CANTINA

Octant v2 Core

Security Review

Cantina Managed review by:

Saw-mon and Natalie, Lead Security Researcher
Hash, Security Researcher

October 29, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Octant is an experiment in participatory public goods funding, utilizing Golem's native ERC-20 token, GLM. Developed by the Golem Foundation, Octant explores motivations for public goods support.

From Jul 16th to Sep 5th the Cantina team conducted a review of octant-v2-core on commit hash 6a426e3c. The review focused on the following scope:

```
src/
   core/
       BaseStrategy.sol
       PaymentSplitter.sol
       TokenizedStrategy.sol
   mechanisms/
       BaseAllocationMechanism.sol
       TokenizedAllocationMechanism.sol
       mechanism/
           QuadraticVotingMechanism.sol
           OctantQFMechanism.sol
       voting-strategy/
            ProperQF.sol
   regen/
       RegenEarningPowerCalculator.sol
       RegenStaker.sol
       RegenStakerBase.sol
       RegenStakerWithoutDelegateSurrogateVotes.sol
   strategies/
       periphery/
           BaseHealthCheck.sol  (needs to be added)
           BaseYieldSkimmingHealthCheck.sol
       yieldDonating/
           MorphoCompounderStrategy.sol
           SkyCompounderStrategy.sol
           YieldDonatingTokenizedStrategy.sol
       yieldSkimming/
            BaseYieldSkimmingStrategy.sol
            LidoStrategy.sol
            RocketPoolStrategy.sol
            YieldSkimmingTokenizedStrategy.sol
   utils/
       Whitelist.sol
   zodiac-core/
        LinearAllowanceExecutor.sol
        modules/
            LinearAllowanceSingletonForGnosisSafe.sol
```

Note that originally, `src/strategies/periphery/UniswapV3Swapper.sol` was not part of the scope, but the Cantina Managed team decided to go over it since it was a dependency for one of the in-scope contracts.

The team identified a total of **59** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	2	2	0
Medium Risk	12	10	2
Low Risk	18	14	4
Gas Optimizations	4	3	1
Informational	23	15	8
Total	**59**	**44**	**15**

The Cantina Managed team confirms that the majority of the findings were fixed on commit hash 389a315b. Additional fixes were added on subsequent commits, which are explicitly declared under each of the affected findings.

3 Findings

3.1 High Risk

3.1.1 `totalUnspent` gets accrued even during the revoke periods

Severity: High Risk

Context: LinearAllowanceSingletonForGnosisSafe.sol#L153-L165, LinearAllowanceSingletonForGnosis-Safe.sol#L226-L248

Description: When a `delegate` which already has an `allowance` rate set for a `safe` gets revoked, the `allowance.lastBookedAtInSeconds` either:

- Does not get updated if since the last update the `newAccrued` would be `0`.
- Or it gets updated to the timestamp `_revokeAllowance` was called through some external call flows.

Let's call this timestamp t_r which might be even less than the revoke timestamp. Now assume at some other time $t_f > t_r$, the `safe` decides to set an allowance for this `delegate` and thus one enters into `_setAllowance`:

```
function _setAllowance(address safe, address delegate, address token, uint192 dripRatePerDay) internal {
    // Cache storage struct in memory to save gas
    if (delegate == address(0)) revert AddressZeroForArgument("delegate");
    LinearAllowance memory a = allowances[safe][delegate][token];

    a = _calculateCurrentAllowance(a); // <-- at this point `a.dripRatePerDay == 0`
    a.dripRatePerDay = dripRatePerDay;

    allowances[safe][delegate][token] = a;

    emit AllowanceSet(safe, delegate, token, dripRatePerDay);
}
```

In the above when `_calculateCurrentAllowance(a)` gets called, we would have `a.dripRatePerDay == 0`:

```
function _calculateCurrentAllowance(LinearAllowance memory a) internal view returns (LinearAllowance memory) {
    uint256 newAccrued = _calculateNewAccrued(a); // <--- `newAccrued` would be `0` since `a.dripRatePerDay == 0`

    if (a.lastBookedAtInSeconds == 0) {
        a.lastBookedAtInSeconds = block.timestamp.toUint64();
    } else if (newAccrued > 0) {
        a.totalUnspent += newAccrued;
        a.lastBookedAtInSeconds = block.timestamp.toUint64();
    }

    return a;
}
```

Thus `newAccrued` ends up being `0` since `a.dripRatePerDay == 0`. Therefore the whole `if/else` block gets skipped since.

- `a.lastBookedAtInSeconds` is non-zero and.
- `newAccrued == 0`.

Thus `a.lastBookedAtInSeconds` does not get updated to the timestamp where this `_setAllowance` was called after the previous revoke. Although one should have set `a.lastBookedAtInSeconds` to be the timestamp `_setAllowance` is called which is t_f. Thus upon the next accrual for the period $[t_r, t_f]$ which the `delegate` was revoked an unspent amount is allocated with the drip rate set for this new set allowance.

Proof of Concept: Apply this patch:

```
  diff --git a/test/unit/zodiac-core/LinearAllowanceExecutor.t.sol
  ↪  b/test/unit/zodiac-core/LinearAllowanceExecutor.t.sol
  index 54eed17..7247fb0 100644
-  -- a/test/unit/zodiac-core/LinearAllowanceExecutor.t.sol
+  ++ b/test/unit/zodiac-core/LinearAllowanceExecutor.t.sol
  @@ -31,6 +31,43 @@ contract LinearAllowanceExecutorTest is Test {
               mockToken.mint(address(mockSafe), 10 ether);
         }

+        function testComplexLinearAllowanceTimeline() public {
```

```
+          // using the below calls in series to just set the
+          // allowances[mockSafe][executor][NATIVE_TOKEN].lastBookedAtInSeconds
+          // to a non-zero value
+          vm.startPrank(address(mockSafe));
+          allowanceModule.setAllowance(address(executor), NATIVE_TOKEN, DRIP_RATE);
+          allowanceModule.revokeAllowance(address(executor), NATIVE_TOKEN);
+          vm.stopPrank();
+
+          // Advance time to unspent amount should not accrue since we have called
+          // `revokeAllowance`
+          vm.warp(block.timestamp + 1 days);
+          uint256 totalUnspent0 = executor.getTotalUnspent(allowanceModule, address(mockSafe), NATIVE_TOKEN);
+          assertEq(totalUnspent0, 0, "1. Executor's totalUnspent should be zero before setting the allowance
↪  again after revoke.");
+
+          vm.startPrank(address(mockSafe));
+          allowanceModule.setAllowance(address(executor), NATIVE_TOKEN, DRIP_RATE);
+          vm.stopPrank();
+
+          uint256 totalUnspent1 = executor.getTotalUnspent(allowanceModule, address(mockSafe), NATIVE_TOKEN);
+          assertEq(totalUnspent1, 0, "2. Executor's totalUnspent should be zero, since no time has passed since
↪  setting the allowance again after revoke.");
+
+          // Get executor's balance before transfer
+          uint256 executorBalanceBefore = address(executor).balance;
+
+          // Execute allowance transfer
+          uint256 transferredAmount = executor.executeAllowanceTransfer(allowanceModule, address(mockSafe),
↪  NATIVE_TOKEN);
+
+          // Verify transfer
+          assertEq(transferredAmount, 0, "Should transfer the exact drip rate amount");
+          assertEq(
+              address(executor).balance - executorBalanceBefore,
+              0,
+              "3. Executor balance should increase by the transferred amount"
+          );
+      }
+
       function testExecuteAllowanceTransferWithNativeToken() public {
           // Set up allowance for executor
           vm.prank(address(mockSafe));
```

and run:

```
forge test -vvvv --mt testComplexLinearAllowanceTimeline
```

Recommendation: Define a new `internal` function to atomically update `allowance.dripRatePerDay` and `allowance.lastBookedAtInSeconds` together:

```
function _updateDripRatePerDay(LinearAllowance memory a, uint192 dripRatePerDay) internal view {
   a.dripRatePerDay = dripRatePerDay;
   a.lastBookedAtInSeconds = block.timestamp.toUint64();
}
```

The above function `_updateDripRatePerDay` can be used in:

- `_setAllowance`.
- `_revokeAllowance`.

To replace the lines:

```
- a.dripRatePerDay = X;
+ _updateDripRatePerDay(a, X);
```

Moreover `_calculateCurrentAllowance` can be changed to:

```
function _calculateCurrentAllowance(LinearAllowance memory a) internal view returns (LinearAllowance memory) {
    uint256 newAccrued = _calculateNewAccrued(a);

   if (newAccrued > 0) {
        a.totalUnspent += newAccrued;
        a.lastBookedAtInSeconds = block.timestamp.toUint64();
    }
    return a;
}
```

Octant: Fixed in commit 14639a92.

Cantina Managed: Fix verified.

3.1.2 `_convertToAssets` and `_convertToShares` has differing conversion rates causing users to loose assets

Severity: High Risk

Context: YieldSkimmingTokenizedStrategy.sol#L160-L167

Description: `_convertToAssets` uses `totalAssets/totalSupply` as the share price for conversion while `_convertToShares` uses the underlying exchange rate ie. `_currentRateRay()` for conversion.

```
function _convertToAssets(
    StrategyData storage S,
    uint256 shares,
    Math.Rounding _rounding
) internal view virtual returns (uint256) {
    // Saves an extra SLOAD if totalSupply() is non-zero.
    uint256 supply = _totalSupply(S);

    return supply == 0 ? shares : shares.mulDiv(_totalAssets(S), supply, _rounding);
}
```

```
function _convertToShares(
    StrategyData storage,
    uint256 assets,
    Math.Rounding rounding
) internal view virtual override returns (uint256) {
    return assets.mulDiv(_currentRateRay(), WadRayMath.RAY, rounding);
}
```

This for example causes users to spend different amount of assets to mint the same amount of shares when using mint vs deposit.

Proof of Concept: Place the test in `test/integration/strategies/YieldSkimming/LidoStrategy.t.sol` and run `forge test --mt testPOC_DiffBwConversions -vv`. It is shown that the user has to spend 2x the amount of assets when using `mint`.

```
function testPOC_DiffBwConversions() public {
    TestState memory state;

    // Both users deposit same amount
    state.user1 = user;
    state.user2 = address(0x5678);
    uint depositAmount = 1000e18; // 1000 WSTETH

    // airdrop and approve tokens
    vm.startPrank(address(this));
    airdrop(ERC20(WSTETH), state.user2, depositAmount * 10);
    vm.stopPrank();

     vm.startPrank(state.user2);
    ERC20(WSTETH).approve(address(strategy), type(uint256).max);
    vm.stopPrank();

    // Initial exchange rate: 1e18
    state.initialExchangeRate = 1e18;
    vm.mockCall(WSTETH, abi.encodeWithSignature("stEthPerToken()"), abi.encode(state.initialExchangeRate));

    vm.prank(state.user1);
    uint256 sharesUser1 = vault.deposit(depositAmount, state.user1);
    assert(sharesUser1 == depositAmount);

    vm.clearMockedCalls();

    // Set exchange rate to 2x
    state.newExchangeRate1 = state.initialExchangeRate * 2;
    vm.mockCall(WSTETH, abi.encodeWithSignature("stEthPerToken()"), abi.encode(state.newExchangeRate1));

    uint256 snapshotId = vm.snapshot();

    // functions that internally call _convertToShares uses the latest exchange rate from stETH for conversion.
    ↪  hence will obtain 2x the shares of the first user
    vm.prank(state.user2);
    uint sharesUser2 = vault.deposit(depositAmount, state.user2);

    assert(sharesUser2 == sharesUser1 * 2);

    vm.revertTo(snapshotId);

    // but functions that internally call _convertToAssets computes the assets using totalAsset/totalSupply.
    ↪  hence to mint the same number of shares, 2x the number of assets is required

    vm.prank(state.user2);
    uint256 assets = vault.mint(sharesUser2, state.user2);

    assert(assets == depositAmount * 2);
}
```

Recommendation: Use the underlying exchange rate ie. return value of `_currentRateRay` in both the functions. But this is still problematic if the total asset value falls short, since then it wouldn't be possible to settle all the users at this rate and newer deposits will lose some of their assets instantly. Using totalAssets/totalSupply for conversion will cause newer deposits to earn yield till their higher share/underlying value is satisfied (and possibly drain dragonRouter shares in case burning wasn't enabled earlier). This part doesn't seem solvable if a common exchange rate is used. Either would have to change the implementation to distinguish between older/newer deposits or a quick fix would be to disable deposits and use totalAssets/totalSupply as the conversion rate till the exchange rate comes back up (this would have an almost similar effect to allowing newer deposits to gain yield).

Octant: Fixed in commit a5401a0d.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 `MorphoCompounderStrategy`'s `_harvestAndReport()` does not consider idle funds

Severity: Medium Risk

Context: MorphoCompounderStrategy.sol#L68-L74

Description: The NatSpec for harvestAndReport() mentions that:

```
/**
 // ...
 * @return . A trusted and accurate account for the total amount
 * of 'asset' the strategy currently holds including idle funds.
 */
```

And thus based on the spec of `BaseStrategy`, one should also include idle funds in the return value for `_harvestAndReport()`. For comparison `Base4626Compounder` has the following line which takes into account the idle funds:

```
// Return total balance
_totalAssets = balanceOfAsset() + valueOfVault();
```

Recommendation: Make sure to include `IERC20(asset).balanceOf(address(this))` into the returned `_totalAssets`.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.2.2 `YieldDonatingTokenizedStrategy` invariants

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: A stated invariants of `YieldDonatingTokenizedStrategy` is that:

$$A_{tot} + A_{loss} = \texttt{principals} + \texttt{profits} \qquad (1)$$

We will analyse how this value and the coversion rates (price per shares) are affected in different flows:

$$\left(\frac{A_{tot} + A_{loss}}{S_{tot}}, \frac{A_{tot}}{S_{tot}}\right)$$

1. Deposit (TokenizedStrategy.sol#L470-L504): In this flow $\frac{A_{tot}}{S_{tot}}$ is conserved (more accurately the rounding directions make sure it is slightly improved):

$$(A_{tot}, A_{loss}, S_{tot}) \to (A_{tot} + a_d, A_{loss}, S_{tot} + s_d)$$
$$\frac{A_{tot}}{S_{tot}} = \frac{A_{tot} + a_d}{S_{tot} + s_d}$$
$$(A_{tot} + A_{loss}) \to (A_{tot} + A_{loss}) + a_d$$

(thus invariant (1) is preserved).

And if A_{loss} is non-zero then the conversion rate $\frac{A_{tot}+A_{loss}}{S_{tot}}$ actually decreases:

$$\frac{A_{tot} + A_{loss}}{S_{tot}} > \frac{(A_{tot} + a_d) + A_{loss}}{S_{tot} + s_d}$$

which means that more shares where minted for the user who deposits since A_{loss} was not used for the deposit conversion rate. This would affect the accounting once this losses are covered later on by reports. The portion of recovered losses would account for profit for this user.

2. Withdraw without loss (refs: TokenizedStrategy.sol#L529-L543, TokenizedStrategy.sol#L569-L585, TokenizedStrategy.sol#L914-L973): Just like `deposit/mint` routes the implementation preserves $\frac{A_{tot}}{S_{tot}}$ (up to rounding):

$$(A_{tot}, A_{loss}, S_{tot}) \to (A_{tot} - a_w, A_{loss}, S_{tot} - s_w)$$

$$\frac{A_{tot}}{S_{tot}} = \frac{A_{tot} - a_d}{S_{tot} - s_d}$$

and if A_{loss} is non-zero then the conversion rate $\frac{A_{tot}+A_{loss}}{S_{tot}}$ actually increases:

$$\frac{A_{tot} + A_{loss}}{S_{tot}} < \frac{(A_{tot} - a_d) + A_{loss}}{S_{tot} - s_d}$$

In this case the user would not recover all its principal if they do not wait for all the tracked losses to be recovered. If the user withdraws before the tracked losses are recovered, users who would withdraw after the tracked losses are recovered would receive more assets.

3. Withdraw with loss (refs: TokenizedStrategy.sol#L529-L543, TokenizedStrategy.sol#L569-L585, TokenizedStrategy.sol#L914-L973, TokenizedStrategy.sol#L948-L957): This is the same as case 2. plus the user would also absorb untracked losses. These absorbed losses would not get tracked in the future reports. Also depending on the yield source, if a loss occurs in this case. All the remaining assets would be considered as a loss in the next report. After the next report we would probably have $A_{tot} = 0$ and $A_{loss} > 0$.

4. Report profit:** In this flow we would have $\Delta A_{tot} > 0$:

$$\begin{aligned} A_{tot} &\to A_{tot} + \Delta A_{tot} \\ A_{loss} &\to A_{loss} - \min(A_{loss}, \Delta A_{tot}) \end{aligned}$$

Let $a_p = \max(0, \Delta A_{tot} - A_{loss})$ which is the profit after covering the tracked loss. Then we have:

$$(A_{tot} + A_{loss}) \to (A_{tot} + A_{loss}) + a_p$$

(thus invariant (1) is preserved).

And when minting shares ((s_d, a_p)) for the dragon router one is making sure the following conversion rate is preserved (up to rounding errors, if we include the rounding errors then the conversion rate is (must be) improved):

$$\frac{A_{tot} + A_{loss}}{S_{tot}} = \frac{A_{tot} + A_{loss} + a_p}{S_{tot} + s_d}$$

This also implies that:

$$\frac{A_{tot}}{S_{tot}} \le \frac{A_{tot} + \Delta A_{tot}}{S_{tot} + s_d}$$

i.e., the conversion rate without considering the tracked loss gets improved if there was net profit $a_p > 0$.

$$\begin{aligned} \frac{A_{tot}}{S_{tot}} &= \frac{A_{tot} + A_{loss}}{S_{tot}} - \frac{A_{loss}}{S_{tot}} \\ &= \frac{A_{tot} + A_{loss} + a_p}{S_{tot} + s_d} - \frac{A_{loss}}{S_{tot}} \\ &= \frac{A_{tot} + a_p}{S_{tot} + s_d} - \left(\frac{A_{loss}}{S_{tot}} - \frac{A_{loss}}{S_{tot} + s_d} \right) \\ &\le \frac{A_{tot} + a_p}{S_{tot} + s_d} \\ &\le \frac{A_{tot} + \Delta A_{tot}}{S_{tot} + s_d} \end{aligned}$$

This contradicts the fact that if there are previously tracked loss and in the new report profit is reported that would cover the previously reported loss plus some extra net profit a_p then the conversion rate $\frac{A_{tot}}{S_{tot}}$ used for normal users improves. In this case:

$$(A_{tot}, A_{loss}) \to (A_{tot} + A_{loss} + a_p, 0)$$

And thus if the minted s_d correspoing to a_p with the conversion rate $\frac{A_{tot}+A_{loss}}{S_{tot}}$ are withdrawn it would be the same conversion rate that the user would use at this point since A_{loss} is 0 and $\frac{A_{tot}+A_{loss}}{S_{tot}} = \frac{A_{tot}+A_{loss}+a_p}{S_{tot}+s_d}$. This again points to the fact that during periods that there are uncovered tracked losses (ie $A_{loss} > 0$) interacting with the strategy as a user would potentially use undesired conversion rates.

1. If the user deposits before a period with loss and withdraws during the positive loss period, they would receive a lower conversion rate.
2. If the user deposits during a period with loss and withdraws after the positive loss period, they would receive a higher conversion rate.
3. Report loss without burning enabled:** In this flow we would have $\Delta A_{tot} \le 0$:

$$\begin{aligned} A_{tot} &\to A_{tot} + \Delta A_{tot} \\ A_{loss} &\to A_{loss} - \Delta A_{tot} \\ (A_{tot} + A_{loss}) &\to (A_{tot} + A_{loss}) \end{aligned}$$

(thus invariant (1) is preserved).

$\frac{A_{tot}+A_{loss}}{S_{tot}}$ would be conserved, but:

$$\frac{A_{tot}}{S_{tot}} \geq \frac{A_{tot}+\Delta A_{tot}}{S_{tot}}$$

i.e., the reported loss would be socialized among the share holders at that specific momment. Share holders can wait for the loss to be recovered before they decide to withdraw their shares.

6. Report loss with burning enabled: The goal should be to preserved invariant (1) just like case 5. More specifically:

$$(A_{tot}+A_{loss}) \to (A_{tot}+A_{loss})$$

Moreover just like the report profic cases, one might think that the following conversion rate should be conserved:

$$\frac{A_{tot}+A_{loss}}{S_{tot}}$$

The are a few issue in the codebase right now:

- The asset burnt amount is not calculated correctly.

  Let's analyse this further. We have (s_{burn} is `sharesBurned`):

$$s_{burn} = \begin{cases} x\,\text{min}(s_D, -\Delta A_{tot}) & S_{tot}=0 \\ 0 & \color{red}{A_{tot}=0} \\ \text{min}(s_D, \left\lceil \frac{(-\Delta A_{tot})S_{tot}}{A_{tot}+A_{loss}} \right\rceil) & \text{otherwise} \end{cases}$$

  In the above, for the case $S_{tot}=0$ should imply that $s_D=0$ so we can deduce:

$$s_{burn} = \begin{cases} 0 & S_{tot}=0 \\ 0 & \color{red}{A_{tot}=0} \\ \text{min}(s_D, \left\lceil \frac{(-\Delta A_{tot})S_{tot}}{A_{tot}+A_{loss}} \right\rceil) & \text{otherwise} \end{cases}$$

  And so the dragon router share and the total supply are updated as:

$$\begin{aligned} s_D &\to s_D - s_{burn} \\ S_{tot} &\to S_{tot} - s_{burn} \end{aligned}$$

  And a_{burn} `assetValueBurned` is calculated as:

$$a_{burn} = \begin{cases} s_{burn} & S_{tot}=s_{burn} \\ \left\lceil \frac{s_{burn}\cdot \color{red}{A_{tot}}}{S_{tot}-\color{red}{s_{burn}}} \right\rceil) & \text{otherwise} \end{cases}$$

  If $s_{burn}=0$ this is like case 5. since we can assume otherwise.

- 6.1 If $S_{tot}=s_{burn}=s_D$: Then $a_{burn}=S_{tot}$ and we would end up with:

$$\begin{aligned} A_{tot} &\to A_{tot}+\Delta A_{tot} \\ A_{loss} &\to A_{loss}-\Delta A_{tot}-S_{tot} \\ S_{tot} &\to 0 \end{aligned}$$

  This is an interesting state to end up with since there would be no share but with potentially A_{tot} and/or A_{loss} ending up non-zero. Also note that:

$$(A_{tot}+A_{loss}) \to (A_{tot}+A_{loss})\color{red}{-S_{tot}}$$

- 6.2 If $S_{tot} - s_{burn} > 0$: We need to make sure:

  $$\frac{A_{tot} + A_{loss}}{S_{tot}} = \frac{A_{tot}^{new} + A_{loss}^{new}}{S_{tot}^{new}}$$

  i.e.:

  $$\frac{A_{tot} + A_{loss}}{S_{tot}} = \frac{A_{tot} + A_{loss} - a_{burn}}{S_{tot} - s_{burn}}$$

  One can decompose the actions into two:

  1. Burn as much shares s_burn of the dragon router as possible such that it would be able to cover as much of the upcoming new loss from the report.
  2. Account for the new incoming loss from the report.

  Let's say we need to burn s_{burn} from the the dragon router shares so that the conversion rate including losses is preserved (this is incontrary to the `withdraw` route though):

  $$\frac{A_{tot} + A_{loss}}{S_{tot}} = \frac{A_{tot} + A_{loss} - a_{burn}}{S_{tot} - s_{burn}}$$

  and so a_{burn} should have been calculated as:

  $$a_{burn} = \left\lfloor \frac{A_{tot} + A_{loss}}{S_{tot}} \cdot s_{burn} \right\rfloor$$

  So in this step 1. we get:

  $$(A_{tot}, A_{loss}, S_{tot}) \to (A_{tot} - a_{burn}, A_{loss}, S_{tot} - s_{burn})$$

  Then for step 2. we perform the accounting for the new net loss:

  $$\begin{aligned}(A_{tot}, A_{loss}, S_{tot}) &\overset{(1)}{\to} (A_{tot} - a_{burn}, A_{loss}, S_{tot} - s_{burn}) \\ &\overset{(2)}{\to} (A_{tot} - a_{burn} - (-\Delta A_{tot} - a_{burn}), A_{loss} + (-\Delta A_{tot} - a_{burn}), S_{tot} - s_{burn}) \\ &= (A_{tot} + \Delta A_{tot}, A_{loss} + (-\Delta A_{tot} - a_{burn}), S_{tot} - s_{burn})\end{aligned}$$

  and so one can see that for both steps the conversion rate with the tracked loss included is conserved:

  $$\frac{A_{tot} + A_{loss}}{S_{tot}} = \frac{(A_{tot} + \Delta A_{tot}) + (A_{loss} + (-\Delta A_{tot} - a_{burn}))}{S_{tot} - s_{burn}}$$

  One might ask why the $\frac{A_{tot}+A_{loss}}{S_{tot}}$ was used for the share of the dragon router. Whereas if in step 1. a regular user would have withdrawn its shares then donated the assets back to the strategy one would have use the $\frac{A_{tot}}{S_{tot}}$ ratio. In this case we would end up with ($a_{burn} \approx \frac{A_{tot}}{S_{tot}} s_{burn}$):

  $$\begin{aligned}\frac{A_{tot} + A_{loss}}{S_{tot}} &\to \frac{A_{tot} + A_{loss} - a_{burn}}{S_{tot} - s_{burn}} \quad \text{(ratio improves)} \\ \frac{A_{tot}}{S_{tot}} &\to \frac{A_{tot} + \Delta A_{tot}}{S_{tot} - s_{burn}} = \frac{A_{tot} + \Delta A_{tot}}{S_{tot} - \frac{S_{tot}}{A_{tot}} \cdot a_{burn}}\end{aligned}$$

  Also note that with the same amount of a_{burn} using the other conversion rate the change in the normal conversion rate would have been:

  $$\frac{A_{tot}}{S_{tot}} \to \frac{A_{tot} + \Delta A_{tot}}{S_{tot} - \frac{S_{tot}}{A_{tot}+A_{loss}} \cdot a_{burn}} \leq \frac{A_{tot} + \Delta A_{tot}}{S_{tot} - \frac{S_{tot}}{A_{tot}} \cdot a_{burn}}$$

In summary the difference of the two approaches are:

Value	Note
$\frac{A_{tot}}{S_{tot}}$	2nd method gives a better final rate compared to the 1st method for the same amount of assets
$\frac{A_{tot}+A_{loss}}{S_{tot}}$	1st method preserves this value while the 2nd method improves it

Recommendation:

- 6. One needs to justify which method should be used to calculate (a_{burn}, s_{burn}), then use the correct conversion rates to derive the values. Overall the goal seems like to be either preserve $\frac{A_{tot}+A_{loss}}{S_{tot}}$ or imporove it (this is the case for 4. and 5.).

Octant: Fixed in commit 389a315b.

Cantina Managed: In the new implementation at commit 389a315b, the concept of loss recovery in YieldDonating is removed in favor of dragon router share burning on report. The invariant is roughly:

$$A_{tot} = \texttt{principals} + \texttt{profits} - \texttt{loss}$$

and thus in some cases where there is a loss and the loss cannot be covered by the profit shares of the dragon router, those losses would eat into the pricinpals. Therefore in such cases users might not be able to fully recover their principal.

`Deposit/mint/withdraw/redeem/report profit` would roughly preserve:

$$\frac{A_{tot}}{S_{tot}}$$

But if losses are reported this value could decrease. Moreover during `withdraw/redeem` unreported loses could be absorbed by the user.

Marked as acknowledged because in general not all principal could be recovered.

Footnote:

parameter	description
A_{tot}	`_strategyStorage().totalAssets` before calling the endpoint such as `report`
A_{tot}^{old}	`_strategyStorage().totalAssets` before calling the endpoint such as `report`
A_{tot}^{new}	`_strategyStorage().totalAssets` after calling the endpoint such as `report`
A_{tot}^{report}	the value returned by `IBaseStrategy(address(this)).harvestAndReport()`
ΔA_{tot}	$A_{tot}^{new} - A_{tot}^{old}$
A_{loss}	`_strategyStorage().lossAmount` before calling the endpoint such as `report`
A_{loss}^{old}	`_strategyStorage().lossAmount` before calling the endpoint such as `report`
A_{loss}^{new}	`_strategyStorage().lossAmount` after calling the endpoint such as `report`
S_{tot}	`_strategyStorage().totalSupply` before calling the endpoint such as `report`
S_{tot}^{old}	`_strategyStorage().totalSupply` before calling the endpoint such as `report`
S_{tot}^{new}	`_strategyStorage().lossAtotalSupplymount` after calling the endpoint such as `report`
s_D	shares of the dragon router in the strategy

3.2.3 Inconsistencies between `compoundRewards` and `claimAndStakeMore{OnBehalf}`

Severity: Medium Risk

Context: RegenStakerBase.sol#L518-L575

Description: Based on the implementation of `compoundRewards`, this endpoint is supposed to perform (when the stake and reward tokens are the same):

- Claim the rewards (following the original `claimReward` from the `Staker` contract).

- Stake the claimed rewards back into the contract (potentially following a mix of `stakeMore` and `stakeMoreOnBehalf`).

But currently there are the following inconsistensies.

1. `msg.sender` has been checked to see whether it is allowed to interact with this endpoint versus the `deposit.owner` In the `{permidAnd}Stake{OnBehalf}` and `{permitAnd}StakeMore{OnBehalf}` it is the `deposit.owner` that has been checked.
2. If `unclaimedAmount` is `0` the call would revert versus in `Staker._claimReward` there is no check for this revert.
3. `compoundRewards` returns `0` if `unclaimedAmount < fee`. But in `Staker._claimReward` one would get an arithmetic underflow revert.
4. If `compoundedAmount == 0` (ie `unclaimedAmount == fee`) the flow continues in `compoundRewards` even though in `Staker._claimReward` one returns early without distributing the fees to `claimFeeParameters.feeCollector` since there would be no net rewards to be sent out.
5. `deposit.scaledUnclaimedRewardCheckpoint` is set to `0` at the end in `compoundRewards` whereas in `Staker._claimReward` one keeps track of the sub-wei dust:

```
deposit.scaledUnclaimedRewardCheckpoint =
    deposit.scaledUnclaimedRewardCheckpoint - (unclaimedAmount * SCALE_FACTOR);
```

6. `RewardClaimed` and `StakeDeposited` are missing but the same information can be gathered from `RewardCompounded` event (almost). This is something to note.

```
emit RewardClaimed(_depositId, _claimer, _payout, _newEarningPower);
emit StakeDeposited(deposit.owner, _depositId, _amount, _newBalance, _newEarningPower);

// vs

emit RewardCompounded(_depositId, msg.sender, compoundedAmount, newBalance, newEarningPower);
```

   - `RewardClaimed._payout == StakeDeposited._amount == RewardCompounded.compoundedAmount`.
   - `RewardClaimed._claimer == RewardCompounded.msg.sender`.
   - `RewardClaimed._newEarningPower` might not necesssarily be the same as `StakeDeposited._newEarningPower` or `RewardCompounded.newEarningPower`.
   - `StakeDeposited._newBalance == RewardCompounded.newBalance`.
   - `RewardCompounded` does not emit `deposit.owner`.

7. Regarding `surrogates` the following:

```
DelegationSurrogate _surrogate = surrogates(deposit.delegatee);
// ...
_stakeTokenSafeTransferFrom(deposit.owner, address(_surrogate), _amount)
```

   has been replaced by the `_transferForCompound(deposit.delegatee, compoundedAmount)` hook.

8. If the `msg.sender` is the `deposit.claimer`, in this route this entity would also be allowed to stake more into the `deposit` which might not be a role anticipated by the `deposit.owner`.

Recommendation:

1. Make sure to perform the allow list check for `deposit.owner`.
2. This is fine as long as the devs are aware and it is documented.
3. Perhaps should be documented.
4. This edge case needs to be fixed and one should return early and not perform any further checks or operations.
5. Make sure to keep track of the sub-wei dust.
6. Make sure `RewardCompounded` also emits `deposit.owner` which could be different than the `msg.sender` (the claimer and staker on behalf).

7. Perhaps should be documented as this is different than the `...StakeMore...` route which uses the surrogates.
8. Perhaps signature should be passed to this endpoint to make sure `deposit.owner` has authorised `msg.sender` (when it is `deposit.claimer`) to stake more on its behalf. If not at least this should be documented.

Octant: Fixed in commit e017ddf7.

Cantina Managed: Fix verified.

3.2.4 Decimal difference is not handled inside `calculateOptimalAlpha`

Severity: Medium Risk

Context: ProperQF.sol#L263-L268

Description: The calculateOptimalAlpha compares the possible shares amount with the total funds available to determine the extent of quadratic funding. But since the voting power and hence the quadratic and linear sum is always scaled to 18 decimals, the condition of `totalAssetsAvailable <= linearSum` will almost always be true for tokens with lower number of decimals causing these to choose pure linear funding.

```
/// @notice Calculate optimal alpha for 1:1 shares-to-assets ratio given fixed matching pool amount
/// @param matchingPoolAmount Fixed amount of matching funds available
/// @param totalUserDeposits Total user deposits in the mechanism
/// @return optimalAlphaNumerator Calculated alpha numerator
/// @return optimalAlphaDenominator Calculated alpha denominator
/// @dev Uses current mechanism state for quadratic and linear sums
function calculateOptimalAlpha(
    uint256 matchingPoolAmount,
    uint256 totalUserDeposits
) external view returns (uint256 optimalAlphaNumerator, uint256 optimalAlphaDenominator) {
    return _calculateOptimalAlpha(matchingPoolAmount, totalQuadraticSum(), totalLinearSum(), totalUserDeposits);
}

    function _calculateOptimalAlpha(
    uint256 matchingPoolAmount,
    uint256 quadraticSum,
    uint256 linearSum,
    uint256 totalUserDeposits
) internal pure returns (uint256 optimalAlphaNumerator, uint256 optimalAlphaDenominator) {
    // Handle edge cases
    if (quadraticSum <= linearSum) {
        // No quadratic funding benefit, set alpha to 0
        optimalAlphaNumerator = 0;
        optimalAlphaDenominator = 1;
        return (optimalAlphaNumerator, optimalAlphaDenominator);
    }

    uint256 totalAssetsAvailable = totalUserDeposits + matchingPoolAmount;
    uint256 quadraticAdvantage = quadraticSum - linearSum;

    // We want:   × quadraticSum + (1- ) × linearSum = totalAssetsAvailable
    // Solving for  :    × (quadraticSum - linearSum) = totalAssetsAvailable - linearSum
    // Therefore:   = (totalAssetsAvailable - linearSum) / (quadraticSum - linearSum)

    if (totalAssetsAvailable <= linearSum) {
        // Not enough assets even for linear funding, set alpha to 0
        optimalAlphaNumerator = 0;
        optimalAlphaDenominator = 1;
```

Recommendation: Scale the token amounts to 18 decimals as well.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

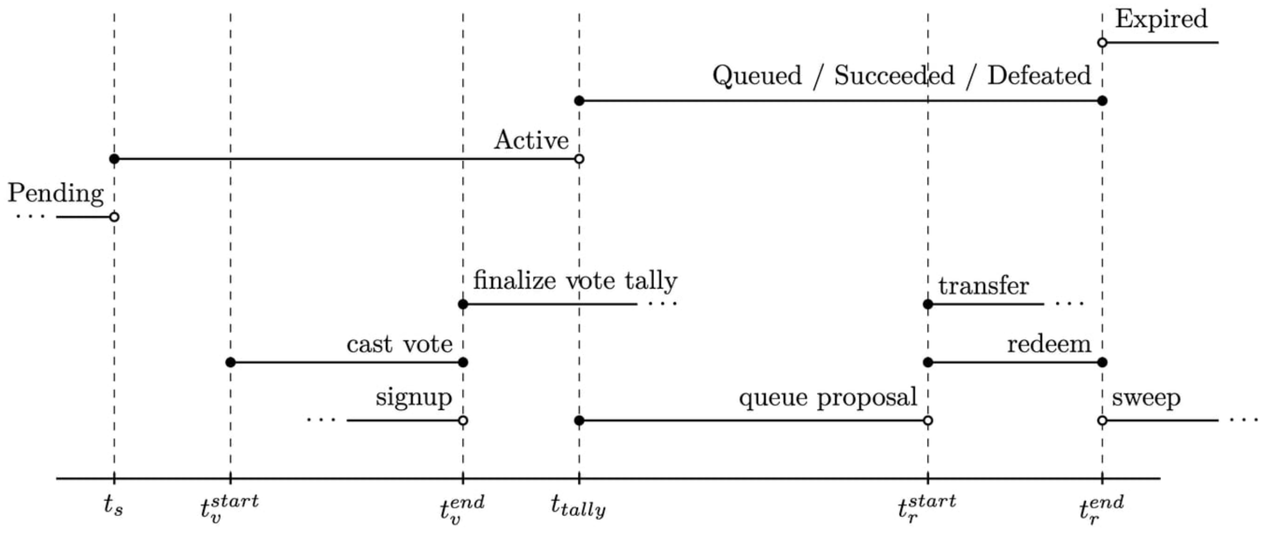
3.2.5 Mechanism timeline and state machine

Severity: Medium Risk

Context: TokenizedAllocationMechanism.sol#L60

Description:

- Timeline: Currently the mechanism timeline looks like this:



parameter	description
t_s	`startBlock`
t_v^{start}	`startBlock + votingDelay`
t_v^{end}	`startBlock + votingDelay votingPeriod`
t_{tally}	the `block.timestamp` that `finalizeVoteTally()` was called by the `owner`
t_r^{start}	`globalRedemptionStart`
t_r^{end}	`globalRedemptionStart + gracePeriod`

One can notice a few inconsistencies:

1. Mix use of `block.number` for pre-tally events and `block.timestamp` for post-tally events.
2. Event time/block points can be cached in storage.
3. The timeline that a registered user could cast a vote and the owner calling `finalizeVoteTally` could have an intersection point at t_v^{end} as show in the diagram.
4. One cannot `signup` at t_v^{end} but can cast a vote if registered prior, ie registering and casting a vote in the same block at this edge timestamp is not possible.
5. After the votes are tallied, canceling a succeeded proposal can be frontrun by a call to `queueProposal`.
6. `transfer...` endpoints do not have right boundaries. Proposal receivers could transfer shares after a proposal has expired. Although those shares would not be redeemable.
7. There are no time-based restrictions for calling `propose(...)`, one can (not considering the hook based checks) create proposals after $t_s, t_{tally}, t_r^{end}, \cdots$. These proposals would end up being useless but could create confusing storage paramters or even event emitions. The same applies to canceling these non-useful proposals. One can for example create instantly defeatable or cancelable proposals.
8. Somewhere it was mentioned that users should only be able to register/sign up during the $[t_-{v}^{start}, t_v^{end}]. But the lower bound is not checked.

- State Machine: Let's look at the state machine:

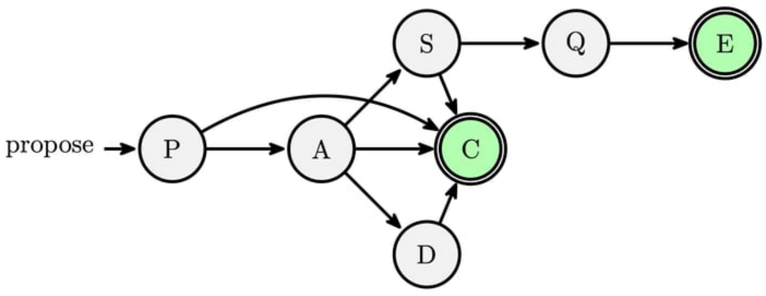


Figure 1: Current `Proposal` finite automata

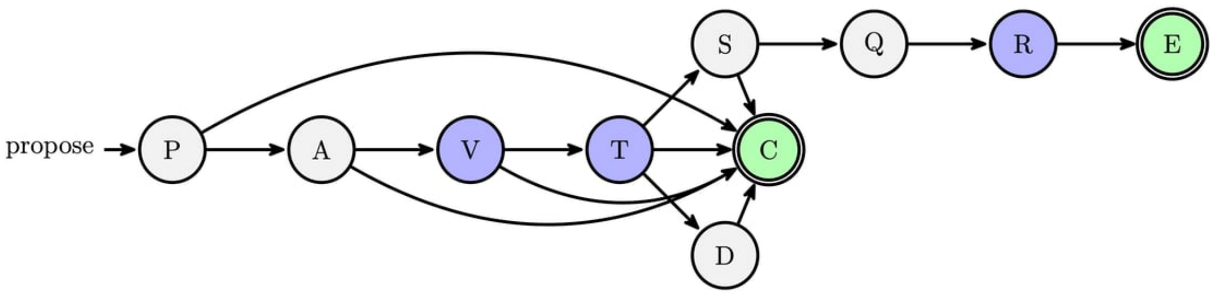


Figure 2: Suggested `Proposal` finite automata

state	description
P	Pending
A	Active
C	Canceled
S	Succeeded
D	Defeated
Q	Queued
E	Expired
V	Votable
T	Tallyable
R	Redeemable (can be transferred also)

In the suggested figure, one is introducing 3 extra states V, T, R to have a more granular state status.

Recommendation:

- Timeline:
  1. Might be best to use either `block.number` or `block.timestamp`.
  2. To save storage reads from the event time points which are used across different checks, it might make sense to cache these values instead of summing the initial point plus the deltas.
  3. It might be best to disallow `finalizeVoteTally` to be called at t_v^{end}.
  4. It might make sense to also allow registering at this point.
  5. Perhaps can be documented.
  6. It would be best to revert transfer calls for when the system is in the expired mode.
  7. It would make sense to at least restrict the call to `propose(...)` to $(-\infty, t_v^{end})$.
  8. If time check is required for `signup...` to make sure users cannot call this endpoint before t_v^{start}, it needs to be implemented.

- State Machine: The design can be refined futher plus define state/status of the mechanism as a whole (global system state). The global system states would be mostly time-based.

`GlobalTimelinePhase` state	description
`Pending`	$(-\infty, t_s)$
`Delay`	$[t_s, t_v^{start})$
`CanCastVote`	$[t_v^{start}, t_v^{end}]$
`CanFinalizeVoteTally`	$(t_v^{end}, \cdots)$ till `finalizeVoteTally` is called.
`CanQueueProposal`	$[t_{tally}, t_r^{start})$
`CanRedeem`	$[t_r^{start}, t_r^{end}]$
`CanSweep`	$[t_r^{end}, \infty)$

```
enum GlobalTimelinePhase {
    Pending,
    Delay,
    CanCastVote,
    CanFinalizeVoteTally,
    CanQueueProposal,
    CanRedeem,
    CanSweep
}

function _globalTimelinePhase() internal view returns (GlobalTimelinePhase globalTimelinePhase) {
    // based on the timeframe and the status of
    // `tallyFinalized` returns the correct phase
}
```

and

```
enum ProposalState {
    DoesNotExist,    // new
    Pending,
    Active,          // or Delayed
    Votable,         // new
    Tallyable,       // new
    Canceled,
    Defeated,
    Succeeded,
    Queued,
    Redeemable,      // new
    Expired
}

function state(uint256 pid) external view onlyInitialized returns (ProposalState) {
    // change accordingly and incorporate the
    // `GlobalTimelinePhase` by calling `_globalTimelinePhase(...)`
}
```

For example:

```
function state(uint256 pid) external view onlyInitialized returns (ProposalState state) {
    state = _state(pid)
    if (state == ProposalState.DoesNotExist) revert InvalidProposal(pid);
}

/// @dev Internal state computation for a proposal
function _state(uint256 pid) internal view returns (ProposalState) {
    if (IBaseAllocationStrategy(address(this)).validateProposalHook(pid)) {
        return ProposalState.DoesNotExist;
    }

    AllocationStorage storage s = _getStorage();
    Proposal storage p = s.proposals[pid];

    if (p.canceled) {
        return ProposalState.Canceled;
    }

    GlobalTimelinePhase globalTimelinePhase = _globalTimelinePhase();
    if (globalTimelinePhase == GlobalTimelinePhase.Pending) {
        return ProposalState.Pending;
    } else if (globalTimelinePhase == GlobalTimelinePhase.Delay) {
        return ProposalState.Active;
    } else if (globalTimelinePhase == GlobalTimelinePhase.CanCastVote) {
        return ProposalState.Votable;
    } else if (globalTimelinePhase == GlobalTimelinePhase.CanFinalizeVoteTally) {
        return ProposalState.Tallyable;
    } else if (globalTimelinePhase == GlobalTimelinePhase.CanQueueProposal) {
        if (!IBaseAllocationStrategy(address(this)).hasQuorumHook(pid)) {
            return ProposalState.Defeated;
        } else if (s.proposalShares[pid] == 0) {
            return ProposalState.Succeeded;
        }

        return ProposalState.Queued;
    } else if (globalTimelinePhase == GlobalTimelinePhase.CanRedeem) {
        return ProposalState.Redeemable;
    } else if (globalTimelinePhase == GlobalTimelinePhase.CanSweep) {
        return ProposalState.Expired;
    }

    // should be unreachable
    revert();
}
```

In the above rough implementation `_state(uint256 pid)` returns a finer state compared to `_globalTimelinePhase()`. Moreover it fixes some edge cases where when `globalTimelinePhase == GlobalTimelinePhase.CanSweep` the old returned proposal state might have been in `Succeeded` state.

Then the call restrictions to the end points can be more simplified. Instead of the inline checks in each endpoint, one might need to fetch `_globalTimelinePhase(...)` and `state(pid)`. This would help refactor the global timeline phase and proposal states into 2 atomic routes. Using the above new rough implementation of `_state(uint256 pid)` one can refactor many of the endpoint checks into querying the state of a proposal.

endpoint	GlobalTimelinePhase	ProposalState (`pid`)
`propose(...)`	?	?
`signup...(...)`	?	?
`castVote...(...)`	`CanCastVote`	`Votable`
`finalizeVoteTally()`	`CanFinalizeVoteTally`	-
`queueProposal(...)`	`CanQueueProposal`	`Succeeded`
`cancelProposal(...)`	?	If not in `Queued`, `Redeemable`, `Expired`
`sweep(...)`	`CanSweep`	-
`redeem(...)`	`CanRedeem`	-
`maxRedeem(...)`	`CanRedeem`	-
`previewRedeem(...)`	`CanRedeem`	-
`balanceOf(...)`	`CanRedeem`	-
`convertToAssets(...)`	?	-
`convertToShares(...)`	?	-
`totalSupply()`	`CanRedeem`	-
`transfer...(...)`	`CanRedeem`	-

Restriction of the endpoints with `?` needs to be further analysed as currently they are missing. The other restriction need to be set as suggested in the above table.

Octant: Fixed in commits:

- 65b17c54.
- 0c662f29.
- 5b2affe4.

Cantina Managed: Fixed in the series of commits:

- 65b17c54.
- 0c662f29.
- 5b2affe4.

Important to view the diffs between the commit and also the commit reviewed in the scope of this review.

item	status	description	note
1	✓		
2	$\pm$	Event time/block points can be cached in storage.	X timestamps are cached in storage, but during event or error emissions values are read multiple times
3	✓		
4	✓		
5	✓	After the votes are tallied, canceling a succeeded proposal can be frontrun by a call to `queueProposal`.	canceling a proposal can now only happen before finalise tally
6	✓	`transfer...` timestamp boundaries	
7	✓	There are no time-based restrictions for calling `propose(...)`, ...	ok
8	ok	... users should only be able to register/sign up during the $[t_v^{start}, t_v^{end}]$. But the lower bound is not checked.	

Other notes/issues ...

- ✓ A succeeded proposal that is not queued but expired can be canceled: fixed.
- ✓ `redeem(...)` timestamp boundary check.
- ✓ `maxRedeem(...)` timestamp boundary check.
- ✓ `previewRedeem(...)` timestamp boundary check.
- ok: `balanceOf(...)` not required based on expected spec. Accepted.
- ok: `totalSupply()` not required based on expected spec. Accepted.
- ok: `convertToAssets(...)` not required based on expected spec. Accepted.
- ok: `convertToShares(...)` not required based on expected spec. Accepted.

✓ Also in general check like below which is performed in `previewRedeem` ...

3.2.6 Quadratic Voting Mechanism

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description/Recommendation:

parameter	description
M_{QV}	Quadratic Voting Mechanism
P	The set of proposals
P_c	The set of canceled proposals
P_q	The set of proposals that have been queued in their lifetime
P_a	The set of proposals that have not been canceled or queued yet
U_v	the set of users that can cast a vote
U_r	the set of users who hold shares
T	`asset` which in the context of Regen staker would be the reward token
po_u^{og}	initial voting power of a registered user u

po_u^r	the current remaining voting power of a registered user u

Borrowing a modified notation from the Liberal Radicalism paper, we have:

$$\Phi : \mathbb{R}^{P \times U_v} \to \mathbb{R}^P$$

Φ is the general voting allocation mechanism. $\mathbb{R}^{P \times U_v}$ is the vector space of all possible contribution vectors c and $\mathbb{R}^P$ is the space of voting levels for all proposals. Examples of the mechanisms used in the Octant implementation are:

$$\begin{aligned}
\Phi^{CAP}(c) &= (\sum_{u \in U_v} |c_{p,u}|)_{p \in P} &&= (\|c_{p,\cdot}\|_1)_{p \in P} \\
\Phi^{LR}(c) &= ((\sum_{u \in U_v} |c_{p,u}|^{1/2})^2)_{p \in P} &&= ((\|c_{p,\cdot}\|_{1/2}))_{p \in P} \\
\Phi^{CLR}_\alpha(c) &= \alpha \cdot \Phi^{LR}(c) + (1-\alpha) \cdot \Phi^{CAP}(c)
\end{aligned}$$

Note that how $\|x\|_p$ p-norms are used in the above definitions except $\|x\|_{1/2}$ is not a norm. And we have for every $x \in \mathbb{R}^{U_v}$:

$$\|x\|_1 \leq \|x\|_{1/2} \leq |U_v| \cdot \|x\|_1$$

Thus for the component of the above allocation functions we have:

$$\begin{aligned}
\Phi^{CAP}(c)_p \leq \Phi^{CLR}_\alpha(c)_p &\leq \Phi^{CAP}(c)_p \cdot (\alpha|U_v| + (1-\alpha)) \\
(\alpha + (1-\alpha)\frac{1}{|U_v|}) \cdot \Phi^{LR}(c)_p \leq \Phi^{CLR}_\alpha(c)_p &\leq \Phi^{LR}(c)_p
\end{aligned}$$

Some of the notations below are borrowed from ”Mechanism timeline and state machine” For a proposal $p \in P$ the value $\Phi^{CLR}_\alpha(c)_p$ increases as more registered users cast some vote for this proposal (currently users can cast only once) in the interval $[t_v^{start}, t_v^{end}]$. Using the weights provided during the vote casting $w_{p,u}$ we get:

$$\begin{aligned}
\Phi^{CAP}(c)_p &= \sum_{u \in U_v} w_{p,u}^2 \\
\Phi^{LR}(c)_p &= (\sum_{u \in U_v} w_{p,u})^2 \\
\Phi^{CLR}_\alpha(c)_p &= \alpha \cdot \Phi^{LR}(c)_p + (1-\alpha) \cdot \Phi^{CAP}(c)_p = \alpha \cdot \left(\sum_{u \in U_v} w_{p,u}\right)^2 + (1-\alpha) \cdot \left(\sum_{u \in U_v} w_{p,u}^2\right)
\end{aligned}$$

Then when `finalizeVoteTally()` is called afterwards by the owner O, the snapshot of the total assets (asset balance of the mechanism M_{QV} at that time) is recorded in storage to be used later on. Let's call this snapshot value A_{tot}^{tally}. At this point the values of $\Phi^{CLR}_\alpha(c)_p$ would also freeze since users cannot cast votes anymore. If a proposal p does not get canceled and also:

$$s_q \leq \Phi^{CLR}_\alpha(c)_p$$

satisfies the quorum (ie the value above is at least the minimum amount required to mint shares s_q) then one can queue this proposal and the shares minted for it would be exactly $\Phi^{CLR}_\alpha(c)_p$ which gets minted for the proposal receiver $u_{p,r}$. This user can after minting of shares to transfer portions of its shares to other users (currently transferring shares is only timelocked below and is NOT bounded to the redeem period).

1. Voting Powers: When a user signups/registers with deposit amount a_u. It's original voting power would be:

$$po_u^{og} = \left\lfloor a_u \cdot 10^{18-T.dec()} \right\rfloor$$

then as it casts votes for different porjects we would get:

$$\left\lfloor a_u \cdot 10^{18-T.dec()} \right\rfloor = po_u^{og} = \left(\sum_{p \in P} w_{p,u}^2 \right) + p_u^r$$

If we sum over all users $u \in U_v$ we get:

$$\left\lfloor A_{tot}^{deposit} \cdot 10^{18-T.dec()} \right\rfloor \geq \sum_{u \in U_v} \left\lfloor a_u \cdot 10^{18-T.dec()} \right\rfloor = \sum_{u \in U_v} \left(\sum_{p \in P} w_{p,u}^2 \right) + \sum_{u \in U_v} p_u^r = \|\Phi^{CAP}(c)\|_1 + \sum_{u \in U_v} p_u^r$$

Note that $A_{tot}^{deposit} = \sum_{u \in U_v} a_u$.

- Splitting one's deposit amounts into multiple separate users would give potentially less voting powers overall due to voting powers getting rounded for assets with decimals greater than 18.
- The weight squared $w_{p,u}$ might make it so that one would not be able to use all its voting power due to the original voting power might not be the exact sum of the squares desired. To avoid dust one needs to pre-calculate the deposit amount based on the amount of weights it wants to use for different proposals so that the value of p_u^r would be as small as possible at the end of voting period.
- Deposit amounts are normalised so that po_u values would have 18 decimal precisions. That would mean that either the weights would have 9 decimal precisions or if one would like to use 18 decimal precisions for the weights as well then their multiplications/squaring should use `mulWadDown` in calculating $\Phi_\alpha^{CLR}(c)$ and `mulWadUp` for updating po_u values.
- Unused voting powers cannot be withdrawn.

2. Optimal α: The `owner` O can find and set optimal α so that:

$$\|\Phi_\alpha^{CLR}(c)\|_1 = \left\lfloor A_{pool} \cdot 10^{18-T.dec()} \right\rfloor + \left\lfloor A_{tot}^{deposit} \cdot 10^{18-T.dec()} \right\rfloor \leq \left\lfloor (A_{pool} + A_{tot}^{deposit}) \cdot 10^{18-T.dec()} \right\rfloor$$

A_{pool} is the pool matched assets. $A_{tot}^{deposit}$ is the total amount of deposited assets. Currently it is not clear whether the values provided to `_calculateOptimalAlpha` are normalised by $10^{18-T.dec()}$ or not (see "None").

- Solving for the optimal α tries to keep the price per share around $10^{T.dec()-18}$.
- Canceled proposal voting levels are still considered in the above equation, $\sum_{p \in P_c} \Phi_\alpha^{CLR}(c)_p$.
- Solving the above equation assumes that the $\|\Phi_\alpha^{CLR}(c)\|_1$ value is fixed. But this might not be true in general. It would be best to solve this equation and set α during $[t_{tally}, t_v^{start})$. Since the value $\|\Phi_\alpha^{CLR}(c)\|_1$ would be fixed. Moreover, it might also makes sense to introduce another timestamp $t_q^{end} \in [t_{tally}, t_v^{start})$ such that ONLY during $[t_{tally}, t_q^{end})$ proposals can be queued and not after. Then during $[t_q^{end}, t_v^{start})$, the owner can calculate the proper value for α, since at this time the set of queued proposals would be fixed P_q and could not grow anymore. For the issue of not including the proposals that are not queued in the above equation when solving for α see the next subsection.

3. Minted Proposal Shares:

$$\|\Phi_\alpha^{CLR}(c)\|_1 \geq \sum_{p \in P_q} \Phi_\alpha^{CLR}(c)_p = \sum_{p \in P_q} s_p = \sum_{u \in U_r} s_u = S_{tot} \geq |P_q| \cdot s_q$$

- The matched pool and deposit amounts allocated to canceled proposals or proposals that did not meet the quorum (their voting level is less than s_q) would get allocated to the queued proposals. That is roughly equivalent to:

$$\left(\sum_{p \in P \setminus P_q} \Phi_\alpha^{CLR}(c)_p \right) \cdot 10^{T.dec()-18}$$

Currently proposing a proposal is safeguarded by the keeper K or the management M. But the above creates an incentive to cancel proposals so that more funds would get allocated to the ones that did not get canceled. In general it would make sense for the canceled proposals the voting powers would get sent back to the users and potentially those users would be able to withdraw their unused voting powers.

4. Effect of splitting a proposal into sub-proposals and merging sub-proposals into a bigger proposal:** Let's assume a proposal p can be split into n different sub-proposals $\{p_1, \cdots, p_n\}$ or vice versa these sub-proposals can be merged into a bigger proposal p. For each user, let's assume the sum of its contributions to the main proposal would stay the same, i.e:

$$c_{p,u} = \sum_{i=1}^{n} c_{p_i,u}$$

or in other words:

$$w_{p,u}^2 = \sum_{i=1}^{n} w_{p_i,u}^2$$

and let's assume all proposals get queued. Let's compare the shares minted:

$$\Phi_\alpha^{CLR}(c)_p = \alpha \cdot \left(\sum_{u \in U_v} w_{p,u} \right)^2 + (1-\alpha) \cdot \left(\sum_{u \in U_v} w_{p,u}^2 \right)$$

$$\sum_{i=1}^{n} \Phi_\alpha^{CLR}(c)_{p_i} = \alpha \cdot \sum_{i=1}^{n} \left(\sum_{u \in U_v} w_{p_i,u} \right)^2 + (1-\alpha) \cdot \sum_{i=1}^{n} \left(\sum_{u \in U_v} w_{p_i,u}^2 \right)$$

We have:

$$(1-\alpha) \cdot \left(\sum_{u \in U_v} w_{p,u}^2 \right) = (1-\alpha) \cdot \sum_{i=1}^{n} \left(\sum_{u \in U_v} w_{p_i,u}^2 \right)$$

i.e., the linear voting levels match up:

$$\Phi^{CAP}(c)_p = \sum_{i=1}^{n} \Phi^{CAP}(c)_{p_i}$$

But for the quadratic levels we have:

$$\left(\sum_{u \in U_v} w_{p,u} \right)^2 = \left(\sum_{u \in U_v} \left(\sum_{i=1}^{n} w_{p_i,u}^2 \right)^{1/2} \right)^2 \geq \sum_{i=1}^{n} \left(\sum_{u \in U_v} w_{p_i,u} \right)^2$$

Thus:

$$\Phi^{LR}(c)_p \geq \sum_{i=1}^{n} \Phi^{LR}(c)_{p_i}$$

and equality only happens if there exist a vector $w_\star \in \mathbb{R}^P$ and scalars $\lambda_u \in \mathbb{R}_{\geq 0}$ for all $u \in U_v$ such that:

$$(w_{p_i,u})_{i=1}^{n} = \lambda_u w_\star$$

i.e., for each sub-proposal the vector of weights for all users sit on the same ray in $\mathbb{R}^P$. Thus we have:

$$\Phi_\alpha^{CLR}(c)_p \geq \sum_{i=1}^{n} \Phi_\alpha^{CLR}(c)_{p_i}$$

So:

- Merging sub-proposals into a bigger proposal would increase the minted share.
- Splitting a proposal into multiple sub-proposals would decrease the minted share.

And perhaps that is why proposal recipeint can be used only once. But again this can be gamed with multiple recipients colluding.

5. Effect of splitting a vote to proposal into multiple vote casts: Currently, it is not allowed for a user u to cast votes to a proposal this user has already casted. But, let's assume one keep cast votes for the same proposal multiple times. Let's assume a user wants to cast in total $c_{p,u}$ contributions and decides to divide this into multiple cast calls with contributions $c_{p,u,i}$ where $i \in \{1, \cdots, n\}$:

$$c_{p,u} = \sum_{i=1}^{n} c_{p,u,i}$$

or using the languare of weights:

$$w_{p,u}^2 = \sum_{i=1}^{n} w_{p,u,i}^2$$

Let's calculate the effect on the to-be-minted shares (or voting levels) for the proposal p. Based on above it is obvious that the linear term $\Phi^{CAP}(c)_p$ would be the same for both cases. For the $\Phi^{LR}(c)_p$ term we have:

$$\begin{aligned}\left(\left(\sum_{v \in U_v \setminus \{u\}} w_{p,v}\right) + w_{p,u}\right)^2 &= \left(\left(\sum_{v \in U_v \setminus \{u\}} w_{p,v}\right) + \left(\sum_{i=1}^{n} w_{p,u,i}^2\right)^{1/2}\right)^2 \\ &\leq \left(\left(\sum_{v \in U_v \setminus \{u\}} w_{p,v}\right) + \left(\sum_{i=1}^{n} w_{p,u,i}\right)\right)^2\end{aligned}$$

Thus:

$$\Phi_\alpha^{CLR}(c)_p \leq \Phi_\alpha^{CLR}(c)_{p,split}$$

Thus, spliting one's vote to a proposal p into multiple cast vote calls but with the same total contribution towards this proposal would increase the to-be-minted shares value $\Phi_\alpha^{CLR}(c)_p$ of the proposal if $(w_{p,u,i})_{i=1}^n \notin \mathbb{R}_{\geq 0}(1)_{i=1}^n$ (ie, uneven spliting of the total contribution). Therefore it makes sense to not allow casting for the same proposal multiple times. Although this can still be gamed by an entity to signup as mutiple users to split its total voting powers among those colluded users.

Octant: Acknowledged. Nine decimal shares precision secures WETH at \$1M down to one tenth of a cent. Design is intended regarding voting power NOT being refundable. Sybil attacks are out of scope of the base mechanism design and should be handled by custom hook implementations, smaller concerns addressed.

Cantina Managed: Acknowledged. It would be helpful to document those assumptions and design choices.

3.2.7 Incorrect conversion of shares to assets or vice versa when total supply is zero

Severity: Medium Risk

Context: TokenizedAllocationMechanism.sol#L1085-L1112

Description: The mechanism shares have 18 decimals precision, but the deposit amounts/assets have `asset.decimals()` $T.dec()$ precisions. When `_totalSupply(S)` is 0, both `_convertToShares` and `_convertToAssets` return incorrect values if $T.dec() \neq 18$.

Recommendation: Make sure to incorporate the decimal precision differences when calculating the final value when `_totalSupply(S)` $S_{tot} = 0$:

$$\texttt{\_convertToShares}(a) = \left\lfloor a \cdot 10^{18-T.dec()} \right\rfloor$$
$$\texttt{\_convertToAssets}(s) = \left\lfloor s \cdot 10^{T.dec()-18} \right\rfloor$$

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.2.8 Different decimal precision parameters are arithmetically combined

Severity: Medium Risk

Context: BaseYieldSkimmingHealthCheck.sol#L151-L166, YieldSkimmingTokenizedStrategy.sol#L149-L159

Description:

- `currentExchangeRate` r_{curr} has 18 decimal precision.
- `newExchangeRate` r_{new} has `decimalsOfExchangeRate()` decimal precision d which might not be 18.

Thus one cannot add or subtract these values until they are normalised into the same decimal precision.

Recommendation: Make sure r_{curr} and r_{new} have the same decimal precision before combined and compared. One can normalise r_{new} to 18 decimals perhaps.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. The recommendation has been applied and the precisions have been normalised to 27 decimals.

3.2.9 `_emergencyWithdraw` should not use `maxWithdraw`

Severity: Medium Risk

Context: MorphoCompounderStrategy.sol#L62-L66

Description: In `MorphoCompounderStrategy` when `_emergencyWithdraw` is called, one calls in `IERC4626.maxWithdraw(...)` which in general it is supposed to underestimate the amount of withdrawable assets. In theory one might be able to withdraw more. In some scenarios it might even return `0` even though one could still be able to withdraw.

Recommendation: It would be best to avoid taking the minimum and blocking `_emergencyWithdraw` from full potential withdrawals:

```
function _emergencyWithdraw(uint256 _amount) internal override {
    _freeFunds(_amount);
}
```

The above change would introduce a race condition between admin emergency withdrawal calls and user `withdraw` calls. But perhaps using private `rpc` endpoints could avoid such scenarios where the front-running user calls could potentially create scenarios where the `_amount` provided in `_emergencyWithdraw` would not be able to be fully withdrawn and thus the call would `revert`.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

Footnote: The same suggestion could apply to the `withdraw` and `redeem` flows where the call to `IBaseStrategy(address(this)).availableWithdrawLimit(owner)` would query `IERC4626(compounderVault).maxWithdraw(address(this))` which gives a lower estimate for a potential amount that could be withdrawn.

This issue comes from the architectural design of the Yearn V3 vaults and is not specific to this codebase.

3.2.10 The zero `PPS` branch should also include losses

Severity: Medium Risk

Context: TokenizedStrategy.sol#L803-L807

Description: In `_convertToSharesWithLoss` there is a branch where one returns a `0` price per share:

```
uint256 totalAssets_ = _totalAssets(S);
// If assets are 0 but supply is not PPS = 0.
if (totalAssets_ == 0) return 0;
```

Recommendation: The condition should also include `S.lossAmount` as in the general settings that value is also used in the denominator:

```
if (totalAssets_ + S.lossAmount == 0) return 0;
```

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. The corresponding function and loss accounting has been removed and thus the issue does not apply anymore.

Footnote: Related to "`YieldDonatingTokenizedStrategy` invariants"

3.2.11 Tokenized Vault's `decimals()` endpoint returns the underlying assets decimals

Severity: Medium Risk

Context: TokenizedAllocationMechanism.sol#L1058-L1060

Description: Tokenized Vault's `decimals()` endpoint returns the underlying assets decimals:

```
s.decimals = ERC20(address(_asset)).decimals();
```

In the case the underlying asset's decimals is not 18 the returned value by `decimals()` would be incorrect.

Recommendation: Make sure to return 18.

Octant: Fixed in commit 601ff671.

Cantina Managed: Fix verified.

3.2.12 `YieldSkimmingTokenizedStrategy` Changes

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The below issues are present in `a5401a0d814d09ed71f9c931d1f5c435e9786bd2` for the `YieldSkimmingTokenizedStrategy` contract:

1. Handling edges cases in `_convertToShares` and `_convertToAssets` are omitted. For example when total supply, total asset or current conversion rate is zero.
2. `TokenizedStrategy._withdraw` has a `virtual` tag.
3. `YieldSkimmingTokenizedStrategy._performWithdrawal` has the same implementation as `TokenizedStrategy._withdraw` with only some comment modification.
4. `redeem` / `withdraw`: resetting the debt parameters are only performed in the case the `owner` is not `S.dragonRouter`.
5. `redeem` / `withdraw` / `transfer{From}`: checking vault solvency is missing at the end. This is only checked at the beginning of these routes.
6. `S.dragonRouter = S.pendingDragonRouter;` can be optimised.
7. `emit UpdateDragonRouter(S.dragonRouter);` can be optimised.
8. `uint256 totalAssets = S.totalAssets;` can be optimised.

9. `max{Deposit,Mint,Withdraw,Redeem}` might return incorrect values for dragon router.
10. `YS.totalUserDebtInAssetValue` should be the same as the balance of the current dragon router.
11. `YS.totalUserDebtInAssetValue` should be the total supply minus the balance of the current dragon router (ie the sum of balances of all users except the current dragon router).

Recommendation:

1. Make sure to handle the edge cases separately. For example one should follow the patterns from `TokenizedStrategy` when the strategy is not solvent (one can call the `super` function) and when the vault is solvent but the current rate is 0 a pattern similiar to `_handleDragonLossProtection` should be used for conversion.
2. `TokenizedStrategy._withdraw` is not going be overriden the `virtual` tag can be removed.
3. Instances of `YieldSkimmingTokenizedStrategy._performWithdrawal` can be replaced by calls to `_withdraw`.
4. Make sure to reset the debt parameters out of the `if/else` branch at the end if the total supply is zero.
5. Make sure to perform solvency checks at the end of `redeem` and `withdraw` as well. Post solvency check for the transfer endpoints might not be necessary. Also note that the solvency checks use the stale value of the total assets (in the meantime some assets might have been donated in the underlying asset contract to this strategy. These assets are accounted for in the `report` endpoint).
6. `S.dragonRouter = oldDragonRouter;`.
7. `emit UpdateDragonRouter(newDragonRouter)`.
8. `uint256 totalAssets = S.totalAssets;` these values at this point would be the same as `totalAssetsBalance`.
9. `max{Deposit,Mint,Withdraw,Redeem}` need to be upadated to make sure when dragon router is supplies and the dragon router is not allowed to interact with the corresponding endpoints, these functions should return 0.
10. And 11. Not sure why these debt parameter values are introduced. They both can be derived simply from other storage values and their current usage adds unrequired complexity to the codebase.

Octant: Fixed in several commits.

Cantina Managed:

1. Commit bc9cc324: irrelevant changes introduced for the `TokenizedStrategy.max...` endpoints perhaps to be used with other commits. For the other changes that directly affect this issue, when the strategy is solvent and the conversion rate is 0 the fallback conversion should be analysed. Having both of these conditions should imply that the total debt value should be 0 which should indicate that the total supply should be 0. And so in this case the fallback conversion rate should be 1:1 after normalizing the corresponding decimal precisions. Although this does not match exatly with the conversion used in `_handleDragonLossProtection`.

The remaining issues regarding the zero conversion edge case has been acknowledged by the Octant team.

2. Commit 8486e426: fixed.
3. Commit bdd9d292: fixed.
4. Commit 8486e426: fixed.
5. Comment from Octant:

> We dont want to be unable to withdraw when vault is insolvant - adding it would prevent withdrawing.

Calling `_requireDragonSolvency(owner)` at the end of the flows does not prevent withdrawing for all users. Only withdrawing for the dragon router would be blocked if the strategy would end up being insolvent at the end.

On a later update: commit 028732e0 fixes this issue.

6. Commit 6a2740f7: fixed.
7. Commit 6a2740f7: fixed.
8. Commit 6a2740f7: fixed.
9. Commit d21f4b5c: fixed. (The irrevalent changed in 1. commit hash was required for this fix).
10. Not fixed.
11. Not fixed.

3.3 Low Risk

3.3.1 Minor recommendations, comments, typos,

Severity: Low Risk

Context: *(See each case below)*

- Minor recommendations:
  - LinearAllowanceSingletonForGnosisSafe.sol#L247: Can be simplified to `return newAccrued;`.
  - LinearAllowanceSingletonForGnosisSafe.sol#L218: Use `encodeCall`.
  - MorphoCompounderStrategy.sol#L41: Make sure the `asset` provided is the `compounderVault`'s asset.
  - TokenizedStrategy.sol#L336: This `API_VERSION` is copied from `Yearn` but should be specialised for `Octant` since there is also a change from that version of `Yearn`.
  - TokenizedStrategy.sol#L364: The root of storage is `BASE_STRATEGY_STORAGE` which is derived from `yearn.base.strategy.storage` which might suggest that the storage layout should also be the same as `Yearn` but this is not the case. It might be best to use a custom path for `Octant`.
  - TokenizedStrategy.sol#L599-L601: `totalAssets()` could return stale values is it does not query for an updated yield from the yield sources and thus not fully conform to the `EIP-4626` standard.
  - TokenizedStrategy.sol#L1324: Use a different prefix for `Octant` tokenised strategies symbols to distinguish than from `Yearn`.
  - TokenizedStrategy.sol#L1620-L1622: Cache `Permit` type hash as a contract level constant.
  - TokenizedStrategy.sol#L1654: Use `Octant Vault` as the `name` in `DOMAIN_SEPARATOR()`.
  - TokenizedStrategy.sol#L1653-L1655: Cache the `EIP712Domain` type hash, `name` and `version` hashes as contract level constants.
  - .env.template: `TEST_RPC_URL_POLYGON` is missing. (Out of Scope).
  - AccessControl.t.sol#L592-L617, AccessControl.t.sol#L549-L574: These tests are failing due to `vm.warp: timestamp must be less than 2^64 - 1` (Out of Scope).
  - UniswapV3Swapper.sol#L23: Make `UniswapV3Swapper` an `abstract` contract. (Out of Scope).
  - YieldSkimmingTokenizedStrategy.sol#L34: Use consistent custom storage slot derivation like `ALLOCATION_STORAGE_SLOT`. Also for both, one can also clear up the last byte from the final result to follow the same patterns as in OpenZeppelin.
  - TokenizedAllocationMechanism.sol#L281: `keccak256(bytes(EIP712_VERSION))` value can be cached as a constant.
  - BaseAllocationMechanism.sol#L213-L230, BaseAllocationMechanism.sol#L264-L294: There are 2 internal view helper sections. They can be consolidated into one section and perhaps places after the `fallback()` function.
  - TokenizedAllocationMechanism.sol#L732-L735: `getRemainingVotingPower(...)` has the same implementation as `votingPower(...)` and it is not used. Perhaps `getRemainingVoting-Power(...)` can be removed.
  - TokenizedAllocationMechanism.sol#L845: Incorrect event name `OwnershipTransferInitiated` for `cancelOwnershipTransfer()`.

  - TokenizedAllocationMechanism.sol#L1054-L1056: Setter function is missing for `emergencyAdmin`.
  - TokenizedAllocationMechanism.sol#L927-L958: Both `redeem` functions with or without `maxLoss` are missing the `onlyInitialized` modifier. This is not an issue in particular, but in some scenarios and some edge cases it might have some undesired consequences.
  - TokenizedAllocationMechanism.sol#L897-L911: Emitting the `Swept` event can be refracted out of the `if/else` block.
  - TokenizedAllocationMechanism.sol#154: `sharesRequested` is not used currently. `sharesRequested` could replace `proposalShares` and perhaps be renamed to `sharesRequested -> mintedProposalShares`. This way all the `proposals` related info would be accessed in one route instead of two.
  - ProperQF.sol#L101: `solc 0.8.0` and up check for multiplication overflows so if `voteWeight * voteWeight` were to overflow, the compiler would throw an arithmetic overflow revert and one would not be able to reach `VoteWeightOverflow()`. This line can be removed:

```
        - if (voteWeightSquared / voteWeight != voteWeight) revert VoteWeightOverflow();
```

  - QuadraticVotingMechanism.sol#L238-L242: `receive` function has been overwritten to prevent accidental native token deposits. It would be best to also override `fallback` to prevent such deposit. With the current implementation of `TokenizedAllocationMechanism` sending native tokens through the `fallback` function fails.
  - TokenizedAllocationMechanism.sol#L60: `TokenizedAllocationMechanism` can inherit from `ReentrancyGuardTransient` instead of `ReentrancyGuard`. the `solc` pragma needs to be modified accordingly.
  - TokenizedAllocationMechanism.sol#L92: `forVotes` and `againstVotes` are unused in the `NoQuorum` error.
  - ProperQF.sol#L133-L135: These errors should not be reachable with the new `solc` compilers. You can write unit tests to confirm.
  - TokenizedStrategy.sol#L1654: `_strategyStorage().name` is not used in `DOMAIN_SEPARATOR()`.
  - TokenizedStrategy.sol#L1252-L1254: Store the `effectiveTimestamp` instead of `dragonRouterChangeTimestamp` in storage for gas optimisation.

- Comments:
  - PaymentSplitter.sol#L57-L65: The NatSpec comments for the `constructor` need to be moved for the `initialize` instead.
  - TokenizedAllocationMechanism.sol#L190-L191: The comment for both `keeper` and `management` are incorrect. Currently in the implemented hooks these two are the only entities that can call `propose(...)`
  - TokenizedAllocationMechanism.sol#L188: The comment is incorrect here, the `totalAssets` is snapshots in the current implementation of the hooks to query the balance of address `this` of the assets. Any airdrops before calling finalise vote tally would be accounted for.
  - TokenizedAllocationMechanism.sol#L867-L871: `emergencyAdmin` should also be able to call `pause()`.
  - TokenizedAllocationMechanism.sol#L1164: `maxLoss` is unused.
  - QuadraticVotingMechanism.sol#L63: `assetDecimals` can be read from `decimals`, i.e. TokenizedAllocationMechanism.sol#L1058 one should compare which design would induce less gas. The `_tokenizedAllocation()` might already be warm, but `IERC20Metadata(address(asset)).decimals()` might return without touching storage. `asset` should also be a warm addressed as it has been touched in `_executeSignup` before the hook call `_getVotingPowerHook`.

- Typos:
  - README.md#L5: `funtion $\rightarrow$ function` (Out of Scope).
  - README.md#L32: It should be `yarn run init` (Out of Scope).

- Missing or Incomplete Events:

  - BaseHealthCheck.sol#L123-L125, BaseYieldSkimmingHealthCheck.sol#L125-L127.
  - BaseHealthCheck.sol#L93-L97, BaseYieldSkimmingHealthCheck.sol#L95-L99.
  - BaseHealthCheck.sol#L113-L116, BaseYieldSkimmingHealthCheck.sol#L115-L118).
  - RegenStakerBase.sol#L269-L271: Emitting events are missing for the `IWhitelist` fields set in `_initializeSharedState`.
  - TokenizedAllocationMechanism.sol#L366-L385: Besides the `owner` emitting events for all the other parameters is missing in `_initializeAllocation`.
  - YieldSkimmingTokenizedStrategy#L41: `DonationBurned` event is not emitted after dragon router shares are burned. Also `DonationMinted`,`DonationBurned` events are not present in `YieldDonatingTokenizedStrategy.sol`.
  - ProperQF.sol#L46-L47: Emitting `AlphaUpdated` is missing.
  - TokenizedAllocationMechanism.sol#L494: It might be useful to also emit the `payer` and `deposit` in `UserRegistered`.
  - TokenizedAllocationMechanism.sol#L613: It might be useful to also emit `choice`, `oldPower` and `newPower` in `VotesCast`.
  - TokenizedAllocationMechanism.sol#L637: Let `VoteTallyFinalized` emit `s.totalAssets`.
  - ProperQF.sol#L122: Emitting an event is missing with all the storage parameters updated in `_processVoteUnchecked`.
  - TokenizedAllocationMechanism.sol#L680: `ProposalQueued` event can be modified to also emit `recipient`, `customDistributionHandled` and avoid emitting `s.globalRedemptionStart` since that is the same for all queued proposals and it is already emitted in finalise vote tally.
  - TokenizedAllocationMechanism.sol#L845: A new event should be used in `cancelOwnershipTransfer()` like `OwnershipTransferCanceled`.
  - TokenizedStrategy.sol#L1292-L1294: Emitting an event is missing when `setName` is called.

Octant: Partially fixed.

Cantina Managed: Most items are acknowledged except:

1. MorphoCompounderStrategy.sol#L41: Make sure the `asset` provided is the `compounderVault`'s asset.
2. TokenizedAllocationMechanism.sol#L845: Incorrect event name `OwnershipTransferInitiated` for `cancelOwnershipTransfer()`.
3. TokenizedAllocationMechanism.sol#L1054-L1056: Setter function is missing for `emergencyAdmin`.

The above have been fixed in commit a7650c24.

3.3.2 `balanceOfAsset()` is not added to the final `_totalAssets` when the strategy is not shutdown

Severity: Low Risk

Context: SkyCompounderStrategy.sol#L194-L203

Description: `balanceOfAsset()` is not added to the final `_totalAssets` when the strategy is not shutdown. In case the current `balance` is less than or equal to `ASSET_DUST` the unstaked balance in left in the strategy is not added to `_totalAssets`.

Recommendation: It might make sense to change the code in the context to:

```
uint256 balance = balanceOfAsset();
if (!TokenizedStrategy.isShutdown() && balance > ASSET_DUST) {
    _deployFunds(balance);
}

_totalAssets = balanceOfAsset() + balanceOfStake();
```

The above also makes sure that if not all the `balance` is staked in the staking contract when `_deployFunds(balance)` due to different implementations then the final returned `_totalAssets` would be correct.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.3.3 `_allocationMechanismWhitelist` is not checked in `_initializeSharedState`

Severity: Low Risk

Context: RegenStakerBase.sol#L271

Description: `_allocationMechanismWhitelist` is only used in `contribute(...)` to check whether a provided `_allocationMechanismAddress` is allowed to be used. In the `constructor` if the value for `_allocationMechanismWhitelist` is set to `address(0)` until this value is updated to a non-zero value the calls to `contribute(...)` would fail since the call below would fail:

```
// IWhitelist(address(0)).isWhitelisted(_allocationMechanismAddress) fails
sharedState.allocationMechanismWhitelist.isWhitelisted(_allocationMechanismAddress)
```

Also the `setAllocationMechanismWhitelist` has the following check which is omitted in the `constructor` flow:

```
require(
   address(_allocationMechanismWhitelist) != address(0),
   DisablingAllocationMechanismWhitelistNotAllowed()
);
```

Recommendation: Unify the flows and add the same check as above for the `constructor` flow in `_initializeSharedState`.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. In the new implementation, the code checks that `_allocationMechanismWhitelist` is a non-zero address in all routes and also it is a different address than the other whitelist contracts.

3.3.4 The active reward period check is different for `setRewardDuration` and `setMinimumStakeAmount`

Severity: Low Risk

Context: RegenStakerBase.sol#L286, RegenStakerBase.sol#L393-L396

Description: The active reward period check is different for `setRewardDuration` and `setMinimumStakeAmount`:

```
block.timestamp >  rewardEndTime // setRewardDuration
block.timestamp >= rewardEndTime // setMinimumStakeAmount
```

Recommendation: It would be best to unify these checks. The stricter inequality might be best.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.3.5 Earning power parameters are not updated when `RegenStakerBase.contribute` is called

Severity: Low Risk

Context: RegenStakerBase.sol#L475

Description: Earning power parameters are not updated when `RegenStakerBase.contribute` is called. this is in contrast to the implementations in `Staker` contract in `_claimReward` function and also other functions where the following pattern is used:

```
function f(/*...*/) /*...*/ {
   // ...
   _checkpointGlobalReward();
   _checkpointReward(deposit);

   // 1. perform some operations and checks
```

```
    // 2. update earning power storage parameters since they affect the next reward related calculation
}
```

Also `RewardClaimed` emits the potentially old `deposit.earningPower` versus the should-be-calculated new one.

Severity::

- Likelihood: depends on many things and also the implementation of the `RegenEarningPowerCalculator`. It was originally set to `medium` but for example if you had bot/keepers that would call `bumpEarningPower` frequently and mixing that with the assumption that one is using the current implementation of `RegenEarningPowerCalculator` and the listing and delisting a user from allowed list is not that frequent, then likelihood would be `low`.
- Impact: Again based on the above assumptions (which are very specific) the impact could be `medium` or `low`.

Overall the `severity` based on the above assumptions would be `medium` or `low` (leaning towards `low`).

Recommendation: Make sure to update the earning power related storage parameters and emit the correct new value in the `RewardClaimed`. Here is a rough patch:

```
  diff --git a/src/regen/RegenStakerBase.sol b/src/regen/RegenStakerBase.sol
  index 2a8615f..0298282 100644
-  -- a/src/regen/RegenStakerBase.sol
+  ++ b/src/regen/RegenStakerBase.sol
  @@ -472,7 +472,17 @@ abstract contract RegenStakerBase is Staker, Pausable, ReentrancyGuard, EIP712,
           uint256 scaledAmountConsumed = _amount * SCALE_FACTOR;
           deposit.scaledUnclaimedRewardCheckpoint = deposit.scaledUnclaimedRewardCheckpoint -
           ↪  scaledAmountConsumed;

-          emit RewardClaimed(_depositId, msg.sender, amountContributedToAllocationMechanism,
↪  deposit.earningPower);
+          uint256 _newEarningPower =
+              earningPowerCalculator.getEarningPower(deposit.balance, deposit.owner, deposit.delegatee);
+
+          emit RewardClaimed(_depositId, msg.sender, amountContributedToAllocationMechanism, _newEarningPower);
+
+          totalEarningPower =
+              _calculateTotalEarningPower(deposit.earningPower, _newEarningPower, totalEarningPower);
+          depositorTotalEarningPower[deposit.owner] = _calculateTotalEarningPower(
+              deposit.earningPower, _newEarningPower, depositorTotalEarningPower[deposit.owner]
+          );
+          deposit.earningPower = _newEarningPower.toUint96();

           // approve the allocation mechanism to spend the rewards
           SafeERC20.safeIncreaseAllowance(
```

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. Recommendation has been applied.

3.3.6 Staking token invariants might get broken when staking and reward tokens are the same

Severity: Low Risk

Context: RegenStakerBase.sol#L217-L218

Description: Staking token invariants might get broken when staking and reward tokens are the same. The big invariant is that (when the staking and reward tokens are different).

$$\sum_{d \in D} b_d = \sum_{a \in U} (b_{su_a} - b_{su_d, donation}) \quad (1)$$

i.e. the sum of the `deposit` balances should be equal to the sum of balances of staking tokens for all the surrogates (minus any donations to those surrogates).

Parameter	Description
D	the set of deposits `deposits`

d	ranges over `depositIds`
b_d	`Deposit.balance` for the deposit id d
U	the set of all delegates
su_a	the surrogate of the delegatee a
b_x	the staking token balance of `x`
$b_{x,donation}$	any donated staking token to `x` that has not come through the `Staker` contract

An issue arise when the staking token T_S is the same as the reward token T_R. The reward calculations and distribution is not permission less and it is prone to mistakes due to rounding errors as well as potentially incorrect reward `amount` notification through `notifyRewardAmount` by the privileged reward token notifiers. There are no internal accounting or parameter used to account for the injected reward tokens to the `Staker` contract. So due to the above reasons if $T_S = T_R$ potentially the invariant in (1) above might get broken. Thus some depositors might not be able to withdraw their staked token balances b_d for some deposit ids.

The above issue applies to `RegenStakerWithoutDelegateSurrogateVotes` since surrogates are the same as the staker contract itself and the invariant becomes (B is the staking token balance of the `Staker` contract, B_R is the assumed reward tokens that need to be present in the `Staker` contract to be claimed):

$$B_R + \sum_{d \in D} b_d = B - B_{donation} \tag{1}$$

And as one can see for an incorrect value of B_R the excess amount would affect the $\sum_{d \in D} b_d$.

This issue does not apply to `RegenStaker` since the staked tokens for deposits are siloed out to the deposit's delegatee's surrogate contract. And another point to note that in the current variants of the `Staker` contract in this project it is the `RegenStakerBase` that hold the rewards tokens which is a separate contract in the case of `RegenStaker` (cannot be a surrogate contract) and so the reward and staking tokens cannot collude.

Recommendation: For `RegenStakerWithoutDelegateSurrogateVotes` the above issue when $T_S = T_R$ needs to be documented. To prevent the issue many changes to the underlying design of the `Staker` contract might be needed which might not be desirable.

Octant: Fixed in commit 7dabed96.

Cantina Managed: Fix verified. New mechanism has been added to track total reward amount and as well as total reward claim amount to address this issue.

3.3.7 MorphoCompounderStrategy's `availableDepositLimit` doesn't consider idle assets

Severity: Low Risk

Context: TokenizedStrategy.sol#L480, MorphoCompounderStrategy.sol#L46-L48, MorphoCompounder-Strategy.sol#L54-L56

Description: MorphoCompounderStrategy's `availableDepositLimit` function returns the maximum deposit limit of the underlying strategy without considering the fact that in addition to the user's deposit amount, the idle assets of the contract will also be deployed to the strategy. Hence if there are any idle assets, the actual deposit amount could be greater than the allowed deposit limit potentially causing a revert.

```
// src/strategies/yieldDonating/MorphoCompounderStrategy.sol:MorphoCompounderStrategy

function availableDepositLimit(address /*_owner*/) public view override returns (uint256) {
   return IERC4626(compounderVault).maxDeposit(address(this));
}
```

```
// src/core/TokenizedStrategy.sol:TokenizedStrategy

function maxDeposit(address receiver) external view returns (uint256) {
    return _maxDeposit(_strategyStorage(), receiver);
}

function _maxDeposit(StrategyData storage S, address receiver) internal view returns (uint256) {
    // Cannot deposit when shutdown or to the strategy.
    if (S.shutdown || receiver == address(this)) return 0;
    return IBaseStrategy(address(this)).availableDepositLimit(receiver);
}

function _deposit(StrategyData storage S, address receiver, uint256 assets, uint256 shares) internal virtual {
     // Cache storage variables used more than once.
    ERC20 _asset = S.asset;

    // Need to transfer before minting or ERC777s could reenter.
    _asset.safeTransferFrom(msg.sender, address(this), assets);

    // We can deploy the full loose balance currently held.
    IBaseStrategy(address(this)).deployFunds(_asset.balanceOf(address(this)));
```

Recommendation: Return `IERC4626(compounderVault).maxDeposit(address(this)) < _asset.balanceOf(address(this)) ? 0 : IERC4626(compounderVault).maxDeposit(address(this)) - _asset.balanceOf(address(this))` as the `availableDepositLimit` in `MorphoCompounderStrategy`.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.3.8 Incorrect/insufficient parameters in error emission

Severity: Low Risk

Context: LinearAllowanceSingletonForGnosisSafe.sol#L45

Description: If the length comparison of more than 2 arrays results in a mismatch, the `ArrayLengthsMismatch` error is emitted with only 2 parameters with no handling done to ensure that the emitted parameters were causing the mismatch.

- Occurrence 1: LinearAllowanceSingletonForGnosisSafe.sol#L89:

```
function executeAllowanceTransfers(
    address[] calldata safes,
    address[] calldata tokens,
    address[] calldata tos
) external nonReentrant returns (uint256[] memory transferAmounts) {
    uint256 length = safes.length;
    require(length == tokens.length && length == tos.length, ArrayLengthsMismatch(length,
    ↪  tokens.length));
```

- Occurrence 2: LinearAllowanceSingletonForGnosisSafe.sol#L37-L46:

```
function setAllowances(
    address[] calldata delegates,
    address[] calldata tokens,
    uint192[] calldata dripRatesPerDay
) external nonReentrant {
    uint256 length = delegates.length;
    require(
        length == tokens.length && length == dripRatesPerDay.length,
        ArrayLengthsMismatch(length, tokens.length)
```

Recommendation: Emit 3 parameters to include all the array lengths, e.g.: `ArrayLengthsMismatch(length, tokens.length, dripRatesPerDay.length)`.

Octant: Fixed in commit 4d36cd1c.

Cantina Managed: Fix verified.

3.3.9 Voting is possible after finalizing if finalisation is done in the first possible block

Severity: Low Risk

Context: TokenizedAllocationMechanism.sol#L622-L623

Description: Both finalizeVoteTally and _executeCastVote allows invocations to happen at `block.number == s.startBlock + s.votingDelay + s.votingPeriod`:

```
function finalizeVoteTally() external onlyOwner nonReentrant onlyInitialized {
    AllocationStorage storage s = _getStorage();

    if (block.number < s.startBlock + s.votingDelay + s.votingPeriod)
        revert VotingNotEnded(block.number, s.startBlock + s.votingDelay + s.votingPeriod);
```

```
function _executeCastVote(address voter, uint256 pid, VoteType choice, uint256 weight) private {
  // .....

  if (block.number < s.startBlock + s.votingDelay || block.number > s.startBlock + s.votingDelay +
  ↪  s.votingPeriod)
      revert VotingClosed(
          block.number,
          s.startBlock + s.votingDelay,
          s.startBlock + s.votingDelay + s.votingPeriod
      );
```

In case a vote is performed after `finalizeVoteTally` is invoked, it might not be counted at all or the total shares can increase past the expected amount resulting in a possible deviation from the expected funding.

Recommendation: Update the check in finalizeVoteTally to use <= rather than <:

```
if (block.number <= s.startBlock + s.votingDelay + s.votingPeriod)
    revert VotingNotEnded(block.number, s.startBlock + s.votingDelay + s.votingPeriod);
```

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.3.10 User registration with zero deposit is allowed

Severity: Low Risk

Context: TokenizedAllocationMechanism.sol#L490-L494

Description: If the used hooks in `_executeSignup` would not check for zero deposits, user registration with zero deposit would be allowed. There would be the following scenarios:

1. `getVotingPowerHook` would return `0` for `0` deposits and thus one would be able to call `_executeSignup` multiple times which would make it emit its events multiple times. That could be problematic for off-chain agents or.
2. `getVotingPowerHook` would return a non-zero value for a `0` deposit each could also be undesirable.

Recommendation: To avoid the above issues, it would be best to `revert` if the provided deposit is `0` upon signing up.

Octant: Acknowledged. There are possible mechanisms where zero deposit signups make sense and others where multiple signups do as well (Quadratic Voting and Quadratic funding variants respectively). Rather than revert at the common implementation level, behavior is left to the signup hook and pattern is documented better for future implementers.

Cantina Managed: Acknowledged.

3.3.11 `_beforeSignupHook` does not check the Security Assumptions

Severity: Low Risk

Context: OctantQFMechanism.sol#L38-L46, QuadraticVotingMechanism.sol#L55-L58

Description: `_beforeSignupHook` has the following security assumptions in the provided docs that are not checked for:

1. MUST return false for address(0) to prevent zero address registration.
2. MUST NOT allow re-registration if user already has voting power.

The assumption 2. is checked in `_executeSignup`:

```
if (s.votingPower[user] != 0) revert AlreadyRegistered(user);
```

Recommendation:

1. This check is missing both in `OctantQFMechanism` and in `QuadraticVotingMechanism` and needs to be implemented.
2. Perhaps it might not need to be implemented since there is a check in `_executeSignup`. But there is an edge case where a user uses all its voting powers then signups again.

Cantina Managed: Octant: Fixed in commit d656cc86.

1. This has been implemented in `_executeSignup(...)`. The spec doc has been updated in commit d656cc86.
2. Multiple registration is now allowed.

3.3.12 `_validateProposalHook` does not implement all security assumptions

Severity: Low Risk

Context: QuadraticVotingMechanism.sol#L50-L53

Description: The following security assumptions are stated in the docs for `_validateProposalHook` that are not implemented:

1. MUST NOT validate canceled proposals as valid.
2. SHOULD be used consistently before any proposal state access.

Currently, the validation of canceled proposals are performed non-atomically in almost all the flows where `_validateProposalHook` is used:

Endpoint	cancelation check
_executeCastVote(...)	Yes
queueProposal(...)	Yes
state(...)	No (needs to return for canceled orders without reverting)
cancelProposal(...)	Yes
getProposalFunding(...)	No (should revert or return for a canceled proposal here)

Recommendation: Either the hook implementation needs to be updated to account for the security assumptions and also the above mentioned route's implementation need to be adjusted. Or the security assumptions in the docs need to be updated.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

```
1. MUST NOT validate canceled proposals as valid
```

The above has been removed from the spec docs and also `getProposalFunding(...)` has been updated to return `0` values for cancelled proposals.

3.3.13 `_calculateTotalAssetsHook` does not conform to the spec

Severity: Low Risk

Context: QuadraticVotingMechanism.sol#L180-L189, TokenizedAllocationMechanism.sol#L629-L631

Description: `_calculateTotalAssetsHook` does not conform to the spec:

Spec	Issue
SHOULD snapshot values if they might change after finalization	This currently happens upstream in `finalizeVoteTally()` and not atomically in the hook

MUST NOT double-count assets or include **unauthorized** funds	`asset.balanceOf(address(this))` could include unauthorised funds as there are no book-keeping for the funding allocation.

Recommendation: Either the documents needs to be updated or the above issues need to be addressed.

Octant: Fixed in commit 406f1588.

Cantina Managed: The above assumptions have been removed from the spec docs in commit 406f1588.

3.3.14 `_convertVotesToShares` does not conform to specs

Severity: Low Risk

Context: QuadraticVotingMechanism.sol#L124-L133, TokenizedAllocationMechanism.sol#L658-L662

Description: `_convertVotesToShares` does not conform to specs in the docs:

> MUST return `0` shares for proposals that don't meet quorum.

Quorum check is not performed in this hook to return `0` in case the quorum is not met. With the current implementation of `TokenizedAllocationMechanism` this is not an issue since the call to `hasQuorumHook` happens before the call to `convertVotesToShares` in (`queueProposal(...)`:

```
if (!IBaseAllocationStrategy(address(this)).hasQuorumHook(pid)) revert NoQuorum(pid, 0, 0, s.quorumShares);
// ...
uint256 sharesToMint = IBaseAllocationStrategy(address(this)).convertVotesToShares(pid);
```

But in the general setting if one were to create new hooks with another `TokenizedAllocationMechanism` implementation, one needs to make sure the specs are up to date.

Recommendation: Either the spec or implementation needs to be updated.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. The spec has been updated:

> - `_convertVotesToShares(uint256 pid)` - Converts vote tallies to vault shares.
>   - Security Assumptions:
>   - ...
>   - Note: Quorum enforcement is handled by `queueProposal()` which calls `hasQuorumHook()` before `convertVotesToShares()`.

3.3.15 `previewRedeem` could turn non-zero value outside of the redemption period

Severity: Low Risk

Context: TokenizedAllocationMechanism.sol#L1011-L1022

Description: `previewRedeem` could turn non-zero value outside of the redemption period. But only should returns 0 if $t \notin [t_r^{start}, t_r^{end}]$. For a more in-depth analysis refer to "Mechanism timeline and state machine"'s last table.

Recommendation: Make sure to return 0 when $t \notin [t_r^{start}, t_r^{end}]$.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.3.16 Make sure `_startBlock` is not in the past during initialisation

Severity: Low Risk

Context: TokenizedAllocationMechanism.sol#L359

Description: `_initializeAllocation` only checks that $t_s \neq 0$:

```
if (_startBlock == 0) revert ZeroStartBlock();
```

But one could provide value that is in the past $t_s < t_{now}$.

Recommendation: It might make sense to add a check to make sure $t_s \geq t_{now}$:

```
if (_startBlock < block.number) revert StartBlockInThePast();
```

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.3.17 Quadratic Voting's total funding value might be slightly higher than the sum of all proposal funding values

Severity: Low Risk

Context: ProperQF.sol#L84-L87, ProperQF.sol#L152-L153, ProperQF.sol#L156-L166, ProperQF.sol#L233-L234

Description: `_getProperQFStorage().totalFunding` is calculated as:

$$F(c) = \lfloor \alpha \cdot \|(\Phi^{LR}(c)_p)_{p\in P}\|_1 \rfloor + \lfloor (1-\alpha) \cdot \|(\Phi^{CAP}(c)_p)_{p\in P}\|_1 \rfloor$$

where as individual proposal voting/funding values are calculated as:

$$F_p(c) = \lfloor \alpha \cdot \Phi^{LR}(c)_p \rfloor + \lfloor (1-\alpha) \cdot \Phi^{CAP}(c)_p \rfloor$$

Thus one can see:

$$\sum_{p\in P} F_p(c) \leq F(c)$$

or

$$\epsilon + \sum_{p\in P} F_p(c) = F(c)$$

where the error term $\epsilon \in \{0, 1, \cdots, 2(|P|-1)\}$, i.e. it is bounded by twice the number of proposals. $\sum_{p\in P_q} F_p$ is the total number of shares that would be minted for the queued proposals and we have:

$$\sum_{p\in P_q} F_p \leq \epsilon + \sum_{p\in P} F_p(c) = F(c)$$

Recommendation: The above discrepancy should be documented. `totalFunding()` is not used directly in the codebase but could be quried by on or off chain agents. To avoid the discrepancy, one could also keep track of the individual $\alpha \cdot \Phi^{LR}(c)_p$ and $(1-\alpha) \cdot \Phi^{CAP}(c)_p$ values per proposal as well as their total sum to avoid the error introduced when splitting values:

$$\lfloor a_1 \rfloor + \lfloor a_2 \rfloor + \cdots + \lfloor a_n \rfloor \leq \lfloor a_1 + a_2 + \cdots + a_n \rfloor$$

The down-side to keeping track of all the weighted voting levels would be that when α is set to a different value all these storage parameters for all proposals would need to get updated.

Octant: Acknowledged. Shares represent a percentage of the total assets, in the end even the dust will get distributed and any other funds left over can be swept by the admin at the end of the redemption window. The discrepancy has been documented in commit 73812a51.

Cantina Managed: Acknowledged.

Footnote: For the notations refer to ”Quadratic Voting Mechanism”.

3.3.18 Ability to redeem or withdraw during a paused state is not consistent across different contracts

Severity: Low Risk

Context: TokenizedStrategy.sol#L515-L543, TokenizedStrategy.sol#L554-L574, TokenizedAllocationMechanism.sol#L927-L947, RegenStakerBase.sol#L647

Description: Ability to redeem or withdraw during a paused state is not consistent across different contracts:

- In `TokenizedAllocationMechanism` and `TokenizedStrategy`(`withdraw` as well). Currently `TokenizedStrategy` and its derived contracts do not have a pausing functionality but the emergency admin can shutdown the contract to stop deposits, one can redeem during the paused state, but...
- In `RegenStakerBase` withdrawing is not allowed when the contract is paused.

Recommendation: It might make sense to unify the pausing pattern used when funds need to be taken out of the contract during pausing period.

If one decides to block redeeming during the pausing period for `TokenizedAllocationMechanism`, the `maxRedeem` and `previewRedeem` would also need to be updated to reflect this choice. They would need to return `0` during this period.

Octant: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Gas Optimization

3.4.1 `totalRewards` can be refactored in `_notifyRewardAmountWithCustomDuration`

Severity: Gas Optimization

Context: RegenStakerBase.sol#L305-L315

Description/Recommendation: `totalRewards` can be refactored in `_notifyRewardAmountWithCustomDuration`:

```
uint256 totalRewards = _amount * SCALE_FACTOR;

if (rewardEndTime > block.timestamp) {
    totalRewards += scaledRewardRate * (rewardEndTime - block.timestamp);
}

scaledRewardRate = totalRewards / sharedState.rewardDuration;
```

Also in this function and many other `Staker` contract (or inherited contracts) storage parameters are not cached at all and thus there might be many unnecessary `SLOAD`s per parameter.

Octant: Acknowledged.

Cantina Managed: Acknowledged.

3.4.2 `DOMAIN_SEPARATOR()` can be further optimised in case of a changing `block.chainid`

Severity: Gas Optimization

Context: TokenizedAllocationMechanism.sol#L265-L272

Description: The current mechanism uses storage parameters (versus similar implementations where immutable variables are used). Thus in the case recompute is needed for the domain separator, one can store the new values in the storage parameters.

Recommendation: After recompute store the values back into the storage:

```
function DOMAIN_SEPARATOR() public view returns (bytes32) {
    AllocationStorage storage s = _getStorage();
    if (block.chainid == s.initialChainId) {
        return s.domainSeparator;
    } else {
        s.initialChainId = block.chainid;
        bytes32 domainSeparator = _computeDomainSeparator(s);
        s.domainSeparator = domainSeparator;
        return domainSeparator;
    }
}
```

and perhaps `s.initialChainId` field name can be changed to `s.chainId`.

Octant: Fixed in commit 8f582c3b.

Cantina Managed: Fix verified.

3.4.3 Redundant checks already performed in `_setAlpha` can be removed

Severity: Gas Optimization

Context: QuadraticVotingMechanism.sol#L32-L33, QuadraticVotingMechanism.sol#L217-L219

Description/Recommendation: Redundant checks already performed in `_setAlpha` can be removed:

```
if (_alphaNumerator > _alphaDenominator) revert AlphaMustBeLEQOne();
if (_alphaDenominator == 0) revert AlphaDenominatorMustBePositive();
```

In:

1. `QuadraticVotingMechanism.constructor`.
2. `QuadraticVotingMechanism.setAlpha`.

Octant: Fixed in commit f8cd9600.

Cantina Managed: Fix verified.

3.4.4 `linearFunding` can be removed from `Project` struct

Severity: Gas Optimization

Context: ProperQF.sol#L27

Description: `linearFunding` and `sumContributions` hold the same values. Thus keeping both in storage in not necessary.

Recommendation: `linearFunding` can be removed from `Project` struct and wherever it is used its value can be replaced by `sumContributions`.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. Both `quadraticFunding` and `linearFunding` have been removed from the `Project` struct.

3.5 Informational

3.5.1 Inconsistencies between `_notifyRewardAmountWithCustomDuration` and the original `notifyRewardAmount`

Severity: Informational

Context: RegenStakerBase.sol#L322-L325

Description: The check here is stricter by `REWARD_DURATION` compared to the original `Staker` implementation:

```
// Staker
(scaledRewardRate * REWARD_DURATION) > (REWARD_TOKEN.balanceOf(address(this)) * SCALE_FACTOR)
```

- `Staker` check:

$$\left\lfloor \frac{R}{\Delta t} \right\rfloor \Delta t = r\Delta t \leq B$$

where:

- r: `scaledRewardRate` after update.
- B: `REWARD_TOKEN.balanceOf(address(this)) * SCALE_FACTOR`.
- R : `totalRewards`.

The current check in `_notifyRewardAmountWithCustomDuration` is $R \leq B$ and we have:

$$r\Delta t = \left\lfloor \frac{R}{\Delta t} \right\rfloor \Delta t \leq R \leq B$$

We have:

$$R - \left\lfloor \frac{R}{\Delta t} \right\rfloor \Delta t \in [0, \Delta t)$$

But during the reward distribution only up to (if we ignore the roundings, with roundings it would be less) $r\Delta t$, ie $\lfloor \frac{R}{\Delta t} \rfloor \Delta t$.

So in some scenarios in the check we would require more reward tokens to be owned by the `RegenStakerBase` than it will be distributed.

Moreover the following comment was removed just before the check:

```
// This check cannot _guarantee_ sufficient rewards have been transferred to the contract,
// because it cannot isolate the unclaimed rewards owed to stakers left in the balance. While
// this check is useful for preventing degenerate cases, it is not sufficient. Therefore, it is
// critical that only safe reward notifier contracts are approved to call this method by the
// admin.
```

Recommendation: The above inconsistencies regarding the balance check needs to be documented. And also it would be great to add back the original comment.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. Implementation has been changed to follow the original `Staker` contract.

3.5.2 Inconsistency of using hooks in `_transferForCompound`

Severity: Informational

Context: RegenStaker.sol#L111-L116, RegenStakerWithoutDelegateSurrogateVotes.sol#L127-L129

Description: `_transferForCompound` is only used in `compoundRewards` at the end of the flow where one is imitating the `_stakeMore` flow:

1. (RegenStaker) When `compoundRewards` is called the deposit should already exist and if it had been needed the surrogate must have been deployed initially in the `_stake` call that had created the deposit. One can also see that in the original `Staker` contract implementation in the `_stakeMore` flow one calls the `surrogates(...)` hook to fetch the surrogate.
2. (RegenStaker, RegenStakerWithoutDelegateSurrogateVotes) In the original `_stakeMore` flow of the `Staker` contract, the `_stakeTokenSafeTransferFrom` hook (virtual function) is used at the end to transfer the stake tokens. Where as in the `_transferForCompound` the current inherited implementation of this hook has been inlined in `RegenStaker` and not used at all in `RegenStakerWithoutDelegateSurrogateVotes` (potentially causing future hook dependancy foot guns). If a custom inherited contract changes the `_stakeTokenSafeTransferFrom` logic it would not be reflected here in the `compoundRewards`'s `_transferForCompound` internal function call.

```
// Staker: parent contract

function _stakeMore(/*...*/) /*...*/ {
  //...

  DelegationSurrogate _surrogate = surrogates(deposit.delegatee);

  // ...
```

```
    _stakeTokenSafeTransferFrom(deposit.owner, address(_surrogate), _amount);
}
```

Recommendation:

1. This is not a big issue. But at least should be noted.
2. Currently not an issue in this project, but for a custom contract based on `RegenStaker` or `RegenStakerWithoutDelegateSurrogateVotes` could potentially cause an issue. It might be best to use the `_stakeTokenSafeTransferFrom`:

   ```
   // RegenStaker
   _stakeTokenSafeTransferFrom(address(this), address(_surrogate), _amount);

   // RegenStakerWithoutDelegateSurrogateVotes
   _stakeTokenSafeTransferFrom(address(this), address(this), _amount);
   ```

   If the suggestion for `RegenStakerWithoutDelegateSurrogateVotes` is not going to be implemented (since one can just skip the call) at least it should be documented for devs to avoid future mistakes.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.5.3 Caveats regarding skipping transfers for `RegenStakerWithoutDelegateSurrogateVotes`

Severity: Informational

Context: RegenStakerWithoutDelegateSurrogateVotes.sol#L111-L117

Description: Caveats regarding skipping transfers for `RegenStakerWithoutDelegateSurrogateVotes` in `_stakeTokenSafeTransferFrom` and `_transferForCompound` are the following:

1. `Transfer` event would not be emitted:

   ```
   emit Transfer(address(this), address(this), amount);
   ```

2. `RegenStakerWithoutDelegateSurrogateVotes` staked token balance is not checked to make sure to be at least the amount being transferred.

Currently:

- `_stakeTokenSafeTransferFrom` with both `_from` and `_to` address being `address(this)` is only possible in the `_alterDelegatee` flow. In the more general settings where the stake token transfers are not skipped if the balance of `RegenStakerWithoutDelegateSurrogateVotes` would be very low, lower than the deposit's balance, one would not be able to alter the delegatee in general. But in the case of `RegenStakerWithoutDelegateSurrogateVotes` one can.
- `_transferForCompound` is only used in `compoundRewards`and again if there are not enough tokens (reward and stake tokens are the same in this case) in the contract, in contrast to one would believe the call to `compoundRewards` would not revert.

Recommendation: If one still decides to skip in the above cases, they should be documented.

Octant: Fixed in commit ebe5bd7b.

Cantina Managed: Fix verified.

3.5.4 Deploying `_surrogate` deterministically might be beneficial

Severity: Informational

Context: RegenStaker.sol#L95-L97, RegenStaker.sol#L106

Description: Currently in `RegenStaker`, surrogate contracts are deployed using the `RegenStaker`'s contract (the surrogate factory) nonce and the calls to `surrogates(_delegatee)` before deployment returns `address(0)`.

Recommendation: It might be best to use deterministic address for surrogates of the delegate by using other deterministic deploy patterns such as `create2` with `salt` so one can query `surrogates(_delegatee)` and get the correct address before it being deployed.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.5.5 Deriving _delegatee from the _surrogate address

Severity: Informational

Context: RegenStaker.sol#L106-L107

Description: Currently it is possible to query for the _surrogate address knowing the _delegatee. But the inverse is not possible.

Recommendation: It would be useful for on or off chain reasons to be able to derive in the reverse direction the _delegatee address from the _surrogate address.

This can be accomplished in many ways. Here are two possibilities:

1. Introduce a new storage mapping parameter _surroageToDelegatee:

   ```
   mapping(DelegationSurrogate => address) private _surroageToDelegatee;
   ```

   and set the corresponding key/value in _fetchOrDeploySurrogate when a new DelegationSurrogateVotes is created. And expose a getter function for it.

2. Or create a new contract DelegationCustom that inherits from DelegationSurrogateVotes with a modified constructor that would set the _delegatee as an immutable parameter in its implementation.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fixed using:

```
// RegenStaker.sol

function getDelegateeFromSurrogate(address _surrogate) external view returns (address) {
     return VOTING_TOKEN.delegates(_surrogate);
 }
```

3.5.6 Current implementation doesn't have a concept of loss deviating from natspec and function definitions

Severity: Informational

Context: TokenizedAllocationMechanism.sol#L1155-L1156

Description: Natspec and function definition of the _withdraw function is written with the indication that losses are handled when withdrawing. But the current implementation doesn't have a concept of loss and doesn't use the maxLoss variable at all.

```
/**
 * ...
 * If we are not able to withdraw the full amount needed, it will
 * be counted as a loss and passed on to the user.
 */
function _withdraw(
    AllocationStorage storage S,
    address receiver,
    address shareOwner,
    uint256 assets,
    uint256 shares,
    uint256 maxLoss // <---
) internal returns (uint256) {
```

Recommendation: Remove the unwanted maxLoss variable and the line from natspec.

Octant: Fixed in commit da2acf22.

Cantina Managed: Fix verified.

3.5.7 `recipient` could get DoSed from receiving a proposal if `beforeProposeHook` does not implement restrictions correctly

Severity: Informational

Context: TokenizedAllocationMechanism.sol#L509-L531

Description: `recipient` could get DoSed from receiving a proposal if `beforeProposeHook` does not implement restrictions correctly. The only check performed in `propose(...)` that would prevent a malicious (or colluded) entity to frontrun another `propose(...)` call with the same `recipient` would be in `beforeProposeHook`. If these checks are not performed correctly, the front-running issue could make the legit `propose(...)` calls to `revert` due to the `s.recipientUsed[recipient]` flag.

As an example in the derived `QuadraticVotingMechanism` contract, this hook is implemented in such a way that would restrict the callers to `propose(...)` to only the `keeper` or `management` accounts (there would be still the risk of rogue accounts):

```
/// @notice Only keeper or management can propose
function _beforeProposeHook(address proposer) internal view virtual override returns (bool) {
    if (proposer == address(0)) revert ZeroAddressCannotPropose();

    // Get keeper and management addresses from TokenizedAllocationMechanism
    address keeper = _tokenizedAllocation().keeper();
    address management = _tokenizedAllocation().management();

    // Allow if proposer is either keeper or management
    return proposer == keeper || proposer == management;
}
```

Also some recommendations about this hook has been documented in the docs:

- `_beforeProposeHook(address proposer)` - Validates proposal creation rights.
  - Security Assumptions:
    - ⋆ MUST verify proposer has legitimate right to create proposals (e.g., voting power > 0, role-based access).
    - ⋆ MUST be view function to prevent state changes during validation.
    - ⋆ SHOULD prevent spam by implementing appropriate restrictions.
    - ⋆ MUST return false for address(0) proposers.
    - ⋆ MAY restrict to specific roles (e.g., QuadraticVotingMechanism restricts to keeper/management only).

Recommendation: It is still recommended that this issue or suggested recommendations be highlighted in the `propose(...)` function either as an inline comment before the hook or in its NatSpec.

Octant: Fixed in commit 3634121c.

Cantina Managed: Fix verified.

3.5.8 Add assumptions and security comments from the docs to the hook NatSpec

Severity: Informational

Context: BaseAllocationMechanism.sol#L75-L146

Description: In the documents there are certain assumptions that are not stated in the NatSpec for the `virtual` hooks defined in this context.

These assumptions are really important for anyone reading the code (auditors, devs who would implement a new mechanism, ...) and should be included in the NatSpec.

Recommendation: Add assumptions and security comments from the docs to the hook NatSpec.

Octant: Acknowledged.

Cantina Managed: Acknowledged.

3.5.9 Misplaced Security Assumptions for `_getVotingPowerHook`

Severity: Informational

Context: QuadraticVotingMechanism.sol#L60-L77

Description: The documents mentions that the following security assumption should be atomically implemented in `_getVotingPowerHook`:

> MUST NOT exceed `MAX_SAFE_VALUE` (`type(uint128).max`) to prevent overflow.

But currently, this is checked in parallel with the call to the hook in `_executeSignup`

Recommendation: Make sure the check is performed atomically in `_getVotingPowerHook` or update the document for security assumptions.

Octant: Fixed in commit 78496af4.

Cantina Managed: Fix verified. The spec docs have been updated in commit78496af4 to reflect the above.

3.5.10 Redundant calculations and error checks in `_processVote`

Severity: Informational

Context: ProperQF.sol#L99-L110, ProperQF.sol#L168-L181

Description/Recommendation: Currently `_processVote` is unused. Some of the checks performed in `_processVote` are redundant.

parameter	description
c	`contribution`
w	`voteWeight`

`solc 0.8.0` and up perform under and overflow arithmetic checks so the following `revert` statement would never be reached and the whole `if` block can be removed:

```
uint256 voteWeightSquared = voteWeight * voteWeight;
if (voteWeightSquared / voteWeight != voteWeight) revert VoteWeightOverflow(); // can be removed
```

The following check:

```
if (voteWeightSquared > contribution) revert SquareRootTooLarge();
```

is to make sure:

$$w^2 \leq c$$

and thus the next check:

```
uint256 actualSqrt = _sqrt(contribution);
uint256 tolerance = actualSqrt / 10; // 10% tolerance
// Only allow vote weight to be lower than actual sqrt, not higher
if (voteWeight < actualSqrt - tolerance || voteWeight > actualSqrt) {
    revert VoteWeightOutsideTolerance();
}
```

if one assumers `_sqrt` is perfectly calculated without any errors:

$$0.9 \cdot \sqrt{c} \leq w \leq \sqrt{c}$$

which is equivalent to:

$$0.9^2 \cdot c \leq w^2 \leq c$$

Note how one of the checks is redundant `voteWeight > actualSqrt` and the other one is:

$$0.81 \cdot c \le w^2$$

or roughly:

$$\left\lfloor \frac{4 \cdot c}{5} \right\rfloor \le w^2$$

So this last block can be replaced by:

```
// do NOT need to calculate `sqrt` or `tolerance`
if (voteWeightSquared < (4 * contribution) / 5 {
    revert VoteWeightOutsideTolerance();
}
```

and so `_sqrt` can also be removed.

Octant: Acknowledged. There is no impact, leaving to document the intent of the design, simplifying the implementation just obscures what is going on and intuition behind quadratic nature.

Cantina Managed: Acknowledged.

3.5.11 Inconsistency in the usage of `minAmountToSell` between UniswapV2 and UniswapV3 swaps

Severity: Informational

Context: SkyCompounderStrategy.sol#L206

Description: The _uniV2swapFrom function only swaps in case the amount is greater than `minAmountToSell`. This is inconsistent with the _swapFrom function of UniswapV3Swapper.sol which swaps even when the amount is equal to `minAmountToSell`.

```
function _uniV2swapFrom(address _from, address _to, uint256 _amountIn, uint256 _minAmountOut) internal {
    if (_amountIn > minAmountToSell) {
        IUniswapV2Router02(UNIV2ROUTER).swapExactTokensForTokens(
```

```
function _swapFrom(
    address _from,
    address _to,
    uint256 _amountIn,
    uint256 _minAmountOut
) internal virtual returns (uint256 _amountOut) {
    if (_amountIn != 0 && _amountIn >= minAmountToSell) {
        _checkAllowance(router, _from, _amountIn);
```

Recommendation: Maintain consistency across both function.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified. `_uniV2swapFrom` has been changed to use $\geq$.

3.5.12 Inaccurate docs or security checks in `_processVoteHook`

Severity: Informational

Context: QuadraticVotingMechanism.sol#L93-L96

Description: The provided documents mention that for `_processVoteHook` we should have:

Implementation Matches Spec	Rule	Notes
✓	Users can only vote once per proposal	has been checked using `hasVoted[pid][voter]` storage parameter.

X	Vote weight cannot exceed user's current voting power	This incorrect, currently the check gets performed against `quadraticCost` which is the weight squared w^2
$\pm$	Votes can only be cast during active voting period	This has been checked upstream and not in the hook
✓	Vote processing reduces user's available voting power	the return value by this hook would be smaller than the original voting power, but the storage updates happen upstream

Recommendation: Either the documentation needs to be updated or the hook needs to be modified to conform to the documentation spec.

Octant: Fixed in commit c5cac762.

Cantina Managed: Fix verified.

3.5.13 Specs and implementation do not match for `_beforeProposeHook`

Severity: Informational

Context: QuadraticVotingMechanism.sol#L40

Description: Specs and implementation do not match for `_beforeProposeHook`. The specs in the docs mentions that:

> MUST return `false` for `address(0)` proposers.

But currently the hook reverts.

Recommendation: Either the specs need to be updated or for the case of `proposer == address(0)` one should return `false`.

Octant: Fixed in commit 133d1825.

Cantina Managed: Fix verified.

3.5.14 Add a check to make sure `_allocationMechanismAddress`'s asset is the same as the rewards token

Severity: Informational

Context: TokenizedAllocationMechanism.sol#L485, RegenStakerBase.sol#L440, RegenStakerBase.sol#L477-L482, RegenStakerBase.sol#L491-L498

Description: When the depositor or the claimer of a deposit calls `contribute(...)` in the `TokenizedAllocationMechanism(_allocationMechanismAddress).signupOnBehalfWithSignature(...)` subcall one performs a token transfer using the `asset` parameter in `_allocationMechanismAddress`. This address should match `REWARD_TOKEN`. Before this call allowances to `_allocationMechanismAddress` gets safely increased to exactly match the amount that needs to be transferred:

```
SafeERC20.safeIncreaseAllowance(
   REWARD_TOKEN,
   _allocationMechanismAddress,
  amountContributedToAllocationMechanism
);
```

Thus with the current implementation in case of a mismatch between `REWARD_TOKEN` and `TokenizedAllocationMechanism(_allocationMechanismAddress).asset()` the call to `contribute(...)` should revert.

Recommendation: Although the check might not be necessary with the current version of the codebase, it might be best to add a check at the beginning of `contribute(...)` to make sure `REWARD_TOKEN` and `TokenizedAllocationMechanism(_allocationMechanismAddress).asset()` match.

Octant: Fixed in commit 389a315b.

Cantina Managed: Fix verified.

3.5.15 `recipient` of a canceled proposal cannot be used for a new proposal

Severity: Informational

Context: TokenizedAllocationMechanism.sol#L509-L512, TokenizedAllocationMechanism.sol#L521, TokenizedAllocationMechanism.sol#L528, TokenizedAllocationMechanism.sol#L716-L728

Description: If a proposer creates proposal for `recipient` and later decides to cancel this proposal (maybe due to incorrect description or maybe by mistake) then this address `recipient` cannot be used for any upcoming proposal due to the check:

```
if (s.recipientUsed[recipient]) revert RecipientUsed(recipient);
```

Recommendation: If this is an undesired behaviour, one should reset `s.recipientUsed[recipient]` for canceled proposals.

Octant: Acknowledged. This is intended behavior.

Cantina Managed: Acknowledged.

3.5.16 Voting powers used for canceled proposals should be able to be reused

Severity: Informational

Context: TokenizedAllocationMechanism.sol#L716

Description/Recommendation: If a proposal gets canceled during the voting period, there should be a mechanism for a the users who have casted votes for the proposal to be able to reclaim the used voting powers and reuse those powers for other proposals. To implement the above, one should introduce a storage parameter:

```
mapping(uint256 proposalId => address voter => uint256) votingPowerUsed;
```

and for canceled `proposalId`, the voters should be able to reclaim their used voting power. This would also mean that perhaps the `projects` storage values for canceled proposals would also need to be updated.

`votingPowerUsed` could replace the boolean flag `hasVoted` since non-zeroness of `votingPowerUsed` indicate that the user has voted for that proposal id. And reseting `votingPowerUsed` does not create an issue for canceled proposals since one cannot cast a vote for a cancelled proposal.

Octant: Acknowledged.

Cantina Managed: Acknowledged. It is intended that buying voting power and voting are binding to prevent attacks, allowing for clawbacks on proposals that get cancelled allows for perverse incentives to game the mechanism.

3.5.17 Time-based limits of `_maxRedeem` should not be allocated to a hook

Severity: Informational

Context: TokenizedAllocationMechanism.sol#L1115-L1129

Description: The time-based limits of `_maxRedeem` should not be allocated to be checked in `IBaseAllocationStrategy(address(this)).availableWithdrawLimit(shareOwner)`. Like other time-based limits they should be performed locally in the `TokenizedAllocationMechanism` contract and if there are any other extra checks that need to be imposed they can be performed in the hook `availableWithdrawLimit`.

Recommendation: Make sure to perform time-based limits in the `TokenizedAllocationMechanism` contract itself and also update to the specification document to reflect this change. For a more in-depth analysis refer to "Mechanism timeline and state machine" where the one would need to redesign and change the proposal states and the global contract phase to reflect all these requirements.

Octant: Acknowledged. This is intentional to allow for more flexibility to implementers.

Cantina Managed: Acknowledged.

3.5.18 The balance check in `_notifyRewardAmountWithCustomDuration` does not account for unclaimed rewards

Severity: Informational

Context: RegenStakerBase.sol#L322-L325

Description: The balance check in `_notifyRewardAmountWithCustomDuration` does not account for unclaimed rewards. Although this is an inherent issue also with the inherited `Staker` implementation, one can easily include the unclaimed rewards in the check.

Recommendation: Define a total unclaimed scaled reward parameter for the contract and update this value accordingly per deltas applied to individual `deposit.scaledUnclaimedRewardCheckpoint` so that:

$$A_r^{un} = \sum_{i \in D} a_{r,i}^{un}$$

and call A_r^{un} `totalScaledUnclaimedRewardCheckpoint` then one needs to enforce:

$$A_{tot} + A_r^{un} \leq 10^{36} \cdot B_{T_R,\text{this}}$$

i.e.:

```
if (totalRewards + totalScaledUnclaimedRewardCheckpoint> (REWARD_TOKEN.balanceOf(address(this)) *
↪  SCALE_FACTOR)) {
   revert Staker__InsufficientRewardBalance();
}
```

Octant: Acknowledged.

Cantina Managed: Acknowledged.

3.5.19 Add `nonReentrant` modifier to `notifyRewardAmount` and `bumpEarningPower`

Severity: Informational

Context: RegenStakerBase.sol#L699-L701

Description: Almost all endpoints in `RegenStakerBase` have the `nonReentrant` modifier, except:

- `notifyRewardAmount`.
- `bumpEarningPower`.

Recommendation: It might be helpful to add this modifier to the above 2 endpoints to avoid potential issues in the future. Also note that in the `Staker` implementation of `bumpEarningPower` we have:

```
function bumpEarningPower(/*...*/) external virtual {
  // ...

  SafeERC20.safeTransfer(REWARD_TOKEN, _tipReceiver, _requestedTip);
  deposit.scaledUnclaimedRewardCheckpoint =
     deposit.scaledUnclaimedRewardCheckpoint - (_requestedTip * SCALE_FACTOR);
}
```

i.e., call to the external contract `REWARD_TOKEN` then the storage `deposit.scaledUnclaimedRewardCheckpoint` parameter is update which should be avoided, thus adding the `nonReentrant` modifier to this endpoint would help with potential re-entrency with reward tokens that have a callback on transfer. Although it might not create issues since `deposit.scaledUnclaimedRewardCheckpoint` only gets decremented and thus total deducted would be bounded above by the original value. But still even after adding the `nonReentrant` modifier, there are possible cross-contract view re-entrancy issues. It would be best to override `bumpEarningPower` and fix the above pattern:

```
function bumpEarningPower(/*...*/) external virtual {
  // ...

  deposit.scaledUnclaimedRewardCheckpoint =
     deposit.scaledUnclaimedRewardCheckpoint - (_requestedTip * SCALE_FACTOR);
  SafeERC20.safeTransfer(REWARD_TOKEN, _tipReceiver, _requestedTip);
}
```

Octant: Fixed in commit 43f7ba1b.

Cantina Managed: Fix verified.

3.5.20 Regen Reward Accumulation

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Let's denote the state of the system by Q and the set of actions (our state transition functions) by $\{S_0, S_1, \cdots\}$ and the set of all words in with this alphabet by $\mathcal{W}$, then we can apply these state transitions to end up at different state WQ where $W \in \mathcal{W}$. Starting with a state Q we can end up at:

$$Q \to S_{i_0}Q \to S_{i_1}S_{i_0}Q \to \cdots$$

One can think of the alphabet set as the endpoints of the contract for example, or one can even refine that to state transitions performed per `OPCODE` in those endpoints. There are even more nuances, but the above notations should give us enough terminology to continue although there might still be a slight ambiguity in the notation. Let's assume Q_{in} is the initial state of the system. We might sometimes write WQ_{in} as just W. Let $W = S_{i_n} \cdots S_{i_1}S_{i_0}$ and $W_{j,k} = \prod_{l=j}^{k} S_{i_l}$ (a sub-word) then (roughly):

$$ch_G(W) = \sum_{j=0}^{n} \left\lfloor \frac{\Delta t_j \cdot r_s(W_{0,j})}{\texttt{pow}_{tot}(W_{0,j})} \right\rfloor$$

and

$$r_S(W) = \left\lfloor \frac{(A_r(W') + a(W')) \cdot f_s}{\Delta t_p(W')} \right\rfloor$$

Thus:

$$[r_S] = \frac{[T_R] \cdot f_s}{[t]}$$

and (in the current case of earning power calculator we have $[\texttt{pow}] = [T_S]$).

$$[ch_G] = \frac{[T_R] \cdot f_s}{[\texttt{pow}]} = \frac{[T_R] \cdot f_s}{[T_S]}$$

Let's focus on the terms for $ch_G(W)$, if one could make it so that the delta timestamps Δt_j would end up small all the time (ie even in $\{0, 1\}$) then the terms in that summation would have the following form:

$$\left\lfloor \frac{\Delta t_j \cdot r_s(W_{0,j})}{\texttt{pow}_{tot}(W_{0,j})} \right\rfloor = \left\lfloor \frac{r_s}{\texttt{pow}_{tot}} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{A_r \cdot f_s}{\Delta t_p} \right\rfloor}{\texttt{pow}_{tot}} \right\rfloor \leq \left\lfloor \frac{A_r \cdot f_s}{\texttt{pow}_{tot} \cdot \Delta t_p} \right\rfloor$$

Now, depending on the values of A_r, $\texttt{pow}_{tot}$ and Δt_p these summation terms might end up being 0. For example when T_R has low decimal precisions (6 for `UDSDC`), T_S has high decimal precision (24 for `NEAR`) and A_r is smallm and $\texttt{pow}_{tot}$ and Δt_p are big. Thus in this case reward tokens allocated would not get distributed to the owners or the claimers of the deposits.

Moreover, frequent updates to ch_G in general even if the terms might not be 0 introduce more errors in the total reward tokens that need to be distributed, this is due to the fact that:

$$\sum_{i=0}^{m} \lfloor a_i \rfloor \leq \left\lfloor \sum_{i=0}^{m} a_i \right\rfloor$$

- Attack to diminish the value of ch_G:
  1. Stake the minimum amount to create a deposit with id i.
  2. Withdraw the staked amount from the deposit with id i. At this point we still have a deposit with an owner, claimer and delegatee, 0 balance and probably 0 earning power.
  3. Call either `claimRewards`, `alterDelegatee` or `alterClaimer` (the last two can also be called when the contract is paused) to trigger a call to `_checkpointGlobalReward()` so that ch_G and t_{ch}^{last} would get updated. Keep doing this for every block or as much as possible (spam the contract to force update the ch_G and t_{ch}^{last} values).
- Effect of calling claim reward multiple times vs. once: Like the above case for ch_G if users reduce the number of calls they make to claim their rewards through different endpoints, they would introduce less errors into the total value of claimed rewards through the life-cycle of the deposit.
- Effect of splitting a staking amount into different deposits: Splitting the deposit into mutiple different sub-deposits introduce more errors when claiming rewards just like the above cases. If possible it would be best to merge all the to-be-staked sub-deposits into a bigger main deposit to avoid error accumulation through splitting.

Recommendation: The above cases need to be documented. It is also important to pay attention to:

1. Decimal differences between T_S and T_R.
2. Picking Δt_p, higher values could affect the errors in ch_G.
3. The amounts provided to `notifyRewardAmount`. Small amounts used with this endpoint can amplify the ch_G issue.
4. Advise users to avoid splitting deposits if possible.
5. For the attack described above there are not whitelist checks on any of the users (the deposit owner or claimer), thus there is not an admin switch to toggle to stop that attack. Unless one also add an allowlist checks for the above mentioned endpoints in step 3. of the attack.

Notations:

parameter	description
ch_G	`rewardPerTokenAccumulatedCheckpoint`
t_{ch}^{last}	`lastCheckpointTime`
r_s	`scaledRewardRate`
$\texttt{pow}_{tot}$	`totalEarningPower`
Δt_j	the time difference between when S_{i_j} and $S_{i_{j-1}}$ were applied
T_S	`STAKE_TOKEN`
T_R	`REWARD_TOKEN`
f_s	`SCALE_FACTOR`
Δt_p	`REWARD_DURATION`
W'	refers to the last time `notifyRewardAmount` gets called before W
$A_r(W')$	`_remainingReward` when the last time `notifyRewardAmount` gets called before W
$a(W')$	`_amount * SCALE_FACTOR` when the last time `notifyRewardAmount` gets called before W
a_{min}	`minimumStakeAmount`
$[X]$	unit and precision of X

Octant: Acknowledged.

Cantina Managed: Acknowledged.

3.5.21 Accounting and Connectivity Analysis of the Payment Splitter

Severity: Informational

Context: PaymentSplitter.sol#L30

Description: If the initial state of the system is Q_{in} and the current state is $Q_f = S_n \cdots S_1 S_0 Q_{in}$. For each user $u \in U$, there is an index j_u such that:

$$r_{T,u} = \left\lfloor \frac{s_u \cdot C_T(Q_{j_u})}{S} \right\rfloor$$

and if `release` would about to be called for this user, we would end up with:

$$r_{T,u} = \left\lfloor \frac{s_u \cdot C_T(Q_f)}{S} \right\rfloor$$

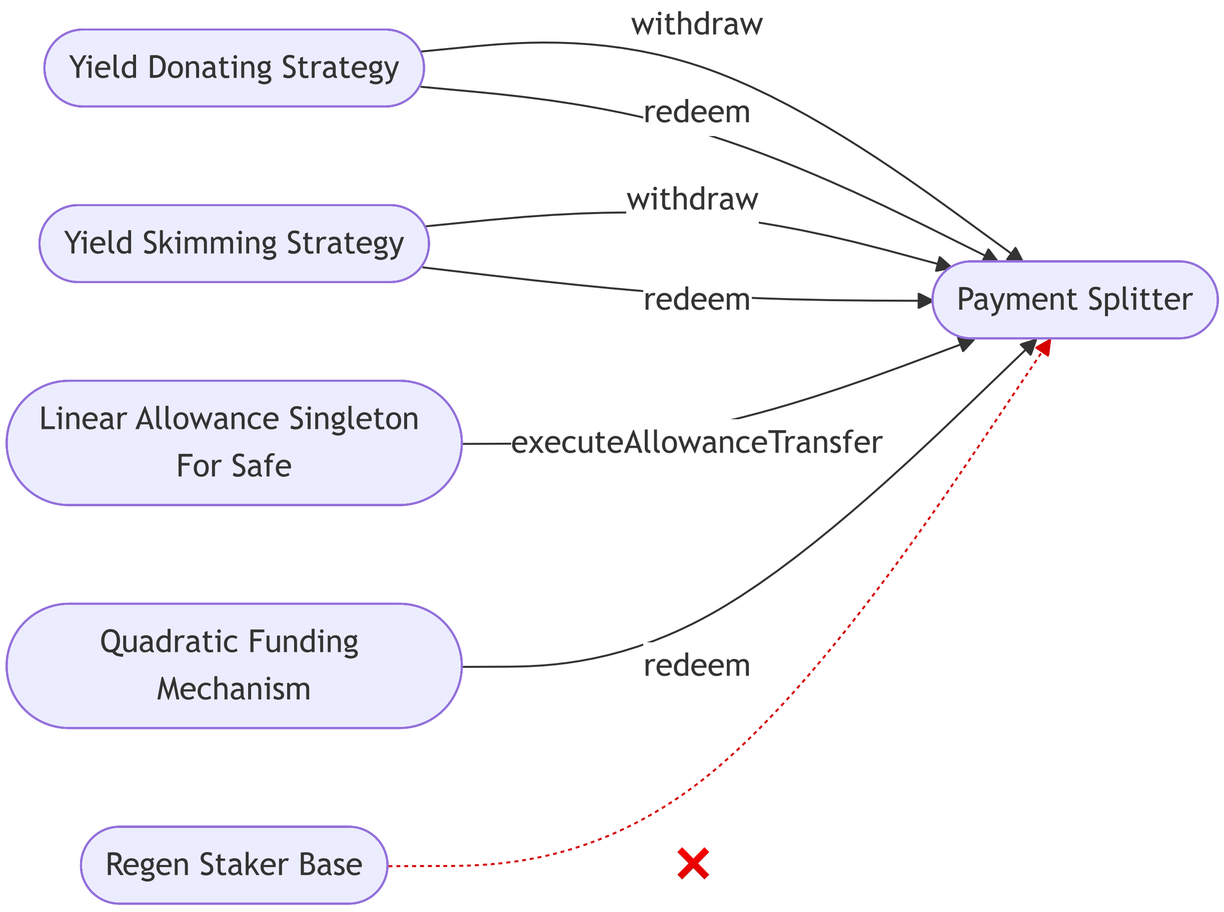
(total amount of tokens T transferred to this user from `PaymentSplitter`). As one can see calling the `release` many times would not at the end accumulate errors introduced at each call. Otherwise one could have unfairly diminish a user's released amount to 0 for some edge cases.

Moreover one can see that if all the release amounts for all users were up to date we would get:

$$C_T(Q_f) \geq \sum_{u \in U} \left\lfloor \frac{s_u \cdot C_T(Q_f)}{S} \right\rfloor$$

And the maximum of the error (difference between the two sides) would be at most the number of users $|U|$.

Connectivity: Part of the current codebase structure is to keep things modular such that different components can be connected together. We can see that:



One cannot directly connect `RegenStakerBase` to `PaymentSplitter` as the `withdraw` and `claimRewards` endpoints do not have an optional `receiver` parameter in `RegenStakerBase`.

Recommendation: If direct connectivity is desired for all components, one can optionally provider an overloaded custom `withdraw` and `claimRewards` endpoints with an optional `receiver` parameter.

The the codebase `PaymentSplitter` acts as a passive contract that does not call into other contracts except when transferring tokens. Another solution would be that potentially this contract would be able to call into specific endpoints, but that would make it more customised and thus perhaps more suited for a child contract.

Octant: Acknowledged. The `RegenStakerBase` requires active withdrawal.

Cantina Managed: Acknowledged.

Notations:

parameter	description
B_T	`token.balanceOf(address(this))`
R_T	`_erc20TotalReleased[token]`
s_u	`_shares[account]`, shares of user u
S	`_totalShares`
$r_{T,u}$	`_erc20Released[token][account]`
C_T	$B_T + R_T$ this quantity **only** increases. Increase happens upon transfers to the `PaymentSplitter`. `release` does not change the value of this quantity. C_T represents the sum of all historical amounts transferred to the `PaymentSplitter` contract
U	the set of payees, `_payees`
Q_j	$S_j \cdots S_1 S_0 Q_{in}$

3.5.22 Deploying donated assets using yield donating morpho strategy can amplify losses

Severity: Informational

Context: TokenizedStrategy.sol#L890-L894

Description: In general, deploying any donated asset when using the `MorphoCompounderStrategy` with `YieldDonatingTokenizedStrategy` can amplify potential to-be-incurred losses.

```
// We can deploy the full loose balance currently held.
IBaseStrategy(address(this)).deployFunds(_asset.balanceOf(address(this)));
```

The above lines does not affect the current yield skimming strategies (as it would be a NO-OP) and the `SkyCompounderStrategy` since staked `USDS` assets would remain in 1:1 relation. The `MorphoCompounderStrategy` is a few degree of separation away from the actual `Morpho` market contract:

`OctantVault` $\rightarrow$ `YearnVault` $\rightarrow$ `MorphoVault` $\rightarrow$ `Morpho`.

And the above imposed donation deployment (and therefore potential amplified loss) can happen along the way in the above flow.

Recommendation: It would be best to document and analyse the above scenario, perhaps have monitoring setup in-case of to-be-incurred loss to stop deposits/mints and avoid injecting donated funds in that case.

Cantina Managed: Comments have been added in commit d006b1a9.

Cantina Managed: Fix verified.

3.5.23 Infinite approvals vs atomic ones in `SkyCompounderStrategy`

Severity: Informational

Context: SkyCompounderStrategy.sol#L64-L65

Description: `SkyCompounderStrategy` inherits from `UniswapV3Swapper` which only performs atomic approvals in:

- `_swapFrom`
- `_swapTo`: This endpoint still does not clear any unused leftover approval at the end of the flow.

But in `SkyCompounderStrategy`'s `constructor` one is giving:

- Unlimited approval of `USDS` token to the `staking` contract.

- Unlimited approval of the reward token (`SKY`) to the `UNIV2ROUTER`.

Recommendation: To unify the approval approach and use the atomic method which would prevent from potential token drainage incase the staking or the router contracts would have a bug, one can use the `_checkAllowance` in both:

- `_uniV2swapFrom`.
- `_deployFunds`.

Although this conservative approach would incur more gas when those two flows are performed.

Also make sure to clear any unused approval in `_swapTo` flow for `UniswapV3Swapper` (this internal function is currently unused).

Octant: Fixed in several commits.

Cantina Managed: Recommendation has been applied in commit 389a315b except for:

> Also make sure to clear any unused approval in `_swapTo` flow for `UniswapV3Swapper` (this internal function is currently unused).

which has been applied in commit 0745cf7e using:

```
ERC20(_from).approve(router, 0)
```

It is still recommended to use `forceApprove` since `approve` requires for the `_from` token to return a `bool` which might not be the case in general since some tokens might not return an `bool`.

`forceApprove` has been used in 54bac781