          _feeCurve = _maxSecondaryBalanceIncrease > _feeCur

226:          secondaryReserveRatio = uint32((secondaryReserveBa

263:      /// @dev Valuation = If current supply is on seconday

266:              return totalSupply() < initialTokenSupply ? (s

297:      /// @dev if current totalSupply < initialTokenSupply A

358:      /// @dev if totalSupply > initialTokenSupply AND _amou

395:      /// @dev bidder needs to send funds equal to current v
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L19

```
File: contracts/Twav/Twav.sol

28:           twavObservations[twavObservationsIndex] = TwavObse

40:               _twav = (_twavObservationCurrent.cumulativeVal
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L28

[N-16] Non-library/interface files should use fixed compiler versions, not floating ones

There is 1 instance of this issue:

```
File: contracts/Utilities/AccessControlMechanism.sol   #1

4:    pragma solidity ^0.8.0;
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L4

[N-17] Typos

There are 14 instances of this issue:

```
File: contracts/NibblVault.sol

/// @audit reenterancy
125:      ///@notice reenterancy guard

/// @audit pausablity
152:      /// @dev pausablity implemented in factory

/// @audit primaryReseveRatio
200:          //curator fee is proportional to the secondary res

/// @audit primaryReseveRatio
201:          curatorFee = (((_secondaryReserveRatio - MIN_SECON

/// @audit continous
250:      /// @dev The max continous tokens on SecondaryCurve is

/// @audit seconday
263:      /// @dev Valuation = If current supply is on seconday

/// @audit continous
270:      /// @param _amount amount of reserve tokens to buy cor

/// @audit continous
282:      /// @param _amount amount of reserve tokens to buy cor

/// @audit Continous
359:      /// @param _amtIn Continous Tokens to be sold

/// @audit recieve
```

```
361:      /// @param _to Address to recieve the reserve token to

/// @audit airdops
512:      /// @notice ERC20s can be accumulated by the underlyir

/// @audit airdops
531:      /// @notice ERC1155s can be accumulated by the underly
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L125

```
File: contracts/Proxy/ProxyBasket.sol

/// @audit internall
26:        * This function does not return to its internall call
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Proxy/ProxyBasket.sol#L26

```
File: contracts/Proxy/ProxyVault.sol

/// @audit internall
26:        * This function does not return to its internall call
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Proxy/ProxyVault.sol#L26

[N-18] File is missing NatSpec

There is 1 instance of this issue:

```
File: contracts/Utilities/EIP712Base.sol (various lines)   #1
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/EIP712Base.sol

[N-19] NatSpec is incomplete

There are 12 instances of this issue:

See original submission for details.

[N-20] Event is missing `indexed` fields

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

There are 5 instances of this issue:

```
File: contracts/Basket.sol

15:        event DepositERC721(address indexed token, uint256 tok
16:        event WithdrawERC721(address indexed token, uint256 to
17:        event DepositERC1155(address indexed token, uint256 to
18:        event DepositERC1155Bulk(address indexed token, uint25
19:        event WithdrawERC1155(address indexed token, uint256 t
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L15

HardlyDifficult (judge) commented:

> Great feedback & it all appears valid.

Gas Optimizations

For this contest, 63 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IllIllI received the top score from the judge.

The following wardens also submitted reports: joestakey, BowTiedWardens, _Adam, m_Rassska, 0xkatana, 0xKitsune, defsec, MiloTruck, c3phas, minhquanym, hansfriese, catchup, robee, UnusualTurtle, 0xNazgul, reassor, TomJ, sashik_eth, pashov, delfin454000, Tomio, slywaters, ElKu, JC, saian, 0x1f8b, Noah3o6, sach1r0, 8olidity, ajtra, ellahi, oyc_109, TerrierLover, simon135, Chom, 0x29A, Waze, fatherOfBlocks, rfa, 0xf15ers, ACai, cRat1st0s, Picodes, Chandr, ych18, 0v3rf10w, Randyyy, zuhaibmohd, StErMi, ynnad, cryptphi, Funen, Lambda, Nyamcil, kenta, Limbooo, IgnacioB, SmartSek, codexploder, exd0tpy, kebabsec, *and* Fitraldys.

	Issue	Instances
1	Setting `DEFAULT_ADMIN_ROLE` as the role admin is redundant	1
2	Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas	23
3	Using `storage` instead of `memory` for structs/arrays saves gas	2
4	State variables should be cached in stack variables rather than re-reading them from storage	1
5	`<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables	7
6	`internal` functions only called once can be inlined to save gas	1
7	Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if`-statement	5
8	`<array>.length` should not be looked up in every loop of a `for`-loop	6
9	`++i`/`i++` should be `unchecked{++i}`/`unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for`- and `while`-loops	6
10	`require()`/`revert()` strings longer than 32 bytes cost extra gas	7
11	Using `bool`s for storage incurs overhead	1
12	Use a more recent version of solidity	1
13	`>=` costs less gas than `>`	1
14	It costs more gas to initialize non-`constant`/non-`immutable` variables to zero than to let the default of zero be applied	6

	Issue	Instances
15	`++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)	6
16	Splitting `require()` statements that use `&&` saves gas	4
17	Usage of `uints`/`ints` smaller than 32 bytes (256 bits) incurs overhead	14
18	Using `private` rather than `public` for constants, saves gas	3
19	Don't use `SafeMath` once the solidity version is 0.8.0 or greater	1
20	Duplicated `require()`/`revert()` checks should be refactored to a modifier or function	3
21	Empty blocks should be removed or emit something	5
22	Use custom errors rather than `revert()`/`require()` strings to save gas	41
23	Functions guaranteed to revert when called by normal users can be marked `payable`	8

Total: 153 instances over 23 issues

[1] Setting `DEFAULT_ADMIN_ROLE` as the role admin is redundant

`DEFAULT_ADMIN_ROLE` is [automatically]() designated as the role admin of any new role, so setting it again is a waste of gas since it involves fetching role-related state variables, updating state variables, and emitting an event

There is 1 instance of this issue:

```
File: contracts/Utilities/AccessControlMechanism.sol   #1

22             _setRoleAdmin(_defaultAdminRole, _defaultAdminRole)
23             _setRoleAdmin(FEE_ROLE, _defaultAdminRole);
24             _setRoleAdmin(PAUSER_ROLE, _defaultAdminRole);
25:            _setRoleAdmin(IMPLEMENTER_ROLE, _defaultAdminRole);
```

[https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/Ac]()

cessControlMechanism.sol#L22-L25

[2] Using `calldata` instead of `memory` for read-only arguments in `external` functions saves gas

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. **Each iteration of this for-loop costs at least 60 gas** (i.e. `60 * <mem_array>.length`). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution.

If the array is passed to an `internal` function which passes the array to another internal function where the array is modified and therefore `memory` is used in the `external` call, it's still more gass-efficient to use `calldata` when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one

There are 23 instances of this issue:

```
File: contracts/Basket.sol

41:        function withdrawMultipleERC721(address[] memory _toke

41:        function withdrawMultipleERC721(address[] memory _toke

68:        function withdrawMultipleERC1155(address[] memory _tok

68:        function withdrawMultipleERC1155(address[] memory _tok

91:        function withdrawMultipleERC20(address[] memory _toker

99:        function onERC721Received(address, address from, uint2

104:       function onERC1155Received(address, address from, uint

109:       function onERC1155BatchReceived(address, address from,

109:       function onERC1155BatchReceived(address, address from,

109:       function onERC1155BatchReceived(address, address from,
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L41

```
File: contracts/NibblVaultFactory.sol

41:            string memory _name,

42:            string memory _symbol,
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L41

```
File: contracts/NibblVault.sol

174:           string memory _tokenName,

175:           string memory _tokenSymbol,

504:       function withdrawMultipleERC721(address[] memory _asse

504:       function withdrawMultipleERC721(address[] memory _asse

523:       function withdrawMultipleERC20(address[] memory _asset

545:       function withdrawMultipleERC1155(address[] memory _ass

545:       function withdrawMultipleERC1155(address[] memory _ass

577:       function onERC1155Received(address, address, uint256,

581:       function onERC1155BatchReceived(address, address, uint

581:       function onERC1155BatchReceived(address, address, uint

581:       function onERC1155BatchReceived(address, address, uint
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.

sol#L174

[3] Using `storage` instead of `memory` for structs/arrays saves gas

When fetching data from a storage location, assigning the data to a `memory` variable causes all fields of the struct/array to be read from storage, which incurs a Gcoldsload (**2100 gas**) for *each* field of the struct/array. If the fields are read from the new memory variable, they incur an additional `MLOAD` rather than a cheap stack read. Instead of declearing the variable with the `memory` keyword, declaring the variable with the `storage` keyword and caching any fields that need to be re-read in stack variables, will be much cheaper, only incuring the Gcoldsload for the fields actually read. The only time it makes sense to read the whole struct/array into a `memory` variable, is if the full struct/array is being returned by the function, is being passed to a function that requires `memory` , or if the array/struct is being read from another `memory` array/struct

There are 2 instances of this issue:

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

38:              TwavObservation memory _twavObservationCurrent
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L38

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

39:              TwavObservation memory _twavObservationPrev =
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L39

[4] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replace each Gwarmaccess (**100 gas**) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

There is 1 instance of this issue:

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

/// @audit secondaryReserveBalance on line 225
226:            secondaryReserveRatio = uint32((secondaryReserveBa
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L226

[5] `<x> += <y>` costs more gas than `<x> = <x> + <y>` for state variables

There are 7 instances of this issue:

```
File: contracts/NibblVault.sol

219:            feeAccruedCurator += _feeCurator;

225:            secondaryReserveBalance += _feeCurve;

242:            feeAccruedCurator += _feeCurator;

320:                    secondaryReserveBalance += _lowerCurveDiff

380:                    primaryReserveBalance -= _saleReturn;

429:                totalUnsettledBids += _buyoutValuationDeposit;
```

```
457:            totalUnsettledBids -= _amount;
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L219

[6] `internal` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There is 1 instance of this issue:

```
File: contracts/Utilities/EIP712Base.sol   #1

27:        function getChainID() internal view returns (uint256 i
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/EIP712Base.sol#L27

[7] Add `unchecked {}` for subtractions where the operands cannot underflow because of a previous `require()` or `if`-statement

`require(a <= b); x = b - a` => `require(a <= b); unchecked { x = b - a }`

There are 5 instances of this issue:

```
File: contracts/NibblVault.sol

/// @audit require() on line 185
201:            curatorFee = (((_secondaryReserveRatio - MIN_SECON
/// @audit require() on line 404
406:         buyoutValuationDeposit = msg.value - (_buyoutBid -

/// @audit require() on line 404
415:             safeTransferETH(payable(msg.sender), (_buyoutE

/// @audit if-condition on line 373
378:                 uint256 _tokensPrimaryCurve = _totalSuppl

/// @audit if-condition on line 414
415:             safeTransferETH(payable(msg.sender), (_buyoutE
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L201

[8] `<array>.length` should not be looked up in every loop of a `for`-loop

The overheads outlined below are *PER LOOP*, excluding the first loop

- storage arrays incur a Gwarmaccess (**100 gas**)
- memory arrays use `MLOAD` (**3 gas**)
- calldata arrays use `CALLDATALOAD` (**3 gas**)

Caching the length changes each of these to a `DUP<N>` (**3 gas**), and gets rid of the extra `DUP<N>` needed to store the stack offset

There are 6 instances of this issue:

```
File: contracts/Basket.sol

43:         for (uint256 i = 0; i < _tokens.length; i++) {

70:         for (uint256 i = 0; i < _tokens.length; i++) {

93:         for (uint256 i = 0; i < _tokens.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L43

```
File: contracts/NibblVault.sol

506:            for (uint256 i = 0; i < _assetAddresses.length; i+

525:            for (uint256 i = 0; i < _assets.length; i++) {

547:            for (uint256 i = 0; i < _assets.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L506

[9] `++i` / `i++` should be `unchecked{++i}` / `unchecked{i++}` when it is not possible for them to overflow, as is the case when used in `for` - and `while` -loops

The `unchecked` keyword is new in solidity version 0.8.0, so this only applies to that version or higher, which these instances are. This saves **30-40 gas** per loop

There are 6 instances of this issue:

```
File: contracts/Basket.sol

43:            for (uint256 i = 0; i < _tokens.length; i++) {

70:            for (uint256 i = 0; i < _tokens.length; i++) {

93:            for (uint256 i = 0; i < _tokens.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L43

```
File: contracts/NibblVault.sol

506:            for (uint256 i = 0; i < _assetAddresses.length; i+

525:            for (uint256 i = 0; i < _assets.length; i++) {

547:            for (uint256 i = 0; i < _assets.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L506

[10] `require()` / `revert()` strings longer than 32 bytes cost extra gas

Each extra memory word of bytes past the original 32 incurs an MSTORE which costs **3 gas**

There are 7 instances of this issue:

```
File: contracts/NibblVaultFactory.sol

48:             require(msg.value >= MIN_INITIAL_RESERVE_BALANCE,

49:             require(IERC721(_assetAddress).ownerOf(_assetToker

107:            require(basketUpdateTime != 0 && block.timestamp >

131:            require(feeToUpdateTime != 0 && block.timestamp >=

141:            require(_newFee <= MAX_ADMIN_FEE, "NibblVaultFactc

149:            require(feeAdminUpdateTime != 0 && block.timestamp

166:            require(vaultUpdateTime != 0 && block.timestamp >=
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L48

[11] Using `bool` s for storage incurs overhead

```
    // Booleans are more expensive than uint256 or any type that
    // word because each write operation emits an extra SLOAD tc
    // slot's contents, replace the bits taken up by the boolear
    // back. This is the compiler's defense against contract upç
    // pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27 Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from 'false' to 'true', after having been 'true' in the past

There is 1 instance of this issue:

```
File: contracts/Utilities/AccessControlMechanism.sol   #1

16:       mapping(bytes32 => mapping(address => bool)) public pe
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L16

[12] Use a more recent version of solidity

Use a solidity version of at least 0.8.2 to get simple compiler automatic inlining Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than `revert()/require()` strings Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

There is 1 instance of this issue:

```
File: contracts/Utilities/AccessControlMechanism.sol   #1
```

```
4:    pragma solidity ^0.8.0;
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L4

[13] `>=` costs less gas than `>`

The compiler uses opcodes `GT` and `ISZERO` for solidity code that uses `>`, but only requires `LT` for `>=`, which saves 3 gas

There is 1 instance of this issue:

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

224:          _feeCurve = _maxSecondaryBalanceIncrease > _feeCur
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L224

[14] It costs more gas to initialize non-`constant`/non-`immutable` variables to zero than to let the default of zero be applied

Not overwriting the default for stack variables saves **8 gas**. Storage and memory variables have larger savings

There are 6 instances of this issue:

```
File: contracts/Basket.sol

43:          for (uint256 i = 0; i < _tokens.length; i++) {

70:          for (uint256 i = 0; i < _tokens.length; i++) {
```

```
93:            for (uint256 i = 0; i < _tokens.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L43

```
File: contracts/NibblVault.sol

506:            for (uint256 i = 0; i < _assetAddresses.length; i+

525:            for (uint256 i = 0; i < _assets.length; i++) {

547:            for (uint256 i = 0; i < _assets.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L506

[15] `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

Saves **6 gas per loop**

There are 6 instances of this issue:

```
File: contracts/Basket.sol

43:            for (uint256 i = 0; i < _tokens.length; i++) {

70:            for (uint256 i = 0; i < _tokens.length; i++) {

93:            for (uint256 i = 0; i < _tokens.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L43

```
File: contracts/NibblVault.sol

506:          for (uint256 i = 0; i < _assetAddresses.length; i+

525:          for (uint256 i = 0; i < _assets.length; i++) {

547:          for (uint256 i = 0; i < _assets.length; i++) {
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L506

[16] Splitting `require()` statements that use `&&` saves gas

See this issue which describes the fact that there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper

There are 4 instances of this issue:

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

107:          require(basketUpdateTime != 0 && block.timestamp >
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L107

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

131:          require(feeToUpdateTime != 0 && block.timestamp >=
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L131

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

```
149:            require(feeAdminUpdateTime != 0 && block.timestamp
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L149

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

166:            require(vaultUpdateTime != 0 && block.timestamp >=
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L166

[17] Usage of `uints` / `ints` smaller than 32 bytes (256 bits) incurs overhead

> When using elements that are smaller than 32 bytes, your contract's gas usage may be higher. This is because the EVM operates on 32 bytes at a time. Therefore, if the element is smaller than that, the EVM must use more operations in order to reduce the size of the element from 32 bytes to the desired size.

https://docs.soliditylang.org/en/v0.8.11/internals/layout_in_storage.html Use a larger size then downcast where needed

There are 14 instances of this issue:

```
File: contracts/NibblVault.sol

28:        uint32 private constant primaryReserveRatio = 200_000;

57:        uint32 public secondaryReserveRatio;

183:            uint32 _secondaryReserveRatio = uint32((msg.value

303:                uint32 _blockTimestamp = uint32(block.timestan
365:                uint32 _blockTimestamp = uint32(block.timestam

445:            uint32 _blockTimestamp = uint32(block.timestamp %

557:            uint8 v,
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L28

```
File: contracts/Twav/Twav.sol

6:              uint32 timestamp;

11:         uint8 public twavObservationsIndex;

12:         uint8 private constant TWAV_BLOCK_NUMBERS = 4; //TWAV

13:         uint32 public lastBlockTimeStamp;

21:         function _updateTWAV(uint256 _valuation, uint32 _block

22:             uint32 _timeElapsed;

37:                 uint8 _index = ((twavObservationsIndex + TWAV_
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Twav/Twav.sol#L6

[18] Using `private` rather than `public` for constants, saves gas

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves **3406-3606 gas** in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table

There are 3 instances of this issue:

```
File: contracts/Utilities/AccessControlMechanism.sol   #1

12:       bytes32 public constant FEE_ROLE = keccak256("FEE_ROLE
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L12

```
File: contracts/Utilities/AccessControlMechanism.sol   #2

13:       bytes32 public constant PAUSER_ROLE = keccak256("PAUSE
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L13

```
File: contracts/Utilities/AccessControlMechanism.sol   #3

14:       bytes32 public constant IMPLEMENTER_ROLE = keccak256('
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L14

[19] Don't use `SafeMath` once the solidity version is 0.8.0 or greater

Version 0.8.0 introduces internal overflow checks, so using `SafeMath` is redundant and adds overhead

There is 1 instance of this issue:

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

```
9:    import { SafeMath } from  "@openzeppelin/contracts/utils/m
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L9

[20] Duplicated `require()` / `revert()` checks should be refactored to a modifier or function

Saves deployment costs

There are 3 instances of this issue:

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

42:           require(_isApprovedOrOwner(msg.sender, 0), "withdr
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L42

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

486:          require(msg.sender == curator,"NibblVault: Only Cu
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L486

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

505:          require(msg.sender == bidder,"NibblVault: Only wir
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L505

[21] Empty blocks should be removed or emit something

The code should be refactored such that they no longer exist, or the block should do something useful, such as emitting an event or reverting. If the contract is meant to be extended, the contract should be `abstract` and the function signatures be added without any default implementation. If the block is an empty `if`-statement block to avoid doing subsequent checks in the else-if/else conditions, the else-if/else conditions should be nested under the negation of the if-statement, because they involve different classes of checks, which may lead to the introduction of errors when the code is later modified (`if(x){}else if(y){...}else{...} => if(!x){if(y){...}else{...}}`)

There are 5 instances of this issue:

```
File: contracts/Basket.sol

114:        receive() external payable {}
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Basket.sol#L114

```
File: contracts/NibblVaultFactory.sol

183:        receive() payable external {    }
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L183

```
File: contracts/NibblVault.sol
```

```
585:        receive() external payable {}
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVault.sol#L585

```
File: contracts/Proxy/ProxyBasket.sol

56:        receive() external payable {    }
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Proxy/ProxyBasket.sol#L56

```
File: contracts/Proxy/ProxyVault.sol

56:        receive() external payable {    }
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Proxy/ProxyVault.sol#L56

[22] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hitby avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

There are 41 instances of this issue:

See original submission for details.

[23] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about **21 gas per call** to the function, in addition to the extra deployment cost

There are 8 instances of this issue:

```
File: contracts/NibblVaultFactory.sol

99:        function proposeNewBasketImplementation(address _newBa

123:        function proposeNewAdminFeeAddress(address _newFeeAddr

140:        function proposeNewAdminFee(uint256 _newFee) external

158:        function proposeNewVaultImplementation(address _newVau

173:        function pause() external onlyRole(PAUSER_ROLE) overri

179:        function unPause() external onlyRole(PAUSER_ROLE) over
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/NibblVaultFactory.sol#L99

```
File: contracts/Utilities/AccessControlMechanism.sol

32:        function setRoleAdmin(bytes32 _role, bytes32 _adminRol

40:        function proposeGrantRole(bytes32 _role, address _to)
```

https://github.com/code-423n4/2022-06-nibbl/blob/8c3dbd6adf350f35c58b31723d42117765644110/contracts/Utilities/AccessControlMechanism.sol#L32

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.

Top