Affine Labs - UltraETH LRT

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Liquid Restaking Token
Timeline	2024-06-10 through 2024-06-26
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• AffineLabs/contracts #2daf109
Auditors	• Hytham Farah Auditing Engineer • Andy Lin Senior Auditing Engineer • Gelei Deng Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	7 Fixed: 4 Acknowledged: 2 Mitigated: 1
High severity findings	0
Medium severity findings	2 Fixed: 2
Low severity findings	4 Fixed: 2 Acknowledged: 1 Mitigated: 1
Undetermined severity findings	0
Informational findings	1 Acknowledged: 1

Summary of Findings

This audit evaluates Affine Lab's Liquid Restaking Token (LRT) ultraETH. The tokenized vault allows users to deposit stETH and receive shares in the form of ultraETH. The staked ETH is delegated to various operators within the Eigenlayer and Symbiotic ecosystems. These operators use stETH as collateral for their actively validated services (AVS). Consequently, user deposits are exposed to both the risks of slashing and the rewards accrued by the operators.

Overall, the codebase is well-written and adequately tested. It is nearly complete but currently undergoing active development. Some changes may have occurred between this audit and deployment.

No high-severity vulnerabilities were found during the audit. Most significant findings relate to potential misuse of the privileged Harvester role, which could adversely affect the protocol (see AFF-1, AFF-2, AFF-3). Additionally, several recommendations were made to enhance security, such as using `safeTransfer` or including storage gaps for upgradeable contracts.

Updates: All serious issues have been adequately addressed, and other issues have been acknowledged by the team with a sufficient explanation. Significant improvements to the NatSpec as well as the test suite were made after the audit.

ID	DESCRIPTION	SEVERITY	STATUS
AFF-1	**Vault Accounting Inconsistency if Harvester Calls Delegator Functions Directly**	• Medium	Fixed
AFF-2	**Risk of Overreporting Assets**	• Medium	Fixed
AFF-3	**Harvester Can Continuously Call `delegateToDelegator()` to Prevent User From `resolveDebt()`**	• Low	Fixed
AFF-4	**Not Using Safe Transfer**	• Low	Fixed
AFF-5	**Contracts without Gap Storage Reserve Used by Upgradeable Contracts**	• Low	Mitigated

ID	DESCRIPTION	SEVERITY	STATUS
AFF-6	One State Variable Cannot Track Both Factories	● Low	Acknowledged
AFF-7	Cannot Control Assets Distribution to Delegators Due to Ordered Liquidation Design	● Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

The scope consisted of the files relevant to the ultraETH Liquid Restaking Token. This mainly included a tokenized vault implementation for the LRT, and two delegator factories to delegate collected assets to EigenLayer and Symbiotic operators.

Files Included

```
/src/vaults/restaking
├── AffineDelegator.sol
├── DelegatorBeacon.sol
├── DelegatorFactory.sol
├── EigenDelegator.sol
├── IDelegator.sol
├── SymDelegatorFactory.sol
├── SymbioticDelegator.sol
```

```
    ├── UltraLRT.sol
    ├── UltraLRTStorage.sol
    └── WithdrawalEscrowV2.sol
```

Files Excluded

`/src/vaults/restaking/AffineReStaking.sol`

Operational Considerations

The protocol allows for limited user functionality to limit the attack surface. There are a few notes to make:

1. **Manual Delegator Selection**: There is no algorithm for distributing deposited assets to delegators. Instead, the Harvester manually chooses the delegator each time user deposits are sent. This approach lacks predictability in asset distribution among validators but offers flexibility for the Affine Labs team to dynamically control asset flow.
2. **Withdrawl Requests do not End Exposure to Profit / Loss**: Users can withdraw funds without centralized approval. If a user initiates a withdrawal and the vault lacks sufficient assets, their request is registered. When an epoch ends, delegators are called to withdraw assets to fulfill the user's request. Share values are calculated at withdrawal based on the assets withdrawn from the vault. This means users remain exposed to losses and rewards even after their shares have been moved from operators/delegators.
3. **Delayed Profit/Loss Reflection**: Since delegated assets are outside the vault, profit/loss isn't immediately reflected in share value. Manual function calls are required to update share values, impacting the assets users receive upon withdrawal. For example, during a significant slashing event, users withdrawing before a `harvest()` call will have their profits unaffected.
4. **Locked Profit Post-Harvest**: Profits after a harvest are vested linearly over time to prevent share inflation attacks, where users deposit large amounts of shares right before a harvest to redeem them immediately after.
5. **First Come First Serve**: Any assets that are in the vault can be immediately redeemed by any user. Even assets that were originally withdrawn to satisfy the withdrawal requests held in the escrow contract. This may lead to unpredictable withdrawal times if the funds are collected and other users withdraw before the Epoch can be resolved, in the worst-case scenario, a malicious Harvester could indefinitely delay withdrawals. (see AFF-3).
6. **No Mechanism for Distributing ERC20 Rewards From AVS**: The current implementation locks and later redeems the native value of the asset but lacks a mechanism for users to obtain the ERC20 rewards that operators receive for participating in AVS.

Key Actors And Their Capabilities

We note that in general the contracts have a high degree of and crucially users can always withdraw deposited assets without permission (unless AFF-4) .

Harvester

Key Responsibilities:

- Choose a delegator to receive user deposits.
- Maintain efficient asset management.
- Process withdrawals accurately.

Trust Assumption:

- The Harvester delegates strategically, remains active, and initiates reasonable withdrawals and liquidations from operators back to the vault.

UltraLRT:

1. `liquidationRequest()` : Initiates withdrawal requests from delegators for a specified amount of assets. Liquidated assets are only returned to the vault.
2. `delegatorWithdrawRequest()` : Selects a delegator to initiate withdrawal from Eigenlayer/Symbiotic for a specified amount of assets.
3. `collectDelegatorDebt()` : Collects all stETH from delegators and transfers them to the Vault.
4. `delegateToDelegator()` : Allocates a specified amount of user deposits to a particular delegator.
5. `harvest()` : Updates `delegatorAssets` based on the current value of delegator assets.

Note: Only the Harvester can call `endEpoch()` until the `LOCK_INTERVAL` (24 hours at the audit commit) has passed. After this period, it becomes publicly callable, allowing users to withdraw funds as they can only do so at the end of the Epoch.

Governance

Key Responsibilities:

- Set contract parameters.
- Manage upgrades.
- Control deposit and withdrawal functionalities.

UltraLRT:

1. `pauseDeposit()` : Pauses deposits into the vault.
2. `unpauseDeposit()` : Resumes deposits into the vault.
3. `setWithdrawalEscrow()` : Configures the withdrawal escrow contract.

4. `setDelegatorFactory()` : Configures the delegator factory contract.
5. `setManagementFee()` : Configures the management fee.
6. `setWithdrawalFee()` : Configures the withdrawal fee.
7. `_authorizeUpgrade()` : Approves upgrades to new contract implementations.

WithdrawalEscrowV2:

1. `sweep()` : Transfers the balance of a specified asset to the governance address.

AffineReStaking:

1. `pauseDeposit()` : Pauses deposits into the contract.
2. `resumeDeposit()` : Resumes deposits into the contract.
3. `approveToken()` : Approves a token for deposit.
4. `revokeToken()` : Revokes approval of a token for deposit.
5. `pause()` : Pauses the entire contract operations.
6. `unpause()` : Resumes the entire contract operations.

NOTE: Governance is also granted the Harvester and Guardian role and can call therefore call their role-restricted functions as well.

Guardian

Key Responsibilities:

- Provide emergency control over the contract.
- Protect against exploits or suspicious activity.

Trust Assumption:

- The Guardian will only pause the contract in case of an emergency.

UltraLRT:

1. `pause()` : Pauses all contract operations.
2. `unpause()` : Resumes all contract operations.

AffineReStaking:

1. `pause()` : Pauses all contract operations.
2. `unpause()` : Resumes all contract operations.

Findings

AFF-1

Vault Accounting Inconsistency if Harvester Calls Delegator Functions Directly

● Medium ⓘ Fixed

Update

We reviewed the fix and confirmed that the team made the following changes to resolve the issue:

- `delegate()` function now transfers tokens from `msg.sender` instead of the vault
- The `withdraw()` function is restricted to the vault

Update

Fixed by the clients:
Addressed in: `136628ef7790204f3885457117fba0094c931fb7`.
The client provided the following explanation:

```
Delegator will transfer assets from msg.sender. This will open access to delegateTo function.
```

File(s) affected: `UltraLRT`, `AffineDelegator`, `EigenDelegator`, `SymbioticDelegator`

Description: The `AffineDelegator` implementation includes the modifier `onlyVaultOrHarvester()`, which checks if the `msg.sender` is the vault or the harvester for several functions. However, the `UltraLRT` vault contract has its own variables for accounting changes to the delegators. If the delegators are changed without going through the vault functions, this can lead to accounting errors.

1. The `UltraLRT.delegateToDelegator()` function updates the `delegatorMap` and `delegatorAssets`. If the harvester calls `AffineDelegator.delegate()` directly, these updates will not be recorded.
2. The `UltraLRT._getDelegatorLiquidAssets()` function records the balances of `delegatorAssets` and `delegatorMap`. Similarly, if the harvester calls `AffineDelegator.withdraw()` directly, these changes will not be recorded.

In the second case, `delegatorAssets` will not record the change from the `withdraw()` function, causing `totalAssets()` to falsely assume it has more assets than it actually does. This discrepancy will result in an incorrect conversion rate of shares to asset amounts.

Recommendation: The team should first clarify if it is necessary to allow the harvester to call functions on the delegator without going through the vault. If not, consider removing this feature. If it is necessary, implement a better mechanism to ensure the vault and the delegators remain in sync.

AFF-2 Risk of Overreporting Assets

● Medium ⓘ Fixed

Update

Fixed by the clients:
Addressed in: `539ade03fd6486dba87cc2f411f1fa646f2f1f1c`.
The client provided the following explanation:

```
Harvester has to record any external withdrawal request manually to complete withdrawal. Delegator keep
track of all the withdrawal requests.
```

File(s) affected: `UltraLRT`

Description: The Harvester may disrupt the accounting of `queuedShares` and can call `EigenDelegator.completeExternalWithdrawalRequest()` to complete a withdrawal without subtracting the withdrawn amount from the `queuedShares` variable. This results in an overrepresentation of the remaining `queuedShares` in the system, and more critically, the `UltraLRT.totalAssets()` function which determines the value of each share.

As it would overreport the number of assets in the system, shares would redeem more assets than they should, potentially leaving the Vault insolvent.

Recommendation: Ensure the `queuedShares` variable is accurate, in particular, consider removing the `completeExternalWithdrawalRequest()` function altogether, or at least, providing some mitigation so it cannot be called incorrectly.

AFF-3 Harvester Can Continuously Call `delegateToDelegator()` to Prevent User From `resolveDebt()`

● Low ⓘ Fixed

Update

Fixed by the clients:
Addressed in: `596d9629cf03181feaab9e26309006ac3e49922b`.
The client provided the following explanation:

```
1. Introduce maxUnresolvedEpochs
2. Delegation will not be possible if max unresolved epoch reached.
```

File(s) affected: `UltraLRT`, `WithdrawalEscrowV2`

Description: The `UltraLRT` and `WithdrawalEscrowV2` contracts include a mechanism that triggers `_liquidationRequest()` when an epoch ends via the `UltraLRT.endEpoch()` function. This function requests withdrawals from the delegators back to the `UltraLRT` contract to fulfill the withdrawal requests queue in the `WithdrawalEscrowV2` contract.

However, in the `UltraLRT` contract, the `HARVESTER` can continuously call the `delegateToDelegator()` function before `resolveDebt()` is executed, thereby blocking users from withdrawing from the system.

Recommendation: The team should first clarify if this behavior aligns with their protocol's design. If not, a mechanism to lock the liquidation request amount is necessary. For instance, consider adding a variable like `liquidationRequestAmount` in the `UltraLRT` contract to record the sum requested in `_liquidationRequest()`. The `delegateToDelegator()` function should respect this `liquidationRequestAmount` and ensure that the required funds remain. Additionally, the `resolveDebt()` function should reduce the `liquidationRequestAmount` once the debt is repaid.

AFF-4 Not Using Safe Transfer

● Low ⓘ Fixed

Update

Fixed by the clients:
Addressed in: `b8c5df95046aecb534242c007ee71e84010c7a8c`.
The client provided the following explanation:

```
Used SafeTransferLib.
```

File(s) affected: `UltraLRT`, `AffineDelegator`

Description: Some parts of the code ignore the return value of the generic ERC20 transfer call. The `transfer` and `transferFrom` functions of ERC20 tokens can return false to indicate a failed transfer instead of reverting directly. Consequently, a failed transfer might occur without being detected. Historically, some ERC20 tokens, such as USDT, do not return the expected boolean value, necessitating the use of safe transfer wrappers.

1. `AffineDelegator.delegate()`: The line `asset.transferFrom(address(vault), address(this), amount)`.
2. `UltraLRT._withdraw()`: The line `IERC20MetadataUpgradeable(asset()).transfer(receiver, assetsToReceive)`.

Although the protocol currently integrates only with stETH, which does not pose this issue, future integrations with other tokens might require attention to this concern.

Recommendation: Consider using safe transfer calls in these instances.

AFF-5 Contracts without Gap Storage Reserve Used by Upgradeable Contracts

• Low ⓘ Mitigated

Update

Gap storage has been added to contracts other than `AffineGovernable`.

Client provided the following explanation:

```
Could not add a gap as this is also used by some other upgradeable contracts.
```

Update

Marked as "Fixed" by the client.
Addressed in: `2bb6fd2581d332f7771c83f408a51714fdd0c4a8`.
The client provided the following explanation:

```
1. Added storage gap
2. Used upgradeable reentrancy
```

File(s) affected: `AffineDelegator`, `EigenDelegator`, `SymbioticDelegator`

Description: Several upgradeable contracts inherit base contract implementations without reserving storage gaps, potentially losing the ability to add new storage variables in future upgrades. The following contracts might require attention:

1. `AffineDelegator` serves as the base contract for `EigenDelegator` and `SymbioticDelegator`, both upgradeable contracts. However, `AffineDelegator` lacks reserved storage gaps.
2. `EigenDelegator` and `SymbioticDelegator` both inherit from `AffineGovernable`, which also lacks gap storage. These contracts might not need to inherit `AffineGovernable`, as it does not include governance guarding functions.
3. `UltraLRT` inherits from `AffineGovernable` and `ReentrancyGuard` without reserving gap storage.

Recommendation:

1. Add gap storage space to the `AffineDelegator` contract as done here.
2. Remove the use of `AffineGovernable` for `EigenDelegator` and `SymbioticDelegator` if unnecessary. If it is needed, add gap storage to the `AffineGovernable` contract and rename it to `AffineGovernableUpgradeable`.
3. Use the upgradeable version of `ReentrancyGuard` as shown here.

AFF-6 One State Variable Cannot Track Both Factories

• Low ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
Vault will only use one beacon at a time. So we are keeping a single interface for all delegator.
```

File(s) affected: `UltraLRT`

Description: The `UltraLRT` contract has only one state variable, `delegatorFactory`, of type `address` to track the factory. This setup cannot track both `SymDelegatorFactory` and `DelegatorFactory`, with the latter being for EigenLayer delegators.

Recommendation: Ensure that the state variables can track multiple types of delegator factories per the needs of the system.

AFF-7
Cannot Control Assets Distribution to Delegators Due to Ordered Liquidation Design

● Informational ⓘ Acknowledged

Update

Marked as "Acknowledged" by the client.
The client provided the following explanation:

```
For now harvester will choose delegator manually.

Harvester can call  delegatorWithdrawRequest()  to collect specific amount of assets withdrawal from
delegator.
```

File(s) affected: `UltraLRT`

Description: In `_liquidationRequest()`, the assets are withdrew from the delegators following their order in the delegator list (`delegatorQueue`). In this case, the early created delegators will always be liquidated first. This makes it hard to maintain a certain balance/distribution to different delegators in the system, unless the manager manually re-distribute the assets.

Recommendation: Consider changing how `_liquidationRequest()` requests withdrawals from the various delegators.

Adherence to Best Practices

1. The repo uses both custom errors and `require` statements, which can lead to inconsistent error handling and potential issues during debugging. Consider using custom error types consistently to ensure clarity across the contracts.
2. The `SymbioticDelegator._delegate()` function is missing the return value from `collateral.deposit()`. Consider implementing a sanity check to ensure the value is at least greater than 0. Below is a list of `TODO` statements which can be found within the audit scope:
3. Finish the list of unfinished TODOs:
    1. **UltraLRT**
        1. `// TODO: calculate fees`
        2. `// TODO -- calculate tvl`
        3. `// TODO: incremental distribution of assets`
        4. `// todo check a valid operator address`
        5. `// TODO map utilization`
    2. **WithdrawalEscrowV2**
        1. `// TODO: epoch end event`

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Files

- `753...a28 ./src/restaking/SymbioticDelegator.sol`
- `a42...cfd ./src/restaking/WithdrawalEscrowV2.sol`
- `dea...382 ./src/restaking/AffineDelegator.sol`
- `0b5...5ff ./src/restaking/SymDelegatorFactory.sol`
- `a25...53b ./src/restaking/DelegatorFactory.sol`
- `ad5...6b1 ./src/restaking/UltraLRTStorage.sol`
- `012...ad9 ./src/restaking/IDelegator.sol`
- `7ac...bce ./src/restaking/UltraLRT.sol`
- `211...143 ./src/restaking/DelegatorBeacon.sol`
- `c77...f86 ./src/restaking/EigenDelegator.sol`

Tests

- `6eb...3e6 ./src/test/SymbitoicDelegator.t.sol`
- `89c...bd0 ./src/test/EigenDelegator.t.sol`
- `5cb...250 ./src/test/UltraLRT.t.sol`

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.10.0

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Results were either dismissed as false positives or included in the report.

Test Suite Results

All the tests we ran our passing though we encourage further unit testing for other critical contracts such as `WithdrawlEscrowV2`.

Update: Many new tests were added significantly increasing the robustness of the test suite.

```
$ forge test --match-contract "(EigenDelegator|SymbioticDelegator|UltraLRT)"

[⠊] Compiling...
[⠘] Compiling 344 files with Solc 0.8.16
[⠒] Solc 0.8.16 finished in 29.77s
Compiler run successful with warnings:
Warning (2519): This declaration shadows an existing declaration.
  --> src/test/integration/StrategyVaultV2.int.sol:41:9:
   |
41 |         StrategyVaultV2 _vault = StrategyVaultV2(address(vault));
```

```
   |         ^^^^^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
  --> src/test/integration/StrategyVaultV2.int.sol:13:5:
   |
13 |     function _vault() internal virtual returns (address) {}
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning (2519): This declaration shadows an existing declaration.
  --> src/test/integration/StrategyVaultV2.int.sol:52:9:
   |
52 |         StrategyVaultV2 _vault = StrategyVaultV2(address(vault));
   |         ^^^^^^^^^^^^^^^^^^^^^^
Note: The shadowed declaration is here:
  --> src/test/integration/StrategyVaultV2.int.sol:13:5:
   |
13 |     function _vault() internal virtual returns (address) {}
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this
warning.
  --> src/test/CommonVault.t.sol:26:23:
   |
26 |     function tokenURI(uint256 id) public view override returns (string memory) {
   |                       ^^^^^^^^^^

Warning (2072): Unused local variable.
   --> src/test/UltraLRT.t.sol:532:9:
    |
532 |         uint256 currentTVL = vault.totalAssets();
    |         ^^^^^^^^^^^^^^^^^^

Warning (2072): Unused local variable.
   --> src/test/UltraLRT.t.sol:563:9:
    |
563 |         uint256 currentTVL = vault.totalAssets();
    |         ^^^^^^^^^^^^^^^^^^

Warning (5667): Unused function parameter. Remove or comment out the variable name to silence this
warning.
  --> src/test/Vault.t.sol:21:23:
   |
21 |     function tokenURI(uint256 id) public view override returns (string memory) {
   |                       ^^^^^^^^^^

Warning (2018): Function state mutability can be restricted to pure
   --> src/test/AaveV2Strategy.t.sol:143:5:
    |
143 |     function _usdc() internal override returns (address) {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
   --> src/test/AaveV2Strategy.t.sol:147:5:
    |
147 |     function _lendingPool() internal override returns (address) {
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/AaveV3Strategy.t.sol:22:5:
   |
22 |     function _usdc() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
```

```
  --> src/test/AaveV3Strategy.t.sol:26:5:
   |
26 |     function _lendingPool() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/AaveV3Strategy.t.sol:40:5:
   |
40 |     function _usdc() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/AaveV3Strategy.t.sol:44:5:
   |
44 |     function _lendingPool() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/CommonVault.t.sol:26:5:
   |
26 |     function tokenURI(uint256 id) public view override returns (string memory) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
   --> src/test/DeltaNeutralLp.t.sol:321:5:
    |
321 |     function decodeStartPositionEvent(bytes memory data)
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
   --> src/test/DeltaNeutralLp.t.sol:329:5:
    |
329 |     function decodeEndPositionEvent(bytes memory data)
    |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
  --> src/test/LevMaticXLoopStrategy.t.sol:57:5:
   |
57 |     function _getPricePerShare() internal returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to view
  --> src/test/LidoLevL2.t.sol:15:5:
   |
15 |     function wstEthToEth(uint256 wstEthAmount) external returns (uint256 ethAmount) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/UltraLRT.t.sol:36:5:
   |
36 |     function test() public returns (uint256) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/Vault.t.sol:21:5:
   |
21 |     function tokenURI(uint256 id) public view override returns (string memory) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/integration/StrategyVaultV2.int.sol:68:5:
   |
68 |     function _vault() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/integration/StrategyVaultV2.int.sol:79:5:
   |
79 |     function _vault() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/integration/StrategyVaultV2.int.sol:90:5:
   |
90 |     function _vault() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/integration/VaultV2.int.sol:38:5:
   |
38 |     function _vault() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Warning (2018): Function state mutability can be restricted to pure
  --> src/test/integration/VaultV2.int.sol:48:5:
   |
48 |     function _vault() internal override returns (address) {
   |     ^ (Relevant source part starts here and spans across multiple lines).

Analysing contracts...
Running tests...

Ran 1 test for src/test/UltraLRT.t.sol:TmpUltraLRT
[PASS] test() (gas: 425)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.58ms (773.71µs CPU time)

Ran 3 tests for src/test/SymbitoicDelegator.t.sol:TestSymbioticDelegator
[PASS] testDeposit() (gas: 311416)
[PASS] testInvalidCollateral() (gas: 4324797)
[PASS] testWithdrawal() (gas: 368753)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 476.54ms (14.74ms CPU time)

Ran 3 tests for src/test/EigenDelegator.t.sol:EigenDelegatorTest
[PASS] testDelegate() (gas: 614850)
[PASS] testReDelegation() (gas: 789892)
[PASS] testZeroShareWithdrawal() (gas: 425075)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 486.73ms (46.64ms CPU time)

Ran 8 tests for src/test/UltraLRTRouter.t.sol:TestUltraLRTRouter
[PASS] testNativeToVault1Deposit() (gas: 347334)
Logs:
  ==> 999999999999999999
  ==> 999999999999999998

[PASS] testNativeToWStEthVaultDeposit() (gas: 426489)
Logs:
  ==> 858178691210958000
  ==> 858178691210958000

[PASS] testStEthToVault1Deposit() (gas: 369882)
[PASS] testStEthToWStEthVaultDeposit() (gas: 401239)
[PASS] testWEthToVault1Deposit() (gas: 461844)
[PASS] testWEthToWStEthVaultDeposit() (gas: 546934)
[PASS] testWStEthToVault1Deposit() (gas: 569086)
Logs:
  ==> 0
```

```
[PASS] testWStEthToWStEthVaultDeposit() (gas: 398489)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 551.90ms (85.30ms CPU time)

Ran 33 tests for src/test/UltraLRT.t.sol:UltraLRTTest
[PASS] testCollectDelegatorDebt() (gas: 965188)
Logs:
  vault balance 999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testCreateDelegator() (gas: 33608715)
Logs:
  vault balance 999999999999999999900000000

[PASS] testDelegateToDelegator() (gas: 759449)
Logs:
  vault balance 999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testDelegateToInactiveDelegator() (gas: 399696)
Logs:
  vault balance 999999999999999999900000000

[PASS] testDelegatorWithdrawReq() (gas: 1003504)
Logs:
  vault balance 999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testDeposit() (gas: 256240)
Logs:
  vault balance 999999999999999999900000000

[PASS] testDepositMintOverMax() (gas: 89621)
[PASS] testDepositPause() (gas: 433571)
Logs:
  Deposit pause 1

[PASS] testDropDelegator() (gas: 3912903)
Logs:
  vault balance 999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testInitializeVaultWithInvalidBeacon() (gas: 11835261)
[PASS] testLiquidationRequest() (gas: 1146754)
Logs:
  vault balance 999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testLossHarvest() (gas: 1479511)
Logs:
  9999999999999999997
  9999999999999999997
  tvl 99999999999999999999

[PASS] testMint() (gas: 250800)
[PASS] testMultipleHarvest() (gas: 259261)
[PASS] testPause() (gas: 264781)
[PASS] testPauseAndUnpause() (gas: 317115)
[PASS] testPermissionedFunctions() (gas: 891231)
Logs:
  vault balance 1999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c
```

```
[PASS] testProfitHarvest() (gas: 1002150)
[PASS] testRedeem() (gas: 293935)
Logs:
  vault balance 99999999999999999999900000000

[PASS] testResolveDebt() (gas: 1631369)
Logs:
  vault balance 99999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testSetDelegatorFactory() (gas: 2304014)
[PASS] testSetManagementFees() (gas: 51563)
[PASS] testSetWithdrawalEscrow() (gas: 13411951)
Logs:
  vault balance 99999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testSetWithdrawalFees() (gas: 51458)
[PASS] testSetWithdrawalQueue() (gas: 1615817)
Logs:
  vault balance 99999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

[PASS] testStEthTransferIssueBuffer() (gas: 636168)
Logs:
  vault balance 99999999999999999999900000000

[PASS] testUpgradeBeacon() (gas: 3099554)
[PASS] testUpgradeVault() (gas: 7771619)
[PASS] testViewOnlyFunctions() (gas: 68373)
[PASS] testWithdrawAndRedeemByAllowance() (gas: 593432)
Logs:
  vault balance 99999999999999999999900000000
  vault balance 100000000000000000000100000000

[PASS] testWithdrawAndRedeemOverMax() (gas: 294410)
Logs:
  vault balance 99999999999999999999900000000

[PASS] testWithdrawFull() (gas: 302774)
Logs:
  vault balance 99999999999999999999900000000

[PASS] testWithdrawalQueueWithDelegatorUpgrade() (gas: 4690165)
Logs:
  vault balance 99999999999999999999900000000
  delegator 0xa38D17ef017A314cCD72b8F199C0e108EF7Ca04c

Suite result: ok. 33 passed; 0 failed; 0 skipped; finished in 552.61ms (526.58ms CPU time)
```

Code Coverage

Note coverage output has been pruned to files within scope.

The coverage for the `/src/vaults/restaking` files is generally high, with `AffineDelegator.sol` and `SymbioticDelegator.sol` achieving 100% in several metrics. However, the coverage for `UltraLRT.sol` (97.24% lines, 96.65% statements, 67.74% branches) and `WithdrawalEscrowV2.sol` (62.50% lines, 60.78% statements, 37.50% branches) should be improved, especially focusing on branch coverage to reach at least 90%, given their critical importance. The `DelegatorBeacon.sol` (50% lines, 57.14% statements) and `SymDelegatorFactory.sol` (0% in all metrics) could also benefit from further coverage.

Update: The updated test suite report highlights significant improvements: `UltraLRT.sol` branch coverage has increased from 67.74% to 80.88%, although line and statement coverage slightly decreased. `WithdrawalEscrowV2.sol` shows substantial gains, with coverage metrics nearly reaching 100% across all categories, a significant jump from previous figures. Both `DelegatorBeacon.sol` and

`SymDelegatorFactory.sol` now boast 100% coverage in all metrics, marking complete coverage enhancements from much lower initial metrics. This focused effort effectively addresses critical deficiencies and enhances the codebase's overall robustness.

File	% Lines	% Statements	% Branches	% Funcs
src/vaults/restaking/AffineDelegator.sol	100.00% (**5/**5)	100.00% (**8/**8)	100.00% (**0/**0)	50.00% (**4/**8)
src/vaults/restaking/DelegatorBeacon.sol	100.00% (**3/**3)	100.00% (**4/**4)	100.00% (**0/**0)	100.00% (**2/**2)
src/vaults/restaking/DelegatorFactory.sol	100.00% (**2/**2)	100.00% (**4/**4)	100.00% (**0/**0)	100.00% (**1/**1)
src/vaults/restaking/EigenDelegator.sol	100.00% (**45/**45)	100.00% (**61/**61)	83.33% (**15/**18)	88.89% (**8/**9)
src/vaults/restaking/SymDelegatorFactory.sol	100.00% (**1/**1)	100.00% (**1/**1)	100.00% (**0/**0)	100.00% (**1/**1)
src/vaults/restaking/SymbioticDelegator.sol	100.00% (**10/**10)	100.00% (**12/**12)	100.00% (**2/**2)	100.00% (**5/**5)
src/vaults/restaking/UltraLRT.sol	91.25% (**146/**160)	91.92% (**239/**260)	80.88% (**55/**68)	92.11% (**35/**38)
src/vaults/restaking/UltraLRTRouter.sol	89.74% (**35/**39)	91.67% (**44/**48)	75.00% (**9/**12)	76.92% (**10/**13)
src/vaults/restaking/WithdrawalEscrowV2.sol	97.22% (**35/**36)	97.87% (**46/**47)	100.00% (**12/**12)	100.00% (**12/**12)

Changelog

- 2024-06-13 - Initial report
- 2024-06-26 - Final Report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:
- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Quantstamp