Quantstamp Security Assessment Certificate

December 13th 2022 — Quantstamp Verified

Sturdy

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

Executive Summary

Type	DeFi
Auditors	Zeeshan Meghji, Auditing Engineer Guillermo Escobero, Security Auditor Poming Lee, Senior Research Engineer
Timeline	2022-08-29 through 2022-09-22
EVM	Paris
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Sturdy Documentation
Documentation Quality	Medium
Test Quality	Undetermined

Source Code

Repository	Commit
sturdy (initial audit)	d5a1660
sturdy (fix review)	0828965

Total Issues	**26**	(14 Resolved)
High Risk Issues	0	(0 Resolved)
Medium Risk Issues	**7**	(7 Resolved)
Low Risk Issues	**7**	(4 Resolved)
Informational Risk Issues	**6**	(1 Resolved)
Undetermined Risk Issues	**6**	(2 Resolved)

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	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.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
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).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Sturdy is a lending protocol that leverages user collateral to produce yield and provide interest-free loans and stable lending rewards. Collateral deposited into the protocol is converted to yield-bearing tokens through another integrated DeFi protocol, such as Lido or Yearn Finance. The yield produced by these protocols is paid out as interest to lenders. Interest rates for borrowers remain at zero until a utilization threshold is reached.

During the audit, we found several serious issues which pose a risk to users of the Sturdy protocol. In particular, QSP-1 allows one compromised vault to drain the entire protocol's funds. QSP-2 results in users being unable to withdraw their collateral under some circumstances. QSP-3 could allow for replay attacks across multiple forks of a chain.

The Sturdy team has implemented tests to verify the protocol's behavior in numerous situations. These include integration tests based on a local mainnet fork. However, we cannot verify the quality of these tests as no method of obtaining code coverage results has been provided by the team. It is thus possible that significant parts of the code have not been sufficiently tested.

The protocol relies on oracles that are not within the scope of the audit. If the oracles do not function as expected, the protocol could catastrophically fail. The protocol also depends on many external protocols, including Lido and Yearn Finance. These protocols often hold the users' collateral. If these protocols were compromised, the users' funds could be lost regardless of the security of the Sturdy contracts.

The Sturdy team has cooperated with the auditors and has helpfully answered our questions. We strongly recommend that the Sturdy team fixes all issues found in the report. We further recommend that the Sturdy team provides a way to evaluate code coverage to ensure that every part of the protocol has been tested.

Update:

The Sturdy team has either fixed, mitigated, or acknowledged all issues found within the report. All vulnerabilities of medium severity or higher have been either fixed or sufficiently mitigated. However, we still recommend that the team fixes some of the acknowledged issues, including QSP-13, QSP-14, QSP-17, QSP-19. QSP-21, QSP-22, and QSP-26.

The Sturdy team has not provided a way to check the code coverage of the protocol's contracts. Without a code coverage report, we cannot determine whether parts of the protocol remain untested. We recommend that the Sturdy team implement a code coverage report and increase the test coverage of any contracts lacking it.

ID	Description	Severity	Status
QSP-1	One Compromised Vault Can Drain All Funds	^ Medium	Mitigated
QSP-2	User May Be Unable to Withdraw Assets	^ Medium	Fixed
QSP-3	Replay Attack on Forked Chain	^ Medium	Fixed
QSP-4	Inappropriate Operations Can Be Performed on Reserve	^ Medium	Mitigated
QSP-5	Cannot Unregister Vault	^ Medium	Fixed
QSP-6	Violation of Checks-Effects-Interactions Pattern in `LendingPoolCollateralManager`	^ Medium	Fixed
QSP-7	Violation of Checks-Effects-Interactions Pattern in `ConvexCurveLPVault`	^ Medium	Fixed
QSP-8	`YearnVault.processYield()` May Run Out of Gas	v Low	Acknowledged
QSP-9	Can Enable Borrowing and Collateral on Same Reserve	v Low	Fixed
QSP-10	Assets with More Decimal Places than `18` Are Implicitly Disallowed	v Low	Fixed
QSP-11	`YearnRETHWstETHVault` May Claim Less Yield if Curve Pool Is Manipulated	v Low	Fixed
QSP-12	Can Receive Less than Expected Amount From Swap	v Low	Fixed
QSP-13	Ownership Can Be Renounced	v Low	Acknowledged
QSP-14	Reentrancy Risks Can Be Mitigated	v Low	Acknowledged
QSP-15	Reliance on External Contracts to Secure Funds	O Informational	Acknowledged
QSP-16	User Funds Will Be Blocked if Sent Directly to Vaults	O Informational	Acknowledged
QSP-17	Unlocked Pragma	O Informational	Acknowledged
QSP-18	Clone-and-Own	O Informational	Acknowledged
QSP-19	Token Balances Not Validated	O Informational	Acknowledged
QSP-20	Missing Input Validation	O Informational	Fixed
QSP-21	`vaultYieldInPrice` Returns Incorrect Value for All Vaults	? Undetermined	Acknowledged
QSP-22	Stable Debt Functionality Not Removed	? Undetermined	Acknowledged
QSP-23	Misleading Behavior of `withdrawOnLiquidation()` in `LidoVault`	? Undetermined	Fixed
QSP-24	`getYieldAmount()` Returns Incorrect Value for `ConvexCurveLPVault`	? Undetermined	Fixed
QSP-25	Reliance on Oracle Contracts that Are Not Within the Scope	? Undetermined	Acknowledged
QSP-26	Fixed Slippage Value	? Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

DISCLAIMER:
If the final commit hash provided by the client contains features that are not in the scope of the audit or a fix review, those features are excluded from the consideration in this report. Please note that this audit only covers the following contracts:

- contracts/protocol/configuration/LendingPoolAddressesProvider.sol
- contracts/protocol/configuration/LendingPoolAddressesProviderRegistry.sol
- contracts/protocol/lendingpool/CollateralAdapter.sol
- contracts/protocol/lendingpool/DefaultReserveInterestRateStrategy.sol
- contracts/protocol/lendingpool/LendingPool.sol
- contracts/protocol/lendingpool/LendingPoolCollateralManager.sol
- contracts/protocol/lendingpool/LendingPoolConfigurator.sol
- contracts/protocol/libraries/sturdy-upgradeability/BaseImmutableAdminUpgradeabilityProxy.sol
- contracts/protocol/libraries/sturdy-upgradeability/InitializableImmutableAdminUpgradeabilityProxy.sol
- contracts/protocol/libraries/sturdy-upgradeability/VersionedInitializable.sol

- contracts/protocol/libraries/logic/GenericLogic.sol
- contracts/protocol/libraries/logic/ReserveLogic.sol
- contracts/protocol/libraries/logic/ValidationLogic.sol
- contracts/protocol/tokenization/DebtTokenBase.sol
- contracts/protocol/tokenization/AToken.sol
- contracts/protocol/tokenization/ATokenForCollateral.sol
- contracts/protocol/tokenization/SturdyInternalAsset.sol
- contracts/protocol/tokenization/VariableDebtToken.sol
- contracts/protocol/vault/ethereum/LidoVault.sol
- contracts/protocol/vault/ethereum/YearnRETHWstETHVault.sol
- contracts/protocol/vault/ethereum/ConvexVault/ConvexCurveLPVault.sol
- contracts/protocol/vault/GeneralVault.sol
- contracts/protocol/vault/IncentiveVault.sol
- contracts/protocol/vault/fantom/YearnVault.sol

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

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. 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:
   i. 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.
   ii. 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, clarify, 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.

Toolset

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

Setup

Tool Setup:

- Slither v0.8.3

Steps taken to run the tools:

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

Findings

QSP-1 One Compromised Vault Can Drain All Funds

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `protocol/lendingpool/LendingPool.sol`

Description: Several critical functions on the `LendingPool` contract are only callable by vaults and can be used to remove an arbitrary amount of any asset. These functions include `LendingPool.withdrawFrom()` and `LendingPool.getYield()`. However, both of these functions allow any vault to remove any asset from the `LendingPool` as seen on `LendingPool.sol#L196:require(_availableVaults[msg.sender] == true, Errors.VT_PROCESS_YIELD_INVALID);`. This violates the principle of least privilege as vaults should only be able to remove assets they deal with. For example, the `LidoVault` should only be able to withdraw stETH from the `LendingPool`. Currently, if even one vault was compromised, it could be used to drain the entire `LendingPool` of all assets.
We also note that any vault can deposit any asset through the `LendingPool.deposit()` and `LendingPool.depositYield()` functions. While the impact of this is less obvious, vaults should only be able to deposit assets that they deal with.

Recommendation: Modify the following functions so that vaults can only withdraw or deposit the assets they deal with. For example, the `LidoVault` should only be able to deposit and withdraw stETH.

- `LendingPool.withdrawFrom()`
- `LendingPool.getYield()`
- `LendingPool.deposit()`
- `LendingPool.depositYield()`

Update: The Sturdy team has mitigated the issue by restricting the `LendingPool.withdrawFrom()`, `LendingPool.getYield()` and `LendingPool.deposit()` functions such that they can only be called by vaults for the appropriate collateral assets. However, the same restriction has not been applied to the `LendingPool.depositYield()` function. We recommend that the team also adds the same restriction to the `LendingPool.depositYield()` function.

QSP-2 User May Be Unable to Withdraw Assets

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `protocol/lendingpool/LendingPool.sol`

Description: By design, the protocol requires users to withdraw their collateral from a vault. However, it is technically possible for users to withdraw directly from the `LendingPool` contract directly using the `LendingPool.withdraw()` function. For some tokens, such as stETH, directly withdrawing from the `LendingPool` contract does not appear to cause any issues. However, if a `SturdyInternalAsset` token is withdrawn directly from the `LendingPool` contract, there will be no way for a user to redeem the `SturdyInternalAsset` for the actual token it represents. The user will be left holding the valueless `SturdyInternalAsset` token and will not be able to withdraw their original assets.

Recommendation: Modify the `LendingPool.withdraw()` function so that if withdrawing a collateral token, it is only callable by the appropriate vault contract.

Update: The Sturdy team has fixed the issue by adding validation to the withdrawal functions so that only vaults can withdraw collateral from the lending pool.

QSP-3 Replay Attack on Forked Chain

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `protocol/tokenization/AToken.sol`, `protocol/tokenization/ATokenForCollateral.sol`

Description: The `AToken` and `ATokenForCollateral` contracts implement EIP-712 to accept off-chain signatures. While the signatures must include a `chainId`, we note that the `DOMAIN_SEPARATOR` for both contracts is defined as follows within the `initialize()` function:

```
assembly {
  chainId := chainid()
}

DOMAIN_SEPARATOR = keccak256(
  abi.encode(
    EIP712_DOMAIN,
    keccak256(bytes(aTokenName)),
    keccak256(EIP712_REVISION),
    chainId,
    address(this)
  )
);
```

Note that the `chainId` is only obtained during the initial computation of the `DOMAIN_SEPARATOR` during the initialization of the contract. If the chain is forked, then the signature could be replayed across both forks of the chain.

Recommendation: Rather than computing the `DOMAIN_SEPARATOR` once during initialization, it should be computed dynamically every time it is needed using a function. Thus the `chainId` component of the `DOMAIN_SEPARATOR` will differ across forks of the chain, and the signature will not be replayable.

Update: The Sturdy team has fixed the issue by dynamically computing the domain separator instead of using the cached value.

QSP-4 Inappropriate Operations Can Be Performed on Reserve

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `protocol/lendingpool/LendingPool.sol`, `protocol/lendingpool/LendingPoolConfigurator.sol`, `protocol/libraries/logic/ValidationLogic.sol`

Description: Unlike AAVE, the Sturdy protocol draws a line between collateral and debt assets. A collateral asset can never be a debt asset and vice versa. However, additional care must be taken to ensure that certain actions can only be performed on reserves of a particular type (debt or collateral). We have listed the cases where additional checks must be added to ensure that each reserve type is acted on appropriately:

- `LendingPool`
  - `depositYield()`: Validate that `asset` is not a collateral asset.
  - `getYield()`: Validate that `asset` is a collateral asset.

-
  - `_withdraw()`: Only emit `ReserveUsedAsCollateralDisabled` if `user.isUsingAsCollateral()` was `true` before.
  - `finalizeTransfer()`:
    - Only emit `ReserveUsedAsCollateralDisabled` if `user.isUsingAsCollateral()` was `true` before.
    - Only set `toConfig.setUsingAsCollateral(reserveId, true)` if the reserve represents a collateral asset.

- `LendingPoolConfigurator`:
  - `updateVariableDebtToken()`: Validate that `input.asset` is not a collateral asset.
  - `configureReserveAsCollateral()`:
    - If `asset` is a collateral asset, then validate that `ltv`, `liquidationThreshold` and `liquidationBonus` are non-zero.
    - It `asset` is not a collateral asset, then validate that `ltv`, `liquidationThreshold` and `liquidationBonus` are zero.

- `ValidationLogic`
  - `validateSetUseReserveAsCollateral()`: Validate that `reserve` represents a collateral asset if `useAsCollateral` is `true`.

Recommendation: Add the suggested checks to ensure whether the asset is a collateral asset or not. The `reserve.getCollateralEnabled()` can be used to check this where appropriate.

Update: The Sturdy team has mitigated the issue by adding most of the suggested checks to distinguish between collateral and debt assets. However, the team chose not to add the validation for the `ltv` value within the `LendingPool.configureReserveAsCollateral()` function for convenience of testing.

QSP-5 Cannot Unregister Vault

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `protocol/lendingpool/LendingPoolConfigurator.sol`

Description: A vault can be registered through the `LendingPoolConfigurator.registerVault()` function but cannot currently be unregistered in any way. Registered vaults have a high level of access to the `LendingPool` contract through the `withdrawFrom()`, `getYield()`, `deposit()` and `depositYield()` functions. Thus, if a vault is compromised in any way, it is important to have the ability to unregister it quickly.

Recommendation: Add an `unregister()` function to the `LendingPoolConfigurator` contract, which unregisters a vault so it can no longer access the `LendingPool` contract.

Update: The Sturdy team has fixed the issue by adding the function `LendingPoolConfigurator.unregisterVault()`, which allows the pool admin to unregister a vault.

QSP-6 Violation of Checks-Effects-Interactions Pattern in `LendingPoolCollateralManager`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `protocol/lendingpool/LendingPoolConfigurator.sol`

Description: The `LendingPoolCollateralManager.liquidationCall()` function does not follow the Check-Effects-Interactions pattern. This is because the function obtains the debt reserve from the liquidator after completing the internal accounting process in regard to the repayment of debt. It may be possible for the liquidator to sell the obtained collateral into debt tokens and then reenter the contract and finish the final external call.

Recommendation: Always follow the Check-Effects-Interactions pattern to avoid reentrancies. This can be done by obtaining the `debtReserve` provided by the liquidator before actually completing the internal accounting process in regard to the repayment of debt.

Update: The Sturdy team has fixed the issue by retrieving the `debtReserve` provided by the liquidator before completing the internal accounting.

QSP-7 Violation of Checks-Effects-Interactions Pattern in `ConvexCurveLPVault`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `protocol/vault/ethereum/ConvexVault/ConvexCurveLPVault.sol`

Description: The `ConvexCurveLPVault._withdraw()` function does not follow the Check-Effects-Interactions pattern. Specifically, `L239` should be performed after `L242` instead of the current order.

Recommendation: Always follow the Check-Effects-Interactions pattern to avoid reentrancy. This can be done by moving the call to `IERC20(curveLPToken).safeTransfer(_to, _amount)` to the end of the aforementioned function. Specifically, `L239` should be performed after `L242` instead of the current order.

Update: The Sturdy team has fixed the issue by refactoring the function to follow the checks-effects-interactions pattern.

QSP-8 `YearnVault.processYield()` May Run Out of Gas

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `protocol/vault/ethereum/YearnVault.sol`

Description: The `YearnVault.processYield()` function iterates over all debt assets multiple times and performs a swap for each asset. If the number of debt assets is sufficiently large, the function could easily run out of gas and fail.
The function first iterates over all reserves when it calls `LendingPool.getBorrowingAssetAndVolumes()` through the `YearnVault._getAssetYields()` function. Then the `YearnVault._getAssetYields()` function iterates over every debt asset as shown from `L274-L286` below:

```
for (uint256 i; i < length; ++i) {
  assetYields[i].asset = assets[i];
  if (i == length - 1) {
    // without calculation, set remained extra amount
    assetYields[i].amount = extraWETHAmount;
  } else {
```

```
    // Distribute wethAmount based on percent of asset volume
    assetYields[i].amount = _amount.percentMul(
      (volumes[i] * PercentageMath.PERCENTAGE_FACTOR) / totalVolume
    );
    extraWETHAmount -= assetYields[i].amount;
  }
}
```

The `YearnVault.processYield()` function then performs another iteration over every debt asset as shown from `L57-L62` below:

```
for (uint256 i; i < length; ++i) {
  // FTM -> Asset and Deposit to pool
  if (assetYields[i].amount != 0) {
    _convertAndDepositYield(assetYields[i].asset, assetYields[i].amount);
  }
}
```

Since `YearnVault._convertAndDepositYield()` will perform a swap, this will result in significantly increased gas usage, possibly enough to make the entire transaction fail.

Recommendation: Several solutions are possible:

1. Implement an upper bound for the number of debt assets.
2. Refactor the `YearnVault.processYield()` function so it can be called individually for each debt asset or for a limited number of debt assets at a time.
3. Break the `YearnVault.processYield()` function into two or more parts to decrease gas usage.

Update: The Sturdy team has indicated that only three debt assets are supported on Fantom. Since the number of assets is bounded, the function is unlikely to run out of gas. If more assets are added in the future, the team may consider refactoring the logic.

QSP-9 Can Enable Borrowing and Collateral on Same Reserve

Severity: *Low Risk*

Status: Fixed

File(s) affected: `protocol/lendingpool/LendingPoolConfigurator.sol`

Description: The `LendingPoolConfigurator.enableBorrowingOnReserve()` and `LendingPoolConfigurator.enableCollateralOnReserve()` functions enable borrowing and collateral on a reserve respectively. A reserve which can be borrowed should never be used as collateral and vice versa. However, it is currently possible to enable borrowing and collateral on the same reserver by calling the two functions with the same reserve `asset`.

Recommendation:

1. Add validation to the `LendingPoolConfigurator.enableBorrowingOnReserve()` to ensure that the reserve is not a collateral asset.
2. Add validation to the `LendingPoolConfigurator.enableCollateralOnReserve()` function to ensure that the reserve is not a debt asset.

Update: The Sturdy team has fixed the issue by implementing the suggested validation checks.

QSP-10 Assets with More Decimal Places than `18` Are Implicitly Disallowed

Severity: *Low Risk*

Status: Fixed

File(s) affected: `protocol/lendingpool/LendingPool.sol`, `protocol/lendingpool/LendingPoolCollateralManager.sol`, `protocol/lendingpool/LendingPoolConfigurator.sol`, `protocol/tokenization/ATokenForCollateral.sol`

Description: Multiple areas within the code implicitly assume that assets with more decimal places than `18` may not be used. The assumption is best illustrated by `ATokenFromCollateral#L159-L160`:

```
if (decimal < 18) amount = (share * 10**(18 - decimal)).rayMul(index);
else amount = share.rayMul(index);
```

We can see that the first condition accounts for when an asset token has less than `18` decimal places. The second `else` statement assumes that the asset has exactly `18` decimal places. However, no condition dealing with the case for greater than `18` decimal places exists. The same set of conditions can be found in multiple contracts, including `LendingPoolCollateralManager` and `LendingPool`.
There is currently no explicit validation check to ensure that the external asset has less than or equal to `18` decimal places. An ideal place to add this check would be when a reserve is initialized within the `LendingPoolConfigurator.batchInitReserve()` function.

Recommendation: Add a check to the `LendingPoolConfigurator.batchInitReserve()` function to ensure that any reserve being initialized is for an asset with `18` or fewer decimal places.

Update: The Sturdy team has fixed the issue by implementing a validation check to ensure that reserves cannot be created with more than `18` decimals.

QSP-11 `YearnRETHWstETHVault` May Claim Less Yield if Curve Pool Is Manipulated

Severity: *Low Risk*

Status: Fixed

File(s) affected: `protocol/vault/ethereum/YearnRETHWstETHVault.sol`

Description: The `YearnRETHWstETHVault.processYield()` function involves taking many steps to convert the yield token into WETH tokens. One of these steps is withdrawing liquidity from a Curve pool to retrieve wstETH as seen on `L125-L129`:

```
amountWstETH = ICurvePool(_poolAddress).remove_liquidity_one_coin(
  _amount,
  1,
  minWstETHAmount,
  address(this)
);
```

If a Curve pool is imbalanced, then the liquidity provider may have to pay a withdrawal fee when removing liquidity. A malicious user may cause an imbalance in the Curve pool before calling `YearnRETHWstETHVault.processYield()` to force the function to pay the penalty by withdrawing liquidity from the Curve pool.

Recommendation: Make the `YearnVault.processYield()` function only callable by a trusted address. Furthermore, the function should be called using Flashbots at unpredictable times so that the curve pool cannot be manipulated beforehand.

Update: The Sturdy team has fixed the issue by adding access-control to the function so that only the yield processor can call it.

QSP-12 Can Receive Less than Expected Amount From Swap

Severity: *Low Risk*

Status: Fixed

File(s) affected: `protocol/vault/ethereum/YearnVault.sol`

Description: The `YearnVault.processYield()` function performs multiple swaps to obtain debt assets from FTM. There is a broken validation check performed after the swap to ensure that the correct amount of debt asset has been received on `L126-L129`:

```
require(
  IERC20(_tokenOut).balanceOf(address(this)) >= receivedAmounts[1],
  Errors.VT_PROCESS_YIELD_INVALID
);
```

The current check validates that the contract's current token balance exceeds the expected amount returned by the swap. This check is insufficient if the contract had an existing balance of the token before the swap.

Recommendation: Validate that the difference between the balance after the swap and the balance before the swap is greater than the expected amount of tokens received.

Update: The Sturdy team has fixed the issue by implementing the recommended check to ensure the expected amount of tokens has been received from the swap.

QSP-13 Ownership Can Be Renounced

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `protocol/configuration/LendingPoolAddressesProvider.sol`, `protocol/configuration/LendingPoolAddressesProviderRegistry.sol`

Description: If the owner renounces their ownership, all ownable contracts will be left without an owner. Consequently, any function guarded by the `onlyOwner` modifier will no longer be able to be executed. Many of the contracts have key administrative functions which are only accessible by the owner.

Recommendation: If ownership should never be renounced, the `renounceOwnership()` function should be overridden to revert.

Update: The Sturdy team indicated they may want to renounce ownership of the contracts in the future.

QSP-14 Reentrancy Risks Can Be Mitigated

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `contracts/protocol/lendingpool/LendingPool.sol`, `contracts/protocol/vault/GeneralVault.sol`

Description: To avoid reentrancy risks and reduce the attack surface, it is highly recommended to inherit from OpenZeppelin's `ReentrancyGuard` contract and add the `nonReentrant` modifier to all functions that:

1. Involve asset manipulation.
2. Could be accessed by any external address.
3. Are not expected to be reentered

The following is a non-exhaustive list of functions we recommend adding a `nonReentrant` modifier to:

- `LendingPool`
  - `deposit()`
  - `withdraw()`
  - `borrow()`
  - `repay()`
  - `liquidationCall()`
  - `setUserUseReserveAsCollateral()`
- `GeneralVault`
  - `depositCollateral()`
  - `depositCollateralFrom()`
  - `withdrawCollateral()`
  - `processYield()`

Recommendation: Add a `nonReentrant` modifier to all suggested functions.

Update: The Sturdy team indicated they could not use the `nonReentrant` modifier because of storage collisions.

QSP-15 Reliance on External Contracts to Secure Funds

Severity: *Informational*

Status: Acknowledged

File(s) affected: `protocol/vault/ethereum/LidoVault.sol`, `protocol/vault/ethereum/YearnRETHWstETHVault.sol`, `protocol/vault/ethereum/ConvexVault/ConvexCurveLPVault.sol`, `protocol/vault/fantom/YearnVault.sol`

Description: The platform generates yields by sending the collateral tokens stored by borrowers to other protocols such as Lido and Convex. If any of these yield-generating protocols got hacked, the protocol users would also lose their funds.

Recommendation: Make this warning explicit to the protocol's users by adding this risk statement to the protocol's public-facing documentation.

Update: The Sturdy team has already added information about these risks to their public documentation.

QSP-16 User Funds Will Be Blocked if Sent Directly to Vaults

Severity: Informational

Status: Acknowledged

File(s) affected: `protocol/vault/ethereum/LidoVault.sol`, `protocol/vault/fantom/YearnVault.sol`

Description: Some vault contracts implement the `receive()` function to receive ETH or FTM. However, it also opens the possibility for a user to send these native tokens directly to the contract by mistake.

Recommendation:

- `LidoVault.sol`: Only allow `provider.getAddress('STETH_ETH_POOL')` (Curve) to call the `receive()` function as LIDO will not send native `ETH` when withdrawing/yield, just `WETH`.
- `LidoVault.sol`: Only allow `provider.getAddress('STETH_ETH_POOL')` (Curve) to call the `receive()` function.
- `YearnVault.sol`: Only allow `provider.getAddress('WFTM')` to call the `receive()` function.

Update: The Sturdy team decided not to restrict the callers of the `receive()` functions because the Curve pool may change.

QSP-17 Unlocked Pragma

Severity: Informational

Status: Acknowledged

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity ^0.8.0`. The `^` before the version number implies an unlocked pragma, meaning that the compiler will use the specified version and above, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, we recommend removing the `^` to lock the file onto a specific Solidity version.

Update: The Sturdy team indicated that they always use the same Solidity version to compile contracts. We still recommend fixing the version as added protection.

QSP-18 Clone-and-Own

Severity: Informational

Status: Acknowledged

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries. Many contracts reference a locally cloned copy of OpenZeppelin contracts stored in the `protocol/dependencies/openzeppelin/*` folder.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as using libraries. If the file is cloned anyway, a comment including the repository, the commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve the traceability of the file.

Update: The Sturdy team prefers to copy only the parts of dependent contracts they need to keep the contract size small. This is why they use the clone-and-own approach.

QSP-19 Token Balances Not Validated

Severity: Informational

Status: Acknowledged

File(s) affected: `protocol/lendingpool/LendingPool.sol`, `protocol/lendingpool/LendingPoolCollateralManager.sol`, `protocol/vault/ethereum/ConvexVault/ConvexCurveLPVault.sol`, `protocol/vault/ethereum/LidoVault.sol`, `protocol/vault/ethereum/YearnRETHWstETHVault.sol`, `protocol/vault/fantom/YearnVault.sol`

Description: It is highly recommended to check the changes in the balance after the token transfer is expected to avoid most of the non-standard behavior of external token contracts (e.g. deflationary tokens) from being used by a hacker as a tool to manipulate the contract. Specifically, we recommend verifying the token balances at the following locations in the code:

1. `LendingPool.sol#L146`
2. `LendingPool.sol#L186`
3. `LendingPool.sol#L480`
4. `LendingPoolCollateralManager.sol#L250`
5. `ConvexCurveLPVault.sol#L192`
6. `LidoVault.sol#L133`
7. `YearnRETHWstETHVault.sol#L167`
8. `YearnVault.sol#L182`

Recommendation: Verify the token balances in the suggested locations.

Update: The Sturdy team believes the external calls would fail if the wrong amount of tokens were sent to the contract. We still recommend adding these checks to guarantee the behavior of the external contracts.

QSP-20 Missing Input Validation

Severity: Informational

Status: Fixed

File(s) affected: `protocol/lendingpool/LendingPool.sol`, `protocol/lendingpool/LendingPoolConfigurator.sol`, `protocol/tokenization/AToken.sol`, `protocol/vault/GeneralVault.sol`, `protocol/libraries/logic/ReserveLogic.sol`

Description:

1. `LendingPool.initialize()`: `provider` should not be zero.
2. `LendingPoolConfigurator.initialize()`: `provider` should not be zero.
3. `AToken.sol.initialize()`: `aTokenDecimals`, `pool`, `treasury`, `underlyingAsset`, `incentivesController` should not be zero.
4. `GeneralVault.initialize()`: `_provider` should not be zero.
5. `ReserveLogic.init()`: `yieldAddress`, `stableDebtTokenAddress`, `variableDebtTokenAddress`, `interestRateStrategyAddress` should not be zero.

Recommendation: Add the missing validation checks.

Update: The Sturdy team has fixed the issue by implementing all suggested validation checks.

QSP-21 `vaultYieldInPrice` Returns Incorrect Value for All Vaults

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `protocol/vault/GeneralVault.sol`

Description: The `GeneralVault.vaultYieldInPrice()` function always returns `0` and is not overridden by any vault contracts. As such, this function always returns the incorrect value. This can be seen in the function definition below:

```
function vaultYieldInPrice() external view virtual returns (uint256) {
  return 0;
}
```

Recommendation: The `GeneralVault.vaultYieldInPrice()` function should either be removed or implemented to always return the correct value.

Update: The Sturdy team indicated that the `vaultYieldInPrice()` function will be implemented in future vaults. We still feel that the current behavior of the function is misleading and could cause confusion.

QSP-22 Stable Debt Functionality Not Removed

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `protocol/lendingpool/DefaultReserveInterestRateStrategy.sol`, `protocol/lendingpool/LendingPool.sol`, `protocol/lendingpool/LendingPoolCollateralManager.sol`, `protocol/lendingpool/LendingPoolConfigurator.sol`, `protocol/libraries/logic/GenericLogic.sol`, `protocol/libraries/logic/ReserveLogic.sol`, `protocol/libraries/logic/ValidationLogic.sol`

Description: Although the stable debt functionality is not currently used by the Sturdy Protocol, the original AAVE code for the stable debt functionality has not been removed. Since the Sturdy Protocol does not currently utilize the stable debt functionality, it is unlikely to have been carefully tested and could thus result in unexpected bugs. The additional code also results in unnecessary additional attack surface for the protocol.
There are many areas within the code where stable debt functionality still exists, including the following:

- `DefaultReserveInterestRateStrategy`
- `LendingPool`
- `LendingPoolCollateralManager`
- `LendingPoolConfigurator`
- `GenericLogic`
- `ReserveLogic`
- `ValidationLogic`

Recommendation: Remove the unused stable debt functionality from the code.

Update: The Sturdy team may decide to activate the stable debt functionality in the future. If the team does so, they will extensively test the functionality. The auditing team believes there is still some risk in letting the dead code for stable debt remain within the contracts.

QSP-23 Misleading Behavior of `withdrawOnLiquidation()` in `LidoVault`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `protocol/vault/ethereum/LidoVault.sol`

Description: The `GeneralVault.withdrawOnLiquidation()` function is not overridden in `LidoVault`. This means that the `LidoVault.withdrawOnLiquidation()` will have the exact same code as `GeneralVault.withdrawOnLiquidation()`. It will thus be callable by anyone and will return the `_amount` passed in as a parameter without doing anything else. This could result in callers of the function being misled into believing that a liquidation has occurred successfully. Furthermore, the `withdrawOnLiquidation()` function is generally only meant to be callable by the `LendingPool` contract. However, `LidoVault.withdrawOnLiquidation()` is callable by anyone.

Recommendation: Override `withdrawOnLiquidation()` within `LidoVault` to either include a correct implementation or to always revert.

Update: The Sturdy team has fixed the issue by overriding the `withdrawOnLiquidation()` function to always revert within the `LidoVault` contract.

QSP-24 `getYieldAmount()` Returns Incorrect Value for `ConvexCurveLPVault`

Severity: *Undetermined*

Status: Fixed

File(s) affected: `protocol/vault/ethereum/ConvexCurveLPVault.sol`

Description: The `ConvexCurveLPVault.getYieldAmount()` function should return the amount of pending yield for the vault. However, it is currently returning the result of the `GeneralVault._getYieldAmount()` function, which returns the difference between the `ATokenForCollateral` supply and the amount of underlying token held within the `ATokenForCollateral` contract. This calculation does not apply to the `ConvexCurveLPVault`, which retrieves yield based on how much the Convex pool awards it.

Recommendation: Reimplement `ConvexCurveLPVault.getYieldAmount()` to return the correct pending yield.

Update: The Sturdy team has fixed the issue by modifying the function's implementation to return the yield earned from the Convex pool.

QSP-25 Reliance on Oracle Contracts that Are Not Within the Scope

Severity: *Undetermined*

Status: Acknowledged

Description: The `SturdyOracle` and lending rate oracle contracts are heavily used among the vault, lending pool, and interest rate contracts. However, they are not within the scope of the current audit. So it is worth noting here that the security of the platform relies on the security of these oracles. If the oracles are breached, the platform could behave unexpectedly, for instance, liquidating assets. And users could suffer from a great loss of their funds.

Recommendation: Make this warning explicit to the protocol's users by adding this risk statement to the protocol's public-facing documentation.

Update: The Sturdy team has indicated they use Chainlink oracles to secure the protocol. We still recommend improving the public documentation by informing users of the oracle risks within the protocol.

QSP-26 Fixed Slippage Value

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `protocol/vault/fantom/YearnVault.sol`

Description: The `LidoVault` contract implements the `setSlippage()` function, allowing an administrator to change the slippage value when swapping stETH for ETH in Curve.

1. `YearnVault` uses a fixed one-percent slippage when swapping tokens in Uniswap.
2. `YearnRETHWstETHVault` uses a fixed two-percent slippage when swapping tokens in Curve.

Recommendation: Confirm this is intended by design. If not, consider implementing a similar approach in `YearnVault` and `YearnRETHWstETHVault`.

Update: The Sturdy team has indicated that this is the intended behavior of the protocol. We still recommend adding the suggested functions to modify slippage.

Automated Analyses

Slither

Slither was run to analyze the smart contracts under audit. Most issues found by Slither were false positives; the rest have been incorporated into the report.

Code Documentation

1. Mappings should be documented with a code comment indicating what the keys and values represent. For example, the mapping in `LendingPoolAddressProvider.sol#21:mapping(bytes32 => address) private _addresses;` should have comment above stating something similar to `(id => proxyAddress)`. (**Update:** Fixed)
2. Return values for functions should be documented using the NatSpec standard. The following are examples of functions which do not have their return values documented:
    1. `LendingPool.getTotalBalanceOfAssetPair()` (**Update:** Fixed)
    2. `LendingPool.getBorrowingAssetAndVolumes()` (**Update:** Fixed)
3. The code comment above `LendingPool.withdrawFrom()` indicates the `Caller is anyone`. However, this is incorrect, as only a vault contract may call the function. (**Update:** Fixed)
4. The code comment above the `IncentiveVault` contract definition has an incorrect title, `@title GeneralVault,` which should be corrected to `@title IncentiveVault`. (**Update:** Fixed)
5. Typographical errors:
    1. `ConvexCurveLPVault.sol#L58`: `_interalToken` (**Update:** Fixed)
    2. `LendingCollateralManager.sol#L69`: "thIS" (**Update:** Fixed)
6. `protocol/vault/ethereum/LidoVault.sol#L136`: the code comment is incorrect. The code just approves the allowance for the lending pool to perform the transfer rather than "Make lendingPool to transfer required amount". (**Update:** Fixed)
7. `protocol/libraries/logic/GenericLogic.sol#L144`: the comment should be for `reservesCount` instead of `reserves`. (**Update:** Fixed)

Adherence to Best Practices

1. The `CollateralAdapter` contract should implement the `ICollateralAdapter` interface. (**Update:** Fixed)
2. `ReserveLogic` is imported twice within `LendingPool.sol` and only needs to be imported once. (**Update:** Fixed)
3. Since the flash loan functionality is not currently in use, the `LendingPool.FLASHLOAN_PREMIUM_TOTAL()` function can be removed. (**Update:** Fixed)

4. The `LendingPoolCollateralManager.liquidationCall()` function currently tries to update the state and interest rates of the collateral reserve. Since the liquidity and debt indexes and rates of collateral tokens should remain constant, the calls to `collateralReserve.updateState()` and `collateralReserve.updateInterestRates()` can be removed.
5. The `AToken.transferOnLiquidation()` is not required as `AToken` represents debt assets that cannot be liquidated as collateral.
6. Consider using a different value than the zero address to represent ETH and FTM within the vault contracts. The zero address can be accidentally used due to human error.
7. The `ConvexCurveLPVault` contract inherits from the `GeneralVault` contract. However, the implementations for `GeneralVault._getYieldAmount()` and `GeneralVault._getYield()` are not applicable to the `ConvexCurveLPVault` contract. The functions should either be overridden with the correct logic or overridden to revert to avoid future errors.
8. Consider creating a constant variable for maximum fee value (`GeneralVault.sol#L214`).
9. Although the main flash loan function was removed from `LendingPool`, there are still some references to flash loans in `LendingPool#L92` and `LendingPool#L720`. There is also another reference on `LendingPoolStorage.sol#L25`.
10. ABI coder v2 is activated by default. The pragma `pragma experimental ABIEncoderV2;` is still valid, but it is deprecated and has no effect. Please use `pragma abicoder v2;` instead. (**Update:** Fixed)
11. Naming of the interface `IYearnVault` can be confusing: one could think that it is the interface for `YearnVault.sol`, but it is the interface of Yearn Finance's external contract.
12. Some function names do not follow the Solidity Style Guide naming convention. Internal variables and functions should start with an underscore:
    1. `ConvexCurveLPVault.sol`: `curveLPToken`, `internalAssetToken` and `convexPoolId`. (**Update:** Fixed)
13. `protocol/vault/fantom/YearnVault.sol#L179`: should revert if `msg.value != 0`. (**Update:** Fixed)
14. `protocol/lendingpool/LendingPool.sol#L92`: an unfinished TODO.
15. `protocol/lendingpool/LendingPool.sol#L764`: consider reverting the transaction directly when `from == to`. (**Update:** Fixed)
16. `protocol/vault/fantom/YearnVault.sol#L279-L285`: consider revert the transaction whenever `volumes[i] == 0` to avoid unexpected behavior of the platform.
17. `protocol/tokenization/ATokenForCollateral.sol`: the `_decimals` currently are the same as the decimals of the internal asset token. However, the `ATokenForCollateral` supply aims to be equal to the external asset token. This would cause confusion when other protocols seek to integrate with the platform. Consider adding a new decimal variable that is dedicated to storing the decimal of the internal asset token, and performing all the decimal adjustments that are currently implemented in the code based on this newly added decimal variable. Then, leave the original `_decimals` variable for the external asset token. This way, as expected, the decimal for the `ATokenForCollateral` is equal to the decimal of the external asset token.
18. `protocol/vault/GeneralVault.sol::withdrawOnLiquidation()`: consider not implementing this function and letting the child contracts override and implement this function to increase maintainability. This would decrease the chances of creating a vault but forgetting to implement some logic to replace the internal collateral asset amount with the external collateral asset amount inside `withdrawOnLiquidation()` when it is necessary for that created new vault. (**Update:** Fixed)
19. Consider renaming the `aTokenBalance` used on `protocol/lendingpool/LendingPool.sol#L225` into something more like an `assetBalanceAcknowledgedByAToken` to prevent ambiguity of the naming. (**Update:** Fixed)
20. `protocol/libraries/logic/ValidationLogic.sol::validateSwapRateMode()` is not used anywhere in the code. (**Update:** Fixed)
21. `protocol/libraries/logic/ValidationLogic.sol::validateRebalanceStableBorrowRate()` is not used anywhere in the code. (**Update:** Fixed)
22. `protocol/libraries/logic/ValidationLogic.sol::validateFlashloan()` is not used anywhere in the code. (**Update:** Fixed)
23. `protocol/lendingpool/LendingPool.sol::_isInterestRateAvailable()`: the function name should be renamed as `_isInterestRateNotAvailable`. (**Update:** Fixed)
24. Since the dev team stated that "stable debt" is not going to be used, should consider substituting a `revert` for `protocol/lendingpool/LendingPoolCollateralManager.sol#L189-L200` since this `else` condition would be unexpected.
25. `protocol/lendingpool/LendingPoolCollateralManager.sol::_processInternalCollateralAsset()`: the variable name `amountCollateral` is ambiguous since before `L311` it is a collateral internal asset and a collateral external asset after `L311`. Consider renaming it to a better name such as `amountCollateralExternal` since, at the end, the variable will be assigned with the value of the amount of external collateral asset. (**Update:** Fixed)

Test Results

Test Suite Results

The tests were run using the command `yarn audit:test`. This command does not run all the tests available in the repository. Furthermore, we note that the tests include integration tests based on a local mainnet fork and do not just rely on unit tests. However, due to the lack of a code coverage script, we cannot fully evaluate the quality of the tests. We cannot guarantee that all the smart contract code has been sufficiently tested. We highly recommend that the Sturdy team implements a code coverage script to determine whether all the appropriate code has been tested.

Update: The tests were successfully rerun during the fix review phase. The Sturdy team has still not provided a code coverage script.

```
yarn run v1.22.19
warning package.json: License should be a valid SPDX license expression
$ rm -f deployed-contracts.json && npm run compile && npm run sturdy:evm:fork:mainnet:migration && FORK=main SKIP_DEPLOY=true TS_NODE_TRANSPILE_ONLY=1 hardhat test ./test-suites/test-sturdy/__setup.deploy.spec.ts ./test-
suites/test-sturdy/audit/*.audit.ts --network localhost

> @sturdy/protocol-v1@1.0.1 compile
> SKIP_LOAD=true hardhat compile

Solidity 0.8.10 is not fully supported yet. You can still use Hardhat, but some features, like stack traces, might not work correctly.

Learn more at https://hardhat.org/reference/solidity-support"

Nothing to compile
Creating Typechain artifacts in directory types for target ethers-v5
Successfully generated Typechain artifacts!

> @sturdy/protocol-v1@1.0.1 sturdy:evm:fork:mainnet:migration
> FORK=main hardhat sturdy:mainnet --network localhost

- Enviroment
  - Fork Mode activated at network:  main
  - Provider URL: eth-mainnet.alchemyapi.io
  - Network : localhost
Migration started

1. Deploy address provider
*** LendingPoolAddressesProvider ***

Network: localhost
tx: 0x0ca79baa4982c18ddb1d0ed30d929512e298db0925a9bfcad83003dc32201a09
contract address: 0x3a5e7db2E0EA9e69fB53Cd8582e64D4001746E8c
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
```

```
gas price: 61946393506
gas used: 1622365

******

- Deploying a new Address Providers Registry:
Signer 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
Balance 999999.89950033929963831
*** LendingPoolAddressesProviderRegistry ***

Network: localhost
tx: 0xe6b2728f64a163c562df014e0ae4a50ba1e7e78d194cc832e09bed84f75aafd9
contract address: 0xda87577f9eb8B15B26C00619FD06d4485880310D
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 524198

******

Deployed Registry Address: 0xda87577f9eb8B15B26C00619FD06d4485880310D
Added LendingPoolAddressesProvider with address "0x3a5e7db2E0EA9e69fB53Cd8582e64D4001746E8c" to registry located at 0xda87577f9eb8B15B26C00619FD06d4485880310D
Pool Admin 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
Emergency Admin 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb
2. Deploy lending pool
        Deploying new lending pool implementation & libraries...
*** ReserveLogic ***

Network: localhost
tx: 0x244ce16d18231bc4fb2be5e86e7ef787f47dbae9bb700f425832ba049e47a350
contract address: 0xA6f48C8190be8F92A4c31aAE4756289Ef3d91477
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 222959

******

*** GenericLogic ***

Network: localhost
tx: 0x0232d6e69fd21cabbd1b26b029182a2aca1e02a6bc67d9fc9a4ceaa3396e9288
contract address: 0xa51d9BD713E0a812e5289D926F8f721eb1d71112
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 777394

******

*** ValidationLogic ***

Network: localhost
tx: 0xa20843af31d0c048f74a06cbd66c0e763fcd4193a71f183943a4a1cf37e46a5c
contract address: 0x5aE47822Da849508020c25E2E92c4a4cC4E03001
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1530613

******

*** LendingPool ***

Network: localhost
tx: 0xc59a96b4c60485463983ee6735d3fd65ac55489b1cc5a6b21d6339a808dbfea3
contract address: 0x677B89Ac909215B7e6B6bA46e229eBCe08d25e79
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 4645452

******

        Setting lending pool implementation with address: 0x677B89Ac909215B7e6B6bA46e229eBCe08d25e79
        Deploying new configurator implementation...
*** LendingPoolConfigurator ***

Network: localhost
tx: 0xe6cf1f0cd133c857141dd418fadc4b5e491465c8b7b5757de69e1e68add72409
contract address: 0xa222efa1133c6702a83eFD04bD5E7Ef941EDb2EC
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 4054008

******

        Setting lending pool configurator implementation with address: 0xa222efa1133c6702a83eFD04bD5E7Ef941EDb2EC
*** StableAndVariableTokensHelper ***

Network: localhost
tx: 0xe134ec25aaaad9f3f33a319655f279e48f1bb31e5e1fe807180f9ef3e9f1bc9b
contract address: 0x58a749D6760e7dd926dF737Ec45b65F687aBbE52
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 466236

******

*** ATokensAndRatesHelper ***

Network: localhost
tx: 0x01beb473a65bb8184976abceebde75147188ed021094649fcc10ae99f0f065e0
contract address: 0x2e9f55f7266d8C7e07d359daBA0e743e331B7A1A
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 513276

******

3. Deploy oracles
*** SturdyOracle ***

Network: localhost
tx: 0xb7417040be27093e64948c33fd2c8542c773366c09054405dfc850abf7f93aad
contract address: 0x609D79AD1935BB9aEce827B8eC111e87122928a7
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 829197

******

*** LendingRateOracle ***

Network: localhost
tx: 0xd43c313cdae8909482dc96761ebc4a78eefa9e064bc96f33f4f32d677602cf6e
contract address: 0x093D3c551bc022bB1A311bF5d9DAB02b4e575472
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 307034

******

- Oracle borrow initalization in 1 txs
  - Setted Oracle Borrow Rates for: DAI, USDC, USDT
Sturdy Oracle: 0x093D3c551bc022bB1A311bF5d9DAB02b4e575472
Lending Rate Oracle: 0x093D3c551bc022bB1A311bF5d9DAB02b4e575472
4. Deploy Data Provider
*** SturdyProtocolDataProvider ***

Network: localhost
tx: 0x460b67c10232a87f92556f34f7a44facfcb27f094b7e305ccf1faa422e177416
contract address: 0x16cC86D9c42208eE9e22b377e3d7927630733a06
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1554305

******

5. Deploy Incentives impl
Duplicate definition of RewardsClaimed (RewardsClaimed(address,address,uint256), RewardsClaimed(address,address,address,uint256))
*** StakedTokenIncentivesControllerImpl ***

Network: localhost
tx: 0x485856cdfa379f1aa5d662780a33756d09df3a248082a215ca12c5692bfe35e0
contract address: 0x72FC13b5C367e20Dce174a8595925E83557E5737
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1993938

******

Duplicate definition of RewardsClaimed (RewardsClaimed(address,address,uint256), RewardsClaimed(address,address,address,uint256))
- Incentives proxy address 0xA897716BA0c7603B10b8b2854c104912a6058542
*** SturdyTokenImpl ***
```

```
Network: localhost
tx: 0x03d0c562c321274c116e1d6c3ac0738a906ab6a6b4b1f17eb5e1c1b1903ddaef
contract address: 0x9310dC480CbF907D6A3c41eF2e33B09E61605531
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 699304

******

- Incentives sturdy token proxy address 0x7Ff897A14796072a1d23e0002f3Dca9Bc80af677
*** StableYieldDistributionImpl ***

Network: localhost
tx: 0x3957eca038141aa5761ba5741a83722e32f566abfa0f117401b1a2ad745b41e6
contract address: 0xBf61048590B6FAd46Fb446aA241fA33f7a22851b
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1660374

******

- Stable Yield Distributor Impl address 0xBf61048590B6FAd46Fb446aA241fA33f7a22851b
*** VariableYieldDistributionImpl ***

Network: localhost
tx: 0x21a864232959d532df7231d5a34a60363f32672eb18188cb038bcb453343443f
contract address: 0x3c3eaa7B2953170BbffF0a477f9Cbcb0280B9c95
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1491984

******

- Variable Yield Distributor Impl address 0x3c3eaa7B2953170BbffF0a477f9Cbcb0280B9c95
5-1. Deploy FXS Stable Yield Distributor
- Incentives proxy address 0x1A4313DdE95487EAAbfF58A181fd34cbe3638041
5-2. Deploy Variable Yield Distributor
- VariableYieldDistributor proxy address 0x8d58c3574A6D32F5F848Abe8E7A03E8B92577c15
6. Deploy Lido vault
*** CurveswapAdapter ***

Network: localhost
tx: 0xc48b2bf3bad9ad8823bb2db4eef6de929a268597b3af93c4ba06d07d4ea7f737
contract address: 0x79CA3AA3287FA27EB96Bc018De53B9f84C6b04ca
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 721300

******

*** LidoVaultImpl ***

Network: localhost
tx: 0x4e65654cdf527bf0165bae177df22669a53ecde0205b7897dfd6bc607fcba479
contract address: 0x6E0745b6B18d0233708554049EEAaB0CB81c4Ab0
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1873391

******

LidoVault.address 0x830647b95f38Af9e5811B4f140a9053b93322f08
       Finished LidoVault deployment
6-3. Deploy Convex FRAX 3CRV vault
*** ConvexFRAX3CRVVaultImpl ***

Network: localhost
tx: 0x1abcc375126b46ad9b04c08501bff65042e1d526001401acc11fcab123563f6b
contract address: 0xD13a7D8A728692eB2c56135B5EB5A1951b3F8395
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3452047

******

internal token: 0xA0ecbe4a0e87b1559C962bb6E1F46286D41394Bf
*** FRAX3CRVOracle ***

Network: localhost
tx: 0x187dd93568c443dac0d023d6ea8d1b8df8ca9f0ec924d03cbd02b51c43c7b9df
contract address: 0xeb05945568527aE1d314870eCeDF78290F44283a
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 311841

******

642803791620348
703518260000000
3413207935471490
ConvexFRAX3CRVVault.address 0xb2E308Ef09697ed2b2d740c769CD3b0246DD1e6c
       Finished ConvexFRAX3CRVVault deployment
6-6. Deploy Convex MIM 3CRV vault
*** ConvexMIM3CRVVaultImpl ***

Network: localhost
tx: 0x5361315484f3f611cf21b1c715c361529ede8358d9f0408fba492997e34e136c
contract address: 0x0a57c238a9C1F158e7a0fF70DA834a3c43c9E246
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3452047

******

internal token: 0xE72aAEEFA3821594c1D7fe2aB81361F0eC9e6e3e
*** MIM3CRVOracle ***

Network: localhost
tx: 0xb6a9e6d9936650958c17f4daa65eec3e25fc2dd47a62886031ff073b26812f90
contract address: 0x0f9F87AcEf7d8216E78A9092b872d35c93210619
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 359068

******

645673418115539
ConvexMIM3CRVVault.address 0xfDEba3d5Fd0819864FD652f5d6b0beC7a3DE5BF8
       Finished ConvexMIM3CRVVault deployment
6-7. Deploy Convex DAI USDC USDT SUSD vault
*** ConvexDAIUSDCUSDTSUSDVaultImpl ***

Network: localhost
tx: 0xf6a8f4b9736ff1f49c6b556f0d17d551f8f6d8a9628a0df2f1d90b3abf3de255
contract address: 0xf116cC24ACDFd53Ba8d5d77B51F46cD369DdFA2a
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3452047

******

internal token: 0x6DFC56b52a2B7F32A3d4348A53cF71a43C0Ed3a7
*** DAIUSDCUSDTSUSDOracle ***

Network: localhost
tx: 0x7f5068787666e471295c80568c1ebc2610f7adac59c068cdc3976e3bbbb2560d
contract address: 0x0676EAfe01B6515530b9153996915DB32D07f154
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 301893

******

678059532352184
ConvexDAIUSDCUSDTSUSDVault.address 0xF44275a19A24D016364ddD9541FC9B17735De2f2
       Finished ConvexDAIUSDCUSDTSUSDVault deployment
6-9. Deploy Convex Iron Bank vault
*** ConvexIronBankVaultImpl ***

Network: localhost
tx: 0x8ff613838f3a567dc46d7a188a685a7fca205c63f720a517a5081318377d71a8
contract address: 0x0103fc84eAd421Ee4d438E280B32B90a22eAfcCB
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3452047

******

internal token: 0x5d6565e260551Dd8872E03A561E43ea4000dea5E
*** IronBankOracle ***
```

```
Network: localhost
tx: 0x8c71bd8c7edeac384432d40c939392ba5369ed8ff192379507a8152773dc69c1
contract address: 0xf121De9FE385D7b62d2C8cE8954788A674cAAA8B
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 272541

******

670728448168149
ConvexIronBankVault.address 0x89D02f895778F6a0eDb620a162B0018847EAe8f6
        Finished ConvexIronBankVault deployment
6-10. Deploy Convex FRAX USDC vault
localhost
*** ConvexFRAXUSDCVaultImpl ***

Network: localhost
tx: 0x2d58d5a151c6287bc82b51621286ce152d37b77a3606da4deb615a98ea1e00df
contract address: 0xecBC2AA619f8fFC19826E90c2b1d6FDEA4AeDD24
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3452047

******

internal token: 0xa84aE48Eb99D069A6205D383cD9d77CA76B9115d
*** FRAXUSDCOracle ***

Network: localhost
tx: 0xdf484e49f53ac8e5566eb727dded8952b065c1e6509e51c27a83ca5406e4e723
contract address: 0x82b828664197F9B42001b6dB4d4487A651f7Bf3e
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 243176

******

637899945371872
ConvexFRAXUSDCVault.address 0x8c30CE2bCE4523681DA9644cA63e1E3f5f41c3d6
        Finished ConvexFRAXUSDCVault deployment
8. Initialize lending pool
*** StableDebtToken ***

Network: localhost
tx: 0x1f9722b4123c65302ca86eb98fa91a5dd14662aec2f333e90363610a926190d7
contract address: 0x2381d96259DC68051f16448C9c0548228B294b61
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1511770

******

*** VariableDebtToken ***

Network: localhost
tx: 0x02ec47dab9b1b5976a19c6746eee3d09fb12a330906d4f75d913c8b7a6d0b227
contract address: 0x76d513064b531fa55BD09A2BF16440e7D139d4F2
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1239540

******

Duplicate definition of RewardsClaimed (RewardsClaimed(address,address,uint256), RewardsClaimed(address,address,address,uint256))
*** AToken ***

Network: localhost
tx: 0x5b8e9888011404dc61e9dfa921b7c4de06e91601201e07faa0d303e6a2216b9b
contract address: 0xf8C7407850bdc63632D32c260D331C50af18782f
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 2130486

******

*** ATokenForCollateral ***

Network: localhost
tx: 0x06321194639c778847e5505bfcbb427a588948719feba3dc6044e32d2071f695
contract address: 0x45652Da975f14612CD42C24CDEC18f7f4886Ab01
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 2151070

******

*** DefaultReserveInterestRateStrategy ***

Network: localhost
tx: 0x15a89e04fe29b0680693001e0cc477928a47b4e400545de590b5456b6e97399f
contract address: 0xB0c3823eC6597673F9952A41dc6d3a0e656d110d
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 669603

******

Strategy address for asset DAI: 0xB0c3823eC6597673F9952A41dc6d3a0e656d110d
Strategy address for asset USDC: 0xB0c3823eC6597673F9952A41dc6d3a0e656d110d
Strategy address for asset USDT: 0xB0c3823eC6597673F9952A41dc6d3a0e656d110d
*** DefaultReserveInterestRateStrategy ***

Network: localhost
tx: 0x93420f1d7114c73e7548e007338d35afa36dbda360cd6a1c6d28cd20598669c6
contract address: 0xe7E1994Ef5A6D8c88e4B8e27E63A1B78db1072A4
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 669495

******

Strategy address for asset stETH: 0xe7E1994Ef5A6D8c88e4B8e27E63A1B78db1072A4
- Skipping init of yvRETH_WSTETH due token address is not set at markets config
- Skipping init of cvxRETH_WSTETH due token address is not set at markets config
*** DefaultReserveInterestRateStrategy ***

Network: localhost
tx: 0x945fa0106b8b45b7a83146f8cc1c470dc17d85ca740a790d0224666a9fcd3752
contract address: 0xd74F685A403607BB6548eb73d161e9A65da04833
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 669735

******

Strategy address for asset cvxFRAX_3CRV: 0xd74F685A403607BB6548eb73d161e9A65da04833
- Skipping init of cvxSTECRV due token address is not set at markets config
- Skipping init of cvxDOLA_3CRV due token address is not set at markets config
*** DefaultReserveInterestRateStrategy ***

Network: localhost
tx: 0x0de7d05a68292458b4cbf435c187ab73f8866f35d88d9ba567e3508362a1bd20
contract address: 0xFBBEaEf1AAE52F1e34A5D4307dd94184ecA84198
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 669735

******

Strategy address for asset cvxMIM_3CRV: 0xFBBEaEf1AAE52F1e34A5D4307dd94184ecA84198
*** DefaultReserveInterestRateStrategy ***

Network: localhost
tx: 0x9a3c66b02a79ddb15d8511149cfa260ec9d24333e745f594f4516208bb1d9fe3
contract address: 0x616a916Af314884c4dAEfcA6AA3c98a561278f01
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 669735

******

Strategy address for asset cvxDAI_USDC_USDT_SUSD: 0x616a916Af314884c4dAEfcA6AA3c98a561278f01
- Skipping init of cvxHBTC_WBTC due token address is not set at markets config
*** DefaultReserveInterestRateStrategy ***

Network: localhost
tx: 0x48b449ba6838801438e429a662d2336c1f10529caf5d9b25dfe0fcd332cce7f5
contract address: 0x591d0398952f25291957d9bA30ac0009870acC33
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
```

```
gas used: 669735

******

Strategy address for asset cvxIRON_BANK: 0x591d0398952f25291957d9bA30ac0009870acC33
*** DefaultReserveInterestRateStrategy ***

Network: localhost
tx: 0xabaa7b32b619ac0c4129692551c7a4c6dae93fa3ef175c3ec7cf6b2537257fac
contract address: 0x1eD95f15B4f8d1653471eA3a2015cF41DaDf4c5c
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 669735

******

Strategy address for asset cvxFRAX_USDC: 0x1eD95f15B4f8d1653471eA3a2015cF41DaDf4c5c
- Reserves initialization in 3 txs
----------chunkedSymbols---------------- [
  [ 'DAI', 'USDC', 'USDT', 'stETH' ],
  [
    'cvxFRAX_3CRV',
    'cvxMIM_3CRV',
    'cvxDAI_USDC_USDT_SUSD',
    'cvxIRON_BANK'
  ],
  [ 'cvxFRAX_USDC' ]
]
  - Reserve ready for: DAI, USDC, USDT, stETH
    * gasUsed 8819599
  - Reserve ready for: cvxFRAX_3CRV, cvxMIM_3CRV, cvxDAI_USDC_USDT_SUSD, cvxIRON_BANK
    * gasUsed 9231463
  - Reserve ready for: cvxFRAX_USDC
    * gasUsed 2346502
- Skipping init of yvRETH_WSTETH due token address is not set at markets config
- Skipping init of cvxRETH_WSTETH due token address is not set at markets config
- Skipping init of cvxSTECRV due token address is not set at markets config
- Skipping init of cvxDOLA_3CRV due token address is not set at markets config
- Skipping init of cvxHBTC_WBTC due token address is not set at markets config
  - Init for: DAI, USDC, USDT, stETH, cvxFRAX_3CRV, cvxMIM_3CRV, cvxDAI_USDC_USDT_SUSD, cvxIRON_BANK, cvxFRAX_USDC
*** LendingPoolCollateralManager ***

Network: localhost
tx: 0x145ebbce6b6967470db71a794509c2a05419f2cd084037f2eaca4d5ed434d3ed
contract address: 0x497d9c622BC27EfD06d2632021fDc3CC5038e420
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 2441508

******

        Setting lending pool collateral manager implementation with address 0x497d9c622BC27EfD06d2632021fDc3CC5038e420
        Setting SturdyProtocolDataProvider at AddressesProvider at id: 0x01 0x497d9c622BC27EfD06d2632021fDc3CC5038e420
*** WalletBalanceProvider ***

Network: localhost
tx: 0xfb278987f114947d905bec097f1030714b57895d3ca38b323770dd5b24af61b2
contract address: 0x71d010EeFb6d629e9E7ad9e7650c17F97078AFa9
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 666616

******

*** UiPoolDataProvider ***

Network: localhost
tx: 0x4026aa03a92c0f787da6d10d5e66d969cd082b35eaf8a645397feaeb2ffb744a
contract address: 0x4D97Bd8eFaCf46b33c4438Ed0B7B6AABfa2359FB
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3278198

******

UiPoolDataProvider deployed at: 0x4D97Bd8eFaCf46b33c4438Ed0B7B6AABfa2359FB
*** UiIncentiveDataProvider ***

Network: localhost
tx: 0xaadd648ff1c08a3b80ad8968ab88a5003b3673109087232af87fffba6e5fb90b
contract address: 0x34f2c3b7FcEA156d981337Efd4729C553E5ec5DA
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 2474425

******

UiIncentiveDataProvider deployed at: 0x34f2c3b7FcEA156d981337Efd4729C553E5ec5DA
8-1. Deploy Collateral Adapter
*** CollateralAdapterImpl ***

Network: localhost
tx: 0x540d1e4d57ed8f079475353e6be8535b7b3b101371c1718e5f137a29920c070f
contract address: 0x58d44cc71C52A9968425a420BE69C31Eb313486F
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 308931

******

{
  DAI: '0x6B175474E89094C44Da98b954EedeAC495271d0F',
  USDC: '0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48',
  USDT: '0xdAC17F958D2ee523a2206206994597C13D831ec7',
  stETH: '0xae7ab96520DE3A18E5e111B5EaAb095312D7fE84',
  cvxFRAX_3CRV: '0xA0ecbe4a0e87b1559C962bb6E1F46286D41394Bf',
  cvxMIM_3CRV: '0xE72aAEEFA3821594c1D7fe2aB81361F0eC9e6e3e',
  cvxDAI_USDC_USDT_SUSD: '0x6DFC56b52a2B7F32A3d4348A53cF71a43C0Ed3a7',
  cvxIRON_BANK: '0x5d6565e260551Dd8872E03A561E43ea4000dea5E',
  cvxFRAX_USDC: '0xa84aE48Eb99D069A6205D383cD9d77CA76B9115d'
}
- Skipping init of yvRETH_WSTETH due token address is not set at markets config
- Skipping init of cvxRETH_WSTETH due token address is not set at markets config
- Skipping init of cvxSTECRV due token address is not set at markets config
- Skipping init of cvxDOLA_3CRV due token address is not set at markets config
- Skipping init of cvxHBTC_WBTC due token address is not set at markets config
CollateralAdapter.address 0xAd7D921B94aDA77A38AbE4c70049574b639BBEBA
        Finished CollateralAdapter deployment
8-3. Deploy Vault Helper
*** DeployVaultHelper ***

Network: localhost
tx: 0x9dc412d4dea84a07422be52e6155a2bb3a042dc1acc2a4e9d4f96f3a66a1b0d1
contract address: 0x9210b37BcF0cBA6C1b6b3DE2512487127A66B108
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1208281

******

DeployVaultHelper.address 0x9210b37BcF0cBA6C1b6b3DE2512487127A66B108
        Finished DeployVaultHelper deployment
8-4. Deploy Yield Manager
*** UniswapAdapter ***

Network: localhost
tx: 0xabc05a323150ba3afb9e3eb61d168f67d3e7703ca72fad1b2e039d3e5586f3bf
contract address: 0x0B3955132AE8b8B7eFE9bfF0207C882920b67475
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 947497

******

*** YieldManagerImpl ***

Network: localhost
tx: 0xd3f1f3705176987cefbda6ad28727f9ae484f9ac10256d4cf14fbd6f92af79c7
contract address: 0x8BE4eDB1930bdedd94142B09503eB6daC3F2b53D
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1761859

******

YieldManager.address 0x06eaCcB0eE9e84c4af04A291A1fA60dD59Af28e6
        Finished YieldManager deployment
8-5. Deploy Leverage Swap Manager
*** LeverageSwapManagerImpl ***
```

```
Network: localhost
tx: 0xff3359ad4da7b604b47eb9ae9b429557aa95c2e72c12325e4068d59ac0f03481
contract address: 0xb16778EDD00233E042998bEd0b8b83699C54bcfa
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 262358

******

*** FRAX3CRVLevSwap ***

Network: localhost
tx: 0x2cbc5d771d40a047da5fdaaa3ea1aad98ee61b104785841a2bcaea484c458e9a
contract address: 0x7bCeDa3f6554c5fC7AD043840a8546DbE8b90402
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3376355

******

FRAX3CRVLevSwap: 0x7bCeDa3f6554c5fC7AD043840a8546DbE8b90402
*** DAIUSDCUSDTSUSDLevSwap ***

Network: localhost
tx: 0xbfd13d835f648a6b8519e1b92ddacfc89403698c52d85447434984c3751bb6dc
contract address: 0xFa73E18353C9a44b677090708E46e7d1bCed2515
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3292237

******

DAIUSDCUSDTSUSDLevSwap: 0xFa73E18353C9a44b677090708E46e7d1bCed2515
*** FRAXUSDCLevSwap ***

Network: localhost
tx: 0xaa1bacc5d3bed649a46dcdfd8e935a318b3e6f1b613667e4b2ed6bd0dc5ca888
contract address: 0x882A796B6f1f1679BBa1dd127EcE41Fd4f7aEacc
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3394177

******

FRAXUSDCLevSwap: 0x882A796B6f1f1679BBa1dd127EcE41Fd4f7aEacc
*** IRONBANKLevSwap ***

Network: localhost
tx: 0x21c2aac38bb8e32e7ed4a443f72d0dc9a77f5ac52a98791e3080f01c99e22b4d
contract address: 0xD436e756Cf41318ADeC62E8dCbEF2608753Ae068
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3254919

******

IRONBANKLevSwap: 0xD436e756Cf41318ADeC62E8dCbEF2608753Ae068
*** MIM3CRVLevSwap ***

Network: localhost
tx: 0x8505f4ffbda2adaa869c195833f058b10482017bef92048290c0e3eb27b0e060
contract address: 0xed60B470AD09D99aeF8363F52d75910Bd0079bBa
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 3376355

******

MIM3CRVLevSwap: 0xed60B470AD09D99aeF8363F52d75910Bd0079bBa
LeverageSwapManager.address 0x1d7c9d1d3D30669cAD26ef3d9E8fb788BD62740F
        Finished LeverageSwapManager deployment
8-6. Deploy APR Data Provider
*** SturdyAPRDataProvider ***

Network: localhost
tx: 0xacd5ce38fde00fabf1c1387bfbf1845f29fcad0e668b8aab1a4f5861e23b29b4
contract address: 0x9E55F61F92f1dA099b913593b75Bafd0c7f9215B
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1600970

******

SturdyAPRDataProvider.address 0x9E55F61F92f1dA099b913593b75Bafd0c7f9215B
        Finished SturdyAPRDataProvider deployment

Finished migrations
Contracts deployed at localhost
---------------------------------
N# Contracts: 73
LendingPoolAddressesProvider: 0x3a5e7db2E0EA9e69fB53Cd8582e64D4001746E8c
LendingPoolAddressesProviderRegistry: 0xda87577f9eb8B15B26C00619FD06d4485880310D
ReserveLogic: 0xA6f48C8190be8F92A4c31aAE4756289Ef3d91477
GenericLogic: 0xa51d9BD713E0a812e5289D926F8f721eb1d71112
ValidationLogic: 0x5aE47822Da849508020c25E2E92c4a4cC4E03001
LendingPoolImpl: 0x677B89Ac909215B7e6B6bA46e229eBCe08d25e79
LendingPool: 0x2A4d822BFB34d377c978F28a6C332Caa2fF87530
LendingPoolConfiguratorImpl: 0xa222efa1133c6702a83eFD04bD5E7Ef941EDb2EC
LendingPoolConfigurator: 0x35D6445E0D43fdDc23ac4cdb6222cDEdF5E715eE
StableAndVariableTokensHelper: 0x58a749D6760e7dd926dF737Ec45b65F687aBbE52
ATokensAndRatesHelper: 0x2e9f55f7266d8C7e07d359daBA0e743e331B7A1A
SturdyOracle: 0x609D79AD1935BB9aEce827B8eC111e87122928a7
LendingRateOracle: 0x093D3c551bc022bB1A311bF5d9DAB02b4e575472
SturdyProtocolDataProvider: 0x16cC86D9c42208eE9e22b377e3d7927630733a06
StakedTokenIncentivesControllerImpl: 0x72FC13b5C367e20Dce174a8595925E83557E5737
StakedTokenIncentivesController: 0xA897716BA0c7603B10b8b2854c104912a6058542
SturdyTokenImpl: 0x9310dC480CbF907D6A3c41eF2e33B09E61605531
SturdyToken: 0x7Ff897A14796072a1d23e0002f3Dca9Bc80af677
StableYieldDistributionImpl: 0xBf61048590B6FAd46Fb446aA241fA33f7a22851b
VariableYieldDistributionImpl: 0x3c3eaa7B2953170BbffF0a477f9Cbcb0280B9c95
FXSStableYieldDistribution: 0x1A4313DdE95487EAAbfF58A181fd34cbe3638041
VariableYieldDistribution: 0x8d58c3574A6D32F5F848Abe8E7A03E8B92577c15
CurveswapAdapter: 0x79CA3AA3287FA27EB96Bc018De53B9f84C6b04ca
LidoVaultImpl: 0x6E0745b6B18d0233708554049EEAaB0CB81c4Ab0
LidoVault: 0x830647b95f38Af9e5811B4f140a9053b93322f08
ConvexFRAX3CRVVaultImpl: 0xD13a7D8A728692eB2c56135B5EB5A1951b3F8395
ConvexFRAX3CRVVault: 0xb2E308Ef09697ed2b2d740c769CD3b0246DD1e6c
FRAX3CRVOracle: 0xeb05945568527aE1d314870eCeDF78290F44283a
ConvexMIM3CRVVaultImpl: 0x0a57c238a9C1F158e7a0fF70DA834a3c43c9E246
ConvexMIM3CRVVault: 0xfDEba3d5Fd0819864FD652f5d6b0beC7a3DE5BF8
MIM3CRVOracle: 0x0f9F87AcEf7d8216E78A9092b872d35c93210619
ConvexDAIUSDCUSDTSUSDVaultImpl: 0xf116cC24ACDFd53Ba8d5d77B51F46cD369DdFA2a
ConvexDAIUSDCUSDTSUSDVault: 0xF44275a19A24D016364ddD9541FC9B17735De2f2
DAIUSDCUSDTSUSDOracle: 0x0676EAfe01B6515530b9153996915DB32D07f154
ConvexIronBankVaultImpl: 0x0103fc84eAd421Ee4d438E280B32B90a22eAfcCB
ConvexIronBankVault: 0x89D02f895778F6a0eDb620a162B0018847EAe8f6
IronBankOracle: 0xf121De9FE385D7b62d2C8cE8954788A674cAAA8B
ConvexFRAXUSDCVaultImpl: 0xecBC2AA619f8fFC19826E90c2b1d6FDEA4AeDD24
ConvexFRAXUSDCVault: 0x8c30CE2bCE4523681DA9644cA63e1E3f5f41c3d6
FRAXUSDCOracle: 0x82b828664197F9B42001b6dB4d4487A651f7Bf3e
StableDebtToken: 0x2381d96259DC68051f16448C9c0548228B294b61
VariableDebtToken: 0x76d513064b531fa55BD09A2BF16440e7D139d4F2
AToken: 0xf8C7407850bdc63632D32c260D331C50af18782f
aTokenImpl: 0xf8C7407850bdc63632D32c260D331C50af18782f
ATokenForCollateral: 0x45652Da975f14612CD42C24CDEC18f7f4886Ab01
aTokenForCollateralImpl: 0x45652Da975f14612CD42C24CDEC18f7f4886Ab01
DefaultReserveInterestRateStrategy: 0x1eD95f15B4f8d1653471eA3a2015cF41DaDf4c5c
rateStrategyStableTwo: 0xB0c3823eC6597673F9952A41dc6d3a0e656d110d
rateStrategySTETH: 0xe7E1994Ef5A6D8c88e4B8e27E63A1B78db1072A4
rateStrategyCVXFRAX_3CRV: 0xd74F685A403607BB6548eb73d161e9A65da04833
rateStrategyCVXMIM_3CRV: 0xFBBEaEf1AAE52F1e34A5D4307dd94184ecA84198
rateStrategyCVXDAI_USDC_USDT_SUSD: 0x616a916Af314884c4dAEfcA6AA3c98a561278f01
rateStrategyCVXIRON_BANK: 0x591d0398952f25291957d9bA30ac0009870acC33
rateStrategyCVXFRAX_USDC: 0x1eD95f15B4f8d1653471eA3a2015cF41DaDf4c5c
LendingPoolCollateralManagerImpl: 0x497d9c622BC27EfD06d2632021fDc3CC5038e420
LendingPoolCollateralManager: 0x497d9c622BC27EfD06d2632021fDc3CC5038e420
WalletBalanceProvider: 0x71d010EeFb6d629e9E7ad9e7650c17F97078AFa9
UiPoolDataProvider: 0x4D97Bd8eFaCf46b33c4438Ed0B7B6AABfa2359FB
UiIncentiveDataProvider: 0x34f2c3b7FcEA156d981337Efd4729C553E5ec5DA
CollateralAdapterImpl: 0x58d44cc71C52A9968425a420BE69C31Eb313486F
CollateralAdapter: 0xAd7D921B94aDA77A38AbE4c70049574b639BBEBA
DeployVaultHelper: 0x9210b37BcF0cBA6C1b6b3DE2512487127A66B108
UniswapAdapter: 0x0B3955132AE8b8B7eFE9bfF0207C882920b67475
YieldManagerImpl: 0x8BE4eDB1930bdedd94142B09503eB6daC3F2b53D
YieldManager: 0x06eaCcB0eE9e84c4af04A291A1fA60dD59Af28e6
LeverageSwapManagerImpl: 0xb16778EDD00233E042998bEd0b8b83699C54bcfa
LeverageSwapManager: 0x1d7c9d1d3D30669cAD26ef3d9E8fb788BD62740F
FRAX3CRVLevSwap: 0x7bCeDa3f6554c5fC7AD043840a8546DbE8b90402
DAIUSDCUSDTSUSDLevSwap: 0xFa73E18353C9a44b677090708E46e7d1bCed2515
FRAXUSDCLevSwap: 0x882A796B6f1f1679BBa1dd127EcE41Fd4f7aEacc
```

```
IRONBANKLevSwap: 0xD436e756Cf41318ADeC62E8dCbEF2608753Ae068
MIM3CRVLevSwap: 0xed60B470AD09D99aeF8363F52d75910Bd0079bBa
SturdyAPRDataProvider: 0x9E55F61F92f1dA099b913593b75Bafd0c7f9215B

Creating Typechain artifacts in directory types for target ethers-v5
Successfully generated Typechain artifacts!

- Enviroment
  - Fork Mode activated at network:  main
  - Provider URL: eth-mainnet.alchemyapi.io
  - Network : localhost
Duplicate definition of RewardsClaimed (RewardsClaimed(address,address,uint256), RewardsClaimed(address,address,address,uint256))

***************
Setup and snapshot finished
***************

  AddressesProviderRegistry
     ✓ Checks the addresses provider is added to the registry (68951 gas)
     ✓ tries to register an addresses provider with id 0 (68951 gas)
     ✓ Registers a new mock addresses provider (146287 gas)
     ✓ Removes the mock addresses provider (102815 gas)
     ✓ Tries to remove a unregistered addressesProvider (25479 gas)
     ✓ Tries to remove a unregistered addressesProvider (25479 gas)
     ✓ Tries to add an already added addressesProvider with a different id. Should overwrite the previous id (60398 gas)

  AToken: Modifiers
     ✓ Tries to invoke mint not being the LendingPool (68951 gas)
     ✓ Tries to invoke burn not being the LendingPool (68951 gas)
     ✓ Tries to invoke transferOnLiquidation not being the LendingPool (68951 gas)
     ✓ Tries to invoke transferUnderlyingTo not being the LendingPool (68951 gas)

  AToken: Permit
     ✓ Checks the domain separator (68951 gas)
     ✓ Get aDAI for tests (418409 gas)
     ✓ Reverts submitting a permit with 0 expiration (251458 gas)
     ✓ Submits a permit with maximum expiration length (334548 gas)
     ✓ Cancels the previous permit (127096 gas)
     ✓ Tries to submit a permit with invalid nonce (44006 gas)
     ✓ Tries to submit a permit with invalid expiration (previous to the current block) (44006 gas)
     ✓ Tries to submit a permit with invalid signature (44006 gas)
     ✓ Tries to submit a permit with invalid owner (44006 gas)

  AToken: Transfer
     ✓ User 0 deposits 1000 DAI, transfers to user 1 (569886 gas)
     ✓ User 1 tries to transfer a small amount of DAI back to user 0 (297716 gas)

  Deposit ETH as collateral and other as for pool liquidity supplier
==================================================
Supplier 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb deposited: 7000 USDC
totalDebtETH:  0
availableBorrowsETH:  0
currentLiquidationThreshold:  0



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 deposited: 1 stETH
totalDebtETH:  0
availableBorrowsETH:  0.701872424405069194
currentLiquidationThreshold:  0.0000000000000075



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 borrowed: 1039000000 USDC
totalDebtETH:  0.66651698367386198
availableBorrowsETH:  0.035355440731207214
currentLiquidationThreshold:  0.0000000000000075



     ✓ User1 deposits USDC, User deposits ETH as collateral and borrows USDC (1205902 gas)

  Deposit stETH as collateral and other as for pool liquidity supplier
==================================================
Supplier 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb deposited: 7000 USDC
totalDebtETH:  0
availableBorrowsETH:  0
currentLiquidationThreshold:  0



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 deposited: 1 stETH
totalDebtETH:  0
availableBorrowsETH:  0.68148488038106462
currentLiquidationThreshold:  0.0000000000000075



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 borrowed: 1009000000 USDC
totalDebtETH:  0.64727202745613738
availableBorrowsETH:  0.03421285292492724
currentLiquidationThreshold:  0.0000000000000075



     ✓ User1 deposits USDC, User deposits stETH as collateral and borrows USDC (1284874 gas)

  Deposit stETH as collateral and other as for pool liquidity supplier
==================================================
Supplier 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb deposited: 7000 USDT
totalDebtETH:  0
availableBorrowsETH:  0
currentLiquidationThreshold:  0



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 deposited: 1 stETH
totalDebtETH:  0
availableBorrowsETH:  0.68148488038106462
currentLiquidationThreshold:  0.0000000000000075



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 borrowed: 1005000000 USDT
totalDebtETH:  0.64726738575
availableBorrowsETH:  0.03421749463106462
currentLiquidationThreshold:  0.0000000000000075



     ✓ User1 deposits USDT, User deposits stETH as collateral and borrows USDT (1262413 gas)

  borrow stETH
==================================================
Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 deposits: 5 stETH
totalDebtETH:  0
availableBorrowsETH:  3.509359788100155587
currentLiquidationThreshold:  0.0000000000000075



     ✓ Should revert if borrow stETH. User1 deposits stETH, User2 deposits ETH as collateral and borrows stETH (620257 gas)

  LendingPoolConfigurator
     ✓ Reverts trying to set an invalid reserve factor (68951 gas)
     ✓ Deactivates the ETH reserve (187046 gas)
     ✓ Rectivates the ETH reserve (176614 gas)
     ✓ Check the onlySturdyAdmin on deactivateReserve  (58519 gas)
     ✓ Check the onlySturdyAdmin on activateReserve  (58519 gas)
     ✓ Freezes the ETH reserve (117038 gas)
     ✓ Unfreezes the ETH reserve (117047 gas)
     ✓ Check the onlySturdyAdmin on freezeReserve  (58528 gas)
     ✓ Check the onlySturdyAdmin on unfreezeReserve  (58528 gas)
     ✓ Deactivates the ETH reserve for borrowing (114213 gas)
     ✓ Activates the USDC reserve for borrowing (115046 gas)
     ✓ Check the onlySturdyAdmin on disableBorrowingOnReserve  (59361 gas)
     ✓ Check the onlySturdyAdmin on enableBorrowingOnReserve  (59361 gas)
     ✓ Deactivates the ETH reserve as collateral (179539 gas)
     ✓ Activates the ETH reserve as collateral (181345 gas)
     ✓ Check the onlySturdyAdmin on configureReserveAsCollateral  (61167 gas)
     ✓ Disable stable borrow rate on the ETH reserve (116917 gas)
     ✓ Enables stable borrow rate on the USDC reserve (111402 gas)
     ✓ Check the onlySturdyAdmin on disableReserveStableRate (55652 gas)
     ✓ Check the onlySturdyAdmin on enableReserveStableRate (55652 gas)
     ✓ Changes the reserve factor of stETH (114720 gas)
     ✓ Check the onlyLendingPoolManager on setReserveFactor (59068 gas)
     ✓ Reverts when trying to disable the DAI reserve with liquidity on it (413314 gas)

  ConvexDAIUSDCUSDTSUSDVault - Deposit & Withdraw
     ✓ should be reverted if try to use an invalid token as collateral (68951 gas)
```

```
    ✓ should be reverted if try to use any of coin other than DAIUSDCUSDTSUSD-f as collateral (68951 gas)
    ✓ deposit DAI-USDC-USDT-SUSD for collateral (1233807 gas)
    ✓ transferring aCVXDAI_USDC_USDT_SUSD should be success after deposit DAI_USDC_USDT_SUSD_LP (1594478 gas)
    ✓ withdraw from collateral should be failed if user has not enough balance (526889 gas)
    ✓ withdraw from collateral (1479950 gas)

convexDAIUSDCUSDTSUSDVault - Process Yield
    ✓ send yield to YieldManager (1746705 gas)

ConvexFRAX3CRVVault - Deposit & Withdraw
    ✓ should be reverted if try to use an invalid token as collateral (68951 gas)
    ✓ should be reverted if try to use any of coin other than FRAX3CRV-f as collateral (68951 gas)
    ✓ deposit FRAX-3CRV for collateral (1168100 gas)
    ✓ transferring aCVXFRAX_3CRV should be success after deposit FRAX_3CRV_LP (1538659 gas)
    ✓ withdraw from collateral should be failed if user has not enough balance (542777 gas)
    ✓ withdraw from collateral (1400006 gas)

ConvexFRAX3CRVVault - Process Yield
    ✓ send yield to YieldManager (1581999 gas)

ConvexFRAXUSDCVault - Deposit & Withdraw
    ✓ should be reverted if try to use an invalid token as collateral (68951 gas)
    ✓ should be reverted if try to use any of coin other than FRAX-USDC as collateral (68951 gas)
    ✓ deposit FRAX-USDC for collateral (1059780 gas)
    ✓ transferring aCVXFRAX_USDC should be success after deposit FRAX_USDC_LP (1358418 gas)
    ✓ withdraw from collateral should be failed if user has not enough balance (464502 gas)
    ✓ withdraw from collateral (1253435 gas)

ConvexFRAXUSDCVault - Process Yield
    ✓ send yield to YieldManager (1440857 gas)

ConvexIronBankVault - Deposit & Withdraw
    ✓ should be reverted if try to use an invalid token as collateral (68951 gas)
    ✓ should be reverted if try to use any of coin other than hCRV as collateral (68951 gas)
    ✓ deposit IRON_BANK_LP for collateral (1013326 gas)
    ✓ transferring aCVXIRON_BANK should be success after deposit IRON_BANK_LP (1470017 gas)
    ✓ withdraw from collateral should be failed if user has not enough balance (622485 gas)
    ✓ withdraw from collateral (1364894 gas)

  ConvexMIM3CRVVault - Deposit & Withdraw
    ✓ should be reverted if try to use an invalid token as collateral (68951 gas)
    ✓ should be reverted if try to use any of coin other than MIM3CRV-f as collateral (68951 gas)
    ✓ deposit MIM-3CRV for collateral (1093615 gas)
    ✓ transferring aCVXMIM_3CRV should be success after deposit MIM_3CRV_LP (1489643 gas)
    ✓ withdraw from collateral should be failed if user has not enough balance (568246 gas)
    ✓ withdraw from collateral (1355625 gas)

  convexMIM3CRVVault - Process Yield
    ✓ send yield to YieldManager (1507514 gas)

  AddressesProviderRegistry
    ✓ Checks the addresses provider is added to the registry (68951 gas)

  LendingPoolAddressesProvider
    ✓ Test the accessibility of the LendingPoolAddressesProvider (97625 gas)
*** ReserveLogic ***

Network: localhost
tx: 0xf12af9f62b10183a90384d0d5d9143a2987ec9ff82f4e6ecf7533c1b6163f38e
contract address: 0xB95Ad562EDE8DD78BBFC287fA18150e802b09D9F
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 222959

******

*** GenericLogic ***

Network: localhost
tx: 0xd9f73272afa54f560a4d700038683d9e5a26de57695f76de2d5ca96e66fa91ff
contract address: 0xE1a595Cfb10D588F2FC8F335650fd2B0E06D9d6C
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 777394

******

*** ValidationLogic ***

Network: localhost
tx: 0x6e155a077d195ba91bb5bb9fc301117a95dead6c1a84bed2ff3e7f43f0aabadd
contract address: 0x2f96e37D417Db81feDEDB880Dc00f52d6326d909
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1530613

******

*** LendingPool ***

Network: localhost
tx: 0x373b89fa9a178f2ce75d422ede6139c5be74584055c310e3cb5078333f7d53cd
contract address: 0x0D9039Cf7F38ca6C572F4df7C791B83bA4530cd7
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 4645392

******

     ✓ Tests adding  a proxied address with `setAddressAsProxy()` (7788498 gas)
     ✓ Tests adding a non proxied address with `setAddress()` (631894 gas)

  LidoVault
    ✓ failed deposit for collateral without ether (68951 gas)
    ✓ deposit ETH for collateral (463472 gas)
    ✓ stETH & aStETH balance check after deposit for collateral (394521 gas)
    ✓ transferring aStETH should be success after deposit ETH (602422 gas)
    ✓ withdraw from collateral should be failed if user has not enough balance (207901 gas)
    ✓ withdraw from collateral (637683 gas)

  LidoVault - use other coin as collateral
    ✓ Should revert to use any of coin other than ETH, stETH as collateral.  (68951 gas)

  LendingPool liquidation - liquidator receiving aToken
    ✓ Deposits stETH, borrows DAI/Check liquidation fails because health factor is above 1 (1226300 gas)
    ✓ Drop the health factor below 1 (390777 gas)
    ✓ Tries to liquidate a different currency than the loan principal (61167 gas)
    ✓ Tries to liquidate a different collateral than the borrower collateral (61167 gas)
    ✓ Liquidates the borrow (593493 gas)
    ✓ User 3 deposits 7000 USDC, user 4 1 stETH, user 4 borrows - drops HF, liquidates the borrow (2213471 gas)

  LendingPool liquidation - liquidator receiving the underlying asset
    ✓ Deposits stETH, borrows DAI (1226300 gas)
    ✓ Drop the health factor below 1 (390777 gas)
    ✓ Liquidates the borrow (676497 gas)
    ✓ User 3 deposits 7000 USDC, user 4 1 stETH, user 4 borrows - drops HF, liquidates the borrow (2403823 gas)

  Deposit ETH as collateral and other as for pool liquidity supplier
==================================================
Supplier 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb deposited: 7000 USDC
totalDebtETH:  0
availableBorrowsETH:  0
currentLiquidationThreshold:  0



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 deposited: 1 stETH
totalDebtETH:  0
availableBorrowsETH:  0.701872424405069194
currentLiquidationThreshold:  0.0000000000000075



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 borrowed: 1039000000 USDC
totalDebtETH:  0.66651698367386198
availableBorrowsETH:  0.035355440731207214
currentLiquidationThreshold:  0.0000000000000075



     ✓ User1 deposits USDC, User deposits ETH as collateral and borrows USDC (1479724 gas)

  Deposit stETH as collateral and other as for pool liquidity supplier
==================================================
Supplier 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb deposited: 7000 USDC
totalDebtETH:  0
availableBorrowsETH:  0
currentLiquidationThreshold:  0
```

```
Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 deposited: 1 stETH
totalDebtETH:  0
availableBorrowsETH:  0.68148488038106462
currentLiquidationThreshold:  0.0000000000000075



Borrower 0x306469457266CBBe7c0505e8Aad358622235e768 borrowed: 1009000000 USDC
totalDebtETH:  0.64727202745613738
availableBorrowsETH:  0.03421285292492724
currentLiquidationThreshold:  0.0000000000000075



     ✓ User1 deposits USDC, User deposits stETH as collateral and borrows USDC (1558696 gas)

  Upgradeability
*** MockAToken ***

Network: localhost
tx: 0x704b6566ddcf56fa26c5b5d06fd50c2595716d19cc4e13816a312e2a47085cfc
contract address: 0x23f5F6eb3A078383a5De5B0dD2A1dB90d9694b38
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 2130925

******

*** MockStableDebtToken ***

Network: localhost
tx: 0x6516096a0f6093351f9976807621f05564f03a94aa7503392e7defcc312876aa
contract address: 0xE1a595Cfb10D588F2FC8F335650fd2B0E06D9d6C
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1511770

******

*** MockVariableDebtToken ***

Network: localhost
tx: 0x1d23d6cc5a3a2af3f69399f3530b416ce016707b05ee8df34373ad3c01323700
contract address: 0x0D9039Cf7F38ca6C572F4df7C791B83bA4530cd7
deployer address: 0xb4124cEB3451635DAcedd11767f004d8a28c6eE7
gas price: 61946393506
gas used: 1239540

******

     ✓ Tries to update the DAI Atoken implementation with a different address than the lendingPoolManager (182525 gas)
     ✓ Upgrades the DAI Atoken implementation  (369652 gas)
     ✓ Tries to update the DAI Stable debt token implementation with a different address than the lendingPoolManager (187127 gas)
     ✓ Upgrades the DAI stable debt token implementation  (354801 gas)
     ✓ Tries to update the DAI variable debt token implementation with a different address than the lendingPoolManager (167674 gas)
     ✓ Upgrades the DAI variable debt token implementation  (317740 gas)

  Variable debt token tests
     ✓ Tries to invoke mint not being the LendingPool (68951 gas)
     ✓ Tries to invoke burn not being the LendingPool (68951 gas)

  Withdraw USDC
=================================================
Supplier 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb deposited: 7000 USDC
totalDebtETH:  0
availableBorrowsETH:  0
currentLiquidationThreshold:  0



Supplier 0x8401Eb5ff34cc943f096A32EF3d5113FEbE8D4Eb withdraw: 7000000000 USDC
totalDebtETH:  0
availableBorrowsETH:  0
currentLiquidationThreshold:  0



     ✓ User1 deposits USDC and then withdraw USDC (688463 gas)

  Yield Manger: configuration
     ✓ Registered reward asset count should be 2 (68951 gas)
     ✓ CRV should be a reward asset. (68951 gas)
     ✓ CVX should be a reward asset. (68951 gas)
     ✓ WETH should be a reward asset. (68951 gas)
     ✓ Should be USDC as an exchange token (68951 gas)
     ✓ Should be failed when set invalid address as an exchange token (68951 gas)
     ✓ Should be failed when use invalid address as a curve pool (68951 gas)
     ✓ All curve pool for USDC -> stable coin should be configured (68951 gas)

  Yield Manager: simulate yield in vaults
     ✓ Lido vault (609325 gas)
     ✓ Convex FRAX vault (1964235 gas)

  Yield Manger: distribute yield
     ✓ Should be failed when use invalid asset index (68951 gas)
     ✓ Should be failed when use invalid swap path (68951 gas)
     ✓ Should be failed when use swap path including invalid tokens (68951 gas)
     ✓ Distribute yield (4639191 gas)
```

Solc version: 0.8.10		Optimizer enabled: true		Runs: 200	Block limit: 6718946 gas	
Methods						
Contract	Method	Min	Max	Avg	# calls	eur (avg)
ConvexCurveLPVault	depositCollateral	282508	1072389	685093	29	-
ConvexCurveLPVault	processYield	376253	508074	430457	9	-
ConvexCurveLPVault	withdrawCollateral	429782	953061	759799	6	-
ERC20	approve	45890	67574	54931	41	-
ERC20	transfer	50953	622485	192140	62	-
LendingPool	borrow	329610	344130	338583	11	-
LendingPool	deposit	251458	272640	263921	18	-
LendingPool	liquidationCall	410595	517955	471977	6	-
LendingPool	repay	-	-	213787	2	-
LendingPool	withdraw	-	-	221212	1	-
LendingPoolAddressesProvider	setAddress	-	-	48428	1	-
LendingPoolAddressesProvider	setAddressAsProxy	-	-	583466	2	-
LendingPoolAddressesProvider	transferOwnership	-	-	28674	2	-
LendingPoolAddressesProviderRegistry	registerAddressesProvider	34919	77336	63197	3	-
LendingPoolAddressesProviderRegistry	unregisterAddressesProvider	-	-	25479	4	-
LendingPoolConfigurator	activateReserve	-	-	58519	4	-
LendingPoolConfigurator	configureReserveAsCollateral	61167	120178	70246	13	-
LendingPoolConfigurator	deactivateReserve	-	-	118095	2	-
LendingPoolConfigurator	disableBorrowingOnReserve	-	-	55685	2	-
LendingPoolConfigurator	disableReserveStableRate	-	-	55750	2	-
LendingPoolConfigurator	enableBorrowingOnReserve	-	-	59361	4	-
LendingPoolConfigurator	enableReserveStableRate	-	-	55652	4	-
LendingPoolConfigurator	freezeReserve	-	-	58519	2	-
LendingPoolConfigurator	setReserveFactor	-	-	59068	3	-
LendingPoolConfigurator	unfreezeReserve	-	-	58528	4	-
LendingPoolConfigurator	updateAToken	-	-	187127	3	-
LendingPoolConfigurator	updateStableDebtToken	-	-	167674	3	-

```
|  LendingPoolConfigurator              ·  updateVariableDebtToken       ·          -  ·          -  ·     150066  ·           1  ·          -  |
·····································|·····························|·····|·····|·····|·····|·····
|  MockBOOForFTM                        ·  permit                        ·      44006  ·      83090  ·      55173  ·           7  ·          -  |
·····································|·····························|·····|·····|·····|·····|·····
|  MockVariableDebtToken                ·  initialize                    ·          -  ·          -  ·     182525  ·           2  ·          -  |
·····································|·····························|·····|·····|·····|·····|·····
|  SturdyAPRDataProvider                ·  registerConvexReserve         ·          -  ·          -  ·      68951  ·          60  ·          -  |
·····································|·····························|·····|·····|·····|·····|·····
|  YieldManager                         ·  distributeYield               ·          -  ·          -  ·    1385487  ·           1  ·          -  |
·····································|·····························|·····|·····|·····|·····|·····
|  Deployments                                                           ·                                ·  % of limit  ·             |
·····································|·····|·····|·····|·····|·····
|  GenericLogic                         ·          -  ·          -  ·     777394  ·      11.6 %  ·          -  |
·····································|·····|·····|·····|·····|·····
|  LendingPool                          ·          -  ·          -  ·    4645392  ·      69.1 %  ·          -  |
·····································|·····|·····|·····|·····|·····
|  ReserveLogic                         ·          -  ·          -  ·     222959  ·       3.3 %  ·          -  |
·····································|·····|·····|·····|·····|·····
|  ValidationLogic                      ·          -  ·          -  ·    1530613  ·      22.8 %  ·          -  |
·--------------------------------------|-------------|-------------|-------------|--------------|-------------·

  129 passing (5m)

 ✓  Done in 441.79s.
```

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.

Contracts

```
c82ce5abc34a8481899a109e605479e4e89012c24be2573f7bec77749fa9667f ./DebtTokenBase.sol
61ff33b3a92b96f2b53f2bcf496ea9c17fe6d1ed059ea40466a2a752cf32eaa9 ./AToken.sol
e280637d4dece26eb868079aeaca963d0587f79ea1a0a318ee1d92ca385e3c01 ./ATokenForCollateral.sol
a60b3678bc7c06afa93859ac47f8a1c73a984bf2120cbf4f0dad990ffd85757d ./SturdyInternalAsset.sol
9d78dc8b132199fb53eab8efa77b631c290b9a40f91e9d3024ca6cf0d50ab8d9 ./VariableDebtToken.sol
bfbd5080a974ccfc9b79e08ddf3769a50de4f251b20d9eb13873ad5a901a23db ./BaseImmutableAdminUpgradeabilityProxy.sol
21f743060068a37d4a07421d588166891cc9a1f78ed584f4cf87fcbf7677ab5d ./InitializableImmutableAdminUpgradeabilityProxy.sol
970fdcc45d4cb8c8d1651d802a9a97753b2f71480ad77f321e9b88674cc43892 ./VersionedInitializable.sol
ab22bd48627ee1f2ed0db10d965835eafdc30cf12abc1107b9ef033104a9e1e3 ./GenericLogic.sol
20a18f925419a1980a8adbbad578a908752d529e984a974513b1fbdd266f8d22 ./ReserveLogic.sol
8a41669cb0e24c44b30034af6b79469f8996ce03acf15bf7451d61e509885714 ./ValidationLogic.sol
8da368d556e3e1c6b8046c8cabe5d670b84a426d4843f83a1af0e7b2443a0a19 ./CollateralAdapter.sol
deb2f02c3dc9195482e52a13d50a4d820836d1de13544f865ef27bcde3ffed98 ./DefaultReserveInterestRateStrategy.sol
1f935cf6241bf9622c8e996c63c2664920ff3240d023ef11d5442c4853e732b7 ./LendingPool.sol
d54bb038ca0b1660707dc51fb0070875b50f51c161d9418597382f362113d962 ./LendingPoolCollateralManager.sol
6b4f5e18dd1d654029631296780796226 5a8df25214bb45949762ae491e26106 ./LendingPoolConfigurator.sol
f3d0c830958a387c48dd8ffdc9e2d3aeaf0d9dc634690256ea498b1983e45923 ./YearnVault.sol
7f4d72b48ef0388007f40ca28ec601779f6213e7225109f9f8aefbd4314dcab4 ./LidoVault.sol
127116ca8af2e9384a176bb189211e39ecdd52ed5ad6a3ba58a1cbed66a6a19c ./YearnRETHWstETHVault.sol
f88710c707bd5cbef7181983fd987c11cc5c91001dc234db940100559388e417 ./ConvexCurveLPVault.sol
ba9c6dac0d9bce035362c156fa1cb22096b94cb999b1f22f27a8eadeb43a9f7e ./GeneralVault.sol
925e6e3dce509d7aa6dafcca1e10c51bf96c9d8cfde3a56342b2f3eeae3f8c03 ./IncentiveVault.sol
b98b40d876e07391179caa8df925832fa3481a43a108b6760716aa3b24f6f88d ./LendingPoolAddressesProvider.sol
0e0741c26a438e8c501e0c5dfab0d894477111a88648b71162226d803355edc0 ./LendingPoolAddressesProviderRegistry.sol
```

Tests

```
c1367ff81ddba62ad37a1ebbd70ae5701b1e29ff33c0ddaf3b369c065c45c31a ./audit/atoken-modifiers.spec.audit.ts
390511529a4d24ee5d13652b610de0d1473e90d847e4140cf52fb9b828588381 ./audit/convex-dai-usdc-usdt-susd-vault.spec.audit.ts
481731938f08a6253c8f496e6d027904b0ad1bd011a386966ea3d178d28e9035 ./audit/configurator.spec.audit.ts
bac5f2af721e058924d3759ef93bac3862d90e06674c453d64200a552f297cdb ./audit/repay.spec.audit.ts
740a3b8097dc2225a18738488eb2e451630fb4301d8a3985bbf957c59ef3448e ./audit/convex-frax-3crv-vault.spec.audit.ts
c52903bd70f366dbd2150f8b8e8229ec77616acb39b224c988e9685eeb5677a2 ./audit/upgradeability.spec.audit.ts
1df7af0d47b4107cc12f80c7c10c7d81fe87932148edb280716be5dfe4475832 ./audit/atoken-permit.spec.audit.ts
92782b628d6536eb12b0487374dcc8f1856c3a247e32211cafedf17dfc7b0d3a ./audit/borrow.spec.audit.ts
b38331bd028a012fe28464fb4cdbbb989ee53b77eb019a6774bde2dc0407d27c ./audit/withdraw.spec.audit.ts
14d646cc4a31388f3132af0327ece43b55e1a60efeb8d7eca576f94fe09dafb0 ./audit/atoken-transfer.spec.audit.ts
f00c143cf2a47b2fa53aa98f886375680d69e34fb253bd03bc4eb4d3a3b4aa32 ./audit/yield-manager.spec.audit.ts
227ef92128e40496f302a53480c77f0c6f284d0d9b70156dd8dc0fe7fd5dac25 ./audit/d.deploy.spec.audit.ts
4d5e8788b277cfc3ff8804394d5caf650f70b4cec836251d391d7055f8f44bdc ./audit/convex-mim-3crv-vault.spec.audit.ts
7e9b59cfa5f3c5fb8726535809617953696 2a4cace1073f4a1529ab55ecb2804 ./audit/liquidation-lido-underlying.spec.audit.ts
f57c48c48e538ad6117f7f036342ccf39e540633398243b98c88a20046cc921e ./audit/convex-iron-bank-vault.spec.audit.ts
901da03b536ce7f80622ff324199197a3e22dee9908a3effecfa0d8e09970b05 ./audit/liquidation-lido-atoken.spec.audit.ts
43470ddd522d39d71f41b23c6fa0149ba06df19413b82dab604250b246321a36 ./audit/lido-vault.spec.audit.ts
ca6e4a58f68930e900ac8e7b0535eab5de0c9a842179441caabe1a853c5bf486 ./audit/convex-frax-usdc-vault.spec.audit.ts
62b71a9398f8ca0e47afd0a138587fc875c6e627e76a79715966b1416d464d31 ./audit/lending-pool-addresses-provider.spec.audit.ts
850c9fcec42d6854b0f8759ea208d3a30049b5fb1d5146d046abae7442850198 ./audit/addresses-provider-registry.spec.audit.ts
deb0f66c9bde952c5c60c40ee1ac7086a0c39e3b0651b78fa2a022ae2a9254ef ./audit/variable-debt-token.spec.audit.ts
```

Changelog

- 2022-09-22 - Initial report
- 2022-10-13 - Final report

About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors 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 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

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, Aave, 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