Nashpoint: smart contracts

Security Review

Cantina Managed review by:

Gerard Persoon, Lead Security Researcher

Kurt Barry, Lead Security Researcher
Jonatas Martins, Associate Security Researcher

February 10, 2025

Contents

1 Introduction **3**
1.1 About Cantina . . . 3
1.2 Disclaimer . . . 3
1.3 Risk assessment . . . 3
1.3.1 Severity Classification . . . 3

2 Security Review Summary **4**

3 Findings **5**
3.1 Medium Risk . . . 5
3.1.1 Routers can access components from other routers as well as other tokens . . . 5
3.1.2 Inflation attack to steal first deposits . . . 6
3.1.3 `_getInvestmentSize()` doesn't retrieve the asset balance correctly . . . 7
3.1.4 Missing slippage control for ERC4626 deposits and withdrawals in the underlying vaults 7
3.1.5 In `_getSwingFactor()`: `reserveImpact >= targetReserveRatio` not handled correctly 7
3.1.6 `_enforceLiquidationQueue()` checks shares in the wrong location . . . 8
3.1.7 `Component asset` could be different than `node asset` . . . 8
3.2 Low Risk . . . 9
3.2.1 Edge cases with `rebalanceCooldown` and `rebalanceWindow` . . . 9
3.2.2 Balance check of `removeComponent()` not robust . . . 9
3.2.3 Old `escrow` might still contain tokens . . . 10
3.2.4 `protocolFeeAddress` could still be `0` if not configured yet . . . 10
3.2.5 `fulfillRedeem` could be unfair . . . 10
3.2.6 Two powerful functions could be combined to reduce impact . . . 10
3.2.7 Functions `deposit()` and `mint()` use different ways to check the maximum assets to be deposited . . . 11
3.2.8 Duplicate `emits` in `deposit()` and `mint()` . . . 11
3.2.9 Incorrect rounding . . . 12
3.2.10 Missing boundaries check for important state-changing functions . . . 12
3.2.11 The quoter doesn't verify whether the node is using same quoter address . . . 13
3.2.12 The `approveMax()` and `unapprove()` functions could be used maliciously by the node owner . . . 13
3.2.13 Invalid Component Allocations Halt Node Rebalancing . . . 13
3.2.14 Node Creation Can Be Frontrun . . . 14
3.2.15 `ERC4626Router.fullfilRedeemRequest()` Rounds in Favor of the User . . . 14
3.2.16 `ERC4626Router.invest()` Is Documented to Return Assets Deposited, but Returns Shares Created Instead . . . 15
3.2.17 `Node.redeem()` Is Documented to Return Assets Redeemed but Returns Shares Redeemed Instead . . . 15
3.2.18 Quoter can be set without validation in the Registry contract . . . 16
3.2.19 `calculateDepositBonus()` doesn't take into account `sharesExiting` . . . 16
3.2.20 The `mint()` function lacks swing price bonus calculation . . . 16
3.2.21 Missing validations in `_approve()` . . . 16
3.2.22 Wrong location of `_approve()` . . . 17
3.2.23 Missing validation for ERC-7540 asynchronous functionality interfaces . . . 17
3.2.24 Round down in `executeAsyncWithdrawal()` could lead to reverts . . . 17
3.2.25 `_enforceLiquidationQueue()` could fail due to one of the underlying `components` failing 18
3.2.26 Function `_liquidate()` doesn't check the actual `assets` received . . . 18
3.2.27 The node contract could contain tokens that are not actively used . . . 18
3.2.28 Withdrawal Penalties Can Be Reduced Via Splitting and Fail to Cover Deposit Bonuses 18
3.3 Gas Optimization . . . 22
3.3.1 `decimals` can be precomputed . . . 22
3.3.2 The `value` Parameter of `Node.execute` Sends Ether But the Contract Has No `payable` Methods . . . 22
3.3.3 Redundant `view` Functions in Node Contract . . . 22
3.3.4 Redundant Check in `ERC4626Router` . . . 23
3.3.5 `_isComponent()` isn't efficient . . . 23
3.3.6 `_decimalsOffset()` related code can be simplified . . . 24
3.3.7 Internal variables in `ERC7540Router` can be moved to function scope . . . 24

3.3.8 Optimize shares to assets conversions . . . 25
3.3.9 Optimize if-statement in invest() . . . 25
3.4 Informational . . . 25
3.4.1 MAX_DEPOSIT can't be changed . . . 25
3.4.2 immutable could be constant . . . 25
3.4.3 Illogical order of statements in 'addRebalancer() . . . 26
3.4.4 setLiquidationQueue() and _setInitialComponents() could contain duplicate components . . . 26
3.4.5 Function enableSwingPricing() doesn't emit maxSwingFactor_ . . . 26
3.4.6 rebalanceCooldown period runs in pallallel with the rebalanceWindow . . . 27
3.4.7 Subtractions and additions can be simplified . . . 27
3.4.8 return in function payManagementFees() is not necessary . . . 27
3.4.9 Function subtractProtocolExecutionFee() doesn't check there is sufficient balance . 28
3.4.10 QuoterV1 knows too many implementation details . . . 28
3.4.11 In function mint(), the return parameter assets isn't used . . . 29
3.4.12 Both msg.sender and _msgSender()' are used . . . 29
3.4.13 Typo errors . . . 29
3.4.14 Missing Natspec . . . 29
3.4.15 Unnecessary Casts . . . 30
3.4.16 NodeRegistry code simplification . . . 30
3.4.17 BaseRouter and BaseQuoter can be defined abstract . . . 30
3.4.18 _safeApprove() can fail with tokens that implement approval race protection . . . 31
3.4.19 Unused events . . . 31
3.4.20 The node parameter is redundant in some events . . . 31
3.4.21 Inconsistent event naming . . . 31
3.4.22 Underflows in Node Cash Calculations in BaseRouter . . . 32
3.4.23 Duplicated Investment Logic Can Be Moved to BaseRouter . . . 32
3.4.24 Not all relevant functions support partial fulfill . . . 33
3.4.25 Ambiguous evaluation in _calculateSharesAfterSwingPricing() . . . 33
3.4.26 Functions _getInvestmentSize() in ERC4626Router and ERC7540Router are very similar 33
3.4.27 The liquidationsQueue processing is not generic . . . 34
3.4.28 Checks Not Using Errors From ErrorsLib.sol . . . 34
3.4.29 Use constants instead of magic numbers . . . 34
3.4.30 Missing maxDeposit() and maxRedeem() validation . . . 35
3.4.31 The owner of the registry can block all nodes . . . 35
3.4.32 onlyRegistryOwner doesn't check the quoter is still a valid quoter . . . 35
3.4.33 Check isComponent() is done on internal and external functions . . . 36
3.4.34 _validateReserveAboveTargetRatio() is similar to _getNodeCashStatus() . . . 36
3.4.35 Functions calculateRedeemPenalty() and calculateDepositBonus() could be generalized . . . 36
3.4.36 target versus component versus vault . . . 37
3.4.37 encodeWithSelector() vs. encodeCall() . . . 37

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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

NashPoint is pioneering a new on-chain banking system and establishing the operating system for global credit markets, bridging traditional finance and DeFi. By empowering credit managers to create onchain financial institutions, NashPoint forms the backbone of an interconnected financial ecosystem-enabling seamless capital allocation across DeFi and Real World Asset (RWA) opportunities.

From Jan 13th to Jan 23rd the Cantina team conducted a review of nashpoint-smart-contracts on commit hash 0c249305. The team identified a total of **81** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	7	5	2
Low Risk	28	24	4
Gas Optimizations	9	9	0
Informational	37	36	1
Total	**81**	**74**	**7**

3 Findings

3.1 Medium Risk

3.1.1 Routers can access components from other routers as well as other tokens

Severity: Medium Risk

Context: BaseRouter.sol#L174-L178, Node.sol#L45, Node.sol#L303-L313, QuoterV1.sol#L29-L30, QuoterV1.sol#L280-L285

Description: Currently `routers` can access `components` from other `routers`. This means if one router contains a bug or a vulnability it could access all funds of all routers in all nodes. This is especially important if other parties would be allowed to create routers in the future.

Note: the code currently assumes `components` are not shared between routers, as keen be seen in `_getTotalAssets()` of `QuoterV1`.

Via `execute()` routers can also access:

- Tokens from `escrow()`, which already belong to controllers.
- The `asset` token from `node`.

Recommendation: Consider isolating the `routers` so they can only access their own components. This can be done in the following way:

- Add `TypeOfVault`: Define an `enum` that identifies the types of `components` supported by the `router`. This is also useful when replacing `routers` with newer version. For example:

```
enum TypeOfVault {
    none,    // 0 - don't use, to avoid confusion with default values
    ERC4626, // 1
    ERC7540  // 2
}
```

- Store the `TypeOfVault` in the Node: Store the `TypeOfVault` per Router in the Node. The best is a mapping in two direction to every combination can be retrieved. Note: the `enum` is not used directly in the mappings to allow future update without changing `Node.sol`.

```
- mapping(address => bool) public isRouter;
+ mapping(address router => uint256 tov) public Router2TOV;
+ mapping(uint256 tov => address router) public TOV2Router;
```

  Also add functions to read and update these value, while keeping them in sync.

- Consistency components with routers in Node: When adding/removing components and routers check that the components and routers are consistent via the `TypeOfVault` that is stored in `QuoterV1`. Also check the components in `liquidationsQueue` stay consistent.
- Update Router: Update the Router to support a specific `TypeOfVault` and add a function to indicate that. Also update the `modifiers`.
- Update `QuoterV1`: Update `QuoterV1` to store a specific `TypeOfVault` per component.

```
- mapping(address => bool) public isErc4626;
- mapping(address => bool) public isErc7540;
+ mapping(address component=> uint256 tov) public Component2TOV;
```

- Update `NodeRegistry`: Consider updating `NodeRegistry` to also store the `TypeOfVault` with the router.
- Restrict `execute()`: Restrict the call to `execute()` so the router can only execute functions of its own components:

```
   function execute(address target, uint256 value, bytes calldata data) ... {
       // ...
+      uint256 TOV=Router2TOV[msg.sender];
+      if (TOV==0 || TOV != quoter.Component2TOV[target])
+          revert ErrorsLib.NotOwnComponent();
       bytes memory result = target.functionCallWithValue(data, value);
       // ...
   }
```

_Note: doublecheck no other `execute()` to other addresses will be required in the future. If so, add an allowlist mechanism to allow that. See the recommendation of finding: Two powerful functions could be combined to reduce impact for a suggestion for `_transferToEscrow()`.

_Note: doublecheck if a token recovery mechanism is required, see finding: The node contract could contain tokens that are not actively used.

NashPoint: Acknowledged.

Cantina Managed: Acknowledged.

3.1.2 Inflation attack to steal first deposits

Severity: Medium Risk

Context: Node.sol#L347-L349

Description: The `updateTotalAssets()` function, callable by the owner or rebalancer, can be exploited to perform an inflation attack against initial depositors. This vulnerability exists because the function updates the contract's total assets and stores it in the `cacheTotalAssets` variable. A malicious node owner can front-run the first deposit through these steps:

- Deposit 1 asset and mint 1 share.
- Transfer assets directly to the node.
- Call `updateTotalAssets()` to update the `cacheTotalAssets`.

Consequently, when the first legitimate depositor calls `deposit()`, they receive 0 shares because the share price has been artificially inflated.

Proof of Concept: Using the `test/unit/Node.t.sol` as reference.

```
function test_inflationAttack() public {
    uint256 assets = 100e18;

    deal(address(asset), address(user), assets);
    deal(address(asset), address(owner), 3*assets);

    //@audit Front-run the first deposit
    vm.startPrank(owner);
    asset.approve(address(node), 1);
    node.deposit(1, owner);
    asset.transfer(address(node), 2*assets);
    node.updateTotalAssets();
    vm.stopPrank();

    vm.startPrank(user);
    asset.approve(address(node), assets);
    node.deposit(assets, user);
    vm.stopPrank();

    //@audit The owner still have 1 share and the user didnt get any share
    assertEq(node.balanceOf(owner), 1);
    assertEq(node.balanceOf(user), 0);
}
```

Recommendation: Consider implementing the OZ standard solution, which requires depositing a non-trivial amount of assets to make price manipulation infeasible. See ERC4626.sol#L22-L28.

Note that removing `updateTotalAssets()` won't resolve this issue, as the owner can achieve the same effect by calling `payManagementFees()`.

NashPoint: Acknowledged.

Cantina Managed: Acknowledged.

3.1.3 `_getInvestmentSize()` doesn't retrieve the asset balance correctly

Severity: Medium Risk

Context: QuoterV1.sol#L242-L246, ERC4626Router.sol#L179-L192

Description: The function `_getInvestmentSize()` in `ERC4626Router` doesn't retrieve the asset balance correctly. This means investments can't be done effectively. The `balanceOf()` value is in shares and needs to be converted to assets, similar to `QuoterV1::_getErc4626Assets()`.

Recommendation: Consider changing the code to:

```
- uint256 currentBalance = IERC20(component).balanceOf(address(node));
+ uint256 currentBalance = IERC4626(component).convertToAssets(IERC20(component).balanceOf(address(node)));
```

Also consider combining this function with `QuoterV1::_getErc4626Assets()`.

NashPoint: Fixed in PR 230.

Cantina Managed: Fix verified.

3.1.4 Missing slippage control for ERC4626 deposits and withdrawals in the underlying vaults

Severity: Medium Risk

Context: ERC4626Router.sol#L78-L83, ERC4626Router.sol#L213-L214

Description: The functions `liquidate()`, `fulfillRedeemRequest()` and `invest()` in `ERC4626Router` have validation for depositing and withdrawing from the underlying vault, but this validation does not protect against slippage. A sandwich attack could manipulate share values in the vault, causing it to return fewer assets than expected while still passing the current validation.

Note: there is also no slippage protection on `node` functions, but there is no good way to implement this within the `ERC4646` standard.

Note: there is also no slippage protection on the `ERC7540Router` but there is no good way to implement this due the asynchronous nature and the `ERC7540standard`.

Recommendation: Consider adding a slippage parameter `liquidate()`, `invest()` and `fulfillRedeemRequest()` to protect against share price manipulation. The expected number of `assets()`/`shares()` can be supplied and this can be compared to received amount.

NashPoint: Fixed in PR 244.

Cantina Managed: Fix verified.

3.1.5 In `_getSwingFactor()`: `reserveImpact >= targetReserveRatio` not handled correctly

Severity: Medium Risk

Context: QuoterV1.sol#L90-L107, QuoterV1.sol#L214-L226

Description: The situation of `reserveImpact >= targetReserveRatio` is not handled correctly in function `_getSwingFactor()`. The return of 0 results in a 100% bonus, which is not as intended.

Recommendation: Change the code to something like this, please check if this is what it intended:

```
  function calculateDepositBonus(/*...*/) /*...*/ {
      uint256 reserveCash = // ...
      int256 reserveImpact = // ...
+     if (uint256(reserveImpact) >= targetReserveRatio)
+         uint256 adjustedAssets = assets;
+     else
          uint256 adjustedAssets =MathLib.mulDiv(/*...*/);

      uint256 sharesToMint = IERC7575(msg.sender).convertToShares(adjustedAssets);
      return (sharesToMint);
  }
  function _getSwingFactor(/*...*/) /*...*/ {
      if (reserveImpact < 0) {
          revert ErrorsLib.InvalidInput(reserveImpact);
-     } else if (uint256(reserveImpact) >= targetReserveRatio) {
-          return 0;
      } else {
          SD59x18 reserveImpactSd = // ...
          SD59x18 result = // ...
          return uint256(result.unwrap());
      }
  }
```

NashPoint: Fixed in PR 231.

Cantina Managed: Fix verified.

3.1.6 `_enforceLiquidationQueue()` checks shares in the wrong location

Severity: Medium Risk

Context: BaseRouter.sol#L184-L190

Description: Function `_enforceLiquidationQueue()` checks the amount `shares` in the `router` contract. However the `shares` are located in the `node`. This disables the entire `liquidationsQueue` mechanism.

Recommendation: Change the code to:

```
- uint256 candidateShares = IERC20(candidate).balanceOf(address(this));
+ uint256 candidateShares = IERC20(candidate).balanceOf(address(node));
```

Note: it is important that the same `candidate` is not used in multiple `routers`. This currently is not the case.

NashPoint: Fixed in PR 241.

Cantina Managed: Fix verified.

3.1.7 `Component asset` could be different than `node asset`

Severity: Medium Risk

Context: BaseRouter.sol#L68, BaseRouter.sol#L77, QuoterV1.sol#L59-L66, ERC4626Router.sol#L156-L157, ERC7540Router.sol#L168-L169

Description: Functions `ERC4626Router::_deposit()` and `ERC7540Router::_requestDeposit()` rely on the component to retrieve the `asset` address. It is never checked this `asset` is the same as the `node asset`. There could be a difference due to a mistake or the `component` could be malicious and return a wrong `asset` on purpose.

Recommendation: In `ERC4626Router::_deposit()` use the `node.asset()`:

```
- IERC4626(vault).asset()
+ INode(node).asset())
```

In `ERC7540Router::_requestDeposit()` also use the `node.asset()`:

```
- IERC4626(component).asset()
+ INode(node).asset())
```

When adding components, check the `component.asset() == node.asset()`. This could be done one or both of these:

- `setErc4626()` / `setErc7540()`.
- `setWhitelistStatus()` / `batchSetWhitelistStatus()`.

To be optimally flexible use a virtual function so a router can overrride if necessary.

NashPoint: Fixed in PR 237. I added this check to node instead: `component.asset() == node.asset()`. Routers and quoters should not track any relationship between nodes and assets.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Edge cases with `rebalanceCooldown` and `rebalanceWindow`

Severity: Low Risk

Context: Node.sol#L112-L115, Node.sol#L118-L123, Node.sol#L137

Description: If `rebalanceCooldown < rebalanceWindow` then `onlyWhenRebalancing` is immediately true, which is probably unwanted. The `modifiers` `onlyWhenNotRebalancing` and `onlyWhenRebalancing` are each others inverses, but their code is different. If `block.timestamp == lastRebalance + rebalanceWindow` then both the `modifiers` `onlyWhenNotRebalancing` and `onlyWhenRebalancing`, which is not necessary.

Recommendation: Consider making the following changes:

```
  function initialize(address escrow_) external onlyOwner {
      // ...
-     lastRebalance = uint64(block.timestamp - rebalanceCooldown);
+     lastRebalance = uint64(block.timestamp - rebalanceWindow);
      // ...
  }
```

The modifier `onlyWhenNotRebalancing` can then be simplified and <= can be replaced with <:

```
  modifier onlyWhenNotRebalancing() {
-     if (block.timestamp >= lastRebalance && block.timestamp <= lastRebalance + rebalanceWindow) {
+     if (block.timestamp < lastRebalance + rebalanceWindow) {
      // ...
  }
```

NashPoint: Fixed in PR 176.

Cantina Managed: Fix verified.

3.2.2 Balance check of `removeComponent()` not robust

Severity: Low Risk

Context: Node.sol#L159-L161

Description: The balance check of `removeComponent()` is not robust:

- Only the `ERC4626` way of checking the balance is implemented, this is not fully compatible with `ERC7540` and also not compatible with future components.
- There could be dust left that is too small to redeem.
- Someone could frontrun `removeComponent()` and send a small amount of `component` shares to prevent removing it.
- A broken or malicious component could revert in `balanceOf()`.

Recommendation: Consider doing the following:

- Use the equivalent of `quoter.getErc7540Assets()` to check for `ERC7540` assets, preferably via a more generic name. Keep in mind that the `quoter` could also have removed support for this component.
- Have a boolean `force` parameter, that can be optionally used to skip the balance check.

NashPoint: Fixed in PR 240.

Cantina Managed: Fix verified.

3.2.3 Old `escrow` might still contain tokens

Severity: Low Risk

Context: Node.sol#L227

Description: If an `escrow` is removed, it could potentially still contain `assets` or `node shares`. It also still could contain allowances to the `node`.

Recommendation: Consider doing one of the following:

- Check no `assets` or `node shares` are present before removing. As this could be limited by dust or frontrun attacks, a `force` parameter might be helpful.
- Have a way to salvage tokens from the `escrow` contract once it is no longer actively used. Snippet for check:

```
if (node.escrow() == address(this)) revert ErrorsLib.StillActivelyUsed();
```

Consider also removing the allowances, see finding The `approveMax()` and `unapprove()` functions could be used maliciously by the node owner.

NashPoint: Fixed in PR 217.

Cantina Managed: Fix verified.

3.2.4 `protocolFeeAddress` could still be `0` if not configured yet

Severity: Low Risk

Context: Node.sol#L335, Node.sol#L344

Description: In theory `protocolFeeAddress` could still be `0` if not configured yet. In that case the tokens would be lost.

Recommendation: Consider check for the value of `0`.

NashPoint: Fixed in PR 209. Added function calls to configure fees and address in `initialize()`.

Cantina Managed: Fix verified.

3.2.5 `fulfillRedeem` could be unfair

Severity: Low Risk

Context: Node.sol#L357-L361

Description: If multiple `controllers` want to `redeem()` and retrieving `assets` from the underlying vaults takes a long time (e.g. multiple days), then the order in which `_fulfillRedeemFromReserve()` is executed for the different `controllers` determines who gets their `assets` first. This might be unfair. Although the rebalancer is a trusted role, there could be multiple rebalancers with different priorities. The `liquidationsQueue` is a way to guide the `rebalancers` for liquidiations buts there is not comparable function for `fulfillRedeem`.

Recommendation: Consider adding a way to guide the `fulfillRedeem` actions. This could be in the same order that `redeem()` has been done. An alternative could be to give all open requests the same percentage of the assets availabe at that moment.

NashPoint: Acknowledged. I will add this in a future Node V2 contract.

Cantina Managed: Acknowledged.

3.2.6 Two powerful functions could be combined to reduce impact

Severity: Low Risk

Context: BaseRouter.sol#L174-L178, Node.sol#L364-L371, Node.sol#L634-L648, Node.sol#L650-L669, ERC4626Router.sol#L140-L144

Description: The function `BaseRouter::_transferToEscrow` and `node::finalizeRedemption()` each are pretty powerful.

- `BaseRouter::_transferToEscrow` move `assets` to the escrow;
- `node::finalizeRedemption()` burns `shares`, updates `sharesExiting`, `cacheTotalAssets` and updates `request[]`.

However when combining them they net out most each others effect. Normally they also should be executed together.

- The transfer of the `assets` is then done together with the `burn` of the `shares`.
- `cacheTotalAssets -= assetsToReturn;` is also done together with the transfer to the escrow.

Additionally there are differences between:

- `ERC4626Router.::fulfillRedeemRequest() + BaseRouter.sol::_transferToEscrow()` don't check `balance` and use `transfer`.
- `ERC4626Router::_fulfillRedeemFromReserve()` does check `balance` and uses `safeTransfer()`.

Recommendation: Consider combining `BaseRouter::_transferToEscrow` and `node::finalizeRedemption()` in one function in `node`. Inclused a `balance` check and use `safeTransfer()`.

Update `node::_fulfillRedeemFromReserve()` to also use this combined function, including moving the `balance` check.

Note: this recommendation is also important for the recommendation of finding: Routers can access components from other routers.

NashPoint: Fixed in PR 210.

Cantina Managed: Fix verified.

3.2.7 Functions `deposit()` and `mint()` use different ways to check the maximum assets to be deposited

Severity: Low Risk

Context: Node.sol#L433-L434, Node.sol#L445-L446

Description: The functions `deposit()` and `mint()` use different ways to check the maximum assets that can be deposited/minted.

Recommendation: Consider changing the code to:

```
  function deposit(/*...*/) /*...*/ {
-     if (assets > maxDeposit(msg.sender)) {
+     if (assets > maxDeposit(receiver)) {
      // ...
  }
```

NashPoint: Fixed in PR 211.

Cantina Managed: Fix verified.

3.2.8 Duplicate `emits` in `deposit()` and `mint()`

Severity: Low Risk

Context: Node.sol#L433-L440

Description: Function `_deposit()` does an emit of `Deposit` while `deposit()` and `mint()` also do this, to the emit is duplicated. Futhermore the emits in `deposit()` and `mint()` are not correct because they use `receiver` twice.

Recommendation: Consider changing the code to:

```
  function deposit(/*...*/) /*...*/ {
      /*...*/
      _deposit(_msgSender(), receiver, assets, sharesToMint); // also emit
      /*...*/
-     emit IERC7575.Deposit(receiver, receiver, assets, sharesToMint);
      /*...*/
}
  function mint(uint256 shares, address receiver) public returns (uint256 assets) {
      /*...*/
      _deposit(_msgSender(), receiver, assetsToDeposit, shares);
      /*...*/
-     emit IERC7575.Deposit(receiver, receiver, assetsToDeposit, shares);
      /*...*/
  }
```

NashPoint: Fixed in PR 212.

Cantina Managed: Fix verified.

3.2.9 Incorrect rounding

Severity: Low Risk

Context: Node.sol#L445-L449, Node.sol#L456-L465, Node.sol#L538-L539

Description: In several places the rounding direction is down, while it should be rounding up. These are the following location:

- `previewMint()` / `_convertToAssets(shares, MathLib.Rounding.Down);`.
- `withdraw()` / `shares = MathLib.mulDiv(assets, maxShares, maxAssets);`.
- `mint()` / `uint256 assetsToDeposit = convertToAssets(shares);`.

Check the OZ ERC4626.sol for reference.

Recommendation: Change the code to round up.

NashPoint: Fixed in PR 213.

Cantina Managed: Fix verified.

3.2.10 Missing boundaries check for important state-changing functions

Severity: Low Risk

Context: BaseRouter.sol#L155-L165, NodeRegistry.sol#L145-L154, Node.sol#L266-L270, Node.sol#L281-L284, Node.sol#L325-L327

Description and recomendation: Some functions doesn't have any validation for the input value, adding a boundary checks for maximum and minimum value when setting these state variables could prevent the owner from setting unrealistic values for them. Consider implementing the following changes:

1. The `protocolManagementFee` should be <= 100% (WAD), otherwise this will revert: Node.sol#L326.
2. When `protocolExecutionFee` is restricted to <= 100% (WAD) then the check can be removed in BaseRouter.sol#L161-163.

```
function _subtractExecutionFee(uint256 transactionAmount, address node) internal returns (uint256) {
    // ...
-     if (executionFee >= transactionAmount) {
-       revert ErrorsLib.FeeExceedsAmount(executionFee, transactionAmount);
-     }
    // ...
}
```

3. In the `Node` contract, the functions `enableSwingPricing()` should validate the `maxSwingFactor` parameter and `setAnnualManagementFee()` should validate the `newAnnualManagementFee` parameter.

NashPoint: Fixed in PR 214.

Cantina Managed: Fix verified.

3.2.11 The quoter doesn't verify whether the node is using same quoter address

Severity: Low Risk

Context: BaseQuoter.sol#L29-L32

Description: The modifier `onlyValidNode` allows access to all versions of the quoter to a node. Since there can be multiple versions of a quoter, the code should verify that the current quoter matches the one registered in the node through the function `node.quoter()`.

Recommendation: Consider adding the check for `node.quoter() == address(this)`.

COMMENTS:

NashPoint: Fixed in PR 215.

Cantina Managed: Fix verified.

3.2.12 The `approveMax()` and `unapprove()` functions could be used maliciously by the node owner

Severity: Low Risk

Context: Escrow.sol#L32-L43

Description: The functions `approveMax` and `unapprove` are critical for the `Escrow` contract as they handle token approval permissions. If the owner's account is compromised, these functions could be exploited in the following ways:

1. `approveMax`: Could be used to grant self-approval for withdrawing assets from the withdrawal queue.
2. `unapprove`: Could cause a denial of service for withdrawals if the Node loses approval to transfer from the Escrow.

Recommendation: The following changes are recommended:

- Make `approveMax()` an internal function and move it to the constructor or initializer, since it's only needed after Node and Escrow deployment.
- Restrict `unapprove()` to only be callable from `Node.setEscrow()` when updating to a new escrow, see finding: Old escrow might still contain tokens. Additionally, consider making the `asset` an immutable variable set in the constructor or initializer.

```
function unapprove() external onlyNodeOwnerOrNode {
    _safeApprove(node , node , 0);
    emit EventsLib.Approve(node , node , 0);
    _safeApprove(asset, node , 0);
    emit EventsLib.Approve(asset, node , 0);
}
```

NashPoint: Fixed in commit b122386b.

Cantina Managed: Fix verified.

3.2.13 Invalid Component Allocations Halt Node Rebalancing

Severity: Low Risk

Context: Node.sol#L144, Node.sol#L159, Node.sol#L178

Description: Node owners are able to update component allocations and add and remove components without validation that the allocation weights of all components plus the reserve allocation weight sum to 10^{18}. Instead, component weights are validated in `startRebalance()`, and updates to component allocations are blocked while rebalancing is occurring. This means that if a Node owner accidentally leaves Node in an invalid allocation state, rebalancing will be blocked until the error is corrected, which could negatively impact users of the protocol. Additional downsides of this design are that all future changes to the protocol must consider the possibility that component allocations may be inconsistent outside of rebalance windows, and a heavier gas burden on rebalancers (as opposed to owners, which would be the case if consistency were forced for all allocation state updates).

Recommendation: Consider enforcing consistency of allocation values when adding a component, removing a component, or updating an allocation. This will require passing in a full array of component

allocations + reserve allocation in each such operation, but this is likely more efficient anyway than making a separate transaction to update every call, as must be done presently. Note that if there was any intention to use this as a hidden "pause" feature (setting an invalid allocation to halt rebalances and thus withdrawals), that can already be accomplished by owners by simply removing all rebalancers via `removeRebalancer`; and explicit pause feature could be added as well.

NashPoint: Acknowledged.

Cantina Managed: Acknowledged.

3.2.14 Node Creation Can Be Frontrun

Severity: Low Risk

Context: NodeFactory.sol#L33-L45, NodeFactory.sol#L53-L89

Description: The `NodeFactory` contract exposes two function for creating a node, `deployFullNode()` and `createNode()` (and the latter calls the former internally, along with performing some secondary setup). The CREATE2 opcode is used to ensure a deterministic address, with the salt being a parameter to both functions. This means any node creation call could be frontrun to deploy a node at the same address in advance, causing the original transaction to fail.

There are 4 cases:

- `deployFullNode()` frontrun with `deployFullNode()`: the node is created exactly as intended. This could be disruptive to smart contracts that are deploying nodes as part of a function call, but otherwise isn't an issue.
- `deployFullNode()` frontrun with `createNode()`: the node is owned by the factory, and cannot be fully initialized. Any logic that relied on a functional node at the intended address is broken.
- `createNode()` frontrun with `createNode()`: the node is created as intended and owned by the intended owner. Potentially disruptive to node-creating smart contracts.
- `createNode()` frontrun with `deployFullNode()`: generally not possible as `deployFullNode` doesn't allow the initial owner to be set.

Recommendation: Add a dependence on `msg.sender` to the CREATE2 salt in some manner, for example by prepending `msg.sender` to the caller-supplied salt and hashing the result to generate a new salt.

NashPoint: Fixed in PR 219.

Cantina Managed: Fix verified.

3.2.15 `ERC4626Router.fullfilRedeemRequest()` Rounds in Favor of the User

Severity: Low Risk

Context: ERC4626Router.sol#L110-L138

Description: The following logic can be found in `ERC4626Router.fullfilRedeemRequest()`:

```
if (assetsReturned < assetsRequested) {
    uint256 percentReturned = MathLib.mulDiv(assetsReturned, WAD, assetsRequested);
    sharesPending = MathLib.mulDiv(sharesPending, percentReturned, WAD);
    sharesAdjusted = MathLib.mulDiv(sharesAdjusted, percentReturned, WAD);
}
```

Rounding down here favors the user because it means for a given amount of assets returned, fewer of their pending redemption shares will be burned. In an extreme case, it's possible for a user to receive non-zero assets but have zero shares eliminated. While this is unlikely to be exploitable here, it is a best practice to ensure rounding favors the protocol.

Recommendation: Round against the user; note that the intermediate variable `percentReturned` isn't necessary, allowing the computations to be done in fewer steps (resulting in less overall rounding error):

```
if (assetsReturned < assetsRequested) {
    sharesPending = min(sharesPending, MathLib.mulDiv(sharesPending,
        assetsReturned, assetsRequested, MathLib.Rounding.Up));
    sharesAdjusted = min(sharesAdjusted, MathLib.mulDiv(sharesAdjusted,
        assetsReturned, assetsRequested, MathLib.Rounding.Up));
}
```

The `min` function above can be understand to take the minimum of two values (it's purpose is to ensure that underflow is impossible as a result of rounding to a value higher than the original).

NashPoint: Fixed in PR 191.

Cantina Managed: Fix verified.

3.2.16 `ERC4626Router.invest()` Is Documented to Return Assets Deposited, but Returns Shares Created Instead

Severity: Low Risk

Context: ERC4626Router.sol#L38-L42, ERC4626Router.sol#L85

Description: The `invest()` function of `ERC4626Router` is documented in a comment to return the assets deposited, and even declares a return variable named `depositedAssets` which is set correctly during execution of the function:

```
/// @return depositAmount The amount of assets invested.
function invest(address node, address component)
    external
    onlyNodeRebalancer(node)
    onlyWhitelisted(component)
    returns (uint256 depositAmount)
```

However, the function concludes with an explicit return of the shares created by the deposit, overriding the declared return variable:

```
uint256 sharesReturned = _deposit(node, component, depositAmount);
if (sharesReturned < expectedShares) {
    revert ErrorsLib.InsufficientSharesReturned(component, sharesReturned, expectedShares);
}

return sharesReturned;
```

This could cause confusion for rebalancers calling this function and relying on the return value in some way.

Recommendation: Make the documented behavior consistent with the actual return value; returning both assets deposited and shares created, and even the rebalancer fee amount, may be useful.

NashPoint: Fixed in PR 220.

Cantina Managed: Fix verified.

3.2.17 `Node.redeem()` Is Documented to Return Assets Redeemed but Returns Shares Redeemed Instead

Severity: Low Risk

Context: Node.sol#L475, Node.sol#L490

Description: Via `INode.sol`, the `Node.redeem()` function is documented to return the assets redeemed:

```
/// @return assets The amount of assets received
function redeem(uint256 shares, address receiver, address controller) external returns (uint256);
```

The function also declares a return variable named `assets` that is set correctly. However, it overrides this return variable via an explicit statement that returns the amount of shares requested to be redeemed. This discrepancy could create confusion and difficulty for users and integrators.

Recommendation: Remove the erroneous return statement.

NashPoint: Fixed in PR 220.

Cantina Managed: Fix verified.

3.2.18 Quoter can be set without validation in the Registry contract

Severity: Low Risk

Context: Node.sol#L238

Description: The `setQuoter()` function in the `Node` contract lacks verification that the `newQuoter` is registered in the `NodeRegistry` contract. This allows the owner to set any quoter address, including potentially compromised ones.

Recommendation: Consider adding the following check:

```
if (!INodeRegistry(registry).isQuoter(newQuoter)) revert ErrorsLib.NotWhitelisted();
```

NashPoint: Fixed in PR 221.

Cantina Managed: Fix verified.

3.2.19 `calculateDepositBonus()` doesn't take into account `sharesExiting`

Severity: Low Risk

Context: QuoterV1.sol#L90

Description: The function `calculateDepositBonus()` doesn't take into account `sharesExiting`, while `calculateRedeemPenalty()` does. As confirmed by the project, `calculateDepositBonus()` should also do this.

Recommendation: Change function `calculateDepositBonus()` to also use `sharesExiting`. See finding Underflows in Node Cash Calculations in `BaseRouter` for related suggestions.

NashPoint: Fixed in PR 222.

Cantina Managed: Fix verified.

3.2.20 The `mint()` function lacks swing price bonus calculation

Severity: Low Risk

Context: Node.sol#L445-L454

Description: The `mint()` function does not include the bonus calculation when swing pricing is enabled, while the `deposit()` function does implement this feature. This is done on purpose to safe gas cost when the bonus is small. However the different implementations are not obvious and reduce composability.

Recommendation: Reconsider the difference in implementation and at least add documentation to explain this intentional difference.

NashPoint: Fixed in PR 223. Added `natspec` to `INode` to explain the differential and how it can be used by depositors to take advantage of the differential.

Cantina Managed: Fix verified.

3.2.21 Missing validations in `_approve()`

Severity: Low Risk

Context: BaseRouter.sol#L202-L204

Description: The `_approve()` function in `BaseRouter` contract does not validate the return value of the approve function, it should implement similar to the validation implemented in `Escrow._safeApprove()`. This would prevent issues with non-standard ERC-20 tokens.

Recommendation: Consider implementing the same validation pattern as `_safeApprove()`:

```
function _safeApprove(address token, address spender, uint256 amount) internal {
    bytes memory data = INode(node).execute(token, 0, abi.encodeWithSelector(IERC20.approve.selector, spender,
    ↪  amount));
    if (!(data.length == 0 || abi.decode(data, (bool)))) revert ErrorsLib.SafeApproveFailed();
}
```

NashPoint: Fixed in PR 224.

Cantina Managed: Fix verified.

3.2.22 Wrong location of `_approve()`

Severity: Low Risk

Context: ERC7540Router.sol#L183-L185

Description: Function `_mint()` does an `_approve()`, but is isn't needed here because you are supposed to receive the shares. On the other hand `_requestRedeem()` does not do `_approve()`, but it is needed there because it needs to send the shares. Because all shares are `_mint()`ed before they are used in `_requestRedeem()`, the current solution doesn't give an error.

Recommendation: Move `_approve()` from `_mint()` to `_requestRedeem()`.

NashPoint: Fixed in PR 225.

Cantina Managed: Fix verified.

3.2.23 Missing validation for ERC-7540 asynchronous functionality interfaces

Severity: Low Risk

Context: BaseRouter.sol#L68-L89, QuoterV1.sol#L64, QuoterV1.sol#L260

Description: According to the EIP-7540 documentation, vaults can optionally implement asynchronous flows for deposits and redemptions. If a vault doesn't implement asynchronous deposits, calls to `pendingDepositRequest` and `claimableDepositRequest` will revert.

Recommendation: Consider validating vault interfaces using `ERC165.supportsInterface()`:

```
type(IERC7540Redeem).interfaceId == 0x620ee8e4 == Asynchronous redemption
type(IERC7540Deposit).interfaceId == 0xce3bbe50 == Asynchronous deposit
```

Add these checks to either `BaseRouter::setWhitelistStatus()` / `BaseRouter::batchSetWhitelistStatus()` or `Quoter::setErc7540()` to ensure proper router type selection and component compatibility. As this check it specific for the type of vault, its best implement the check itself in the related router.

NashPoint: Acknowledged.

Cantina Managed: Acknowledged.

3.2.24 Round down in `executeAsyncWithdrawal()` could lead to reverts

Severity: Low Risk

Context: ERC7540Router.sol#L131-L151

Description: In function `executeAsyncWithdrawal()` the following happens.

- `_withdraw()` first converts the `assets` to `shares` while rounding down.
- `convertToAssets()` converts them back to `assets` while rounding down.

The dual round down could result in the situation where `assetsReceived < assets` and the transaction reverts without need.

Recommendation: Consider check the asset actually received and compare that with the requested `assets`. This can be done by checking the difference in `balanceOf()`. To prevent reentrancy, where the `balanceOf()` is counted twice, consider using an `nonReentrant` modifier. Alternatively add some tolerance in the comparison.

Also see finding:Function `_liquidate()` doesn't check the actual `assets` received.

NashPoint: Fixed in PR 226.

Cantina Managed: Fix verified.

3.2.25 `_enforceLiquidationQueue()` could fail due to one of the underlying `components` failing

Severity: Low Risk

Context: BaseRouter.sol#L184-L200, Node.sol#L243, ERC4626Router.sol#L110

Description: Function `_enforceLiquidationQueue()` calls `balanceOf()` and `convertToAssets()` of a series of components. If one of these components reverts, due to a temporary problem, a bad implementation or being malicious, then `_enforceLiquidationQueue()` reverts and `ERC4626Router::fulfillRedeemRequest()` reverts.

Recommendation: This can be solved by updating the `liquidationsQueue` via `setLiquidationQueue()`. To be able to do that effectively, it would be helpful to have monitoring in place.

An alternative solution could be to have a `try/catch`, to be able to ignore failing calls to `balanceOf()` and `convertToAssets()` and continue with the next `component`.

NashPoint: Fixed in PR 227.

Cantina Managed: Fix verified.

3.2.26 Function `_liquidate()` doesn't check the actual `assets` received

Severity: Low Risk

Context: ERC4626Router.sol#L199-L221

Description: Function `_liquidate()` checks the return value of `_redeem()` but doesn't check the `assets` have actually been received.

Recommendation: Consider check the asset actually received and compare that with `expectedAssets`. This can be done by checking the difference in `balanceOf()`. To prevent reentrancy, where the `balanceOf()` is counted twice, consider using an `nonReentrant` modifier. Also see finding: Round down in `executeAsyncWithdrawal()` could lead to reverts.

NashPoint: Fixed in PR 228.

Cantina Managed: Fix verified.

3.2.27 The node contract could contain tokens that are not actively used

Severity: Low Risk

Context: Node.sol#L21

Description: The node contract could contain tokens that are not actively used. This could be due to airdrops or shares from inactive components.

Recommendation: Consider having a way to salvage tokens from the node contract. This could be authorized for the node owner and exclude all active tokens.

NashPoint: Fixed in PR 229.

Cantina Managed: Fix verified.

3.2.28 Withdrawal Penalties Can Be Reduced Via Splitting and Fail to Cover Deposit Bonuses

Severity: Low Risk

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

Description: If a user wishing to withdraw funds completes their withdrawal in two transactions instead of one, they will pay a lower net penalty. The penalty incurred will be less than the deposit bonus received by a deposit of equal value to the total withdrawn. The net profit is small (less than a percent in the worst

case investigated) and is only maximized when the cash reserve is nearly full and the total withdrawn is close to the amount of the total cash reserve. The fact that pending redemptions are currently not included in the deposit bonus (see issue #53) increases the difficulty of exploitation, as a user must wait for withdrawals to be processed before they can receive a deposit bonus and risk being frontrun. Note that a rebalancer could have an easier path to exploitation as they could reliably backrun redemption fulfillments. Extracting profits is also non-trivial as it requires waiting for the cash reserve to be refilled to avoid exit penalties erasing most profits. However, an attacker is able to earn income while they wait so the opportunity cost is lessened.

Even in the absence of an easy closed-loop exploit, allowing users to avoid paying full exit penalties by leveraging splitting can over time result in deposit bonuses outweighing exit penalities, unfairly disadvantaging longer-term depositors.

It is worth noting that the worst-case impact has not been comprehensively determined here; for example, splitting exits into three or more requests may yield a superior fee-avoidance, and different combinations of parameters (like the reserve allocation and maximum swing factor) may yield larger or smaller margins of profit.

Proof of Concept: The following test can be added to `test/integration`VaultTests.t.sol`and run via`forge test --match-test testSplitWithdrawalThenRedeposit -vv‘ to view numerical outputs.

```
function testSplitWithdrawalThenRedeposit() public {
    vm.startPrank(user);
    asset.approve(address(node), 900_000 ether);
    node.deposit(900_000 ether, user);
    vm.stopPrank();

    vm.startPrank(user2);
    asset.approve(address(node), 100_000 ether);
    node.deposit(100_000 ether, user2);
    vm.stopPrank();

    vm.startPrank(rebalancer);
    router4626.invest(address(node), address(vault));
    vm.stopPrank();

    // assert reserveRatio is correct before other tests
    uint256 reserveRatio = _getCurrentReserveRatio();
    assertEq(reserveRatio, node.targetReserveRatio());

    // set max discount for swing pricing
    uint64 maxSwingFactor = 2e16;

    // enable swing pricing
    vm.prank(owner);
    node.enableSwingPricing(true, maxSwingFactor);

    uint256 t = 0.24e18;
    uint256 sharesToRedeem = (1e18 - t) * node.balanceOf(user2) / 1e18;

    // user 2 makes their first withdrawl
    vm.startPrank(user2);
    node.approve(address(node), type(uint256).max);
    node.requestRedeem(sharesToRedeem, user2, user2);
    vm.stopPrank();

    vm.prank(rebalancer);
    node.fulfillRedeemFromReserve(user2);

    uint256 user2BalBefore = mockAsset.balanceOf(user2);

    // user 2 withdraws max assets
    uint256 maxWithdraw = node.maxWithdraw(address(user2));
    vm.prank(user2);
    node.withdraw(maxWithdraw, address(user2), address(user2));

    uint256 user2FirstWithdrawal = mockAsset.balanceOf(user2) - user2BalBefore;

    emit log_named_uint("user2 first withdrawal:", user2FirstWithdrawal);

    uint256 user2SecondWithdrawal = 0;
    // Only do second withdrawal if there should be something left to withdraw.
    if (t > 0) {
```

```
        sharesToRedeem = node.balanceOf(user2);  // Redeem the rest

        // user 2 makes there first withdrawl
        vm.startPrank(user2);
        node.approve(address(node), type(uint256).max);
        node.requestRedeem(sharesToRedeem, user2, user2);
        vm.stopPrank();

        vm.prank(rebalancer);
        node.fulfillRedeemFromReserve(user2);

        user2BalBefore = mockAsset.balanceOf(user2);

        // user 2 withdraws max assets
        maxWithdraw = node.maxWithdraw(address(user2));
        vm.prank(user2);
        node.withdraw(maxWithdraw, address(user2), address(user2));

        user2SecondWithdrawal = mockAsset.balanceOf(user2) - user2BalBefore;
    }

    emit log_named_uint("user2 second withdrawal:", user2SecondWithdrawal);
    uint256 totalWithdrawn = user2FirstWithdrawal + user2SecondWithdrawal;
    emit log_named_uint("total withdrawn:", totalWithdrawn);

    vm.startPrank(user2);
    asset.approve(address(node), 100_000 ether);
    node.deposit(totalWithdrawn, user2);
    vm.stopPrank();

    uint256 user2Assets = node.convertToAssets(node.balanceOf(user2));
    emit log_named_uint("user2 in-protocol share value:", user2Assets);

    emit log_named_uint("sum of share value and withdrawn value:", user2Assets + totalWithdrawn);
    emit log_named_uint("total deposited by user2:", 100_000 ether + totalWithdrawn);
}
```

Recommendation: One measure that could help would be to adjust the deposit bonus to be consistently lower, e.g. by reducing the maximum swing factor by a constant fraction for deposits, to ensure it never exceeds collected withdrawal penalties. It is also worth considering different formulations of the swing pricing mechanism that would not introduce an incentive to split deposits (for example, a constant price penalty once the reserves are reduced below the target level).

If there is a desire to maintain a variable penalty and also avoid splitting/aggregation advantages, this can be done by making the spot price of assets in shares proportional to an exponential of the fractional reserve shortfall. This is sketched out mathematically below with a few caveats for any attempts at implementation.

Definitions:

- A: The cached total assets of the protocol (note: unchanged by deposits and withdrawals).
- S: The total extant node shares (`Node.totalSupply()`).
- E: The shares exiting (increased on any withdrawal or redemption by the amount of shares taken from the user).
- C: The current cash balance of the node (i.e. the node's balance of its underlying asset).
- r_T: The target reserve ratio of the node (between 0 and 1).

The "ideal" cash reserve is given by $r_T A$. We can define the shortfall relative to this ideal reserve as:

$$Q \equiv r_T A - (B - A \cdot E/S)$$

and we can further define the shortfall fraction f as the ratio of the shortfall to the ideal reserves:

$$f \equiv \frac{Q}{r_T A}$$

We will treat first the case of a withdrawal. When $f < 0$, we have ample reserves, and assets withdrawn in this regime are converted to shares by multiplying by S/A. When $0 \leq f \leq 1$, the price of an infinitesimal unit of assets in shares will be:

$$p = \frac{S}{A} e^{-kf}$$

where k is a constant that can be simply related to the maximum discount to the spot price (when $f = 1$, $p = \frac{S}{A} e^{-k}$). For example, if the maximum desired discount is 2%, then k would be chosen as $\ln(1/0.98)$: $e^{-k} = e^{\ln(0.98)} = 0.98$.

For the portion of assets redeemed in the $0 \leq f \leq 1$ regime, the adjusted quantity of shares can be obtained by integrated the product $p \cdot da$ (da is the differential of assets withdrawn a) between the initial and final f values:

$$s_{adj} = \int_{f_i}^{f_f} \frac{S}{A} e^{-kf(a)} da$$

We can use $f(a) = (Q + a)/(r_T A)$ to make a change of integration variable to f by noting $da = (r_T A) df$:

$$s_{adj} = r_T S \int_{f_i}^{f_f} e^{-kf(a)} df$$

This integral is straightforward to perform and evaluate; one obtains:

$$s_{adj} = \frac{r_T S}{k} (e^{-kf_i} - e^{-kf_f})$$

Denoting the initial shortfall as Q_i which for assets redeemed a implies $Q_f = Q_i + a$, we arrive at the following:

$$s_{adj}(a, Q_i) = \frac{r_T S}{k} e^{\frac{-kQ_i}{r_T A}} \left(1 - e^{\frac{-ka}{r_T A}}\right)$$

It is easy to verify algebraically the property that $s_{adj}(a_1, Q_i) + s_{adj}(a_2, Q_i + a_1) = s_{adj}(a_1 + a_2, Q_i)$, which demonstrates that no advantage can be gained by splitting or combining withdrawal operations.

Once f exceeds one, the penalty is capped and we calculate $s_{adj}(a) = \frac{S}{A} e^{-k} a$. In an implementation, the calculation would need to be split piecewise based on which f regimes were accessed.

A simple approach to calculating a deposit bonus safely is to give the depositor value in proportion to the withdrawal penalty that would be assessed if their deposit were reversed. Mathematically, for the $0 \leq f \leq 1$ regime, the shares $s_{dep}(a, Q_f)$ awarded for a deposit of assets a that leaves the shortfall at Q_f after the deposit is:

$$s_{dep}(a, Q_f) = \frac{S}{A} \left(a + (a - \frac{A}{S} s_{adj}(a, Q_f)\right)$$

However, this is not as obviously correct--both total assets and shares change after a deposit, so more work is needed to determine whether this is a safe construction.

On the subject of implementation, note that the formula for s_{adj} will not behave well numerically due to precision loss when $e^{\frac{-ka}{r_T A}}$ is close to one. In such a numerical regime, a Taylor expansion around zero is a good approximation for the exponential and can be used to compute the penalty instead.

NashPoint: Fixed in PR 232. Dividing `maxSwingFactor` by 2 instead of 3 for final commit.

Cantina Managed: Fix verified.

3.3 Gas Optimization

3.3.1 `decimals` can be precomputed

Severity: Gas Optimization

Context: Node.sol#L552-L554

Description: The value of `IERC20Metadata(asset).decimals();` shouldn't change, so it can be calculated and stored in the constructor of `node`. This will save some gas.

Recommendation: Consider changing the code to:

```
+ uint8 private immutable _decimals;

  constructor() {
      // ...
+     decimals = IERC20Metadata(asset).decimals();
  }
  function decimals() public view override(ERC20, IERC20Metadata) returns (uint8) {
-     return IERC20Metadata(asset).decimals();
+     return _decimals;
  }
```

NashPoint: Fixed in PR 200.

Cantina Managed: Fix verified.

3.3.2 The `value` Parameter of `Node.execute` Sends Ether But the Contract Has No `payable` Methods

Severity: Gas Optimization

Context: Node.sol#L303-L310

Description: The `Node.execute` function accepts a `value` parameter which specifies an amount of Ether to be sent with the external call to be made. However, the contract implements no `payable` methods, meaning it can only receive Ether through nonstandard and contrived mechanisms like the `SELFDESTRUCT` opcode and block rewards. This would be very cumbersome if sending Ether were an important functionality of `Node.execute`.

Recommendation: According to the NashPoint team, sending Ether via `Node.execute` is not an intended functionality. Therefore, it is recommended to remove the `value` parameter to reduce gas costs and avoid confusion.

NashPoint: Fixed in PR 201.

Cantina Managed: Fix verified.

3.3.3 Redundant `view` Functions in Node Contract

Severity: Gas Optimization

Context: Node.sol#L560-L608

Description: The `Node` contract defines a number of `view` functions that expose data also accessible via automatically-generated getter functions for the underlying `public` state fields (see lines 560-608). While a few of them augment the functionality of the compiler-generated functions (for example returning entire arrays or returning partial data from structs), most of them are redundant. This increases bytecode size and can contribute to runtime gas cost via more expensive function selector resolution.

Recommendation: Consider either removing redundant functions, or making storage fields private if a custom interface is desired.

NashPoint: Fixed in PR 202.

Cantina Managed: Fix verified.

3.3.4 Redundant Check in `ERC4626Router`

Severity: Gas Optimization

Context: BaseRouter.sol#L127-L131, Node.sol#L303-L308, ERC4626Router.sol#L199-L205

Description: In the `_liquidate()` function of `ERC4626Router`, the `_validateNodeAcceptsRouter()` function is invoked to check that the router is compatible with the node being operated on. However, the node itself will do the same check in the call to `execute()` due to the `onlyRouter` modifier. Thus doing the check in both functions is redundant and a significant waste of gas (as an additional external call is added to the router logic).

Recommendation: The node must be ultimately responsible for ensuring that only authorized routers interact with sensitive functions (a malicious router can be implemented to simply omit the check). Thus, it makes sense to eliminate the invocation of `_validateNodeAcceptsRouter()` in `ERC4626Router.liquidate()`. Note that this is the only usage of `_validateNodeAcceptsRouter()` in the codebase (it is defined in `BaseRouter.sol` which is inherited by both `ERC4626Router` and `ERC7540Router`); thus, removing this function entirely is also recommended.

NashPoint: Fixed in PR 203.

Cantina Managed: Fix verified.

3.3.5 `_isComponent()` isn't efficient

Severity: Gas Optimization

Context: Node.sol#L243-L247, Node.sol#L671-L673, Node.sol#L726-L734, QuoterV1.sol#L272-L276

Description: Function `_isComponent()` does a linear search and thus takes a relative large amount of gas. Its main use is in function `setLiquidationQueue()`. With a very large number of `components` this could run out of gas.

Recommendation: Estimate the expected maximum number amount of components and check if the risk of out of gas is present. If it is, consider using the mapping `componentAllocations_[]` to directly check if a component is present.

Note: make sure the `ComponentAllocation` doesn't contain the value 0, perhaps add a extra field with a non zero value.

Note: an alternative would be to use `OpenZeppelin EnumerableMap`, however this increases the gas cost for retrieving the list of `components`. That should be avoided because it is part of `QuoterV1.::getTotalAssets` and thus `node::_updateTotalAssets()`.

Here is an example how `OpenZeppelin EnumerableMap` could be used:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "hardhat/console.sol";
import "@openzeppelin/contracts/utils/structs/EnumerableMap.sol";

contract testEnumerableMap {
    EnumerableMap.AddressToBytes32Map components;
    using EnumerableMap for EnumerableMap.AddressToBytes32Map;

    function setTargetWeightMaxDelta(uint64 targetWeight,uint64 maxDelta) public pure returns(bytes32) {
        return bytes32(bytes8(targetWeight)) >> 64 | bytes32(bytes8(maxDelta));
    }
    function getTargetWeight(bytes32 b) public pure returns(uint64 targetWeight) {
       targetWeight = uint64(bytes8(b << 64) & bytes8(0xffffffffffffffff));
    }
    function getMaxDelta(bytes32 b) public pure returns(uint64 maxDelta) {
       maxDelta = uint64(bytes8(b & bytes8(0xffffffffffffffff)));
    }
    constructor() {
        uint64 targetWeight = 7;
        uint64 maxDelta = 12345678901234567890;
        address a = address(0x789);
        components.set(a,setTargetWeightMaxDelta(targetWeight,maxDelta));
        console.log(components.contains(a)); // true
        uint len = components.length();
        for (uint i=0;i< len;i++) {
            (address key, bytes32 b) = components.at(i);
            console.logAddress(key); // 0x0000000000000000000000000000000000000789
            console.log("targetWeight",getTargetWeight(components.get(key))); // targetWeight 7
            console.log("maxDelta",getMaxDelta(b)); // maxDelta 12345678901234567890
        }
    }
}
```

NashPoint: Fixed in PR 204.

Cantina Managed: Fix verified.

3.3.6 `_decimalsOffset()` related code can be simplified

Severity: Gas Optimization

Context: Node.sol#L552-L554, Node.sol#L748-L758

Description: The value of `_decimalsOffset()` is 0 which is compatible `decimals()`. So this value won't change.

Recommendation: Consider simplifying the code to make it more readable while also saving gas:

```
  function _convertToShares(uint256 assets, MathLib.Rounding rounding) internal view virtual returns (uint256) {
-     return assets.mulDiv(totalSupply() + 10 ** _decimalsOffset(), totalAssets() + 1, rounding);
+     return assets.mulDiv(totalSupply() + 1, totalAssets() + 1, rounding);
  }
  function _convertToAssets(uint256 shares, MathLib.Rounding rounding) internal view virtual returns (uint256) {
-     return shares.mulDiv(totalAssets() + 1, totalSupply() + 10 ** _decimalsOffset(), rounding);
+     return shares.mulDiv(totalAssets() + 1, totalSupply() + 1, rounding);
  }
- function _decimalsOffset() internal view virtual returns (uint8) {
-    return 0;
- }
```

NashPoint: Fixed in PR 205.

Cantina Managed: Fix verified.

3.3.7 Internal variables in `ERC7540Router` can be moved to function scope

Severity: Gas Optimization

Context: ERC4626Router.sol#L19-L22, ERC7540Router.sol#L21-L23

Description and recommendation: The internal variables `totalAssets`, `currentCash`, and `idealCashReserve` are only used within the `investInAsyncVault()` function. Consider declaring these variables directly inside the function to improve code organization and safe gas.

NashPoint: Fixed in PR 206.

Cantina Managed: Fix verified.

3.3.8 Optimize `shares` to `assets` conversions

Severity: Gas Optimization

Context: QuoterV1.sol#L258-L264

Description/Recommendation: The `assets` variable is calculated by converting multiple share amounts to assets individually: `shareBalance`, `pendingRedeemRequest()`, and `claimableRedeemRequest()`. These share amounts could first be summed together before applying the `convertToAssets()` function once. This would reduce gas costs and minimize rounding errors.

NashPoint: Fixed in PR 207.

Cantina Managed: Fix verified.

3.3.9 Optimize if-statement in `invest()`

Severity: Gas Optimization

Context: ERC4626Router.sol#L64-L66

Description/Recommendation: The `depositAmount` validation in `ERC4626Router.invest()` can be simplified using `Math.min()` for better readability:

```
- if (depositAmount > availableReserve) {
-     depositAmount = availableReserve;
- }
+ depositAmount = Math.min(depositAmount, availableReserve);
```

NashPoint: Fixed in PR 208.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 `MAX_DEPOSIT` can't be changed

Severity: Informational

Context: Node.sol#L32

Description: `MAX_DEPOSIT` is set at a large value for testing purposes, but it can't be changed. Being able to change it might be helpful to limit the risk of the `Nodes`.

Recommendation: Consider a way to change `MAX_DEPOSIT`, either via the `constructor` or via an authorized function, in which case the `immutable` has to be removed. Also consider using a lower, more realistic default value.

NashPoint: Fixed in PR 170.

Cantina Managed: Fix verified.

3.4.2 `immutable` could be `constant`

Severity: Informational

Context: Node.sol#L30-L33

Description: The following `immutable` variables are not assigned in the `constructor` and are supposed to be changeable, so they could be `constant`:

```
uint256 internal immutable WAD = 1e18;
uint256 private immutable REQUEST_ID = 0;
uint256 public immutable SECONDS_PER_YEAR = 365 days;
```

Recommendation: Consider replacing `immutable` with `constant`.

NashPoint: Fixed in PR 169.

Cantina Managed: Fix verified.

3.4.3 Illogical order of statements in 'addRebalancer()

Severity: Informational

Context: Node.sol#L211-L215

Description: Function `addRebalancer()` first sets `isRebalancer[]` and then does an additional check that can revert. It is more logical to reverse these statements.

Recommendation: Consider changing the code to:

```
  function addRebalancer(address newRebalancer) external onlyOwner {
      // ...
-     isRebalancer[newRebalancer] = true;
      if (!INodeRegistry(registry).isRebalancer(newRebalancer)) revert ErrorsLib.NotWhitelisted();
+     isRebalancer[newRebalancer] = true;
      // ...
  }
```

NashPoint: Fixed in PR 171.

Cantina Managed: Fix verified.

3.4.4 `setLiquidationQueue()` and `_setInitialComponents()` could contain duplicate components

Severity: Informational

Context: Node.sol#L243-L251

Description: In function `setLiquidationQueue()`, `newQueue` could contain duplicate components. This doesn't seem to hurt in the current code. However it might be an issue in future versions of routers. In function `_setInitialComponents()`, `components_` could contain duplicate components. In that case `componentAllocations[]` would use the latest value.

Recommendation: Consider checking for duplicates. This can be done by sorting the addresses before calling the set functions and check the addresses are increasing (or decreasing) in the function.

NashPoint: Fixed in PR 234.

Cantina Managed: Fix verified.

3.4.5 Function `enableSwingPricing()` doesn't `emit maxSwingFactor_`

Severity: Informational

Context: Node.sol#L266-L270

Description: Function `enableSwingPricing()` doesn't `emit maxSwingFactor_`. Adding this might be useful.

Recommendation: Consider changing the code to:

```
- emit EventsLib.SwingPricingStatusUpdated(status_);
+ emit EventsLib.SwingPricingStatusUpdated(status_, maxSwingFactor_);
```

NashPoint: Fixed in PR 174.

Cantina Managed: Fix verified.

3.4.6 `rebalanceCooldown` period runs in pallallel with the `rebalanceWindow`

Severity: Informational

Context: Node.sol#L291-L300

Description: The `rebalanceCooldown` period runs in pallallel with the `rebalanceWindow`, which requires care with setting the values and isn't intuitive.

Recommendation: Consider starting the `rebalanceCooldown` after the `rebalanceWindow` has closed. This could be implemented in the following way:

```
  function startRebalance() external ... {
      // ...
-     if (block.timestamp < lastRebalance + rebalanceCooldown) revert /*...*/;
+     if (block.timestamp < lastRebalance + rebalanceWindow + rebalanceCooldown) revert /*...*/;
      // ...
  }
```

Note: `isCacheValid()` should be updated too.

NashPoint: Fixed in PR 175.

Cantina Managed: Fix verified.

3.4.7 Subtractions and additions can be simplified

Severity: Informational

Context: Node.sol#L333, Node.sol#L394-L395, QuoterV1.sol#L137

Description: In several place the expression: `a = a - b` is used. This can be replaced with `a -= b` which is more efficient, easier to read and more consistent.

Recommendation: Consider changing the code to:

```
  function payManagementFees() /*...*/ {
      // ...
-     cacheTotalAssets = cacheTotalAssets - feeForPeriod;
+     cacheTotalAssets -= feeForPeriod;
      // ...
  }
```

```
function calculateRedeemPenalty(/*...*/) /*...*/ {
    // ...
-   balance = balance - pendingRedemptions;
+   balance -= pendingRedemptions;
    // ...
}
```

```
  function requestRedeem(/*...*/) /*...*/ {
      // ...
-     request.pendingRedeemRequest = request.pendingRedeemRequest + shares;
+     request.pendingRedeemRequest += shares;
-     request.sharesAdjusted = request.sharesAdjusted + adjustedShares;
+     request.sharesAdjusted += adjustedShares;
      // ...
  }
```

NashPoint: Fixed in PR 177.

Cantina Managed: Fix verified.

3.4.8 `return` in function `payManagementFees()` is not necessary

Severity: Informational

Context: Node.sol#L316-L339

Description: The `return` in function `payManagementFees()` is somewhat confusing, because it is done in the `if`, but not at the end of the function. The return is also not necessary because its also the return parameter.

Recommendation: Consider removing the `return feeForPeriod;`.

NashPoint: Fixed in PR 178.

Cantina Managed: Fix verified.

3.4.9 Function `subtractProtocolExecutionFee()` doesn't check there is sufficient balance

Severity: Informational

Context: Node.sol#L342-L345

Description: Function `subtractProtocolExecutionFee()` doesn't check there is sufficient balance like `payManagementFees()` does. Although `_validateReserveAboveTargetRatio()` is called in the `routers` before calling this, this doesn't guarantee there are enough assets, because `targetReserveRatio()` could be `0`.

Recommendation: For consistency also consider checking there is sufficient balance.

NashPoint: Fixed in PR 179.

Cantina Managed: Fix verified.

3.4.10 `QuoterV1` knows too many implementation details

Severity: Informational

Context: QuoterV1.sol#L252

Description: The `QuoterV1` currently knows implementation details for the different `routers` and types of assets via:

- `_getErc4626Assets()`.
- `_getErc7540Assets()`.

For maintenance and upgrability it is advisable to delegate this calculation to the `router`.

Recommendation: Move `_getErc4626Assets()` and `_getErc7540Assets()` to a `getAssets()` in the `router`. Function `getAssets()` should check the component is of the right type by checking `QuoterV1`.

Note: Have options in `getAssets()` to exclude assets that are in transit (e.g. `pendingDepositRequest`/`claimableDepositRequest` and `pendingRedeemRequest`/`claimableRedeemRequest`). This is important for finding:The `liquidationsQueue` processing is not generic.

Change the code of `_getTotalAssets()` to something like this:

```
function _getTotalAssets(address node) internal view returns (uint256) {
    // ...
    for (uint256 i = 0; i < componentsLength; i++) {
        address component = components[i];
        uint256 typeOfVault = Component2TOV[component];
        if (typeOfVault == 0)
            revert ErrorsLib.InvalidComponent();
        address router = node.TOV2Router(typeOfVault);
        componentAssets += router.getAssets(node, component, /*...*/);
    }
    // ...
}
```

Note: this assumes the recommendations from finding: Routers can access components from other routers are implemented.

Note: this is also relevant for the recommendation of finding: Functions `_getInvestmentSize()` in `ERC4626Router` and `ERC7540Router` are very similar.

NashPoint: Fixed in commit 01417bd4:

1. Created `getComponentAssets()` virtual function to be inherited by routers, moved `_updateTotalAssets()` to Node (directly referencing the router for each component), using `getComponent()` assets as balance check in 'removeComponent()'.
2. Added a force bool as an argument to `removeComponent`

3. Stripped out all component management functionality from the quoter. Quoter is now purely for swing pricing and is using the IQuoterV1 interface and `IQouter` has been deleted.

Cantina Managed: Fix verified.

3.4.11 In function `mint()`, the return parameter `assets` isn't used

Severity: Informational

Context: Node.sol#L445-L454

Description: In function `mint()`, the return parameter `assets` isn't used. Instead `assetsToDeposit` is returned.

Recommendation: Consider removing the name `assets` from the return parameter definition, to prevent confusion.

NashPoint: Fixed in PR 180.

Cantina Managed: Fix verified.

3.4.12 Both `msg.sender` and _msgSender()' are used

Severity: Informational

Context: Node.sol#L438, Node.sol#L450

Description: Most of the time `msg.sender` is used, however sometimes `_msgSender()` is used, which is confusing.

Recommendation: Consider replacing `_msgSender()` with `msg.sender`.

NashPoint: Fixed in PR 181.

Cantina Managed: Fix verified.

3.4.13 Typo errors

Severity: Informational

Context: QuoterV1.sol#L82, QuoterV1.sol#L113

Description and Recommendation: The code contains some typos in the comments.

- `QuoterV1.sol#L82`:

```
- /// @dev Called by Node Contract to alculates the deposit bonus
+ /// @dev Called by Node Contract to calculates the deposit bonus
```

- `QuoterV1.sol#L113`:

```
- /// This is to prevent a situation where requests are pending for withdrawal but no swing pricing
↪  penatly is being applied
+ /// This is to prevent a situation where requests are pending for withdrawal but no swing pricing
↪  penalty is being applied
```

NashPoint: Fixed in PR 167.

Cantina Managed: Fix verified.

3.4.14 Missing Natspec

Severity: Informational

Context: INodeFactory.sol#L33, IQuoter.sol#L8, IQuoter.sol#L13

Description/Recommendation: Some functions linked to this finding lacks of documentation, adding clear and detailed documentation would improve code maintainability and reduce potential integration errors.

NashPoint: Fixed in PR 182.

Cantina Managed: Fix verified.

3.4.15 Unnecessary Casts

Severity: Informational

Context: BaseQuoter.sol#L36

Description: The following are instances of unnecessary casting, which can reduce code readability:

- `src/libraries/BaseQuoter.sol` line 36: in the expression `address(Ownable(address(registry)).owner())` the `owner()` function is already returning type `address`, so the final cast to `address` is not needed.

Recommendation: Remove unnecessary casts to improve readability.

NashPoint: Fixed in PR 183.

Cantina Managed: Fix verified.

3.4.16 `NodeRegistry` code simplification

Severity: Informational

Context: NodeRegistry.sol#L27-L35

Description: In the `NodeRegistry` contract, there is a separate variable for each registry: node, factory, router, quoter, and rebalancer. These variables can be groupped into a single variable using an enum.

For example, we can create an enum with 'NODE=1, FACTORY=2, ROUTER=3, QUOTER=4, REBALANCER=5`and use a single`mapping(address => enum => bool)' to store values for each address. If an address needs multiple roles, we can implement bitflags.

This refactor would reduce the number of add/remove/initialize functions needed.

Recommendation: Consider converting the multiple variables into a single one using the following code as reference:

```
- mapping(address => bool) public isNode;
- mapping(address => bool) public isFactory;
- mapping(address => bool) public isRouter;
- mapping(address => bool) public isQuoter;
- mapping(address => bool) public isRebalancer;

+ enum RegistryType { NODE, FACTORY, ROUTER, QUOTER, REBALANCER }
+ mapping(address => RegistryType => bool) public isRegistered
```

Note: For routers it might be helpful to create a seperate mapping, see finding: Routers can access components from other routers .

NashPoint: Fixed in PR 184:

- I kept node contracts out of the `setRole()` function as only the Factory can add a node to the registry
- There may be some redundant events.

Cantina Managed: Fix verified.

3.4.17 `BaseRouter` and `BaseQuoter` can be defined abstract

Severity: Informational

Context: BaseQuoter.sol#L12, BaseRouter.sol#L16

Description: The contracts `BaseRouter` and `BaseQuoter` are currently defined using the `contract` keyword, but since they serve as base contracts for inheritance, they should be declared as `abstract` contracts.

Recommendation: Consider declaring both contracts as `abstract` to better reflect their intended usage.

NashPoint: Fixed in PR 185.

Cantina Managed: Fix verified.

3.4.18 `_safeApprove()` can fail with tokens that implement approval race protection

Severity: Informational

Context: Escrow.sol#L46-L49

Description/Recommendation: Some tokens, like USDT, require setting the allowance to zero before approving a new allowance value. Otherwise, the current code will revert.

Since the approval would be changed to only used once in the constructor, as discussed in finding: The `approveMax()` and `unapprove()` functions could be used maliciously by the node owner, this shouldn't be an issue.

NashPoint: Fixed in PR 216.

Cantina Managed: Fix verified.

3.4.19 Unused events

Severity: Informational

Context: EventsLib.sol#L25, EventsLib.sol#L28, EventsLib.sol#L31, EventsLib.sol#L55, EventsLib.sol#L64, EventsLib.sol#L67, EventsLib.sol#L73, EventsLib.sol#L100

Description: Several events declared in`EventsLib`are never used: `SetManager`, `AddOperator`, `RemoveOperator` ,`NodeRemoved`, `PricerAdded`, `PricerRemoved`, `SetNodePricer`, and`DepositClaimable`.

Recommendation: Consider removing these unused event declarations.

NashPoint: Fixed in PR 186.

Cantina Managed: Fix verified.

3.4.20 The `node` parameter is redundant in some events

Severity: Informational

Context: EventsLib.sol#L79-L91

Description: The `node` variable is redundant in events since `address(this)` is used and the event is already associated with the `Node` contract address. Here are the events:

```
emit EventsLib.ComponentAdded(address(this), component, allocation);
emit EventsLib.ComponentRemoved(address(this), component);
emit EventsLib.ComponentAllocationUpdated(address(this), component, allocation);
emit EventsLib.ReserveAllocationUpdated(address(this), allocation);
emit EventsLib.RebalanceStarted(address(this), block.timestamp, rebalanceWindow);
```

Recommendation: Consider removing the `node` parameter from these events.

NashPoint: Fixed in PR 187.

Cantina Managed: Fix verified.

3.4.21 Inconsistent event naming

Severity: Informational

Context: EventsLib.sol#L10, EventsLib.sol#L13, EventsLib.sol#L16, EventsLib.sol#L25, EventsLib.sol#L28, EventsLib.sol#L31, EventsLib.sol#L70, EventsLib.sol#L73

Description: Most events use `Added` or `Removed` as suffixes, but there are exceptions:

- `AddRouter` / `RemoveRouter`.
- `AddOperator` / `RemoveOperator`.

Similarly, while `Set` is typically used as a suffix, these events use it as a prefix:

- `SetEscrow`.
- `SetManager`.

- `SetQuoter`.
- `SetNodePricer`.

Recommendation: Consider using a consistent naming pattern for all events.

NashPoint: Fixed in PR 188:

- Reordered events to be be grouped by emitting contract.
- Added missing event `SetAnnualManagementFee`.

Cantina Managed: Fix verified.

3.4.22 Underflows in Node Cash Calculations in `BaseRouter`

Severity: Informational

Context: BaseRouter.sol#L113-L117, BaseRouter.sol#L138-L145, QuoterV1.sol#L121-L138

Description: The `BaseRouter` contract performs the following computations in two separate places:

```
uint256 currentCash = IERC20(INode(node).asset()).balanceOf(address(node))
- INode(node).convertToAssets(INode(node).getSharesExiting());
```

It is possible for this subtraction to underflow, resulting in a revert. The first instance is within the `_validateReserveAboveTargetRatio()` function, and reverting is appropriate behavior, although the error message will not be as informative as it could be. The second instance is within a `view` function `_getNodeCashStatus()` where reverting could lead to unexpected behavior depending on usage (although it currently does not as all calls to `_getNodeCashStatus()` are preceded by a call to `_validateReserveAboveTargetRatio()`).

Note: doublecheck the situation where the result would be negative to see if additional actions have to be taken.

Note: `calculateRedeemPenalty()` does a similar calculation but does check for underflows.

Recommendation: Compute `currentCash` as a signed instead of unsigned integer. Alternatively, set it to zero if underflow would occur; this allows a very specific edge case that if there is negative cash and `idealCashReserve` is zero, then `_validateReserveAboveTargetRatio()` does not revert when it should, but any such transaction will revert for other reasons, and this scenario will likely never occur in practice.

Consider moving this calulation to function in `node` as it is also used by:

- `calculateRedeemPenalty()`.
- `calculateDepositBonus()` once the suggestion for finding `calculateDepositBonus()` doesn't take into account `sharesExiting` is implemented.

NashPoint: Solved by PR 189 and PR 222.

Cantina Managed: Fix verified.

3.4.23 Duplicated Investment Logic Can Be Moved to `BaseRouter`

Severity: Informational

Context: ERC4626Router.sol#L38-L69, ERC7540Router.sol#L39-L69

Description: Both `ERC4626Router` and `ERC7540Router` contain an identical section of logic in their `invest()` and `investInVaultAsync()` functions, respectively (lines 49-69 in both files). This logic checks various preconditions, subtracts the execution fee, and computes the amount to deposit. It is not specific to either Vault type in any way.

Recommendation: Move the logic to `BaseRouter`. An appropriate name could be `_computeDepositAmount()` and its only necessary return value would be the amount to deposit.

NashPoint: Fixed in PR 190.

Cantina Managed: Fix verified.

3.4.24 Not all relevant functions support partial fulfill

Severity: Informational

Context: Node.sol#L634, ERC4626Router.sol#L110

Description: Function `ERC4626Router::fulfillRedeemRequest()` allows for a partial fulfill, however the following comparable function `Node::_fulfillRedeemFromReserve()` doesn't support this. A partial fulfill could be relevant for very large redeems.

Recommendation: Consider adding a partial fulfill functionality to `Node::_fulfillRedeemFromReserve()`. Refactor the code to prevent code duplication.

NashPoint: Fixed in PR 242:

- Due to async nature of 7540 there is no clear way to atomically withdraw from a 7540 component and fulfill a redemption.
- A future version of the 7540 router may include this if it is a requested feature.
- Intended operation is for the rebalancer to liquidate a 7540 component into the reserve in order to fulfill requests.

Cantina Managed: Fix verified.

3.4.25 Ambiguous evaluation in `_calculateSharesAfterSwingPricing()`

Severity: Informational

Context: Node.sol#L736-L737

Description: In function `_calculateSharesAfterSwingPricing()` there is an evaluation that might be ambiguous.

Recommendation: To be sure and for readabilitity consider changing the code to:

```
- if (   totalAssets() == 0 && totalSupply() == 0   || !swingPricingEnabled) {
+ if ( ( totalAssets() == 0 && totalSupply() == 0 ) || !swingPricingEnabled) {
```

NashPoint: Fixed in PR 192:

- Due to async nature of 7540 there is no clear way to atomically withdraw from a 7540 component and fulfill a redemption.
- A future version of the 7540 router may include this if it is a requested feature.
- Intended operation is for the rebalancer to liquidate a 7540 component into the reserve in order to fulfill requests.

Cantina Managed: Fix verified.

3.4.26 Functions `_getInvestmentSize()` in `ERC4626Router` and `ERC7540Router` are very similar

Severity: Informational

Context: ERC4626Router.sol#L179-L192, ERC7540Router.sol#L222-L235

Description: The functions `_getInvestmentSize()` in `ERC4626Router` and `ERC7540Router` are very similar. Combining them will imrpove maintainability.

Recommendation: Consider integrating the functions and move the integrated function to `BaseRouter`. The recommendation from finding: `QuoterV1` knows too many implementation details to create `getAssets()` will be helpful.

Function `_getInvestmentSize()` could then look like:

```
function _getInvestmentSize(/*...*/) /*...*/ {
    uint256 targetHoldings = MathLib.mulDiv(/*...*/);
    uint256 currentBalance = getAssets(node, component);
    uint256 delta = // ...
    return delta;
}
```

NashPoint: Acknowledged. I will keep these separate as the virtual/override we have now leave more room for different types of integrations in future.

Cantina Managed: Acknowledged.

3.4.27 The `liquidationsQueue` processing is not generic

Severity: Informational

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

Description: The `liquidationsQueue` is currently only used in one router: `ERC4626Router`. However it is more logical to use for all available routers.

Recommendation: Consider making the `liquidationsQueue` more generic:

- Possibly move `_enforceLiquidationQueue` to a central place: `node` or `QuoterV1`, to make sure the execution is done in a consistent way.
- Take a a component from the `queue` and check its still present in `node`.
- For the component check the `TypeOfVault` via `QuoterV1`, which also checks the `component` is supported by `QuoterV1`. See finding: Routers can access components from other routers.
- Via the `TypeOfVault` determine the related `router`.
- Query the router via `getAssets()`. See finding `QuoterV1` knows too many implementation details, excluding the `pendingRedeemRequest` and `claimableRedeemRequest`.
- Call the approriate function of the router to `withdraw` or `request a withdrawal`. This will require a new function. Note: function `ERC7540Router::requestAsyncWithdrawal()` also has to be adapted.

NashPoint: Fixed in PR 241.

Cantina Managed: Fix verified.

3.4.28 Checks Not Using Errors From ErrorsLib.sol

Severity: Informational

Context: ERC7540Router.sol#L72, ERC7540Router.sol#L95, ERC7540Router.sol#L121

Description: Most failed checks in the codebase revert with a custom error, but there are some exceptions that use a revert string instead. See the linked code for the locations.

Recommendation: Use custom errors everywhere for consistency and gas efficiency.

NashPoint: Fixed in PR 193.

Cantina Managed: Fix verified.

3.4.29 Use constants instead of magic numbers

Severity: Informational

Context: QuoterV1.sol#L261-L264, ERC7540Router.sol#L72, ERC7540Router.sol#L121

Description: The code uses the literal number 0 instead of the defined constant `REQUEST_ID` in several locations:

- ERC7540Router.sol#L72.
- ERC7540Router.sol#L121.
- QuoterV1.sol#L261-L264.

Recommendation: Replace the hardcoded zeros with the `REQUEST_ID` constant in the identified locations.

```
- require(requestId == 0, "No requestId returned");
+ require(requestId == REQUEST_ID, "No requestId returned");

- assets += IERC7540(component).pendingDepositRequest(0, node);
- assets += IERC7540(component).claimableDepositRequest(0, node);
- assets += IERC4626(component).convertToAssets(IERC7540(component).pendingRedeemRequest(0, node));
- assets += IERC4626(component).convertToAssets(IERC7540(component).claimableRedeemRequest(0, node));
+ assets += IERC7540(component).pendingDepositRequest(REQUEST_ID, node);
+ assets += IERC7540(component).claimableDepositRequest(REQUEST_ID, node);
+ assets += IERC4626(component).convertToAssets(IERC7540(component).pendingRedeemRequest(REQUEST_ID, node));
+ assets += IERC4626(component).convertToAssets(IERC7540(component).claimableRedeemRequest(REQUEST_ID, node));
```

NashPoint: Fixed in PR 194.

Cantina Managed: Fix verified.

3.4.30 Missing `maxDeposit()` and `maxRedeem()` validation

Severity: Informational

Context: ERC4626Router.sol#L72, ERC4626Router.sol#L199, ERC7540Router.sol#L39, ERC7540Router.sol#L106

Description: The `investInAsyncVault()` function in the `ERC7540Router` contract lacks validation for `maxDeposit()`. Both, while `ERC7540Router::requestAsyncWithdrawal()` and `ERC4626Router::_liquidate()` lack validation for `maxRedeem()`. A validation for `maxDeposit()` is present in the `ERC4626Router` contract.

Recommendation: Consider implementing a `maxDeposit()` and `maxRedeem()` validation check in both routers:

NashPoint: Our understanding for the `7540Router`:

- Per the EIP, `maxDeposit()` for a `7540` is a check on the `pendingDepositRequest()`.
- Per the Centrifuge implementation, `maxRedeem()` is a check on the `claimableRedeemRequest()`.
- On the router, `mintClaimableShares()` and `executeAsyncWithdrawal()` both check the correct view functions: `maxMint()` and `maxWithdraw()`.

Implemented the suggested check on maxRedeem for the 4626 Router in PR 195.

Cantina Managed: Fix verified.

3.4.31 The owner of the registry can block all nodes

Severity: Informational

Context: NodeRegistry.sol#L98, NodeRegistry.sol#L114, NodeRegistry.sol#L130, Node.sol#L291-L300, Node.sol#L671-L673, QuoterV1.sol#L59-L66, QuoterV1.sol#L272-L290

Description: The owner of the registry can remove components from `QuoterV1`, via `setErc4626()` and `setErc7540()`. This will block `getTotalAssets()` and thus `startRebalance()` on a lot of nodes. The owner of the registry can also remove `Rebalancers`, `Routers`, `Quoters`.

All of these can effectively stop the functioning of the `Nodes` and lock all the funds, which is a centralising factor. This can also be used as a pause in case of an emergency.

Recommendation: Doublecheck the balance of power between all actors and see if the power of the owner of the registry is desired. Make changes if desired. Doublecheck the need for a pause functionality. If desired, create an explicit pause functionality.

NashPoint: Fixed in PR 239.

Cantina Managed: Fix verified.

3.4.32 `onlyRegistryOwner` doesn't check the quoter is still a valid quoter

Severity: Informational

Context: BaseQuoter.sol#L35-L39

Description: The modifier `onlyRegistryOwner` doesn't check the quoter is still a valid quoter (e.g. registred in `NodeRegistry`). This means an old quoter could still be managed. The risk is limited because the old quoter likely doesn't have any use anymore.

Recommendation: Consider checking the quoter is still valid.

NashPoint: Addressed in PR 215.

Cantina Managed: Fix verified.

3.4.33 Check `isComponent()` is done on `internal` and `external` functions

Severity: Informational

Context: ERC4626Router.sol#L38-L47, ERC4626Router.sol#L93, ERC4626Router.sol#L110, ERC4626Router.sol#L199-L203, ERC7540Router.sol#L39-L47, ERC7540Router.sol#L82-L91, ERC7540Router.sol#L106-L114, ERC7540Router.sol#L131-L139

Description: The check for `isComponent()` is sometimes done on `internal` functions and sometimes on `external` functions. This is not consistent and more difficult to review and maintain.

Recommendation: Move the check from `_liquidate()` to `liquidate()` and `fulfillRedeemRequest()`. Also consider converting the check to a `modifier` in both `ERC4626Router` and `ERC7540Router`.

NashPoint: Addressed in PR 196.

Cantina Managed: Fix verified.

3.4.34 `_validateReserveAboveTargetRatio()` is similar to `_getNodeCashStatus()`

Severity: Informational

Context: BaseRouter.sol#L113-L123, BaseRouter.sol#L138-L147

Description: Function `_validateReserveAboveTargetRatio()` has largely the same code as `_getNodeCashStatus()`. This code duplication makes the code more difficult to review and maintain.

Recommendation: Let the function `_validateReserveAboveTargetRatio()` call `_getNodeCashStatus()`.

NashPoint: Addressed in PR 197.

Cantina Managed: Fix verified.

3.4.35 Functions `calculateRedeemPenalty()` and `calculateDepositBonus()` could be generalized

Severity: Informational

Context: QuoterV1.sol#L90, QuoterV1.sol#L121-L127

Description: The functions `calculateRedeemPenalty()` and `calculateDepositBonus()` do several calls back to the `node` and could be generalized to be more future proof.

Recommendation: Consider doing the following:

- Use the same parameters & return type on both functions.
- Supply some extra parameters that might be useful in a future version.
- Minimize the calls back to node for gas efficiency. Supply extra parameters to be able to do that.
- Do all the parameters and returns in assets (and not in shares), after checking this is sufficient for all use cases.

NashPoint: Addressed in PR 198. *Note: I left the return value in `calculateDepositBonus()` in shares as it is solely used in the `deposit` function to define the shares to mint to a user.*

Cantina Managed: Fix verified.

3.4.36 `target` versus `component` versus `vault`

Severity: Informational

Context: BaseRouter.sol#L23-L27, BaseRouter.sol#L56-L57, BaseRouter.sol#L65-L89, ERC4626Router.sol#L71, ERC4626Router.sol#L73, ERC4626Router.sol#L152-L163, ERC4626Router.sol#L165-L173, ERC7540Router.sol#L39, ERC7540Router.sol#L124

Description: `BaseRouter` uses `target`, while on most other places the term `component` is used. This is confusing. `ERC4626Router` frequently uses `vault`, while `ERC7540Router` mostly uses `component`. This is confusing.

Recommendation: Consider using the same term everywhere.

NashPoint: Addressed in commit c6d27f24.

Cantina Managed: Fix verified.

3.4.37 `encodeWithSelector()` vs. `encodeCall()`

Severity: Informational

Context: Escrow.sol#L47, BaseRouter.sol#L176, BaseRouter.sol#L203, ERC4626Router.sol#L161, ERC4626Router.sol#L171, ERC7540Router.sol#L173, ERC7540Router.sol#L188, ERC7540Router.sol#L201, ERC7540Router.sol#L213

Description: Sometimes `encodeWithSelector()` is used and sometimes `encodeCall()`.

Recommendation: Consider using `encodeCall()` where possible because this enforces type checks.

NashPoint: Fixed in PR 238.

Cantina Managed: Fix verified.