Quantstamp

Ensuro Strategy Vault

Executive Summary

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

Type	Vault
Timeline	2025-02-24 through 2025-03-03
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• https://github.com/ensuro/vaults #bc7bfa8
Auditors	• Jennifer Wu Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Rabib Islam Senior Auditing Engineer

Documentation quality	High
Test quality	High
Total Findings	7 Fixed: 4 Acknowledged: 3
High severity findings	0
Medium severity findings	0
Low severity findings	5 Fixed: 4 Acknowledged: 1
Undetermined severity findings	1 Acknowledged: 1
Informational findings	1 Acknowledged: 1

Summary of Findings

The Ensuro Vault is a multi-strategy vault that accepts user deposits and allocates them across multiple investment strategies (e.g., AaveV3, CompoundV3, or stable swaps) through delegate calls. Each strategy manages its own portion of the deposited funds within a single vault, while the deposit and withdrawal process relies on the vault's queue mechanism to coordinate operations.

Quantstamp reviewed the vault strategies, as well as the vault and access control design, and identified 7 findings and 4 suggestions for code improvements. The most notable issues revolve around unclaimed rewards in `CompoundV3InvestStrategy.disconnect()` (ENS-1), potential collisions in function selectors due to the XOR-based approach in `getForwardToStrategySelector()` (ENS-2), and a partial reset of outflow data when changing slotsize in the `OutflowLimitedAMMSV` contract (ENS-3).

Overall, the codebase is thoughtfully designed and well-tested, with a comprehensive test suite demonstrating solid coverage. We recommend the client to consider all identified issues.

Fix Review: The client resolved all issues by implementing fixes, mitigating, or acknowledging them.

ID	DESCRIPTION	SEVERITY	STATUS
ENS-1	`CompoundV3InvestStrategy.disconnect()` **Ignores Claimable Rewards**	● Low	Fixed
ENS-2	**Inconsistent and Collidable Selector Generation for Strategy Forwarding**	● Low	Fixed
ENS-3	**Inconsistent Use of the** `force` **in** `MSVBase.removeStrategy()`	● Low	Fixed
ENS-4	`_maxWithdrawable()` **and** `_maxDepositable()` **May Overflow**	● Low	Fixed
ENS-5	**Hard-Coded** `_price` **Can Become Stale and Inaccurate in the Event of a Depeg**	● Low	Acknowledged

ID	DESCRIPTION	SEVERITY	STATUS
ENS-6	Potential Double Counting in `totalAssets()` if Multiple Strategies Share the Same Underlying Asset	● Informational ⓘ	Acknowledged
ENS-7	Changing Slot Size Temporarily Allows Excessive Outflow	● Undetermined ⓘ	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

Files Excluded

Repo: https://github.com/ensuro/vaults(bc7bfa872901f90d6e3e2034e5bb82e9b7918ad3)

Files:

- contracts/mock/*
- contracts/dependencies/**
- contracts/SingleStrategyERC4626.sol
- contracts/CompoundV3ERC4626.sol
- contracts/MultiStrategyERC4626.sol
- contracts/PermissionedERC4626.sol

Operational Considerations

The project uses upgradeable contracts, which allow the implementation logic to be changed arbitrarily. While this flexibility enables feature enhancements and bug fixes, it also introduces risks of partial or total loss of funds if upgrades are not properly controlled. The vault is deployed behind a UUPS proxy, meaning the admin can call the `upgradeToAndCall()` function, but no other calls from the admin are forwarded to the implementation.

Once deployed, it is not possible to change the admin based on the vault's implementation. Proper management of the admin role is critical, as improper controls could put user funds at risk. Introducing future updates without safeguards can lead to storage collisions. Following established patterns in OpenZeppelin's upgradeable contracts is recommended.

Each strategy contract is assumed to manage a distinct underlying asset to ensure accurate accounting. In the current design, `totalAssets()` in each strategy reflects only that strategy's holdings (e.g., `aToken`, `cToken`, or a specific asset). Because strategies do not overlap in the assets they control, the vault's global `totalAssets()` calculation remains accurate. However, if a future strategy reuses the same underlying asset as an existing strategy, the vault could double-count those holdings when summing across all strategies. Keeping assets separate for each strategy preserves the integrity of the vault's aggregate asset value.

The selector for forwarding calls to strategies is computed from the strategy index, method ID, and the first four bytes of the strategy's address. Because different inputs can produce the same output, there is a risk of collisions that could overlap access control permissions. Caution is needed when adding or removing strategies to avoid unintended conflicts.

Fix Review: The 4-byte selector for forwarding calls to strategies is now derived by hashing the `strategy` address and a `uint8` method, then extracting the first four bytes of the result. Because different inputs can map to the same four-byte prefix, this approach carries a collision risk that could overlap with access control permissions. Careful management of adding or removing strategies methods is still required to avoid such unintended conflicts.

The `SwapStableInvestStrategy` contract stores a fixed `_price` representing the exchange rate between `_asset` and `_investAsset`. If the peg is lost or `_price` remains unchanged despite market shifts, users could experience large slippage, receive fewer tokens than expected, or lose slippage protection (`maxSlippage`).

Key Actors And Their Capabilities

The vault is deployed behind a UUPS proxy, with the UUPS admin specified during proxy deployment. This admin can invoke the `upgradeToAndCall()` function to modify the vault's implementation contract. Once the vault implementation is deployed, the admin cannot be changed.

The access control within the vault is governed by roles defined in the OpenZeppelin's AccessManager contract. `AccessManager` defines which roles (identified by `uint64`) can call specific functions (identified by a `target` address and a function `selector`). Depending on the delay setting for each role, interactions with the vault can be immediate or subject to a timelock through a scheduled operation. Proper configuration of `AccessManager` is critical; if roles and permissions are set up incorrectly, unauthorized users may gain access to privileged functions.

Findings

ENS-1 `CompoundV3InvestStrategy.disconnect()` Ignores Claimable Rewards • Low Fixed

Update
Addressed in: `511a9fce43f09c970b4f8f714466eb4eba2a9ba7` .
The client provided the following explanation:

```
I applied the recommendation, now disconnect fails if there are rewards to be claimed (unless forced).
```

File(s) affected: `CompoundV3InvestStrategy.sol`

Description: In the `CompoundV3InvestStrategy.disconnect()` function, the logic verifies that the `cToken` balance is zero before allowing a disconnect. However, the function does not account for any claimable rewards that may be pending in the rewards manager, potentially leaving yield unclaimed.

Recommendation: Update the disconnect function to verify not only that the `cToken` balance is zero but also that no claimable rewards remain in the rewards manager.

ENS-2 Inconsistent and Collidable Selector Generation for Strategy Forwarding • Low Fixed

Update
Addressed in: `743b22a44f13f1d1b02fd7fea0c4d785d3f380ef` .
The client provided the following explanation:

> ```
> I applied the recommendation, now getForwardToStrategySelector uses keccak256 instead of XOR, using
> only the address of the strategy and the method as inputs (not using the index anymore).
> ```
>
> The 4-byte selector for forwarding calls to strategies is now derived by hashing the `strategy` address and a `uint8` method, then extracting the first four bytes of the result. Because different inputs can map to the same four-byte prefix, this approach carries a collision risk that could overlap with access control permissions. We recommend having a test function that checks for collisions using expected deployment addresses.

File(s) affected: `AccessManagedMSV.sol`, `MSVBase.sol`

Description: In the `getForwardToStrategySelector()` function, the selector for forwarding calls to strategies is constructed using a combination of the strategy index, method ID, and the first four bytes of the strategy's address. This approach introduces two key issues:

1. Potential Collisions: The use of XOR to combine truncated values is not a secure method of generating unique identifiers. XOR operations are commutative and linear, meaning different combinations of inputs can result in identical outputs, leading to unintended access control overlaps.
2. Selector Instability: Since the strategy index is included in the selector, any change to the order of strategies in the `_strategies` array will change the computed selector. This means that modifying the strategy list (such as calling `MSVBase.removeStrategy()`) will require resetting access control permissions, leading to operational friction and potential misconfigurations.

Recommendation: Instead of using XOR-based manual selector construction, consider the following alternatives:

- Use `keccak256(abi.encode(strategyAddress, methodID))` and extract the first four bytes to generate a more robust and collision-resistant selector.
- Avoid including the strategy index in the selector construction. Instead, base the selector solely on immutable identifiers, such as the strategy's address and the method being invoked.

ENS-3 Inconsistent Use of the `force` in `MSVBase.removeStrategy()`

• Low Fixed

> **Update**
>
> Addressed in: `f0f32edfe7699657a6191efb08ef9bc74a7bd763`.
> The client provided the following explanation:
>
> ```
> I applied the recommendation, now CannotRemoveStrategyWithAssets can only be raised from MSVBase when
> force=false.
> ```

File(s) affected: `MSVBase.sol`

Description: In the `MSVBase.removeStrategy()` function, there is an early check that reverts if `strategy.totalAssets() != 0` without considering the `force` parameter. The function should allow removal by letting the strategy handle asset checks in `strategy.dcDisconnect(force)`.

Recommendation: Use `if (!force && strategy.totalAssets() != 0) revert CannotRemoveStrategyWithAssets();` so that forced removals bypass the check. This update will allow the forced disconnect via `strategy.dcDisconnect(force)` as intended.

ENS-4 `_maxWithdrawable()` and `_maxDepositable()` May Overflow

• Low Fixed

> **Update**
>
> Addressed in: `e272a7f252eac871e6422bf4dd5f75dc6898ea94`.
> The client provided the following explanation:
>
> ```
> I applied the recommendation, now the methods use OZ's Math.tryAdd and return either limit or
> type(uint256).max when add fails.
> ```

File(s) affected: `MSVBase.sol`,

Description: The function `_maxWithdrawable()` sums up the maximum withdrawal amount of each strategy related to the vault and returns the sum up to a `limit`. However, depending on the strategies, the sum may evaluate to greater than `type(uint256).max` before a `return` statement is reached, resulting in the function reverting.

The same principle applies to `_maxDepositable()` which is implemented similarly.

Recommendation: Check whether the sum would ever exceed `type(uint256).max`. If it would, then either return `limit` or `type(uint256).max`.

ENS-5
Hard-Coded `_price` Can Become Stale and Inaccurate in the Event of a Depeg

● Low ⓘ Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

```
The SwapStableInvestStrategy is intented to be used with stablecoins or other situations where the peg
is supposed to be strong. Some use cases are investing in USDM (yield bearing T-bill), or converting
from Bridged USDC to Native USDC to get better yields on AAVE.
  In such cases, is hard to anticipate what to do in a depeg situation, so I don't think any code
change can improve this situation that much.
  The current implementation, playing with the maxSlippage allows both panic sell (setting the
maxSlippage high and withdrawing all the funds), and ad-hoc pause (setting maxSlippage=0).
  Also, another alternative to pause might be setting the strategy at the bottom of the deposit and
withdraw queues.
```

File(s) affected: `SwapStableInvestStrategy.sol`

Description: The `SwapStableInvestStrategy` holds a fixed `_price` for the exchange rate between `_asset` and `_investAsset`. While this is sufficient for closely pegged assets, it becomes a risk if the peg is lost (e.g., stablecoin depegs) or if `_price` remains hard-coded despite market shifts. Users may then encounter large slippage, receive fewer tokens, or lose slippage protection if `_price` no longer matches real conditions. In extreme cases, it may become impossible to withdraw at the assumed `_price` unless `maxSlippage` is raised significantly.

Recommendation: Add a pause mechanism or allow a fast update of `_price` if a peg deviation occurs. Continuously monitor market rates and keep `_price` aligned with real conditions. This ensures prompt action in the event of depegging and avoids stale price risks.

ENS-6
Potential Double Counting in `totalAssets()` if Multiple Strategies Share the Same Underlying Asset

● Informational ⓘ Acknowledged

Update

Addressed in: `c1bedbade0383966cfa944e40d155e6c382e7972`.

The client acknowledged the issue and provided the following explanation:

```
We were aware of this, we added a warning comment in the README to make it more clear for anyone adding
new strategies.
```

File(s) affected: `MSVBase.sol`

Description: In the current design, each strategy's `totalAssets()` reflects that strategy's own asset (e.g. `aToken` and `cToken` and `asset`) holdings. Since the strategies do not overlap in the assets they hold, the vault's global `totalAssets()` remains accurate. However, if a future strategy reuses the same underlying asset as another existing strategy, the vault could mistakenly double-count those holdings when summing across all strategies.

Recommendation: Keep each asset type unique to a single strategy to avoid overlap, or otherwise ensure that `totalAssets()` across multiple strategies is calculated without duplication. If new strategies are added in the future, verify they do not manage the same asset concurrently.

ENS-7
Changing Slot Size Temporarily Allows Excessive Outflow

● Undetermined ⓘ Acknowledged

Update

Addressed in: `8545f261678ce45bad5285a2705aca7587e6b9c9`.

The client acknowledged the issue and provided the following explanation:

```
The method that changes the slotSize is a method that will be granted to an admin. Actually the same
method can also change the limit.
  We were aware about the fact that changing the slots resets the outflows (there is a test around
that).
  We just added a comment to the method to make it more evident.
```

File(s) affected: `OutflowLimitedAMMSV.sol`

Description: In the `OutflowLimitedAMMSV` contract, the `slotsize` is used as part of the key to track outflow data. Changing the `slotsize` shifts the upper bits of the key, effectively using a new "namespace" where old data is kept separate and not included in future checks. For the duration of the next `slotsize`, the `SlotIndex prevSlot = SlotIndex.wrap(SlotIndex.unwrap(slot) - 1);` will point to empty data and not contribute to the delta calculation. This resets the outflow history without deleting past records.

Exploit Scenario:
1. The outflow has reached the maximum for the current `slotsize`.
2. The admin updates the `slotsize`, thereby shifting the keys.
3. New outflow data is stored under keys computed with the new slotsize.
4. Future outflow checks only consider these new keys, ignoring the historical data and temporarily allowing for a greater outflow than might be intended.

Recommendation: If it is vital for the protocol to maintain the outflow limits through the change of the `slotsize`, consider updating the implementation to cache the previous slots outflow.

Auditor Suggestions

S1 Missing Input Validation

Mitigated

Update

Addressed in: `61869e53d69ec1b97c18cbb97bd40c782cc071d5`.

The client provided the following explanation:

```
We applied most of the recommendations, please check the pull request message
https://github.com/ensuro/vaults/pull/30 with the explanations of why we haven't applied some.
```

File(s) affected: `SwapStableInvestStrategy.sol`, `AccessManagedMSV.sol`, `CompoundV3InvestStrategy.sol`

Description: It is crucial to validate inputs, even if the inputs come from trusted addresses, to avoid human error. A lack of robust input validation can only increase the likelihood and impact in the event of mistakes.

The following is a list of places that can potentially benefit from stricter input validation:

1. `SwapStableInvestStrategy.sol`:
   1. Fixed the decimal `asset_` of the `constructor()` function should be less than `18`.
   2. Fixed the decimal `investAsset_` of the `constructor()` function should be less than `18`.
   3. Acknowledged the `swapConfig.maxSlippage` of the `_setSwapConfig` should be less than `WAD`.
2. `AccessManagedMSV.sol`:
   1. Acknowledged the `asset_` of `initialize()` should not be zero address.
3. `CompoundV3InvestStrategy.sol`:
   1. Mitigated the inputs of `constructor()` should not be zero addresses.

Recommendation: Add the validations and checks listed in the description.

S2 Require a Sorted Strategy Array to Save Gas

Acknowledged

Update

The client acknowledged the suggestion and provided the following explanation:

```
We decided not to apply this recommendation. The reasons are:
1. The gas improvement is not so relevant, since it's only on initialization.
2. Forcing the strategies to be in order will make more complex the initialization where you have to
   send several aligned arrays.
3. Forcing an ordered array will make more complex the tools we use to deploy the contracts.
```

File(s) affected: `MSVBase.sol`

Description: In `__MSVBase_init_unchained()`, a nested loop is used to check for duplicate strategy addresses. This check increases gas costs as the number of strategies grows. The intended functionality is to ensure that no strategy appears twice, but the current method is O(n^2).

```
for (uint256 i = 0; i < strategies_.length; i++) {
    for (uint256 j = 0; j < i; j++) {
        if (strategies_[i] == strategies_[j]) revert DuplicatedStrategy(strategies_[i]);
    }
}
```

Recommendation: Require that the input array of strategies is sorted (for example, in ascending order by address). This allows the contract to check for duplicates in a single pass.

```
for (uint256 i = 1; i < strategies_.length; i++) {
    if (strategies_[i] <= strategies_[i - 1]) revert DuplicatedStrategy(strategies_[i]);
}
```

S3 Gas Optimization

Fixed

Update
Addressed in: `4e7e83227bd8e431916d5e6dff4faf1279e2ccfe`.
The client provided the following explanation:

```
Change applied, thanks!
```

Description: Gas can be saved by using `++i` instead of `i++` in order to save gas when incrementing through `for` loops.

Recommendation: Make the adjustment.

S4 'Dead' Code

Fixed

Update
Addressed in: `f06a545282e2f2007d660cc1a9e9f7d164c1ef92`.
The client provided the following explanation:

```
Change applied. Also, I fixed visibility of constant WAD in the same file that was public when it
should be private.
```

File(s) affected: `SwapStableInvestStrategy.sol`

Related Issue(s): SWC-131, SWC-135

Description: "Dead" code refers to code which is never executed and hence makes no impact on the final result of running a program. Dead code raises a concern, since either the code is unnecessary or the necessary code's results were ignored.

The following code is unused:

- `SwapStableInvestStrategy`
  - `bytes32 public constant SWAP_ADMIN_ROLE = keccak256("SWAP_ADMIN_ROLE");`

Recommendation: We recommend removing any unused code.

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

- `79f...187 ./vaults/contracts/SwapStableAaveV3InvestStrategy.sol`
- `e3d...6eb ./vaults/contracts/SwapStableInvestStrategy.sol`
- `ac6...d5b ./vaults/contracts/AaveV3InvestStrategy.sol`
- `464...158 ./vaults/contracts/CompoundV3InvestStrategy.sol`
- `975...f2d ./vaults/contracts/MSVBase.sol`
- `d52...a56 ./vaults/contracts/AccessManagedMSV.sol`
- `c6c...4ae ./vaults/contracts/AccessManagedProxy.sol`
- `86d...5a7 ./vaults/contracts/InvestStrategyClient.sol`
- `bed...6e3 ./vaults/contracts/OutflowLimitedAMMSV.sol`
- `a0f...35c ./vaults/contracts/interfaces/IInvestStrategy.sol`
- `475...6cb ./vaults/contracts/interfaces/IExposeStorage.sol`

Tests

- `3d3...4c4 ./test/test-swap-stable-invest-strategy.js`
- `749...371 ./test/test-compound-v3-vault.js`
- `17e...59f ./test/test-storage-gaps.js`
- `ef7...d5e ./test/test-single-strategy-erc4626.js`
- `7ea...213 ./test/test-limit-outflow.js`
- `e79...ce1 ./test/test-swap-stable-aave-v3-invest-strategy.js`
- `f6a...9fa ./test/utils.js`
- `e39...61b ./test/test-multi-strategy-erc4626.js`
- `109...93b ./test/test-integration-fork.js`

Toolset

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

Setup

Tool Setup:
- Slither v0.11.0

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

We were unable to run Slither on the protocol's codebase.

Test Suite Results

```
  CompoundV3ERC4626 contract tests
    ✔ Checks vault inititializes correctly (204ms)
    ✔ Checks vault constructs with disabled initializer [CompoundV3ERC4626]
    ✔ Checks entering the vault is permissioned, exit isn't [!SwapStableAAVEV3Strategy] (154ms)
    ✔ Checks vault accrues compound earnings (168ms)
    ✔ Checks rewards can be harvested [!AAVEV3Strategy] [!SwapStableAAVEV3Strategy] (468ms)
    ✔ Checks only authorized user can change swap config [!AAVEV3Strategy] (430ms)
    ✔ Checks can't deposit or withdraw when Compound is paused [!AAVEV3Strategy]
[!SwapStableAAVEV3Strategy] (157ms)

  CompoundV3Strategy contract tests
    ✔ Checks vault inititializes correctly (202ms)
    ✔ Checks strategy can't be constructed with rewards=0 [CompoundV3Strategy]
    ✔ Checks reverts if extraData is sent on initialization [!CompoundV3ERC4626] (100ms)
    ✔ Checks entering the vault is permissioned, exit isn't [!SwapStableAAVEV3Strategy] (241ms)
    ✔ Checks vault accrues compound earnings (388ms)
    ✔ Checks rewards can be harvested [!AAVEV3Strategy] [!SwapStableAAVEV3Strategy] (672ms)
    ✔ Checks it can't disconnect without harvesting rewards [CompoundV3Strategy] (728ms)
    ✔ Checks only authorized user can change swap config [!AAVEV3Strategy] (541ms)
    ✔ Checks can't deposit or withdraw when Compound is paused [!AAVEV3Strategy]
[!SwapStableAAVEV3Strategy] (405ms)
    ✔ Checks only authorized can setStrategy [CompoundV3Strategy] (686ms)
    ✔ Checks methods can't be called directly [!CompoundV3ERC4626]

  AAVEV3Strategy contract tests
    ✔ Checks vault inititializes correctly (184ms)
    ✔ Checks reverts if extraData is sent on initialization [!CompoundV3ERC4626] (87ms)
    ✔ Checks entering the vault is permissioned, exit isn't [!SwapStableAAVEV3Strategy] (480ms)
    ✔ Checks vault accrues compound earnings (749ms)
    ✔ Checks can't operate when AAVE is paused [AAVEV3Strategy] [SwapStableAAVEV3Strategy] (1115ms)
    ✔ Checks only authorized can setStrategy [AAVEV3Strategy] [SwapStableAAVEV3Strategy] (1064ms)
    ✔ Checks methods can't be called directly [!CompoundV3ERC4626]

  CompoundV3Strategy+AccessManaged contract tests
    ✔ Checks vault inititializes correctly (547ms)
    ✔ Checks reverts if extraData is sent on initialization [!CompoundV3ERC4626]
    ✔ Checks entering the vault is permissioned, exit isn't [!SwapStableAAVEV3Strategy] (126ms)
    ✔ Checks vault accrues compound earnings (461ms)
    ✔ Checks rewards can be harvested [!AAVEV3Strategy] [!SwapStableAAVEV3Strategy] (965ms)
    ✔ Checks it can disconnect without harvesting rewards if forced [CompoundV3Strategy+AccessManaged]
(456ms)
    ✔ Checks only authorized user can change swap config [!AAVEV3Strategy] (868ms)
    ✔ Checks can't deposit or withdraw when Compound is paused [!AAVEV3Strategy]
[!SwapStableAAVEV3Strategy] (579ms)
    ✔ Checks methods can't be called directly [!CompoundV3ERC4626]

  SwapStableAAVEV3Strategy contract tests
    ✔ Checks vault inititializes correctly (247ms)
    ✔ Checks reverts if extraData is sent on initialization [!CompoundV3ERC4626] (118ms)
    ✔ Checks vault accrues compound earnings (2233ms)
    ✔ Checks only authorized user can change swap config [!AAVEV3Strategy] (593ms)
    ✔ Checks can't operate when AAVE is paused [AAVEV3Strategy] [SwapStableAAVEV3Strategy] (2455ms)
    ✔ Checks only authorized can setStrategy [AAVEV3Strategy] [SwapStableAAVEV3Strategy] (2035ms)
    ✔ Checks methods can't be called directly [!CompoundV3ERC4626]

  MultiStrategy Integration fork tests
    ✔ Can perform a basic smoke test (1969ms)

  AMProxy+OutflowLimitedAMMSV contract tests
    ✔ Initializes the vault correctly (916ms)
    ✔ Handles withdrawal limits correctly for multiple LPs and ensures limits are respected across time
periods (4604ms)
    ✔ Respects withdrawal limits and resets daily limit after time advancement (3717ms)
    ✔ Prevents withdrawal when combined daily limits from consecutive slots are surpassed (3791ms)
    ✔ Prevents withdrawal for consecutive daily limits, but forgets two days ago withdrawals (3405ms)
    ✔ Checks that change in slot size resets the limits (3288ms)
    ✔ Allows accumulated withdrawals up to the daily limit and prevents exceeding it (4827ms)
    ✔ Allows accumulated withdrawals up to the daily limit and prevents exceeding it - mint/redeem
(4452ms)
    ✔ Allows accumulated withdrawals up to the daily limit and prevents exceeding it -
  mint/redeem/deposit/withdraw (8870ms)
      ✔ Sets and resets delta using LOM__changeDelta correctly (615ms)

    MultiStrategyERC4626 contract tests
      ✔ Checks vault constructs with disabled initializer [MultiStrategyERC4626] (210ms)
      ✔ Initializes the vault correctly (188ms)
      ✔ Initialization fails if strategy connect fails (36741ms)
      ✔ It checks calls to forwardToStrategy require permission [MultiStrategyERC4626] (831ms)
      ✔ It sets and reads the right value from strategy storage (711ms)
      ✔ It fails when initialized with wrong parameters (1691ms)
      ✔ It respects the order of deposit and withdrawal queues (2259ms)
      ✔ It respects the order of deposit and authorized user can rebalance (983ms)
      ✔ It can addStrategy and is added at the bottom of the queues (707ms)
      ✔ It can add up to 32 strategies (2290ms)
      ✔ It can removeStrategy only if doesn't have funds unless forced (559ms)
      ✔ It can removeStrategy only if doesn't have funds (1101ms)
      ✔ It can removeStrategy in different order (546ms)
      ✔ It can change the depositQueue if authorized (2452ms)
      ✔ It can change the withdrawQueue if authorized (2305ms)
      ✔ It can replaceStrategy if authorized (1266ms)
      ✔ Initialization fails if any strategy and vault have different assets (103ms)
      ✔ Fails to add strategy to vault if assets are different (268ms)
      ✔ Fails to replace strategy to vault if assets are different (197ms)

    AMProxy+AccessManagedMSV contract tests
      ✔ Checks vault constructs with disabled initializer [!MultiStrategyERC4626] (233ms)
      ✔ Initializes the vault correctly (690ms)
      ✔ Initialization fails if strategy connect fails (34270ms)
      ✔ It checks calls to forwardToStrategy require permission [!MultiStrategyERC4626] (1793ms)
      ✔ It sets and reads the right value from strategy storage (1634ms)
      ✔ It fails when initialized with wrong parameters (1873ms)
      ✔ It respects the order of deposit and withdrawal queues (3553ms)
      ✔ It respects the order of deposit and authorized user can rebalance (1480ms)
      ✔ It can addStrategy and is added at the bottom of the queues (1232ms)
      ✔ It can add up to 32 strategies (2758ms)
      ✔ It can removeStrategy only if doesn't have funds unless forced (1081ms)
      ✔ It can removeStrategy only if doesn't have funds (1801ms)
      ✔ It can removeStrategy in different order (1117ms)
      ✔ It can change the depositQueue if authorized (3093ms)
      ✔ It can change the withdrawQueue if authorized (3087ms)
      ✔ It can replaceStrategy if authorized (1905ms)
      ✔ Initialization fails if any strategy and vault have different assets (101ms)
      ✔ Fails to add strategy to vault if assets are different (629ms)
      ✔ Fails to replace strategy to vault if assets are different (593ms)

    AMProxy+OutflowLimitedAMMSV contract tests
      ✔ Checks vault constructs with disabled initializer [!MultiStrategyERC4626] (251ms)
      ✔ Initializes the vault correctly (742ms)
      ✔ Initialization fails if strategy connect fails (34898ms)
      ✔ It checks calls to forwardToStrategy require permission [!MultiStrategyERC4626] (1822ms)
      ✔ It sets and reads the right value from strategy storage (1621ms)
      ✔ It fails when initialized with wrong parameters (1835ms)
      ✔ It respects the order of deposit and withdrawal queues (3838ms)
      ✔ It respects the order of deposit and authorized user can rebalance (1499ms)
      ✔ It can addStrategy and is added at the bottom of the queues (1283ms)
      ✔ It can add up to 32 strategies (2825ms)
      ✔ It can removeStrategy only if doesn't have funds unless forced (1154ms)
      ✔ It can removeStrategy only if doesn't have funds (1883ms)
      ✔ It can removeStrategy in different order (1113ms)
      ✔ It can change the depositQueue if authorized (3251ms)
      ✔ It can change the withdrawQueue if authorized (3186ms)
      ✔ It can replaceStrategy if authorized (1966ms)
      ✔ Initialization fails if any strategy and vault have different assets (104ms)
      ✔ Fails to add strategy to vault if assets are different (700ms)
      ✔ Fails to replace strategy to vault if assets are different (654ms)

    SingleStrategyERC4626 contract tests
      ✔ Initializes the vault correctly (140ms)
      ✔ Initialization fails if strategy connect fails (71ms)
      ✔ Initialization fails if extra data is sent (67ms)
      ✔ It sets and reads the right value from strategy storage (130ms)
      ✔ If disconnect fails it can't change the strategy unless forced (190ms)
    ✔ Initialization fails if strategy and vault have different assets (73ms)
    ✔ Fails to add strategy to vault if assets are different
    ✔ Checks only GUARDIAN_ROLE can upgrade
    ✔ Checks only DEFAULT_ADMIN_ROLE can setRoleAdmin, then others can set specific roles

  Storage Gaps
    ✔ CompoundV3ERC4626 has a proper storage gap (120ms)
    ✔ SingleStrategyERC4626 has a proper storage gap (123ms)
    ✔ MultiStrategyERC4626 has a proper storage gap (128ms)

  SwapStableAaveV3InvestStrategy contract tests USDC(6)->USDC_NATIVE(6) with AAVE
    ✔ Initializes the vault correctly with AAVE (475ms)
    ✔ Deposit and accounting works (610ms)
    ✔ Withdraw works with original slippage and validates balances (1215ms)
    ✔ maxWithdraw returns correct values initially, afert deposit & withdraw (1443ms)
    ✔ Checks methods can't be called directly
    ✔ Should disconnect when strategy change & when authorized (331ms)
    ✔ Disconnect doesn't fail when changing strategy (903ms)

  SwapStableAaveV3InvestStrategy contract tests USDC(6)->USDT(6) with AAVE
    ✔ Initializes the vault correctly with AAVE (225ms)
    ✔ Deposit and accounting works (507ms)
    ✔ Withdraw works with original slippage and validates balances (1231ms)
    ✔ maxWithdraw returns correct values initially, afert deposit & withdraw (1503ms)
    ✔ Checks methods can't be called directly
    ✔ Should disconnect when strategy change & when authorized (277ms)
    ✔ Disconnect doesn't fail when changing strategy (932ms)

  SwapStableAaveV3InvestStrategy contract tests USDC(6)->DAI(18) with AAVE
    ✔ Initializes the vault correctly with AAVE (221ms)
    ✔ Deposit and accounting works (553ms)
    ✔ Withdraw works with original slippage and validates balances (1257ms)
    ✔ maxWithdraw returns correct values initially, afert deposit & withdraw (1409ms)
    ✔ Checks methods can't be called directly
    ✔ Should disconnect when strategy change & when authorized (270ms)
    ✔ Disconnect doesn't fail when changing strategy (975ms)

  SwapStableInvestStrategy contract tests A(6)->B(6)
    ✔ Initializes the vault correctly (379ms)
    ✔ Deposit and accounting works (259ms)
    ✔ Withdraw function executes swap correctly and emits correct events - currA(6) -> currB(6) (765ms)
    ✔ Withdraw function fails (435ms)
    ✔ Deposit and accounting works when price != 1 (593ms)
    ✔ Deposit and accounting works when price != 1 and slippage (526ms)
    ✔ Checks methods can't be called directly
    ✔ Checks onlyRole modifier & setSwapConfig function (172ms)
    ✔ Should return the correct swap configuration (187ms)
    ✔ setStrategy should work and disconnect strategy when authorized (181ms)
    ✔ Disconnect doesn't fail when changing strategy (484ms)
    ✔ Disconnect without assets doesn't revert (210ms)

  SwapStableInvestStrategy contract tests A(6)->M(18)
    ✔ Initializes the vault correctly (186ms)
    ✔ Deposit and accounting works (253ms)
    ✔ Withdraw function executes swap correctly and emits correct events - currA(6) -> currB(6) (674ms)
    ✔ Withdraw function fails (422ms)
    ✔ Deposit and accounting works when price != 1 (616ms)
    ✔ Deposit and accounting works when price != 1 and slippage (516ms)
    ✔ Checks methods can't be called directly
    ✔ Checks onlyRole modifier & setSwapConfig function (182ms)
    ✔ Should return the correct swap configuration (152ms)
    ✔ setStrategy should work and disconnect strategy when authorized (182ms)
    ✔ Disconnect doesn't fail when changing strategy (496ms)
    ✔ Disconnect without assets doesn't revert (192ms)

  SwapStableInvestStrategy contract tests M(18)->X(18)
    ✔ Initializes the vault correctly (185ms)
    ✔ Deposit and accounting works (240ms)
    ✔ Withdraw function executes swap correctly and emits correct events - currA(6) -> currB(6) (723ms)
    ✔ Withdraw function fails (421ms)
    ✔ Deposit and accounting works when price != 1 (544ms)
    ✔ Deposit and accounting works when price != 1 and slippage (491ms)
    ✔ Checks methods can't be called directly
    ✔ Checks onlyRole modifier & setSwapConfig function (192ms)
    ✔ Should return the correct swap configuration (155ms)
    ✔ setStrategy should work and disconnect strategy when authorized (247ms)
    ✔ Disconnect doesn't fail when changing strategy (515ms)
    ✔ Disconnect without assets doesn't revert (197ms)

  SwapStableInvestStrategy contract tests M(18)->A(6)
    ✔ Initializes the vault correctly (144ms)
    ✔ Deposit and accounting works (255ms)
    ✔ Withdraw function executes swap correctly and emits correct events - currA(6) -> currB(6) (762ms)
    ✔ Withdraw function fails (445ms)
    ✔ Deposit and accounting works when price != 1 (521ms)
    ✔ Deposit and accounting works when price != 1 and slippage (516ms)
    ✔ Checks methods can't be called directly
    ✔ Checks onlyRole modifier & setSwapConfig function (181ms)
    ✔ Should return the correct swap configuration (167ms)
    ✔ setStrategy should work and disconnect strategy when authorized (194ms)
    ✔ Disconnect doesn't fail when changing strategy (516ms)
    ✔ Disconnect without assets doesn't revert (215ms)

  SwapStableInvestStrategy constructor tests
    ✔ It reverts when asset or invest asset has >18 decimals (152ms)


  191 passing (5m)
```

Code Coverage

Code coverage was generated using `npx hardhat coverage` on the Polygon POS chain, and while the overall test suite coverage is high, we recommend adding tests for scheduled calls through the access manager as well as additional tests for any recently fixed issues. We commend the team for their existing, robust test suite, which already covers many scenarios and non-happy paths thoroughly.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.49	92.04	98.66	99.53	
AaveV3InvestStrategy.sol	95.65	81.25	92.31	95.65	103
AccessManagedMSV.sol	100	90	92.86	100	
AccessManagedProxy.sol	100	100	100	100	
CompoundV3ERC4626.sol	97.14	85	100	100	
CompoundV3InvestStrategy.sol	95.35	90.63	100	100	
InvestStrategyClient.sol	100	100	100	100	
MSVBase.sol	100	97.17	100	100	
MultiStrategyERC4626.sol	100	96.43	100	100	
OutflowLimitedAMMSV.sol	100	100	100	100	
PermissionedERC4626.sol	100	78.57	100	100	
SwapStableAaveV3InvestStrategy.sol	96.3	87.5	100	95.83	92

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
SwapStableInvestStrategy.sol	97.22	93.33	100	100	
contracts/interfaces/	100	100	100	100	
IExposeStorage.sol	100	100	100	100	
IInvestStrategy.sol	100	100	100	100	
All files	98.49	92.04	98.66	99.53	

Changelog

- 2025-03-03 - Initial report
- 2025-03-17 - Final report

About Quantstamp

Timeliness of content

Notice of confidentiality

Links to other websites

Disclaimer

Quantstamp