Quantstamp

Sperax - USDs

Executive Summary

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

Type	Yield automating stablecoin
Timeline	2023-10-26 through 2023-11-15
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Sperax Gitbook
Source Code	• Sperax/USDs-v2 #a4cb9ac • Sperax/USDs-v2 #f958ea2
Auditors	• Pavel Shabarkin Auditing Engineer • Ibrahim Abouzied Auditing Engineer • Shih-Hung Wang Auditing Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	32 Fixed: 16 Acknowledged: 16
High severity findings	4 Fixed: 3 Acknowledged: 1
Medium severity findings	3 Fixed: 1 Acknowledged: 2
Low severity findings	21 Fixed: 9 Acknowledged: 12
Undetermined severity findings	0
Informational findings	4 Fixed: 3 Acknowledged: 1

Summary of Findings

USDs is a stablecoin that automatically generates yield for end-users, eliminating the need for them to stake their tokens or actively claim their yield. Users can mint USDs by supplying whitelisted collateral. Later, they have the option to use their USDs to redeem their chosen collateral. Users can choose to opt out of the auto-yield rebasing mechanism. To generate yield, the collateral is invested in liquidity pools across various protocols, including Aave, Compound, and Stargate. Half of the yield is allocated to pay auto-yield to USDs holders, while the other half funds SPA buybacks.

The Sperax team maintains a clean, well-structured, and documented codebase. The overall quality of the code meets industry standards.

During this security assessment, we identified three high-severity and three medium-severity findings. All high-severity findings were related to Oracle integrations. The medium-severity findings were primarily discovered in edge cases of the protocol design, which could impact the protocol under specific circumstances. However, the risk posed by these medium-severity findings is still high. During the fix phase, the Sperax team also identified one high-severity (SPE-4) and one informational-severity (SPE-29) findings.

The USDs token follows the ERC20 interface, but its underlying logic is complex and non-standard. Our review of the USDs rebasing mechanisms revealed no major security concerns. However, it's important to recognize that changes in any of the components USDs relies on could alter the current security model, potentially introducing new vulnerabilities.

Additionally, we identified 20 low-severity findings that warrant attention. Addressing these would strengthen the protocol and help mitigate future security risks.

ID	DESCRIPTION	SEVERITY	STATUS
SPE-1	DoS of SPA Oracle Contract	• High	Acknowledged
SPE-2	Chainlink Oracle Could Process Stale and Incorrect Data	• High	Fixed
SPE-3	VST Oracle Data Could Be Stale	• High	Fixed
SPE-4	The `YieldReserve` Contract Did Not Account Decimals During	• High	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
	Swaps		
SPE-5	User's Collateral Could Be Arbitrary Allocated to Undesired Strategies	Medium	Acknowledged
SPE-6	Temporary Immobilization of Interest Within the Stargate Aggregator in Case of Emergency	Medium	Fixed
SPE-7	Users May Lose USDs Yields when Redeeming Collateral	Medium	Acknowledged
SPE-8	USDs Minting Could Be DoS	Low	Acknowledged
SPE-9	Off-by-One Error in USDs Token Balances	Low	Acknowledged
SPE-10	Privileged Roles and Ownership	Low	Acknowledged
SPE-11	Allocation Functionality Does Not Explicitly Validate Non-Existing Strategy Address	Low	Fixed
SPE-12	Missing Constraints for `maPeriod` Settings	Low	Fixed
SPE-13	User Can Lose Their Collateral by Depositing Directly Into Strategy	Low	Fixed
SPE-14	The `_dstToken` Should Be Early Rejected and Not Be USDs Address	Low	Acknowledged
SPE-15	Fees Are Calibrated Imprecisely	Low	Acknowledged
SPE-16	Dust Funds Can Be Locked in the Stargate Strategy	Low	Fixed
SPE-17	Potential Fund Losses in Emergency Withdraw of the Stargate Strategy	Low	Acknowledged
SPE-18	Use of `safeApprove()` Can Allow Dos Attacks	Low	Fixed
SPE-19	Unable to Handle Fee-on-Transfer Tokens	Low	Acknowledged
SPE-20	TWAP Period for Uniswap V3 Oracles May Be Insufficiently Small	Low	Fixed
SPE-21	Address Aliasing May Affect Cross-Chain Access Control	Low	Acknowledged
SPE-22	Yield Distribution Events Are Predictable	Low	Acknowledged
SPE-23	External Risks that May Affect Protocol Security	Low	Acknowledged
SPE-24	Missing Input Validation	Low	Fixed
SPE-25	Critical Role Transfer Not Following Two-Step Pattern	Low	Acknowledged
SPE-26	Ownership Can Be Renounced	Low	Acknowledged
SPE-27	The Drip Rate Continuously Slows Down	Low	Fixed
SPE-28	Contract Never Initialized	Low	Fixed
SPE-29	Incorrect `vaultAmt` Value Returned in the `redeemView()`	Informational	Fixed

ID	DESCRIPTION	SEVERITY	STATUS
SPE-30	Use of Solidity Version with Known Compiler Bugs	● Informational ⓘ	Fixed
SPE-31	Slippage Restrictions Cannot Be Updated	● Informational ⓘ	Acknowledged
SPE-32	Application Monitoring Can Be Improved by Emitting More Events	● Informational ⓘ	Fixed

Assessment Breakdown

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

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

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

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

Methodology

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

Scope

Files Included

Repo: https://github.com/Sperax/USDs-v2/tree/main(a4cb9aceb191eafe058f890423c9c87b26ed5cda) Files: contracts/*

Files Excluded

Repo: https://github.com/Sperax/USDs-v2/tree/main(a4cb9aceb191eafe058f890423c9c87b26ed5cda) Files: contracts/libraries/StableMath.sol contracts/interfaces utils/BrownieUtils.sol

Findings

SPE-1 DoS of SPA Oracle Contract

● High ⓘ Acknowledged

Update

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

> Given that SPA is a volatile token, we have currently set the deviationBips as 100 (1%) and will configure the diaMaxTimeThreshold in the SPAOracle contract to be 30 minutes. This is to ensure a balance in the number of transactions recorded in the DIAOracle and to manage the staleness of the price. Apart from this we have reached out to DIA Oracle team to implement a heartbeat feature to force push the price transactions at a fixed interval ensuring regular updates in the price feed. We will continue to monitor the price feeds and update the configurations as needed based on market conditions.

File(s) affected: `contracts/oracle/SPAOracle.sol`

Description: The `SPAOracle` contract utilizes the DIA Oracle to get price feed for SPA/USD pair. The configuration of SPA Oracle is aimed to be 600s which is 10 minutes, however the current data feed reporting exceeds this limit. The average time period for data feed update in the oracle smart contract is 3-4h. The check of `lastUpdated` variable in the SPAOracle contract ensures that contract will not proceed with execution if the data in the DIA oracle is stale. Having the feed update on average on 3-4h would make SPAOracle contract and components dependent on this functionality unaccessible. The Sperax protocol could be DoS if the configuration of frequency update will not be changed.

Recommendation: Configure settings of DIA oracle node to have a more frequent push-update (price updates should be completed at least every 10 minutes).

SPE-2 Chainlink Oracle Could Process Stale and Incorrect Data

● High ⓘ Fixed

Update

Quantstamp: Sperax team ensures that the returned price is not a negative number, verifies that the oracle was updated within the expected time period, which can be configured.

Update

Marked as "Fixed" by the client. Addressed in: `f10f3cae40511339c0dbea42af40d43fa1cc81bd`. The client provided the following explanation:

- Implemented a configurable Timeout parameter for individual oracle pairs.
- Incorporated the recommended check into the code.

File(s) affected: `contracts/oracle/ChainlinkOracle.sol`

Description: The Sperax contract is planned to be deployed on the Arbitrum network. The `ChainlinkOracle` contract ensures that sequencer of the Arbitrum network does not have a downtime before processing the data from the feed aggregator. However, the `ChainlinkOracle` contract does not verify that data is stale and correct. This way returned data may not be updated in the expected timeframe, and price data could be incorrect.

Recommendation: Consider adding the following validation checks after pulling data from the ChainLink oracle aggregator. Determine the time window of update for each oracle pair to ensure the last update is not later then `CONFIGURABLE_TIME`.

```
 require(updatedAt != 0, "Round is not complete");
require(block.timestamp - updatedAt <= CONFIGURABLE_TIME, "Stale price");
require(price >= 0, "Invalid price");
```

SPE-3 VST Oracle Data Could Be Stale

● High ⓘ Fixed

Update

Quantstamp: The Sperax team removed VST Oracle from the codebase.

Update

Marked as "Fixed" by the client. Addressed in: `eda0c1f2c32c3316067faddb60b738d00b8238de`. The client provided the following explanation:

> All artifacts related to VST have been removed from the repository.

File(s) affected: `contracts/oracle/VSTOracle.sol`

Description: VST Oracle returns last data update at the `getPriceData` request, however the `VSTOracle.getPrice()` does not verify staleness of the returned data. So, the data returned from the `VSTOracle` could be stale.

Recommendation: Consider adding the following validation checks after pulling data from the VST oracle aggregator.

```
require(block.timestamp - updatedAt <= 600, "Stale price");
```

SPE-4
The `YieldReserve` Contract Did Not Account Decimals During Swaps

● High ⓘ Fixed

Update

Quantstamp: The Sperax team implemented two `onlyOwner` functions `toggleSrcTokenPermission()` and `toggleDstTokenPermission()` to set the conversion factor of the token `10^(18-token.decimals())`.

Within the `getTokenBForTokenA()` function Sperax team implemented calculation taking into account the number of decimals for the source and destination token.

Within the `SPABuyback._getUsdsOutForSpa()` function Sperax team does not take into account tokens' decimals, but both SPA and USDs tokens have the same number of decimals (10^18), which does not pose a threat to a protocol.

File(s) affected: `contracts/buyback/YieldReserve.sol`

Description: During the fix process, the Sperax team identified a severe vulnerability in the `YieldReserve` contract: it failed to account for the number of decimals in the target token.

This allowed an attacker to swap a source token with a higher number of decimals for a significantly larger amount of target tokens with fewer decimals. For example, an attacker could swap DAI (10^18) for USDC (10^6) and directly profit. Conversely, when a source token has fewer decimals, the swapper could lose funds if the target token has a higher number of decimals. Both scenarios could lead to the loss of user funds in some form.

Recommendation: Include amount of decimals of source and target tokens into the calculation.

SPE-5
User's Collateral Could Be Arbitrary Allocated to Undesired Strategies

● Medium ⓘ Acknowledged

Update

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

> The objective of this upgrade is to take a step forward towards decentralization. As part of this, the allocation function is now publicly accessible and protected by a pre-configured allocation cap for each collateral-strategy pair, as determined by the protocol governance. In case of emergencies, we have essential functions at our disposal to:
> 1. Withdraw funds from the strategy.
> 2. Update the allocation cap to maintain an appropriate risk/reward ratio.
> 3. Unlink a strategy for a collateral to prevent further allocations.

Description: When user deposits collateral, the `VaultCore` contract does not immidiately deposit funds into yield aggregators. The `VaultCore` contract implements the `allocate()` function that allocates all the available funds in the vault into the whitelisted strategy. The allocation is limited by the `allocationCap` for strategy, however if there is some strategies that are not optimal for usage anyone would be still able to deposit all the available collateral (limited by `allocationCap`) into that strategy. Other users could allocate funds of users into undesired strategies.

For example, if one of the external dependencies is compromised, the protocol team or the governance has not yet paused the affected collateral from the vault or the affected strategy. In this case, an attacker can call `allocate()` to send more collateral funds to the compromised dependencies and potentially make the vault lose more funds until the allocation cap is reached.

Recommendation: Guard the `allocate` function with `onlyOwner` modifier, or limit the amount of funds a user can allocate based on the their unallocated deposit.

SPE-6
Temporary Immobilization of Interest Within the Stargate Aggregator in Case of Emergency

● Medium ⓘ Fixed

Update
Quantstamp: The Sperax team subtracts the `amtRecv` amount from the `allocatedAmt` variable during the `emergencyWithdrawToVault()` function call.

Update
Marked as "Fixed" by the client. Addressed in: `ee45b507aad621df5a347e15022ea98e42d09f51`. The client provided the following explanation:

> Accounting error in emergencyWithdrawToVault() in the StargateStrategy contract was fixed

File(s) affected: `contracts/strategies/stargate/StargateStrategy.sol`

Description: The `StargateStrategy` contract implements functionality that allows owner of the contract to withdraw all the deposited funds from the aggregator in case of the incident. In case the owner call the `emergencyWithdrawToVault()` function all the funds will be withdrawn to the `VaultCore` contract. However, the `assetInfo[_asset].allocatedAmt` variable is not updated, what will create accounting inconsistencies once the report will continue operations and user will be allowed to deposit into this aggregator. Users will not be able to earn interest until balance in the Stargate aggregator will be higher than the `assetInfo[_asset].allocatedAmt` variable.

Recommendation: In the case of call of the `emergencyWithdrawToVault` function by owner of the contract subtract the `amtRecv` variable from the `assetInfo[_asset].allocatedAmt` variable.

SPE-7 Users May Lose USDs Yields when Redeeming Collateral

• Medium ⓘ Acknowledged

Update
Quantstamp: The Sperax team acknowledged the issue and explained the reason, however users should be aware that USDs yield could be lost if user will not call `rebase()` function before `redeem()`.

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

> We prefer maintaining the current approach for the following reasons:
> 1. If we execute the rebase before _redeem, a user might face difficulty redeeming their entire USDs balance as the rebase could increase the user's balance.
> 2. To address the first issue, we considered adding a redeemFull function that would redeem the entire USDs balance, including any additional amount from the internal rebase. However, this would necessitate users approving additional USDs upfront to accommodate the rebased amount, a step we want to avoid.
> 3. Chronologically, it is logical for the user to initiate the rebase first by explicitly calling the rebase function and then proceed with redemption.

File(s) affected: `contracts/vault/VaultCore.sol`

Description: According to the rebasing mechanism design, USDs holders can earn additional USDs as yields by simply holding the token. In a rebasing event, the rebase manager determines the rebase amount, which will be distributed to all USDs holders (except those who opt out) in proportion to their token balance.

The `VaultCore.rebase()` function implements the rebasing logic, which will be triggered whenever a user calls the `mint()` or `redeem()` function on the vault contract. Anyone can also invoke the `rebase()` function.

However, in the `redeem()` function, `rebase()` is called after the user's USDs tokens are burned but not before, which may result in the user receiving less USDs yields if they have a non-zero amount of USDs left after the redemption. Users may receive less collateral in return if they provide all their USDs when redeeming.

In other words, whether the user explicitly calls `rebase()` before `redeem()` or not will affect the USDs yield they earned or the redeemed collateral amount. Users might suffer from fund loss if they did not call `rebase()` before `redeem()`.

Recommendation: In the `VaultCore._redeem()` function, consider moving the `rebase()` function call after the input validation checks but before any token transfers or state updates, similar to the approach in the `mint()` function. This way, it will ensure that before the user's USDs are burned, their USDs balance is up to date.

SPE-8 USDs Minting Could Be DoS

• Low ⓘ Acknowledged

Update

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

> Considering that a rebase only occurs when the `rebasingSupply` is greater than 0, utilizing the rebasing fund, which is a portion of USDs included in `nonRebasingSupply`, this should theoretically never happen. For reference:
>
> 1. Rebase is initiated in the VaultCore contract, which first attempts to obtain the available fund for rebase from the RebaseManager.
> 2. RebaseManager, based on the available funds for rebase, Principal (rebasing supply), configured APR caps, and the last rebase timestamp, calculates the rebasing amount for the current timestamp.
> 3. VaultCore invokes a USDs rebase only if rebaseAmt > 0. If RebasingSupply is 0 ⇒ rebaseAmt is 0 ⇒ hence rebase would be skipped.

File(s) affected: `VaultCore.sol`

Description: Only `VaultCore` contract can mint USDs tokens. During the investigation of the mint function of the `VaultCore` contract, we have determined that protocol could be potentially DoS. Before calling `mint` function on the USDs contract the `rebase` function is called, which could revert if the `_totalSupply` of the USDs was zero. This means on the early first user deposit the protocol could be DoS. This DoS scenario could also appear if the protocol decides to migrate to another address in case of incident or planned migration, or the protocol will have the full reset of collateral. The USDs contract is currently upgradable and already has total supply, however it still pose a risk that protocol could be DoS in the cases described above.

Recommendation: Ensure that total liquidity of USDs cannot be zero.

SPE-9 Off-by-One Error in USDs Token Balances

• Low ⓘ Acknowledged

Update

Quantstamp: The Sperax team understood main issue and provided an explanation of acknowledgment. However, team did not provide explanation of rounding direction.

Update

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

> This is a recognized issue. We have already addressed this error for non-rebasing wallets by eliminating the credit systems for such wallets and making USDs function like a standard ERC20 token. However, for rebasing wallets, we are maintaining the current state for now. This decision is based on the fact that the amounts involved are minuscule (trillionths of a trillionth of a dollar), which is deemed insignificant, especially considering that holders receive a continuous yield using the same mechanism. We will ensure to emphasize the potential mathematical inaccuracies in our documentation.

File(s) affected: `contracts/vault/VaultCore.sol`

Description: According to the code comments, the `VaultCore.mintView()` function returns the expected minted USDs amount and the fee amount for a given collateral type and amount. However, the actual amount received in USDs after minting may be less than the expected amount by 1. This off-by-one error is caused by the precision loss in USDs internal accounting, which only happens to rebasing accounts.

The same off-by-one error also exists when transferring from or to a rebasing account. Third parties integrating with the `VaultCore` or `USDs` contracts should be aware of such potential inconsistency.

Rounding direction is also a concern when performing integer divisions. Generally, the protocol should round towards a direction (either round down or up) that favors the protocol instead of the users. In the `_burn()` and `_executeTransfer()` functions of the USDs token, `mulTruncateCeil()` should be used to not favor the caller or sender.

Recommendation: Consider whether this off-by-one error could confuse users or external protocols and whether it should be addressed. If not, consider clarifying the expected behaviour on public-facing documentation. Regarding rounding directions, consider modifying the relevant functions to use `mulTruncateCeil()` instead.

SPE-10 Privileged Roles and Ownership

• Low ⓘ Acknowledged

Update

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

> The ownership of all USDs protocol contracts is currently held by a Multisig (with 3 out of 5 signature confirmations), with the intention to transfer it to an on-chain governance once established. Current Owner: 0x5b12d9846F8612E439730d18E1C12634753B1bF1

File(s) affected: `buyback/SPABuyback.sol`, `buyback/YieldReserve.sol`, `oracle/BaseUniOracle.sol`, `oracle/ChainlinkOracle.sol`, `oracle/MasterPriceOracle.sol`, `rebase/Dripper.sol`, `rebase/RebaseManager.sol`, `strategies/InitializableAbstractStrategy.sol`, `token/USDs.sol`, `vault/CollateralManager.sol`, `contracts/vault/VaultCore.sol`

Description: The protocol design includes several privileged roles that can control the contract configurations or execute specific functions. Therefore, users are exposed to the risk of being attacked if the privileged roles are malicious or compromised:

- `BaseUniOracle`: The owner can configure the oracle by calling `updateMasterOracle()` and `setUniMAPriceData()`. +
- `ChainlinkOracle`:The owner can configure the oracle by calling `setTokenData()`
- `CollateralManager`:
  - The owner can configure the accepted collateral types by calling `addCollateral()`, `updateCollateralData()`, and `removeCollateral()`.
  - The owner can configure the investment strategies by calling `addCollateralStrategy()`, `updateCollateralStrategy()`, `removeCollateralStrategy()`, and `updateCollateralDefaultStrategy()`. **A compromised owner can send protocol funds to a false strategy.**
- `Dripper`:
  - `recoverTokens()`: Intended to be an emergency withdrawal function. **A compromised owner can withdraw any amount of any token at any time.**
  - `updateVault()`: **A compromised owner can redirect dripped funds by changing the vault.**
  - `updateDripDuration()`: The owner can configure the duration across which funds are dripped.
- `InitializableAbstractStrategy`:
  - `updateVault()`: **A compromised owner can redirect withdrawn funds by changing the vault.**
  - `updateHarvestIncentiveRate()`: The owner can configure what percentage of rewards are given to the function caller as an incentive to offset gas costs.
  - `updateSlippage()`: The owner can configure the maximum allowed slippage in withdrawals.
  - `recoverERC20()` **: The owner can withdraw all ERC-20 tokens at any time.**
- `MasterPriceOracle`: The owner can configure the price feeds with `updateTokenPriceFeed()` and `removeTokenPriceFeed()`.
- `RebaseManager`: The owner can configure the contract variables with `updateVault()`, `updateDripper()`, `updateGap()`, and `updateAPR()`.
  - The `vault` (through the `onlyVault` modifier) can call `fetchRebaseAmt()` to trigger `dripper.collect()`.
- `SPABuyback`:
  - `withdraw()`: Intended to be an emergency withdrawal function. **A compromised owner can withdraw any amount of any token at any time.**
  - `updateRewardPercentage()`: The owner can modify the percentage of SPA that is burned vs sent as a reward.
  - `updateVeSpaRewarder()`: **A compromised owner can redirect rewards to an address of their choice.**
  - `updateOracle()`: The owner can update the oracle address.
- `USDs`:
  - **The owner can override a user's decision for opting in/out of the rebasing mechanism through `rebaseOptIn()` and `rebaseOptOut()`.**
  - `updateVault()`: **A compromised owner can mint any amount of USDs by assigning themselves as the `vault`.**
  - `pauseSwitch()`: The owner can pause the contract.
  - The `vault` (through the `onlyVault` modifier) can call `rebase()` and `mint()`.
- `VaultCore`:
  - The owner can configure the contract variables by calling: `updateFeeVault()`, `updateYieldReceiver()`, `updateCollateralManager()`, `updateRebaseManager()`, `updateFeeCalculator()`, and `updateOracle()`.
  - **A compromised owner can redirect fees by calling `updateFeeVault()`.**
- `YieldReserve`:
  - The owner can configure which tokens are allowed to be swapped with `toggleSrcTokenPermission()` and `toggleDstTokenPermission()`.
  - `withdraw()`: Intended to be an emergency withdrawal function. **A compromised owner can withdraw any amount of any token at any time.**
  - The owner can configure the contract variables by calling: `updateBuybackPercentage()`, `updateBuybackAddress()`, `updateOracleAddress()`, `updateDripperAddress()`, and `updateVaultAddress()`.
  - **A compromised owner can redirect the USDs intended for buybacks and dripping by changing the `dripper` and `buyback` addresses.**

Recommendation: Consider documenting the risk and impact a compromised privileged role can cause on the protocol and inform the users in detail. As the privileged roles can be the single point of failure of the protocol, consider using a multi-sig or a contract with a timelock feature to mitigate the risk of being compromised or exploited.

SPE-11
Allocation Functionality Does Not Explicitly Validate Non-Existing Strategy Address

• Low ⓘ Fixed

Update

Quantstamp: The Sperax team implemented the verification mechanism to reject non-allowlisted strategy addresses.

Update
Marked as "Fixed" by the client. Addressed in: `acafbef7b12ecbdf13abd913bf636035c589720f`. The client provided the following explanation:

> We have added the missing check for the collateral to Strategy mapping.

File(s) affected: `contracts/vault/VaultCore.sol`

Description: The allocate function of the VaultCore contract does not explicitly reject addresses which are not configured in the whitelist. If the strategy address is not in the whitelist the `collateralStrategyInfo[_collateral][_strategy].allocationCap` variable will have zero value, then multiplication to calculate the `maxCollateralUsage` will result in zero as well. Then, execution will likely revert. However it is always better to define the intention explicitly and harden the protocol having according security checks.

Recommendation: Validate address of the strategy in `allocate` function, revert execution if there is no strategy found.

SPE-12 Missing Constraints for `maPeriod` Settings

• Low ⓘ Fixed

Update
Quantstamp: The Sperax team implemented mechanisms recommended by the Quantstamp team.

Update
Marked as "Fixed" by the client. Addressed in: `943b95d752f95e20e137d0a797bb9d332fba8179`. The client provided the following explanation:

> As recommended we have added the MA_PERIOD_LOWER_BOUND = 10 minutes and MA_PERIOD_UPPER_BOUND = 2 hours to validate the configuration of maPeriod.

Description: The `maPeriod` setting of the oracle configuration does not have an upper limit sanity check, what would allow a malicious owner to set broad time window for oracle updates.

Recommendation: Implement lower and upper boundaries for `maPeriod` setting

SPE-13
User Can Lose Their Collateral by Depositing Directly Into Strategy

• Low ⓘ Fixed

Update
Quantstamp: The Sperax team implemented access control recommended by the Quantstamp team.

Update
Marked as "Fixed" by the client. Addressed in: `ee45b507aad621df5a347e15022ea98e42d09f51`. The client provided the following explanation:

> We have added the recommended modifier for the functions.

File(s) affected: `contracts/strategies/aave/AaveStrategy.sol`, `contracts/strategies/compound/CompoundStrategy.sol`, `contracts/strategies/stargate/StargateStrategy.sol`

Description: The aggregator strategies allow anyone to deposit collateral directly instead of going through the `VaultCore` contract. In the case user decide to deposit collateral directly, the USDs tokens will not be minted and user directly lose the funds. There is no incentive to keep this function public. For general safety it is better to guard this function with `onlyVault` modifier.

Recommendation: Guard `deposit` function with `onlyVault` modifier.

SPE-14
The `_dstToken` Should Be Early Rejected and Not Be USDs Address

• Low ⓘ Acknowledged

Update

Quantstamp: The Sperax team understood main issue and provided an explanation of acknowledgment.

Update

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

> This is not possible for multiple reasons:
> 1. Any USDs collected during a swap is directly forwarded to both the Dripper and SPABuyback contract at a preset proportion (currently 50-50).
> 2. Considering that the allowed source tokens can be either USDs or any of the allowed USDs collaterals (e.g., USDC), swapping USDs with USDs or USDC would yield the same result.
> 3. The purpose of the YieldReserve contract is to convert any yield received via the strategies into USDs; therefore, USDs will never be configured as a source token.

File(s) affected: `contracts/buyback/YieldReserve.sol`

Description: The `_dstToken` variable in the `swap` function of `YieldReserve` contract does not explicitly define that it should not be a USDs address. In case the `dstToken` is configured with USDs address an attacker could deplete the whole collected interest sent to `YieldReserve` contract and leave USDs holders without earnings.

Recommendation: Define explicitly the check that verifies `_dstToken` is not USDs address.

SPE-15 Fees Are Calibrated Imprecisely

• Low ⓘ Acknowledged

Update

Quantstamp: The Sperax team understood an issue and provided an explanation of acknowledgment.

Update

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

> Given that our definition of collaterals for USDs is that each USDs is backed with 1 collateral, imposing the fee calculation based on the quantity of the collateral in the pool makes sense. The downsidePeg should protect against minting USDs with depegged collaterals. And the redemptionCap should protect against the redemption of collaterals with a price greater than $1. We do plan to add more sophisticated fee models that will benefit the protocol.

File(s) affected: `contracts/vault/FeeCalculator.sol`

Description: In `_calibrateFee()`, the mint fee and redeem fee are calibrated to incentivize minting/redeeming to the desired collateral amount. However, the collateral is assumed to be identically priced to USDs. If the collateral or USDs depegs, the fees will not be correctly calibrated.

Recommendation: Clarify if the function is intended to compare the quantities of USDS to the collateral, or the values of USDs to the collateral. If the `upperLimit` and `lowerLimit` do not account for price fluctuations and precise fee calibration based on token values is needed, compare `upperLimit`, `lowerLimit`, and `totalCollateral` based on their token prices before calibrating the fee.

SPE-16 Dust Funds Can Be Locked in the Stargate Strategy

• Low ⓘ Fixed

Update

Quantstamp: The Sperax team introduced `recoverERC20()` function to recover remaining tokens from the the strategy. **This function allows the owner to withdraw all ERC-20 tokens from all strategy contracts at any time**

Update

Marked as "Mitigated" by the client. Addressed in: `ee45b507aad621df5a347e15022ea98e42d09f51`. The client provided the following explanation:

> Added an emergency recoverERC20() function to clear out residual and unwanted tokens from the strategy.

File(s) affected: `strategies/stargate/StargateStrategy.sol`

Description: The Stargate strategy deposits collateral as LP tokens to the Stargate protocol by calling the `addLiquidity()` function on the router with the `_amount` parameter specifying the amount of liquidity to add.

However, according to Stargate's implementation and the on-chain contract, the router may not transfer the entire `_amount` from the strategy but only a rounded down amount based on the pool's conversion rate (see Router#L101-L103):

```
uint256 convertRate = pool.convertRate();
_amountLD = _amountLD.div(convertRate).mul(convertRate);
_safeTransferFrom(pool.token(), msg.sender, address(pool), _amountLD);
```

As a result, a small amount of collateral funds will be left in the Stargate strategy and not accounted for in the allocated amount. These funds are only withdrawable if the strategy implementation logic is upgraded.

Recommendation: Consider allowing a privileged role to withdraw the dust funds in the strategy and return them to the vault.

SPE-17 Potential Fund Losses in Emergency Withdraw of the Stargate Strategy

• Low ⓘ Acknowledged

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

> The bug in Yearn Finance's code occurred because they accepted _amountNeeded, and any amount received less than _amountNeeded was reported as a loss, when, in fact, it was just illiquid. There was no actual loss of funds; it was a reporting error, which they later addressed and fixed. In our strategy, we explicitly specify the asset, check its balance, and then specify the lpTokenAmount to be withdrawn. Stargate's instantRedeemLocal function caps the lpAmount and only transfers the capped lpAmount to the assets that they can return. If the liquidity is lower, they will simply take fewer lpTokens. As we calculate and subtract the amtRecv variable based on the amountSD returned from the strategy and use the pool's convertRate, we ensure accurate handling of the transaction.

File(s) affected: `strategies/stargate/StargateStrategy.sol`

Description: The `emergencyWithdrawToVault()` function of the `StargateStrategy` performs an emergency withdrawal from the Stargate farm contract without receiving any rewards.

Additionally and more importantly, it invokes `instantRedeemLocal()` on the Stargate router contract to redeem the entire received LP token without first ensuring that the withdraw amount is less than the maximum amount, the `deltaCredit` variable in the Stargate pool contract.

As a result, if the strategy attempts to withdraw more than the `deltaCredit` amount of LP tokens, they will receive less liquidity in return, causing a loss of funds. More details can be found in Yearn's Stargate strategy incident report.

Recommendation: Consider clarifying this behaviour in code comments and public-facing documentation so that users and protocol admins know the associated risks of invoking this function.

SPE-18 Use of `safeApprove()` Can Allow Dos Attacks

• Low ⓘ Fixed

Update
Quantstamp: The Sperax team changed all instances of the `safeApprove()` function to `forceApprove()`.

Update
Marked as "Fixed" by the client. Addressed in: `cba249ca90d40cf3468a8c919a76d350cffd99e9`. The client provided the following explanation:

> We replaced the safeApprove function with forceApprove instead of safeIncreaseAllowance to mitigate possible failures due to non-standard implementations by other ERC20 tokens.

File(s) affected: `strategies/aave/AaveStrategy.sol`, `strategies/compound/CompoundStrategy.sol`, `strategies/stargate/StargateStrategy.sol`, `buyback/YieldReserve.sol`

Description: The `safeApprove()` function does not allow the token owner to change the allowance from a non-zero value to another non-zero value. The non-zero allowance has to be reset to 0 first if it needs to be changed.

As a result, such a design may cause DoS to the `deposit()` function of the `StargateStrategy` contract. As described in the "Dust Funds Can Be Locked in the Stargate Strategy" issue, the Stargate router only transfers a rounded-down amount of tokens from the strategy, while the strategy approves the router the entire amount.

As a result, a non-zero allowance amount may be left after the `addLiquidity()` function call if `_amount` is not exactly divisible by `convertRate()`. A non-zero allowance will block subsequent `deposit()` function calls since `safeApprove()` requires the allowance to be set to 0 first.

Recommendation: Since the `safeApprove()` function has been deprecated and removed in the latest OpenZeppelin's `SafeERC20` implementation, consider using the `safeIncreaseAllowance()` function instead.

SPE-19 Unable to Handle Fee-on-Transfer Tokens

• Low ⓘ Acknowledged

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

> Given that the Fee-on-Transfer is not defined as a standard in the ERC20 interface, we believe that its implementation could have a destructive effect, resulting in composability and compatibility issues. We will leave this as it is and address it when there is a standard for tackling such situations.

File(s) affected: `strategies/aave/AaveStrategy.sol`, `strategies/compound/CompoundStrategy.sol`, `strategies/stargate/StargateStrategy.sol`, `buyback/YieldReserve.sol`, `vault/VaultCore.sol`

Description: In several places in the codebase, it is assumed that the caller of `token.safeTransferFrom(address, amount)` will receive the exact `amount` of `token` after the function call. However, this assumption may not be valid regarding fee-on-transfer tokens. For example, USDT is possibly a fee-on-transfer token since it includes a fee-charging mechanism, which the admin may enable in the future.

If an amount fee is charged in the `safeTransferFrom()` call, the caller will receive less than the specified `amount`. Depending on the use cases, this may cause internal accounting errors or transaction failures.

For example, in the `deposit()` function of the Aave strategy contract, the token amount supplied to Aave is the same as what is provided in the `safeTransferFrom()` call.

```
 IERC20(_asset).safeTransferFrom(msg.sender, address(this), _amount);
IERC20(_asset).safeApprove(address(aavePool), _amount);
aavePool.supply(_asset, _amount, address(this), REFERRAL_CODE);
```

Therefore, if the contract received less than `_amount` of tokens, the subsequent `aavePool.supply()` call will fail due to insufficient funds.

Recommendation: Consider calculating the difference in token balances before and after the `safeTransferFrom()` call to get the actual received amount.

SPE-20 TWAP Period for Uniswap V3 Oracles May Be Insufficiently Small

• Low ⓘ Fixed

Update
Quantstamp: The Sperax team enforced period to be at least 10 minutes. If the cardinality of the pool is not sufficiently large, the `consult()` call will revert, which is an acceptable case.

Update
Marked as "Fixed" by the client. Addressed in: `dc8d81d947db407b2992d4d32c7f73537e87369e`. The client provided the following explanation:

> - We have added a constant MIN_TWAP_PERIOD of 10 minutes and the necessary check to ensure the TWAP period is greater than or equal to a reasonable value.
> - We have also increased the cardinality of the relevant pools to be greater than or equal to 200.

File(s) affected: `oracle/BaseUniOracle.sol`

Description: The `BaseUniOracle._getUniMAPrice()` function queries the TWAP value of a token from a Uniswap V3 pool with the TWAP `period` set as follows:

```
 uint32 oldestObservationSecondsAgo = OracleLibrary.getOldestObservationSecondsAgo(pool);
uint32 period = maPeriod < oldestObservationSecondsAgo ? maPeriod : oldestObservationSecondsAgo;
```

`maPeriod` is the target TWAP period configured by the protocol admin. However, if `oldestObservationSecondsAgo` is less than `maPeriod`, a smaller TWAP period will be used, causing the calculated price to be more easily affected by the spot price.

Recommendation: Consider adding a check to ensure that the result in TWAP `period` is at least greater than a reasonable value. Also, consider expanding the `cardinality` of the Uniswap V3 pool to keep the oldest observation in the pool longer if necessary.

SPE-21 Address Aliasing May Affect Cross-Chain Access Control

• Low ⓘ Acknowledged

Update

Quantstamp: The Sperax team is correct that the address aliasing mechanism is intended to prevent cross-chain exploits, but the purpose of such a mechanism is unrelated to this issue. This issue points out that the owner would be unable to control the protocol if the sequencer is down.

Alert

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

> We prefer not to override the Ownable OpenZeppelin contract to implement a fix for this issue. Additionally, the introduction of address aliasing was intended to prevent cross-chain exploits. It's worth noting that no other prominent protocols on Arbitrum are currently addressing this issue through patches.

Description: Most of the protocol contracts are deployed on Arbitrum. Arbitrum allows users to send cross-chain messages to call L2 contracts from L1 so that the interaction with L2 protocols can continue even when the sequencer is down.

However, when an L2 contract receives a cross-chain call from L1, the `msg.sender` value is usually not the sender who initiates the cross-chain call on L1. On Arbitrum, the `msg.sender` is the aliased address of the original sender.

Therefore, access control mechanisms based on the `msg.sender` value should take such changes into account if a privileged role needs to call an L2 contract from L1 in certain circumstances, for example, in an emergency but directly sending L2 transaction is not possible since the sequencer is down.

In the past, the Uniswap Factory deployed on Arbitrum did not consider address aliasing, and its owner was set to the address of the Timelock contract on L1, causing the execution of governance proposals on Arbitrum to be temporarily blocked (reference).

Recommendation: If the sequencer is down, a privileged role needs to call the L2 contracts from L1, consider retrieving the original sender in the code so that the authorization does not fail. For implementation details, please refer to Arbitrum's `AddressAliasHelper` contract.

SPE-22 Yield Distribution Events Are Predictable

• Low ⓘ Acknowledged

Update

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

> The official documentation has not been updated with the new mechanism yet. Previously, we manually performed rebases every 7-12 days to maintain randomness. However, with the upgraded version, anyone can initiate a rebase as long as certain conditions are met. These conditions include a calculated APR for the rebase greater than aprBottom and a time gap between consecutive rebases greater than the configured gap in the RebaseManager. Rebasing involves multiple steps (harvest, buyback) and is semi-automated within the mint and redeem process, accessible to anyone. This makes it difficult to predict and take advantage of. The rebase amount will be relatively small but more frequent, reducing the incentive for front-running. We will closely monitor and adjust parameters over time.

File(s) affected: `vault/VaultCore.sol`, `rebase/RebaseManager.sol`

Description: According to the official documentation:

> Yield is distributed approximately every 7 days. The exact distribution time is determined in a quasi-random way. We have decided on this randomised distribution time to prevent users from timing their USDs minting and redeeming with yield distribution events. Huge spike in minting or redeeming around the time of yield distribution can put strain on the peg and this randomisation works as a defence mechanism for maintaining peg

However, according to the code implementation, the yield distribution events, i.e., rebasing events, can be triggered by anyone by calling the `VaultCore.rebase()` function. As long as the cooldown period has passed and sufficient funds are in the vault or can be collected from `Dripper`, the `rebase()` call will take effect. Therefore, the rebasing events are considered easily predictable and can be executed by users.

Recommendation: Consider clarifying the intention of such a randomization defense mechanism and ensure it matches the code implementation.

SPE-23 External Risks that May Affect Protocol Security

• Low ⓘ Acknowledged

Update

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

> Given that USDs is built to act as a Yield-Automator for its holders. It will always be dependent on external protocols for generating the yield. At this point of time we can only focus on strengthening our process of integrating with other protocols like:
> 1. Assessing the credibility, security and risks involved while selecting a strategy for a collateral.
> 2. Strategies are to be linked only via governance proposals.
> 3. Ensuring diversification in the USDs strategy portfolio.
> 4. Actively monitoring the changes in the other protocols / depeg events impacting USDs.

File(s) affected: `oracle/SPAOracle.sol`, `oracle/USDsOracle.sol`, `strategies/aave/AaveStrategy.sol`, `strategies/compound/CompoundStrategy.sol`, `strategies/stargate/StargateStrategy.sol`, `vault/VaultCore.sol`

Description: The protocol extensively relies on several external applications, especially protocols that generate yields for USDs holders. Therefore, handling the failures of these external applications and controlling the damage is critical to the protocol. This issue highlights the possible failures of external applications and the potential impact:

1. Implementation Changes in External Protocols

The implementation of the strategy contracts highly depends on the behavior of the external protocols and their current configurations. For example, the `checkRewardEarned()` function of the `CompoundStrategy` contract assumes that the reward tokens are the same across all invested markets, which is not guaranteed to hold in the future. Also, the `_convertToCollateral()` function in the `StargateStrategy` contract assumes the conversion formula between LP and underlying tokens, which depends on Stargate implementation details. Therefore, a change in the external contracts in the future could possibly cause inconsistencies. Moreover, the protocol maintains the assumption that LP token and deposited collateral has same precision. For current strategies in review the implementation of the assumption is correct, however it is important to keep track of when adding new strategies to the protocol.

2. Security Incidents of External Protocols and Depeg Events

An attack on these protocols may cause the deposited collateral tokens to be lost. If the loss is significant and makes the USDs tokens not fully backed, USDs holders may likely choose to withdraw from the other unaffected strategies to avoid fund losses. After most or all the funds from the other strategies are withdrawn, the remaining holders withdrawing from the affected strategy will bear the loss for the others.

Similarly, if a stablecoin collateral depegs with a significant price fall, some USDs holders might redeem USDs for other unaffected collateral, further exacerbating the USDs unbacked situation.

3. TWAP Oracle Manipulation

Uniswap V3 pools with low liquidity are more vulnerable to manipulation since attackers require less capital to manipulate the prices. Also, the design of concentrated liquidity can make TWAP manipulations more cost-efficient, especially for the USDs/USDC pair, where most of the liquidity is likely to be concentrated around the center tick. A manipulated TWAP may affect the price evaluation in the `SPABuyback` and `YieldReserve` contracts, which may, for example, allow an attacker to buy USDs at a manipulated low price.

Recommendation:
1. Consider keeping track of the latest status of the external protocols and ensuring that the assumptions still hold if new changes are introduced.
2. Consider implementing on-chain monitoring systems and reacting promptly to security incidents on any external applications or potential TWAP manipulation activities, such as pausing the protocol operations to reduce the damage.
3. Increase the TWAP window to the time window that is sufficient enough to reduce the described risk, and consider adding more liquidity to the Uniswap pool in the central and around central liquidity ticks.

SPE-24 Missing Input Validation

• Low ⓘ Fixed

Update

Quantstamp: The Sperax team implemented validation of the `_asset` variable for `AaveStrategy._withdraw()` and `StargateStrategy._withdraw()` functions. The `CollateralManager.constructor()` now validates the `VAULT` parameter. Each strategy is also validated in the `CollateralManager.validateAllocation()` function now.

The Sperax team is correct about validation of `_data.baseMintFee + _data.baseRedeemFee` variables in the `CollateralManager.addCollateral()` function. Both of the values could be 80%. If a user does mint and then redeem, they will only get back 1 * 0.2 * 0.2 = 0.04 of the original amount.

Update

Marked as "Fixed" by the client. Addressed in: `3c62111de4817d1aba4142090600478406ae5520`. The client provided the following explanation:

> - We have add the missing validations.
> - CollateralManager.addCollateral(): Validate Helpers._isLTEMaxPercentage(_data.baseMintFee + _data.baseRedeemFee) is incorrect/unnecessary.

File(s) affected: `vault/CollateralManager.sol` , `contracts/strategies/aave/AaveStrategy.sol` , `contracts/strategies/stargate/StargateStrategy.sol`

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. The following inputs should be validated:

- `CollateralManager.validateAllocation()` :
  - Validate that `collateralStrategyInfo[_collateral][_strategy].exists` is `true` at the beginning of the function.
- `CollateralManager.constructor()` : Validate that `VAULT` is a non-zero address.
- `CollateralManager.addCollateral()` : Validate `Helpers._isLTEMaxPercentage(_data.baseMintFee + _data.baseRedeemFee)` .
- `AaveStrategy._withdraw()` : Validate that `_asset` is supported.
- `StargateStrategy._withdraw()` : Validate that `_asset` is supported.

Recommendation: Consider adding the validation checks to the above functions.

SPE-25 Critical Role Transfer Not Following Two-Step Pattern

• Low ⓘ Acknowledged

Update

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

> We have decided against transitioning to Ownable2Step because: |
> 1. It would break our USDs contract due to the increase in an internal storage variable. We aim to maintain a consistent Ownable structure across all other files.
> 2. Given that we already have a multi-sig Gnosis Safe acting as the Owner, our transaction confirmation process entails multiple steps.

File(s) affected: `See the Description section`

Description: The owner of the contracts can call `transferOwnership()` to transfer the ownership to a new address. If an uncontrollable address is accidentally provided as the new owner address then the contract will no longer have an active owner, and functions with the `onlyOwner` modifier can no longer be executed.

All the listed contracts are inherited from the OZ `Ownable` contract.

File(s) Affected:

- `buyback/SPABuyback.sol`
- `buyback/YieldReserve.sol`
- `oracle/BaseUniOracle.sol`
- `oracle/ChainlinkOracle.sol`
- `oracle/MasterPriceOracle.sol`
- `rebase/Dripper.sol`
- `rebase/RebaseManager.sol`
- `strategies/InitializableAbstractStrategy.sol`
- `token/USDs.sol`
- `vault/CollateralManager.sol`
- `vault/VaultCore.sol`

Recommendation: Consider using OpenZeppelin's `Ownable2Step` contract to adopt a two-step ownership pattern in which the new owner must accept their position before the transfer is complete.

SPE-26 Ownership Can Be Renounced

• Low ⓘ Acknowledged

Alert

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

> We prefer not to override renounceOwnership() with custom logic to avoid interfering with OpenZeppelin code. Additionally, considering we already have a multi-threshold Gnosis Safe serving as the Owner, our process involves multiple steps before confirming a transaction.

File(s) affected: `See the Description section`

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.

All the listed contracts inherited from OZ `Ownable` contract, and therefore the `owner` , by default, can be renounced. According to the protocol design, it is unlikely that the owner will be renounced since the owner would have to configure the collateral and strategy settings.

File(s) Affected:

- `buyback/SPABuyback.sol`
- `buyback/YieldReserve.sol`
- `oracle/BaseUniOracle.sol`
- `oracle/ChainlinkOracle.sol`
- `oracle/MasterPriceOracle.sol`
- `rebase/Dripper.sol`
- `rebase/RebaseManager.sol`
- `strategies/InitializableAbstractStrategy.sol`
- `token/USDs.sol`
- `vault/CollateralManager.sol`
- `vault/VaultCore.sol`

Recommendation: Consider overriding the `renounceOwnership()` function to revert the function call to avoid accidentally renouncing the ownership.

SPE-27 The Drip Rate Continuously Slows Down

• Low ⓘ Fixed

Update

Quantstamp: The Sperax team implemented drip rate mechanism as recommended by the Quantstamp team.

Update

Marked as "Fixed" by the client. Addressed in: `969732fa250407ffdfcf50b6748d11e68dfc3b26` . The client provided the following explanation:

> We have implemented the recommended changes.

File(s) affected: `rebase/Dripper.sol`

Description: The `Dripper` releases tokens to fund rebases to the vault at a steady `dripRate` . After a call to `collect()` , the `dripRate` is recalculated as follows:

```
dripRate = IERC20(Helpers.USDS).balanceOf(address(this)) / dripDuration;
```

If it is intended for the `Dripper` contract to release its entire balance over the course of the `dripDuration` , calls to `collect()` will continue to extend the duration it takes for the funds to be fully released as the `dripRate` drops with each call to `collect()` . Additionally, `collect()` can be called by any address.

Recommendation: Only adjust the `dripRate` when USDs is sent to the `Dripper` or when the `Dripper` is out of funds.

SPE-28 Contract Never Initialized

• Low ⓘ Fixed

Update

Quantstamp: The Sperax team implemented `initialize()` function in the `USDs` contract.

Update

Marked as "Fixed" by the client. Addressed in: `7d0434d5e53f93204a4a808f289449326e9b66f1` . The client provided the following explanation:

> We have added the initialize() function in all upgradeable contracts for completeness, as recommended.

File(s) affected: `token/USDs.sol`

Description: The `USDs` contracts lacks an `initialize()` function. This leaves the contract ownerless, making many privileged functions impossible to call. It is assumed that this function has been removed as part of a contract upgrade in which the owner has already been initialized, so the severity has been marked as Low. However, if the contract is ever used in a fresh deployment, it is critical for the `initialize()` function to be added.

Recommendation: Consider adding the `initialize()` function for completeness and potential future deployments.

SPE-29 Incorrect `vaultAmt` Value Returned in the `redeemView()`

• Informational ⓘ Fixed

Update
Quantstamp: The Sperax team implemented the fix to ensure that implementation is aligned with the specification.

File(s) affected: `contracts/vault/VaultCore.sol`

Description: During the fix process the Sperax team identified a spec related issue where `vaultAmt` variable is not updated when redemption amount is less than the vault balance.

In our examination of the mathematical error in the `vaultCore` contract's `_redeemView()` function, we have determined that it does not pose a security risk, but rather a discrepancy with the specifications. If the redemption amount (`calculatedCollateralAmt`) is smaller than the vault's balance, it indicates sufficient liquidity to fulfill the user's redemption request. The `redeem()` function will transfer the actual `calculatedCollateralAmt` to the user, updating the vault balance accordingly.

We note that the specification mentions `/// @return vaultAmt amount of Collateral released from Vault`. This suggests the expectation that the `vaultAmt` variable should reflect the amount of collateral moved from the vault contract. However, this issue is more related to the specification and user experience rather than security, and it does not appear to have any financial implications.

Recommendation: Update the `vaultAmt` variable accordingly, even in the case when redemption amount is less than the vault balance.

SPE-30 Use of Solidity Version with Known Compiler Bugs

• Informational ⓘ Fixed

Update
Quantstamp: The Sperax team switched to 0.8.19 compiler version which still has known bugs, but they don't affect the code. The Uniswap library function calls have been moved to a separate contract, UniswapUtils.

Update
Marked as "Fixed" by the client. Addressed in: `cacc8ee5c1dd1c2131fa3a57c30d4cfce39a3e9f`. The client provided the following explanation:

- We have upgraded our contracts from 0.8.16 to 0.8.19
- As part of this update, we have added a wrapper contract named UniswapUtility. This contract exposes all the Uniswap library functions used by the protocol and is consumed via an interface, addressing the versioning limitation (< 0.8.0) of UniV3 libs. This update ensures consistency in the solc version across the repository. https://arbiscan.io/address/0xd2Aa19D3B7f8cdb1ea5B782c5647542055af415e#readContract

File(s) affected: `All contracts`

Description: The in-scope contracts are compiled using Solidity version 0.8.16, which contains known compiler bugs, according to the Solidity official's compiler bug list. Specifically, this version of the compiler has one bug labeled as `medium/high` severity and three bugs labeled as `low` severity. Our examination shows that these bugs are unlikely to affect the code in scope.

Recommendation: Consider updating the Solidity version to the latest or a more recent version to avoid the code from potentially being affected by the compiler bugs. Note that the EVM version needs to be adjusted if a Solidity version >= 0.8.20 is used so that the bytecode does not include `PUSH0` opcodes, which is not yet supported on Aribtrum.

SPE-31 Slippage Restrictions Cannot Be Updated

• Informational ⓘ Acknowledged

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

- It's an assumption that strategies like Aave and Compound always return the exact amount when liquidating the LP; however, strategies involving paired tokens (e.g., Curve/Uniswap) are prone to slippages.
- We retain this function for maintaining a consistent interface.

File(s) affected: `strategies/aave/AaveStrategy.sol`, `strategies/compound/CompoundStrategy.sol`

Description: The `InitializableAbstractStrategy` has setters for the `depositSlippage` and `withdrawSlippage`. The `AaveStrategy` and `CompoundStrategy` both initialize these parameters to `0`. However, these parameters are unused in the implementations. If these parameters are later updated to non-zero values, they will have no effect.

Recommendation: Consider implementing slippage checks in `AaveStrategy` and `CompoundStrategy`.

SPE-32
Application Monitoring Can Be Improved by Emitting More Events

● Informational ⓘ Fixed

Update
Quantstamp: The Sperax team implemented log emits for `_rebaseOptOut()` and `_rebaseOptIn()` functions.

Update
Marked as "Fixed" by the client. Addressed in: `cd22be541d293f969e2ba7c0302699e65ad28dd0`. The client provided the following explanation:

> Implemented the recommended changes.

File(s) affected: `token/USDS.sol`

Description: It is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management:

- `_rebaseOptOut()`
- `_rebaseOptIn()`

Recommendation: Consider emitting the events.

Definitions

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

Automated Analysis

N/A

Test Suite Results

All tests are currently passing.

Command to execute test: `forge test`

```
Running 4 tests for test/buyback/SPABuyback.t.sol:TestInit
[PASS] testCannotInitializeImplementation() (gas: 38257)
[PASS] testCannotInitializeTwice() (gas: 45363)
[PASS] testInit() (gas: 20480)
[PASS] testInitialize() (gas: 1767200)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 4.81s

Running 8 tests for test/buyback/SPABuyback.t.sol:TestSetters
```

```
[PASS] testCannotIfCallerNotOwner() (gas: 71201)
[PASS] testCannotIfInvalidAddress() (gas: 43266)
[PASS] testCannotIfInvalidAddressOracle() (gas: 43224)
[PASS] testCannotIfPercentageIsZero() (gas: 43103)
[PASS] testCannotIfPercentageMoreThanMax() (gas: 43365)
[PASS] testUpdateOracle() (gas: 56218)
[PASS] testUpdateRewardPercentage() (gas: 51717)
[PASS] testUpdateVeSpaRewarder() (gas: 56337)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 419.61ms

Running 2 tests for test/strategy/CompoundStrategy.t.sol:InitializeTests
[PASS] test_initialization() (gas: 175528)
[PASS] test_invalid_address() (gas: 97732)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 5.44s

Running 4 tests for test/strategy/AaveStrategy.t.sol:SetPToken
[PASS] test_RevertWhen_DuplicateAsset() (gas: 129069)
[PASS] test_RevertWhen_InvalidPToken() (gas: 129334)
[PASS] test_RevertWhen_NotOwner() (gas: 53532)
[PASS] test_SetPTokenAddress() (gas: 133626)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 7.61s

Running 4 tests for test/buyback/SPABuyback.t.sol:TestWithdraw
[PASS] testCannotIfCallerNotOwner() (gas: 55652)
[PASS] testCannotWithdrawMoreThanBalance() (gas: 78258)
[PASS] testCannotWithdrawSPA() (gas: 54988)
[PASS] testWithdraw() (gas: 145603)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 3.62s

Running 4 tests for test/strategy/CompoundStrategy.t.sol:RemovePTokenTest
[PASS] test_RemovePToken() (gas: 73980)
[PASS] test_RevertWhen_CollateralAllocated() (gas: 397391)
[PASS] test_RevertWhen_InvalidId() (gas: 45343)
[PASS] test_RevertWhen_NotOwner() (gas: 49154)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 3.52s

Running 6 tests for test/strategy/CompoundStrategy.t.sol:WithdrawTest
[PASS] test_RevertWhen_CallerNotVault() (gas: 54470)
[PASS] test_RevertWhen_InvalidAddress() (gas: 50683)
[PASS] test_RevertWhen_Withdraw0() (gas: 60653)
[PASS] test_Withdraw() (gas: 148631)
[PASS] test_WithdrawToVault() (gas: 152619)
[PASS] test_WithdrawToVault_RevertsIf_CallerNotOwner() (gas: 90658)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 8.96s

Running 4 tests for test/strategy/CompoundStrategy.t.sol:SetPTokenTest
[PASS] test_RevertWhen_DuplicateAsset() (gas: 130202)
[PASS] test_RevertWhen_InvalidPToken() (gas: 125732)
[PASS] test_RevertWhen_NotOwner() (gas: 53564)
[PASS] test_SetPTokenAddress() (gas: 134763)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 21.55ms

Running 2 tests for test/strategy/AaveStrategy.t.sol:InitializeTests
[PASS] test_empty_address() (gas: 171624)
[PASS] test_success() (gas: 176190)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 9.72ms

Running 1 test for test/oracle/SPAOracle.t.sol:Test_FetchPrice
[PASS] test_fetchPrice() (gas: 113436)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.60s

Running 1 test for test/oracle/SPAOracle.t.sol:Test_Init
[PASS] test_initialization() (gas: 22031)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 24.63ms

Running 3 tests for test/oracle/SPAOracle.t.sol:Test_UpdateDIAWeight
[PASS] test_revertsWhen_invalidWeight() (gas: 37000)
[PASS] test_revertsWhen_notOwner() (gas: 10995)
[PASS] test_updateDIAParams() (gas: 48898)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 10.32ms

Running 3 tests for test/oracle/SPAOracle.t.sol:Test_setUniMAPriceData
```

```
[PASS] test_revertsWhen_invalidData() (gas: 44301)
[PASS] test_revertsWhen_notOwner() (gas: 13307)
[PASS] test_setUniMAPriceData() (gas: 77991)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 226.89ms

Running 4 tests for test/oracle/SPAOracle.t.sol:Test_updateMasterOracle
[PASS] test_revertsWhen_invalidAddress() (gas: 36055)
[PASS] test_revertsWhen_notOwner() (gas: 13020)
[PASS] test_revertsWhen_quoteTokenPriceFeedUnavailable() (gas: 52627)
[PASS] test_updateMasterOracle() (gas: 50491)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 29.77ms

Running 3 tests for test/strategy/StargateStrategy.t.sol:ChangeSlippage
[PASS] test_RevertWhen_NotOwner() (gas: 20138)
[PASS] test_RevertWhen_slippageExceedsMax() (gas: 43470)
[PASS] test_UpdateSlippage() (gas: 56643)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 9.18ms

Running 2 tests for test/rebase/Dripper.t.sol:Collect
[PASS] test_CollectDripper() (gas: 205883)
[PASS] test_CollectZeroBalance() (gas: 67282)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 2.50s

Running 2 tests for test/strategy/CompoundStrategy.t.sol:CollectRewardTest
[PASS] test_CheckRewardEarned() (gas: 425952)
[PASS] test_collectReward() (gas: 606815)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 13.32s

Running 3 tests for test/strategy/CompoundStrategy.t.sol:DepositTest
[PASS] test_Deposit() (gas: 376796)
[PASS] test_RevertWhen_InvalidAmount() (gas: 48395)
[PASS] test_deposit_Collateral_not_supported() (gas: 51996)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 32.56ms

Running 4 tests for test/strategy/AaveStrategy.t.sol:RemovePToken
[PASS] test_RemovePToken() (gas: 69884)
[PASS] test_RevertWhen_CollateralAllocated() (gas: 483380)
[PASS] test_RevertWhen_InvalidId() (gas: 45321)
[PASS] test_RevertWhen_NotOwner() (gas: 49132)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 939.53ms

Running 3 tests for test/rebase/Dripper.t.sol:SetDripDuration
[PASS] test_RevertWhen_CallerIsNotOwner(uint256) (runs: 256, μ: 42483, ~: 42483)
[PASS] test_RevertWhen_InvalidInput(uint256) (runs: 256, μ: 36489, ~: 36489)
[PASS] test_UpdateDripDuration(uint256) (runs: 256, μ: 43285, ~: 43285)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 148.07ms

Running 3 tests for test/rebase/Dripper.t.sol:UpdateVault
[PASS] test_RevertWhen_CallerIsNotOwner() (gas: 42083)
[PASS] test_RevertWhen_VaultIsZeroAddress() (gas: 36084)
[PASS] test_UpdateVault() (gas: 42864)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 8.90ms

Running 6 tests for test/strategy/AaveStrategy.t.sol:MiscellaneousTest
[PASS] test_CheckAvailableBalance() (gas: 445202)
[PASS] test_CheckAvailableBalance_InsufficientTokens() (gas: 477789)
[PASS] test_CheckBalance() (gas: 18848)
[PASS] test_CheckInterestEarned_Empty() (gas: 45572)
[PASS] test_CheckRewardEarned() (gas: 12652)
[PASS] test_CollectReward() (gas: 15591)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 5.46s

Running 3 tests for test/strategy/AaveStrategy.t.sol:Deposit
[PASS] test_Deposit() (gas: 462659)
[PASS] test_RevertWhen_InvalidAmount() (gas: 50605)
[PASS] test_deposit_Collateral_not_supported() (gas: 48578)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 11.08ms

Running 1 test for test/strategy/AaveStrategy.t.sol:CollectInterest
[PASS] test_CollectInterest() (gas: 308388)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 14.83s
```

```
Running 5 tests for test/strategy/AaveStrategy.t.sol:WithdrawTest
[PASS] test_RevertWhen_CallerNotVault() (gas: 54470)
[PASS] test_RevertWhen_InvalidAddress() (gas: 50667)
[PASS] test_RevertWhen_Withdraw0() (gas: 60662)
[PASS] test_Withdraw() (gas: 250561)
[PASS] test_WithdrawToVault() (gas: 254571)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 7.22s

Running 4 tests for test/oracle/ChainlinkOracle.t.sol:Test_GetTokenPrice
[PASS] test_getTokenPrice() (gas: 47288)
[PASS] test_revertsWhen_gracePeriodNotPassed() (gas: 34709)
[PASS] test_revertsWhen_sequencerDown() (gas: 33729)
[PASS] test_revertsWhen_unSupportedCollateral() (gas: 15668)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 20.82ms

Running 2 tests for test/oracle/ChainlinkOracle.t.sol:Test_SetTokenData
[PASS] test_revertsWhen_notOwner() (gas: 12623)
[PASS] test_setTokenData() (gas: 86903)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 8.78ms

Running 8 tests for test/token/USDs.t.sol:TestTransfer
[PASS] test_change_vault() (gas: 52861)
[PASS] test_creditsBalanceOf() (gas: 47249)
[PASS] test_revert_balance() (gas: 53340)
[PASS] test_revert_invalid_input() (gas: 50100)
[PASS] test_transfer(uint256) (runs: 256, μ: 104529, ~: 104426)
[PASS] test_transfer_sender_non_rebasing_from() (gas: 134418)
[PASS] test_transfer_sender_non_rebasing_to_and_from_v1() (gas: 177411)
[PASS] test_transfer_sender_non_rebasing_to_and_from_v2() (gas: 180699)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 1.33s

Running 9 tests for test/token/USDs.t.sol:TestTransferFrom
[PASS] test_allowance() (gas: 45033)
[PASS] test_attack_1() (gas: 48790)
[PASS] test_change_vault() (gas: 52883)
[PASS] test_decreaseAllowance() (gas: 68199)
[PASS] test_increaseAllowance() (gas: 74071)
[PASS] test_revert_balance() (gas: 55649)
[PASS] test_revert_invalid_input() (gas: 50297)
[PASS] test_transfer_from(uint256) (runs: 256, μ: 114555, ~: 114421)
[PASS] test_transfer_from_without_approval() (gas: 57911)
Test result: ok. 9 passed; 0 failed; 0 skipped; finished in 282.86ms

Running 1 test for test/token/USDs.t.sol:USDsTest
[PASS] test_change_vault() (gas: 52805)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 10.99ms

Running 1 test for test/token/USDs.t.sol:USDsUpgradabilityTest
[PASS] test_data() (gas: 1853407)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 10.28ms

Running 1 test for test/oracle/USDsOracle.t.sol:Test_FetchPrice
[PASS] test_fetchPrice() (gas: 93918)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 850.53ms

Running 1 test for test/oracle/USDsOracle.t.sol:Test_Init
[PASS] test_initialization() (gas: 18937)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 31.12ms

Running 3 tests for test/rebase/Dripper.t.sol:RecoverTokens
[PASS] test_RecoverTokens(uint128) (runs: 256, μ: 1514900, ~: 1514904)
[PASS] test_RevertWhen_CallerIsNotOwner() (gas: 42002)
[PASS] test_RevertWhen_NothingToRecover() (gas: 48688)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 6.63s

Running 4 tests for test/vault/FeeCalculator.t.sol:TestCalibrateFee
[PASS] test_CalibrateFee_TotalCollateralGTUpperLimit() (gas: 75505)
[PASS] test_CalibrateFee_TotalCollateralIsInDesiredRange() (gas: 74968)
[PASS] test_CalibrateFee_TotalCollateralLTLowerLimit() (gas: 76042)
[PASS] test_revertsIf_InvalidCalibration() (gas: 13202)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 18.38s
```

```
Running 2 tests for test/vault/FeeCalculator.t.sol:TestFeeCalculator
[PASS] testGetMintFee() (gas: 125792)
[PASS] testGetRedeemFee() (gas: 125851)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 18.47ms

Running 1 test for test/vault/FeeCalculator.t.sol:TestFeeCalculatorInit
[PASS] testInitialization() (gas: 947139)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 235.51ms

Running 8 tests for test/oracle/MasterPriceOracle.t.sol:MasterPriceOracleTest
[PASS] test_getPriceFeed() (gas: 112922)
[PASS] test_removeTokenPriceFeed() (gas: 175656)
[PASS] test_revertsWhen_feedNotFetched() (gas: 40460)
[PASS] test_revertsWhen_invalidPriceFeed() (gas: 39527)
[PASS] test_revertsWhen_removingNonExistingFeed() (gas: 40616)
[PASS] test_revertsWhen_unAuthorizedRemoveRequest() (gas: 44230)
[PASS] test_revertsWhen_unAuthorizedUpdate() (gas: 45099)
[PASS] test_updateTokenPriceFeed() (gas: 818806)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 17.48ms

Running 1 test for test/vault/VaultIntegration.t.sol:TestInit
[PASS] test_Initialization() (gas: 2331063)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 621.36ms

Running 4 tests for test/vault/CollateralManager.t.sol:CollateralManager_updateCollateral_Test
[PASS] test_revertsWhen_collateralCompositionExceeded(uint16,uint16,uint16,uint16,uint16) (runs: 256, μ:
158059, ~: 158409)
[PASS] test_revertsWhen_updateNonExistingCollateral(uint16,uint16,uint16,uint16) (runs: 256, μ: 41605, ~:
41605)
[PASS] test_updateCollateral(uint16,uint16,uint16,uint16,uint16) (runs: 256, μ: 168735, ~: 169020)
[PASS] test_updateMultipleCollaterals(uint16,uint16,uint16,uint16,uint16) (runs: 256, μ: 570580, ~:
571051)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 19.18s

Running 7 tests for test/vault/CollateralManager.t.sol:CollateralManager_AddCollateral_Test
[PASS] test_addCollateral(uint16,uint16,uint16,uint16) (runs: 256, μ: 160562, ~: 160902)
[PASS] test_addMultipleCollaterals(uint16,uint16,uint16,uint16) (runs: 256, μ: 520497, ~: 520968)
[PASS] test_revertsWhen_addSameCollateral(uint16,uint16,uint16,uint16) (runs: 256, μ: 156281, ~: 156621)
[PASS] test_revertsWhen_baseMintFeeExceedsMax(uint16,uint16,uint16,uint16) (runs: 256, μ: 41577, ~:
41577)
[PASS] test_revertsWhen_baseRedeemFeeExceedsMax(uint16,uint16,uint16,uint16) (runs: 256, μ: 41711, ~:
41711)
[PASS] test_revertsWhen_collateralCompositionExceeded(uint16,uint16,uint16) (runs: 256, μ: 159327, ~:
159327)
[PASS] test_revertsWhen_downsidePegExceedsMax(uint16,uint16,uint16,uint16) (runs: 256, μ: 41563, ~:
41563)
Test result: ok. 7 passed; 0 failed; 0 skipped; finished in 4.33s

Running 3 tests for test/rebase/RebaseManager.t.sol:UpdateAPR
[PASS] test_RevertWhen_CallerIsNotOwner(uint256,uint256) (runs: 256, μ: 42575, ~: 42575)
[PASS] test_RevertWhen_InvalidConfig(uint256,uint256) (runs: 256, μ: 36790, ~: 36790)
[PASS] test_UpdateAPR(uint256,uint256) (runs: 256, μ: 48555, ~: 48855)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 96.52ms

Running 3 tests for test/rebase/RebaseManager.t.sol:UpdateDripper
[PASS] test_RevertWhen_CallerIsNotOwner() (gas: 42041)
[PASS] test_RevertWhen_DripperIsZeroAddress() (gas: 36091)
[PASS] test_UpdateDripper() (gas: 42932)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 16.74ms

Running 3 tests for test/rebase/RebaseManager.t.sol:UpdateGap
[PASS] test_RevertWhen_CallerIsNotOwner() (gas: 41958)
[PASS] test_UpdateGap(uint256) (runs: 256, μ: 43156, ~: 43156)
[PASS] test_UpdateGap_Zero() (gas: 37929)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 34.03ms

Running 3 tests for test/rebase/RebaseManager.t.sol:UpdateVault
[PASS] test_RevertWhen_CallerIsNotOwner() (gas: 42050)
[PASS] test_RevertWhen_VaultIsZeroAddress() (gas: 36099)
[PASS] test_UpdateVault() (gas: 42898)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 16.41ms
```

```
Running 2 tests for test/vault/VaultCore.t.sol:TestRebase
[PASS] test_Rebase() (gas: 270849)
[PASS] test_Rebase0Amount() (gas: 63569)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 4.97s

Running 2 tests for test/rebase/RebaseManager.t.sol:FetchRebaseAmt
[PASS] test_FetchRebaseAmt_Scenario() (gas: 597180)
[PASS] test_RevertWhen_CallerIsNotOwner() (gas: 42742)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 938.66ms

Running 3 tests for test/vault/CollateralManager.t.sol:CollateralManager_removeCollateral_Test
[PASS] test_removeMultipleCollaterals(uint16,uint16,uint16) (runs: 256, μ: 762483, ~: 762484)
[PASS] test_revertsWhen_removeNonExistingCollateral() (gas: 38347)
[PASS] test_revertsWhen_removeStrategyCollateralStrategyExists(uint16,uint16,uint16,uint16) (runs: 256,
μ: 256933, ~: 259328)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 4.85s

Running 1 test for test/vault/VaultIntegration.t.sol:TestMint
[PASS] test_Mint() (gas: 469600)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 982.22ms

Running 1 test for test/vault/VaultIntegration.t.sol:TestRebase
[PASS] test_Rebase() (gas: 296423)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 17.52ms

Running 6 tests for test/vault/CollateralManager.t.sol:CollateralManager_addCollateralStrategy_Test
[PASS] test_addCollateralStrategy(uint16,uint16,uint16,uint16,uint16) (runs: 256, μ: 257091, ~: 259374)
[PASS] test_addMultipleCollateralStrategies(uint16,uint16,uint16,uint16,uint16) (runs: 256, μ: 329919, ~:
332978)
[PASS] test_revertsWhen_addCollateralstrategyAllocationPerExceeded(uint16,uint16,uint16,uint16,uint16)
(runs: 256, μ: 171935, ~: 172340)
[PASS] test_revertsWhen_addCollateralstrategyNotSupported(uint16,uint16,uint16,uint16) (runs: 256, μ:
163009, ~: 163425)
[PASS] test_revertsWhen_addCollateralstrategyWhenAlreadyMapped(uint16,uint16,uint16,uint16) (runs: 256,
μ: 258152, ~: 259749)
[PASS] test_revertsWhen_collateralDoesntExist(uint16) (runs: 256, μ: 39161, ~: 39161)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 20.89s

Running 5 tests for test/vault/CollateralManager.t.sol:CollateralManager_removeCollateralStrategy_Test
[PASS] test_removeCollateralStrategy(uint16,uint16,uint16,uint16) (runs: 256, μ: 374517, ~: 374878)
[PASS] test_revertsWhen_DefaultStrategy(uint16,uint16,uint16,uint16) (runs: 256, μ: 261338, ~: 261601)
[PASS] test_revertsWhen_DefaultStrategyNotExist(uint16,uint16,uint16,uint16) (runs: 256, μ: 261164, ~:
261569)
[PASS] test_revertsWhen_strategyInUse(uint16,uint16,uint16,uint16) (runs: 256, μ: 285235, ~: 285498)
[PASS] test_revertsWhen_strategyNotMapped(uint16,uint16,uint16,uint16) (runs: 256, μ: 157193, ~: 157489)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 20.90s

Running 4 tests for test/vault/CollateralManager.t.sol:CollateralManager_mintRedeemParams_test
[PASS] test_getMintParams(uint16,uint16,uint16,uint16) (runs: 256, μ: 322174, ~: 322174)
[PASS] test_getRedeemParams(uint16,uint16,uint16,uint16) (runs: 256, μ: 406639, ~: 406639)
[PASS] test_revertsWhen_getMintParams_collateralDoesntExist() (gas: 41484)
[PASS] test_revertsWhen_getRedeemParams_collateralDoesntExist() (gas: 36782)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 274.44ms

Running 3 tests for test/buyback/SPABuyback.t.sol:TestGetters
[PASS] testCannotIfInvalidAmount() (gas: 25676)
[PASS] testGetSpaReqdForUSDs() (gas: 121389)
[PASS] testGetUsdsOutForSpa() (gas: 121400)
Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 9.36ms

Running 5 tests for test/vault/CollateralManager.t.sol:CollateralManager_updateCollateralStrategy_Test
[PASS] test_revertsWhen_updateCollateralstrategyAllocationNotValid(uint16,uint16,uint16,uint16) (runs:
256, μ: 297493, ~: 297865)
[PASS] test_revertsWhen_updateCollateralstrategyAllocationPerExceeded(uint16,uint16,uint16,uint16) (runs:
256, μ: 260410, ~: 260760)
[PASS] test_revertsWhen_updateCollateralstrategyWhenNotMapped(uint16,uint16,uint16,uint16) (runs: 256, μ:
157207, ~: 157568)
[PASS] test_updateCollateralStrategy(uint16,uint16,uint16,uint16,uint16) (runs: 256, μ: 462266, ~:
465680)
[PASS] test_updateMultipleCollateralStrategies(uint16,uint16,uint16,uint16) (runs: 256, μ: 331988, ~:
332382)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 1.85s
```

```
Running 1 test for test/vault/VaultIntegration.t.sol:TestRedeem
[PASS] test_RedeemFromDefaultStrategy() (gas: 930185)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.89s

Running 5 tests for test/buyback/SPABuyback.t.sol:TestBuyUSDs
[PASS] testBuyUSDs() (gas: 542112)
[PASS] testBuyUSDs(uint256,uint256,uint256) (runs: 256, μ: 471638, ~: 489131)
[PASS] testCannotIfInsufficientUSDsBalance() (gas: 60939)
[PASS] testCannotIfSlippageMoreThanExpected() (gas: 45787)
[PASS] testCannotIfSpaAmountTooLow() (gas: 42424)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 2.52s

Running 8 tests for test/vault/VaultCore.t.sol:TestSetters
[PASS] test_revertIf_InvalidAddress() (gas: 98217)
[PASS] test_revertIf_callerIsNotOwner() (gas: 115388)
[PASS] test_updateCollateralManager() (gas: 54255)
[PASS] test_updateFeeCalculator() (gas: 54230)
[PASS] test_updateFeeVault() (gas: 54299)
[PASS] test_updateOracle() (gas: 54156)
[PASS] test_updateRebaseManager() (gas: 54276)
[PASS] test_updateYieldReceiver() (gas: 71488)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 17.24ms

Running 1 test for test/vault/VaultIntegration.t.sol:TestAllocate
[PASS] testFuzz_Allocate(uint256) (runs: 256, μ: 285861, ~: 285893)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 277.85ms

Running 2 tests for test/strategy/CompoundStrategy.t.sol:CheckAvailableBalanceTest
[PASS] test_checkAvailableBalance_MoreThanAllocated() (gas: 37176)
[PASS] test_checkAvilableBalance_LTAllocatedAmount() (gas: 25918)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 10.87ms

Running 24 tests for test/buyback/YieldReserve.t.sol:YieldReserveTest
[PASS] test_getTokenBForTokenA() (gas: 108722)
[PASS] test_getTokenBForTokenA_inputs() (gas: 116337)
[PASS] test_toggleDstTokenPermission() (gas: 73898)
[PASS] test_toggleDstTokenPermission_auth_error() (gas: 44346)
[PASS] test_toggleSrcTokenPermission() (gas: 71702)
[PASS] test_toggleSrcTokenPermission_auth_error() (gas: 44414)
[PASS] test_updateBuybackAddress() (gas: 44831)
[PASS] test_updateBuybackAddress_auth_error() (gas: 44292)
[PASS] test_updateBuybackAddress_inputs() (gas: 36251)
[PASS] test_updateBuybackPercentage() (gas: 42265)
[PASS] test_updateBuybackPercentage_auth_error() (gas: 42083)
[PASS] test_updateBuybackPercentage_inputs() (gas: 40082)
[PASS] test_updateDripperAddress() (gas: 44789)
[PASS] test_updateDripperAddress_auth_error() (gas: 44292)
[PASS] test_updateDripperAddress_inputs() (gas: 36206)
[PASS] test_updateOracleAddress() (gas: 44721)
[PASS] test_updateOracleAddress_auth_error() (gas: 44246)
[PASS] test_updateOracleAddress_inputs() (gas: 36139)
[PASS] test_updateVaultAddress() (gas: 44743)
[PASS] test_updateVaultAddress_auth_error() (gas: 44159)
[PASS] test_updateVaultAddress_inputs() (gas: 36118)
[PASS] test_withdraw() (gas: 204057)
[PASS] test_withdraw_auth_error() (gas: 46604)
[PASS] test_withdraw_inputs() (gas: 54392)
Test result: ok. 24 passed; 0 failed; 0 skipped; finished in 595.63ms

Running 1 test for test/strategy/CompoundStrategy.t.sol:CheckRewardEarnedTest
[PASS] test_CheckRewardEarned() (gas: 425765)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 11.93ms

Running 1 test for test/vault/VaultCore.t.sol:TestInit
[PASS] test_Initialization() (gas: 2331063)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 17.48ms

Running 5 tests for test/vault/VaultCore.t.sol:TestMint
[PASS] testFuzz_RevertsIf_DeadlinePassed(uint256) (runs: 256, μ: 62485, ~: 62835)
[PASS] test_Mint() (gas: 458199)
[PASS] test_MintBySpecifyingCollateralAmt() (gas: 456232)
```

```
[PASS] test_RevertsIf_MintFailed() (gas: 135633)
[PASS] test_RevertsIf_SlippageScrewsYou() (gas: 176328)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 46.73ms

Running 4 tests for test/vault/VaultCore.t.sol:TestMintView
[PASS] test_Fee0If_CallerHasFacilitatorRole() (gas: 121535)
[PASS] test_MintView() (gas: 213524)
[PASS] test_MintView_Returns0When_MintIsNotAllowed() (gas: 113453)
[PASS] test_MintView_Returns0When_PriceLowerThanDownsidePeg() (gas: 61084)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 18.33ms

Running 6 tests for test/strategy/StargateStrategy.t.sol:SetPToken
[PASS] test_RevertWhen_DuplicateAsset() (gas: 176573)
[PASS] test_RevertWhen_InvalidPToken() (gas: 61038)
[PASS] test_RevertWhen_InvalidPid() (gas: 59578)
[PASS] test_RevertWhen_InvalidRewardPid() (gas: 75860)
[PASS] test_RevertWhen_NotOwner() (gas: 29748)
[PASS] test_SetPTokenAddress() (gas: 314742)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 725.98ms

Running 6 tests for test/vault/CollateralManager.t.sol:CollateralManager_validateAllocation_test
[PASS] test_getAllCollaterals(uint16,uint16,uint16,uint16) (runs: 256, μ: 604955, ~: 604955)
[PASS] test_getCollateralStrategies(uint16,uint16,uint16,uint16) (runs: 256, μ: 327516, ~: 327801)
[PASS] test_getZeroCollaterals() (gas: 35949)
[PASS] test_revertsWhen_validateAllocationNotAllowed(uint16,uint16,uint16,uint16) (runs: 256, μ: 268181,
~: 268181)
[PASS] test_validateAllocation(uint16,uint16,uint16,uint16) (runs: 256, μ: 295850, ~: 296190)
[PASS] test_validateAllocationMaxCollateralUsageSup(uint16,uint16,uint16,uint16) (runs: 256, μ: 295908,
~: 296291)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 3.34s

Running 4 tests for test/strategy/StargateStrategy.t.sol:InitializationTest
[PASS] test_InvalidInitialization() (gas: 64442)
[PASS] test_UpdateHarvestIncentiveRate() (gas: 233636)
[PASS] test_UpdateVaultCore() (gas: 231169)
[PASS] test_ValidInitialization() (gas: 246500)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 9.49ms

Running 4 tests for test/strategy/StargateStrategy.t.sol:RemovePToken
[PASS] test_RemovePToken() (gas: 77701)
[PASS] test_RevertWhen_CollateralAllocated() (gas: 238866)
[PASS] test_RevertWhen_InvalidId() (gas: 47386)
[PASS] test_RevertWhen_NotOwner() (gas: 17785)
Test result: ok. 4 passed; 0 failed; 0 skipped; finished in 10.40ms

Running 1 test for test/strategy/CompoundStrategy.t.sol:CollectInterestTest
[PASS] test_CollectInterest() (gas: 214017)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 358.77ms

Running 6 tests for test/token/USDs.t.sol:TestMint
[PASS] test_change_vault() (gas: 52883)
[PASS] test_max_supply() (gas: 114882)
[PASS] test_mint() (gas: 115772)
[PASS] test_mint_owner_check() (gas: 54348)
[PASS] test_mint_paused() (gas: 134206)
[PASS] test_mint_to_the_zero() (gas: 52733)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 9.93ms

Running 10 tests for test/token/USDs.t.sol:TestRebase
[PASS] test_change_vault() (gas: 52838)
[PASS] test_pauseSwitch() (gas: 67508)
[PASS] test_rebase() (gas: 106519)
[PASS] test_rebaseOptIn() (gas: 74856)
[PASS] test_rebaseOptOut() (gas: 98210)
[PASS] test_rebase_no_supply_change() (gas: 59349)
[PASS] test_rebase_opt_in() (gas: 107128)
[PASS] test_rebase_opt_out() (gas: 112060)
[PASS] test_revertIf_IsAlreadyNonRebasingAccount() (gas: 68437)
[PASS] test_revertIf_IsAlreadyRebasingAccount() (gas: 76122)
Test result: ok. 10 passed; 0 failed; 0 skipped; finished in 10.40ms

Running 6 tests for test/token/USDs.t.sol:TestBurn
```

```
[PASS] test_burn() (gas: 78733)
[PASS] test_burn_case2() (gas: 122755)
[PASS] test_burn_case3() (gas: 91550)
[PASS] test_burn_opt_in() (gas: 113203)
[PASS] test_change_vault() (gas: 52816)
[PASS] test_credit_amount_changes_case1() (gas: 104496)
Test result: ok. 6 passed; 0 failed; 0 skipped; finished in 10.67ms

Running 27 tests for test/buyback/YieldReserve.t.sol:SwapTest
[PASS] test_getTokenBForTokenA() (gas: 108700)
[PASS] test_getTokenBForTokenA_inputs() (gas: 116360)
[PASS] test_swap() (gas: 330637)
[PASS] test_swap_non_USDS() (gas: 422750)
[PASS] test_swap_slippage_error() (gas: 150664)
[PASS] test_toggleDstTokenPermission() (gas: 73898)
[PASS] test_toggleDstTokenPermission_auth_error() (gas: 44434)
[PASS] test_toggleSrcTokenPermission() (gas: 71702)
[PASS] test_toggleSrcTokenPermission_auth_error() (gas: 44414)
[PASS] test_updateBuybackAddress() (gas: 44809)
[PASS] test_updateBuybackAddress_auth_error() (gas: 44292)
[PASS] test_updateBuybackAddress_inputs() (gas: 36251)
[PASS] test_updateBuybackPercentage() (gas: 42265)
[PASS] test_updateBuybackPercentage_auth_error() (gas: 42061)
[PASS] test_updateBuybackPercentage_inputs() (gas: 40060)
[PASS] test_updateDripperAddress() (gas: 44834)
[PASS] test_updateDripperAddress_auth_error() (gas: 44315)
[PASS] test_updateDripperAddress_inputs() (gas: 36206)
[PASS] test_updateOracleAddress() (gas: 44744)
[PASS] test_updateOracleAddress_auth_error() (gas: 44224)
[PASS] test_updateOracleAddress_inputs() (gas: 36139)
[PASS] test_updateVaultAddress() (gas: 44721)
[PASS] test_updateVaultAddress_auth_error() (gas: 44204)
[PASS] test_updateVaultAddress_inputs() (gas: 36096)
[PASS] test_withdraw() (gas: 204134)
[PASS] test_withdraw_auth_error() (gas: 46604)
[PASS] test_withdraw_inputs() (gas: 54392)
Test result: ok. 27 passed; 0 failed; 0 skipped; finished in 1.91s

Running 1 test for test/strategy/StargateStrategy.t.sol:EdgeCases
[PASS] test_Balance_nLoss() (gas: 60723)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.05s

Running 12 tests for test/vault/VaultCore.t.sol:TestRedeemView
[PASS] test_RedeemViewApplyDownsidePeg() (gas: 226646)
[PASS] test_RedeemViewFee0AndCollAmtDownsidePegged() (gas: 217340)
[PASS] test_RedeemViewFee0AndCollAmtNotDownsidePegged() (gas: 217487)
[PASS] test_RedeemViewFee0IfCallerIsFacilitator() (gas: 259120)
[PASS] test_RedeemViewWithoutDownsidePeg() (gas: 226340)
[PASS] test_RedeemView_FromDefaultStrategy() (gas: 718328)
[PASS] test_RedeemView_FromOtherStrategy() (gas: 703419)
[PASS] test_RedeemView_RevertsIf_InsufficientCollateral() (gas: 169931)
[PASS] test_RedeemView_RevertsIf_InvalidStrategy() (gas: 121916)
[PASS] test_RedeemView_WhenDefaultStrategySetButBalanceIsNotAvailable() (gas: 319612)
[PASS] test_RevertsIf_CollateralAmtMoreThanVaultAmtAndDefaultStrategyNotSet() (gas: 323628)
[PASS] test_RevertsIf_RedeemNotAllowed() (gas: 53649)
Test result: ok. 12 passed; 0 failed; 0 skipped; finished in 4.13s

Running 2 tests for test/token/USDs.t.sol:TestEnsureRebasingMigration
[PASS] test_change_vault() (gas: 52783)
[PASS] test_nocode_to_code() (gas: 1964987)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 1.18s

Running 8 tests for test/strategy/StargateStrategy.t.sol:Withdraw
[PASS] test_RevertWhen_CallerNotVault() (gas: 56809)
[PASS] test_RevertWhen_EnoughFundsNotAvailable() (gas: 344760)
[PASS] test_RevertWhen_InsufficientRwdInFarm() (gas: 183928)
[PASS] test_RevertWhen_SlippageCheckFails() (gas: 276672)
[PASS] test_RevertWhen_Withdraw0() (gas: 62826)
[PASS] test_Withdraw() (gas: 397126)
[PASS] test_WithdrawToVault() (gas: 401170)
[PASS] test_withdraw_InvalidAddress() (gas: 52947)
Test result: ok. 8 passed; 0 failed; 0 skipped; finished in 2.55s
```

```
Running 4 tests for test/vault/VaultCore.t.sol:TestAllocate
[PASS] testFuzz_Allocate(uint256) (runs: 256, μ: 287233, ~: 287262)
[PASS] test_Allocate() (gas: 563972)
[PASS] test_revertIf_AllocationNotAllowed() (gas: 152497)
[PASS] test_revertIf_CollateralAllocationPaused() (gas: 69668)
Test result: FAILED. 5 passed; 1 failed; 0 skipped; finished in 7.95s

Running 1 test for test/strategy/StargateStrategy.t.sol:CollectReward
[PASS] test_CollectReward(uint16) (runs: 256, μ: 325383, ~: 328268)
Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 4.01s

Running 2 tests for test/strategy/StargateStrategy.t.sol:CollectInterest
[PASS] test_CollectInterest() (gas: 841440)
[PASS] test_RevertWhen_UnsupportedAsset() (gas: 28566)
Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 14.95s

Running 5 tests for test/vault/VaultCore.t.sol:TestRedeem
[PASS] test_RedeemFromDefaultStrategy() (gas: 921181)
[PASS] test_RedeemFromSpecificOtherStrategy() (gas: 906248)
[PASS] test_RedeemFromSpecifiedDefaultStrategy() (gas: 921479)
[PASS] test_RedeemFromVault() (gas: 490371)
[PASS] test_RedeemFromVault_RevertsIf_SlippageMoreThanExpected() (gas: 333674)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 15.28s

Running 5 tests for test/strategy/StargateStrategy.t.sol:Deposit
[PASS] testFuzz_Deposit(uint256) (runs: 256, μ: 1401446, ~: 1527383)
[PASS] test_RevertWhen_DepositSlippageViolated() (gas: 930196)
[PASS] test_RevertWhen_InvalidAmount() (gas: 57727)
[PASS] test_RevertWhen_NotEnoughRwdInFarm() (gas: 1184153)
[PASS] test_RevertWhen_UnsupportedCollateral() (gas: 417867)
Test result: ok. 5 passed; 0 failed; 0 skipped; finished in 12.71s
Ran 83 test suites: 347 tests passed, 0 failed, 0 skipped (350 total tests)
```

Code Coverage

Overall code coverage is high enough, however coverage of the `VaultCore` and `SPAOracle` contracts could be further improved.

File	% Lines	% Statements	% Branches	% Funcs
contracts/buyback/SPABuyback.sol	100.00% (**56**/56)	100.00% (**69**/69)	100.00% (**6**/6)	84.62% (**11**/13)
contracts/buyback/YieldReserve.sol	100.00% (**53**/53)	100.00% (**66**/66)	93.75% (**15**/16)	91.67% (**11**/12)
contracts/libraries/Helpers.sol	100.00% (**6**/6)	100.00% (**13**/13)	100.00% (**12**/12)	100.00% (**6**/6)
contracts/libraries/StableMath.sol	0.00% (**0**/7)	0.00% (**0**/14)	100.00% (**0**/0)	0.00% (**0**/5)
contracts/oracle/BaseUniOracle.sol	100.00% (**20**/20)	100.00% (**25**/25)	87.50% (**7**/8)	100.00% (**5**/5)
contracts/oracle/ChainlinkOracle.sol	100.00% (**12**/12)	100.00% (**18**/18)	100.00% (**6**/6)	100.00% (**2**/2)
contracts/oracle/MasterPriceOracle.sol	100.00% (**17**/17)	100.00% (**23**/23)	100.00% (**8**/8)	100.00% (**5**/5)
contracts/oracle/SPAOracle.sol	100.00% (**15**/15)	100.00% (**23**/23)	66.67% (**4**/6)	100.00% (**3**/3)

File	% Lines	% Statements	% Branches	% Funcs
contracts/oracle/USDsOracle.sol	100.00% (**4/**4)	100.00% (**7/**7)	100.00% (**0/**0)	100.00% (**1/**1)
contracts/oracle/VSTOracle.sol	100.00% (**2/**2)	100.00% (**2/**2)	100.00% (**0/**0)	100.00% (**1/**1)
contracts/rebase/Dripper.sol	100.00% (**21/**21)	100.00% (**28/**28)	100.00% (**4/**4)	100.00% (**5/**5)
contracts/rebase/RebaseManager.sol	100.00% (**28/**28)	100.00% (**44/**44)	100.00% (**4/**4)	85.71% (**6/**7)
contracts/strategies/InitializableAbstractStrategy.sol	100.00% (**43/**43)	100.00% (**47/**47)	100.00% (**6/**6)	100.00% (**7/**7)
contracts/strategies/aave/AaveStrategy.sol	98.08% (**51/**52)	95.77% (**68/**71)	81.25% (**13/**16)	94.12% (**16/**17)
contracts/strategies/compound/CompoundStrategy.sol	98.57% (**69/**70)	98.90% (**90/**91)	85.71% (**12/**14)	94.12% (**16/**17)
contracts/strategies/stargate/StargateStrategy.sol	93.46% (**100/**107)	91.14% (**144/**158)	90.00% (**27/**30)	90.48% (**19/**21)
contracts/token/USDs.sol	98.35% (**119/**121)	97.22% (**140/**144)	96.15% (**50/**52)	92.86% (**26/**28)
contracts/vault/CollateralManager.sol	100.00% (**112/**112)	99.26% (**134/**135)	92.50% (**37/**40)	100.00% (**17/**17)
contracts/vault/FeeCalculator.sol	100.00% (**24/**24)	100.00% (**31/**31)	87.50% (**7/**8)	100.00% (**5/**5)
contracts/vault/VaultCore.sol	93.40% (**99/**106)	93.44% (**114/**122)	65.79% (**25/**38)	100.00% (**19/**19)
Total	97.14% (**851/**876)	96.02% (**1086/**1131)	88.68% (**243/**274)	92.34% (**181/**196)

Changelog

- 2023-11-17 - Initial report
- 2023-12-11 - Final report

About Quantstamp

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

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

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

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

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos

- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

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

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.