Quantstamp Security Assessment Certificate

June 6th 2022 — Quantstamp Verified

Mercurial Finance

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

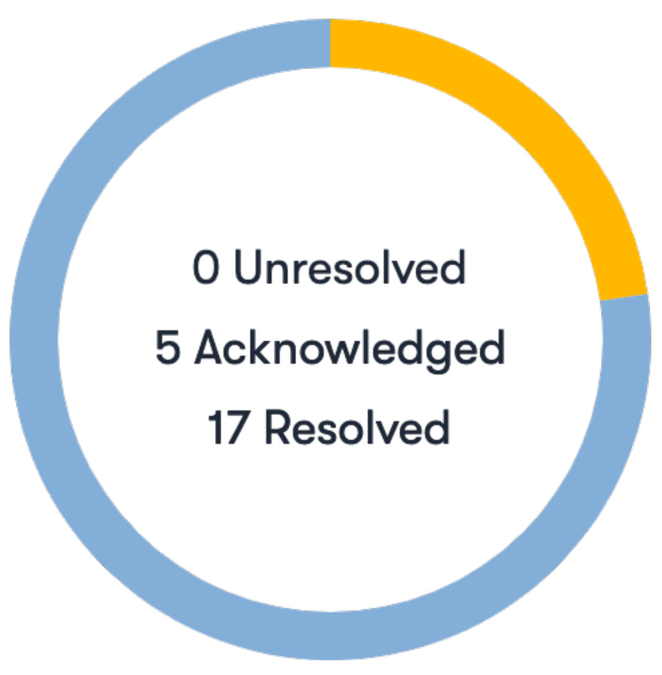
Executive Summary

Type	Lending Aggregator and Yield Generator on Solana
Auditors	Danny Aksenov, Security Auditor Andy Lin, Senior Auditing Engineer Marius Guggenmos, Senior Research Engineer
Timeline	2022-05-09 through 2022-05-20
Languages	Rust
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Mercurial Vault Technical Doc
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
mercurial-v2	3a9ce59
main_audit	0658689

Total Issues	22	(17 Resolved)
High Risk Issues	2	(1 Resolved)
Medium Risk Issues	3	(3 Resolved)
Low Risk Issues	9	(8 Resolved)
Informational Risk Issues	5	(3 Resolved)
Undetermined Risk Issues	3	(2 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Fixed	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

During this audit, Quantstamp found 22 potential issues of various levels of severity: 2 high-severity issues, 3 medium-severity issues, 9 low-severity issues, 3 undetermined-severity issues, as well as 5 informational-severity issues. We also made 15 best practices recommendations.
Overall, we found the code easy to follow, and while the technical document could've been more extensive, it provided adequate context for the auditors. However, since programs in Solana require the users to pass in all the data, the responsibility for strict input validations falls on the developers. The validation in the accounts is not thorough enough, and as a result we found some high and medium severity level issues.
Disclaimer:

1. Readers should be aware that Quantstamp was requested and had audited the files scoped in the audit process, which did not cover all the strategies implemented, other than the ones found in `solend_with_lm.rs` and `solend_without_lm.rs`.
2. This project utilized SPL programs, the Anchor framework, and many more external dependencies. All of these implementations are not part of this audit.

ID	Description	Severity	Status
QSP-1	Using Unaudited Anchor Framework Programs	High	Acknowledged
QSP-2	Vaults can Own Strategies Intended for Other Vaults	Medium	Fixed
QSP-3	Admin/Operator can Steal Funds Through Fake `obligation` and `obligation_owner`	High	Fixed
QSP-4	Incorrect Data Validation in `transfer_fee_vault`	Medium	Fixed
QSP-5	`total_amount` can be Overridden Resulting in a Denial of Service	Medium	Fixed
QSP-6	Transfer of Admin Rights Could Lead to an Inoperable Vault	Low	Fixed
QSP-7	Arithmetic Overflow/Underflow	Low	Fixed
QSP-8	Parameter Discrepancy in Interface	Low	Fixed
QSP-9	Counterfeit `strategy_program`	Low	Mitigated
QSP-10	Tokens can be Locked in the Vault or Strategy	Low	Acknowledged
QSP-11	Able to Provide Invalid VaultBumps to `vault`	Low	Fixed
QSP-12	Rounding Error	Low	Fixed
QSP-13	Rounding Gap on `withdraw_directly_from_strategy`	Low	Fixed
QSP-14	Missing Validations in Solend Strategies	Low	Fixed
QSP-15	Privileged Roles and Ownership	Informational	Acknowledged
QSP-16	No Error for Reaching Max Strategies on `add_strategy`	Informational	Fixed
QSP-17	Consider Enforcing Rewards Collection on Strategy Removal	Informational	Acknowledged
QSP-18	Estimating Amounts in `StrategyWithdraw`	Informational	Fixed
QSP-19	Risk of Setting the Same Strategy Twice to the Same Vault	Informational	Fixed
QSP-20	Profit Lockup	? Undetermined	Fixed
QSP-21	Effectiveness of Sandwich Attack Prevention	? Undetermined	Mitigated
QSP-22	Lack of Protection Against Black Swan Events	? Undetermined	Acknowledged

Quantstamp Audit Breakdown

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

DISCLAIMER:
This report only covers the files scoped in the audit process. Please note that this audit did not cover all the strategies implemented, other than the ones found in solend_with_lm.rs and solend_without_lm.rs.

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

- Missing account-ownership check
- Missing account-signature check
- Missing account-type check
- Missing account-program-id check
- Unsafe external calls
- Integer overflow/ underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversight
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

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

Toolset

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

Setup

Tool Setup:

- Cargo Audit v0.16.0
- Rust-Clippy 0.1.60
- Soteria build 1643895818

Steps taken to run the tools:

1. `cargo install cargo-audit`
2. `cargo audit`
3. `rustup component add clippy`
4. `cargo clippy`
5. `docker run -v ${ProjectDirectory}:/workspace -it greencorelab/soteria:latest /bin/bash`
6. `soteria .`

Findings

QSP-1 Using Unaudited Anchor Framework Programs

Severity: *High Risk*

Status: Acknowledged

Description: Mercurial Finance relies on the Anchor framework for the Solana Sealevel runtime. Anchor developers state in their official documentation that:

- Anchor is in active development, so all APIs are subject to change.
- This code is unaudited. Use at your own risk.

Production code should not rely on unstable, unaudited and thus insecure code.

Recommendation: Users should be notified of the inherent risk that Mercurial is taking by relying on the anchor framework.

QSP-2 Vaults can Own Strategies Intended for Other Vaults

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `programs/vault/src/lib.rs`, `programs/vault/src/context.rs`

Description: The instruction `lib.rs::vault::enable_strategy` will add the strategy to the vault if it does not exist in the `vault.strategies` array. However, it does not verify whether the strategy is correspondent to the vault or not. This can result in potential accounting issues.

Recommendation: We need to ensure only the same vault can use the strategy. The team can add a new field `vault` to the `strategy` and add checks that it is the same as the vault account. Alternatively, from the observation in `context.rs::SetStrategy::strategy`, the `strategy` PDA uses the following seeds: `[vault.key().as_ref(), reserve.key().as_ref(), &[bumps.strategy_index]`. The implementation could verify the `strategy` account is indeed a PDA derived by the `vault.key()` inside the `enable_strategy` instruction. Note that all strategy-related instructions should add this check.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-3 Admin/Operator can Steal Funds Through Fake `obligation` and `obligation_owner`

Severity: *High Risk*

Status: Fixed

File(s) affected: `programs/vault/strategy/solend_with_lm.rs`

Description: `obligation` holds the user-related information on Solend. It is used to have the collateral of the Mercurial vault on Solend. However, both `obligation` and `obligation_owner` are accounts being passed as `remaining_accounts` and lack strict validation to be the PDA of the program. As a result, the admin or the operator can give invalid `obligation` and `obligation_owner` accounts and later interact with Solend directly to withdraw from the `obligation`.
The following are a list of methods using `obligation` or `obligation_owner`:

- `SolendWithLMHandler::init_strategy`
- `SolendWithLMHandler::deposit`
- `SolendWithLMHandler::withdraw`
- `SolendWithLMHandler::get_collateral_amount`

Exploit Scenario: The admin or the operator sets the strategy with a fake `obligation_owner` that they own. Then the operator calls `withdraw_obligation_collateral` to Solend to collect the fund directly from Solend.

Recommendation: Confirm the PDA is indeed derived from the expected seeds.

- `obligation`: Verify the `Pubkey` of the account is indeed the PDA of `obligation_seeds` defined in `solend_with_lm.rs::L47-51` (`[SOLEND_OBLIGATION_PREFIX, vault_pubkey, bump]`).
- `obligation_owner`: Verify the `Pubkey` of the account is indeed the PDA of `obligation_owner_seeds` defined in `solend_with_lm.rs::L69-72` (`[SOLEND_OBLIGATION_OWNER_PREFIX, vault_pubkey, bump]`).

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-4 Incorrect Data Validation in `transfer_fee_vault`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `programs/vault/src/lib.rs`, `programs/vault/src/context.rs`

Description: The function `transfer_fee_vault` requires that the new fee vault's `mint` is equal to the vault's `token_mint`. As seen in the following code:

```
#[account(constraint = new_fee_vault.mint == vault.token_mint)]
pub new_fee_vault: Box<Account<'info, TokenAccount>>
```

This is an issue, since the `fee_vault` receives the fees in the form of `vault.lp_mint`.

Recommendation: Fix the constraint on `new_fee_vault` by requiring that `new_fee_vault.mint == vault.lp_mint`. Additionally, we recommend enforcing the same constraints as when initializing a new vault, i.e. require that the new fee vault's owner is the treasury.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-5 `total_amount` can be Overridden Resulting in a Denial of Service

Severity: *Medium Risk*

Status: Fixed

Description: In `state.rs::deposit_liquidity`, it directly overrides the `self.total_amount` as the `token_amount` when the unlocked amount is zero. However, the unlocked amount can be zero if the users withdraw all the tokens. It is possible to override the `self.total_amount` incorrectly if `deposit` is called. If the user overrides `total_amount` to an amount lower than the locked profit, the calculation of `get_unlocked_amount` will fail, blocking further calls to deposit or withdraw.

Exploit Scenario:

1. Alice deposits 100. Now `vault.total_amount` is 100 and `locked_profit` is 0.
2. After a while, the vault earns 50. Now `vault.total_amount` is 150 and `locked_profit` is 50.
3. Now Alice withdraws all tokens (100 unlocked tokens). Now `vault.total_amount` is 50 and `locked_profit` is 50.

4. Within the same transaction, Alice also deposits 10. Now `vault.total_amount` is overridden to 10 and `locked_profit` is 50. `locked_profit` is now greater than `vault.total_amount`.

Recommendation: Do the direct overriding only when `self.total_amount` is 0. When the unlocked amount is zero but `self.total_amount` is not zero, return `None` to let the `libs.rs::vault::deposit` instruction return an `VaultError::MathOverflow` error.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-6 Transfer of Admin Rights Could Lead to an Inoperable Vault

Severity: Low Risk

Status: Fixed

File(s) affected: `programs/vault/src/lib.rs`

Description: The `transfer_admin` function only requires a signature from the current admin to transfer the admin rights. In the case that an admin makes a mistake during this operation, the admin can be set to an account outside of their control, rendering that particular vault inoperable.

Recommendation: Consider requiring signatures for both the current and the new admin to make sure both accounts are under the admin's control when transferring the rights to a new account.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-7 Arithmetic Overflow/Underflow

Severity: Low Risk

Status: Fixed

File(s) affected: `programs/vault/src/state.rs`, `programs/vault/src/utils.rs`

Description: Arithmetic operations in Rust do not promise to fail in release mode when there is overflow or underflow. In the recommendation, we cover places where there might be a concern of overflow/underflow.

Recommendation:

- `state.rs::L56`: use `checked_mul` for `duration * locked_profit_degradation`.
- `state.rs::L80`: use `checked_add` for `locked_profit += gain`.
- `util.rs::L21`: use `checked_sub` for `total_amount_after - performance_fee`.
- In `utils::calculate_performance_fee`, should check that `lp_mint_more` does not exceed `u64::MAX` or alternatively update the total amount that the vault contains by first adding the new vault/strategy amounts and then subtracting the old amounts.
- In `state.rs::update_total_liquidity`, it is safer to do the calculation in `u128` to avoid overflow and cast it back to `u64` after finishing the computation.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-8 Parameter Discrepancy in Interface

Severity: Low Risk

Status: Fixed

File(s) affected: `programs/vault/strategies/base.rs`

Description: The order of the parameters: `admin`, `vault`, and `collatoral_mint` in the trait of `base.rs::StrategyHandler.init_strategy` does not match the order of parameters in the other strategy handler implementations.

Recommendation: Change the order of `base.rs::StrategyHandler::init_strategy` to match the order of the other implementations.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-9 Counterfeit `strategy_program`

Severity: Low Risk

Status: Mitigated

File(s) affected: `programs/vault/src/lib.rs`, `programs/vault/src/context.rs`

Description: `strategy_program` is one of the accounts being passed on the following instruction-context pair related to strategy in the `libs.rs`:

- `set_strategy`: `SetStrategy`
- `disable_strategy`: `DisableStrategy`
- `deposit_strategy`: `RebalanceStrategy`
- `withdraw_strategy`: `RebalanceStrategy`
- `withdraw_directly_from_strategy` : `WithdrawDirectlyFromStrategy`

However, we never check the program id in `lib.rs`, `context.rs`. Without validation, the attacker can try to pass in fake programs to interact with the wrong program.
Fortunately, the implementation in `solend_with_lm.rs` and `solend_without_lm.rs` hardcodes the instruction program with `get_solend_program_id()`. If the `strategy_program` were passed incorrectly, it would fail implicitly.

Recommendation: The implementation should validate that the `strategy_program` is the expected program id of the third-party platform. We can validate the program id on the `set_strategy` instruction and save it to the `strategy.strategy_program` state (as a new field). The other instructions should add a `has_one = strategy_program` constraint on the `strategy` account.

Update: The Mercurial Finance team has stated that they have intentionally hard coded the strategy program address. Their fix for this issue was to add `get_solend_program_id()` for the `init_obligation` instruction. While there is still no explicit check that the program id provided by the instruction is the same as `get_solend_program_id()`, Quantstamp determines that there is no direct exploit for this issue. The implementation for this issue is in the following PR.

QSP-10 Tokens can be Locked in the Vault or Strategy

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `programs/vault/src/state.rs`, `programs/vault/src/lib.rs`

Description: The `lib.rs::rebalance_strategy_wrapper` function relies on `token_vault.amount` to get the `liquidity_in_vault_before` value before calling `rebalance_strategy_fn()` in `L683-709`. Later, the value `liquidity_in_vault_before` will be used to calculate the `vault.total_amount` with the function `vault.update_total_liquidity`. In `state.rs::Vault::update_total_liquidity`, it will consider the value `liquidity_in_vault_before` as what was used in the previous rebalance crank and subtract it and add the new value `liquidity_in_vault_after` to update. However, the assumption that `liquidity_in_vault_before` is the same as the value of `liquidity_in_vault_after`, used in the previous rebalance crank, can be wrong. If anyone transfers tokens to the `token_vault` directly between the two rebalance cranks, it can break this assumption and break the formula that `vault.total_amount = liquidity_in_vault + sum(liquidity_in_strategies)`.
Fortunately, the implication of this is that those N tokens being transferred will be locked somewhere in the `token_vault` or the `strategy`. The Mercurial vault's essential functions seem to work continuously despite breaking the formula.

Exploit Scenario:

1. At time `t0`: `total_amount=100`, `liquidity_in_vault=50`, and `liquidity_in_strategy=50`
2. At time `t1`: Alice transfers 50 tokens directly to the `token_vault` -> `total_amount=100`, `liquidity_in_vault=100`, and `liquidity_in_strategy=50`
3. At time `t2`: Alice withdraws 100 tokens from the Mercurial vault ->`total_amount=0`, `liquidity_in_vault=0`, and `liquidity_in_strategy=50`
4. Now, 50 tokens are locked in the strategy.

Recommendation: Instead of using `token_vault.amount` as the `liquidity_in_vault_before`, we should store this value somewhere (e.g., in the `vault`) and use the stored value to update for the next rebalance. If anyone transfers tokens directly, the Mercurial vault will treat those tokens as the profit.

Update: The Mercurial Finance team has acknowledge the issue and has added a test to determine that all of the vault's intended functionality remains operational in the following PR.

QSP-11 Able to Provide Invalid VaultBumps to `vault`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `programs/vault/src/lib.rs`, `programs/vault/src/context.rs`

Description: In the `lib.rs::vault::initialize` instruction, `bumps` are passed as an extra data field and are assigned and saved to `vault.bumps`. There are two fields: `vault_bump` and `token_vault_bump` in the `context.rs::VaultBumps` struct. The `vault_bump` is used in several strategy-related instructions. The wrong bump values would cause those functions to fail later with the wrong value being stored and used.

Recommendation: For `vault_bump` use `*ctx.bumps.get("vault").unwrap()` to get the bump instead. After anchor `0.21.0`, it restricts the usage of `bump = <:pass-in-bump>` but allows you to get it via `tx.bumps.get("<:pda-account-name")`. See the change log: link. Note that this removes the need to pass in the bump as extra instruction parameters. We can remove the `bumps: VaultBumps` parameter from the `initialize` instruction.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-12 Rounding Error

Severity: *Low Risk*

Status: Fixed

Description: The implementations of the Solend strategies rely on the crate `solend-token-lending` (version 0.1.1) to calculate the amount between collateral and liquidity. However, the implementation of `solend_token_lending::state::Reserve` uses `try_round_u64()` to do the final rounding on `CollateralExchangeRate::collateral_to_liquidity` and `CollateralExchangeRate::liquidity_to_collateral`. This allows it to round up if the remaining decimal is larger than or equal to 0.5. This allows Solend users to get more liquidity as the collateral with `collateral_to_liquidity`. Note that the Solend team has a fixed PR (link) for this, and it was merged to the `mainnet` branch.
This would have impact to the `from_collateral_to_liquidity` and `from_liquidity_to_collateral` methods of the Solend strategy handlers. `from_collateral_to_liquidity` is used to calculate `liquidity_in_strategy_after` in `lib.rs::rebalance_strategy_wrapper`. The result will potentially cause the `total_value` to have more than what the vault owns. `from_liquidity_to_collateral` is used to decide how much to withdraw from the lending platform in `lib rs::vault::withdraw_strategy` and `lib.rs::vault::withdraw_directly_from_strategy` instructions. The rounding issue might cause the function to fail without enough collateral for the liquidity.
Following are the places using the mis-implemented functions:

- `solend_with_lm.rs::L245`: the line `collateral_exchange_rate.collateral_to_liquidity`
- `solend_with_lm.rs::L254`: the line `collateral_exchange_rate.liquidity_to_collateral`
- `solend_without_lm.rs::L151`: the line `collateral_exchange_rate.collateral_to_liquidity`
- `solend_without_lm.rs::L169`: the line `collateral_exchange_rate.liquidity_to_collateral`

Recommendation: It is critical to first confirm with the Solend team whether the fix has been deployed or not. The calculation used in Mercurial finance must be consistent with them. Also, Quanstamp recommends the Mercurial finance team ask the Solend team to release the library with the fix. If there is a new library, the implementation can use the updated version.

Update: The Mercurial Finance team's solution for this issue was to create their own fork of the solend-token-lending library containing the recommended fix. The implementation for this issue is in the following PR.

QSP-13 Rounding Gap on `withdraw_directly_from_strategy`

Severity: *Low Risk*

Status: Fixed

Description: The instruction `lib.rs::vault::withdraw_directly_from_strategy` would first decide the `out_amount` from the `unmint_amount`, then calculate the remaining amount needed from the strategy and then withdraw those from the strategy. However, to withdraw from the strategy, it needs to go through the conversion of liquidity to collateral. The calculation here is decimal based and there will be some gap in the rounding for the final result. The rounding issue will also occur on the lending platform when the platform tries to convert the collateral to the withdrawable liquidity back. As a consequence, it is not guaranteed that the vault will get the amount expected when calling `withdraw` on the strategy handler. This is mitigated by performing a `checked_add(1)`. However this can still lead to a failure if the strategy does not have enough collateral after the addition.

Recommendation: In `lib.rs::L644`, instead of using `out_amount` for transfer, we should check if `out_amount` is less than or equal to the amount held in `token_vault`. If the `out_amount` is larger, we transfer whatever is kept in `token_vault` instead. Note that this potential loss in the rounding should be documented to the users/clients interacting with this instruction.

Update: The Mercurial Finance team has fixed this issue in the following PR.

QSP-14 Missing Validations in Solend Strategies

Severity: Low Risk

Status: Fixed

File(s) affected: `programs/vault/src/lib.rs`, `programs/vault/src/context.rs`, `programs/vault/src/strategy/solend_without_lm.rs`, `programs/vault/src/strategy/solend_with_lm.rs`

Description: It is hard to validate all incoming accounts in the context of the strategy handlers as they are abstracted concepts shared by shared 3rd party platforms. So some of the validations have to be done at the strategy handler level. Nonetheless, some checks are missing and can cause invalid data stored in accounts. Additionally, the strategy handler implementations rely on `remaining_accounts` to fetch different accounts needed to call the 3rd party platform. Those accounts require separate validations as well.
Following is the list of validations with a lower impact that we found. Those with higher severity will have separate issues:

- In `solend_without_lm.rs::SolendWithoutLMHandler::init_strategy`
  - . `reserve(:AccountInfo).owner` should be the program id of the Solend program.
  - . `reserve(:SolendReserve).collateral.mint_pubkey` should be the same as `_collateral_mint.key()`
  - . `_bumps` should be an empty array
- In `solend_without_lm.rs::SolendWithoutLMHandler::deposit`, we recommend deserializing the `reserve` to the `SolendReserve` type and check that:
  - . the `owner` of the reserve account info is the program id of Solend
  - . `reserve.liquidity.supply` matches `solend_accounts.reserve_liquidity_supply`
  - . `reserve.collateral.mint` matches `solend_accounts.reserve_collateral_mint`
  - . `reserve.lending_market` matches `solend_accounts.lending_market`
- In `solend_without_lm.rs::SolendWithoutLMHandler::withdraw`, we recommend deserializing the `reserve` to the `SolendReserve` type and check that:
  - . the `owner` of the reserve account info is the program id of Solend
  - . `reserve.liquidity.supply` matches `solend_accounts.reserve_liquidity_supply`
  - . `reserve.collateral.mint` matches `solend_accounts.reserve_collateral_mint`
  - . `reserve.lending_market` matches `solend_accounts.lending_market`
- In `solend_with_lm.rs::SolendWithLMHandler::init_strategy`
  - . `reserve(:AccountInfo).owner` should be the program id of the Solend program.
  - . `reserve(:SolendReserve).collateral.mint_pubkey` should be the same as `_collateral_mint.key()`
  - . `_bumps` should have exactly two items.
  - . `remaining_accounts.lending_market` should be the same as `reserve(:SolendReserve).lending_market`
- In `solend_without_lm.rs::SolendWithLMHandler::deposit`, we recommend deserializing the `reserve` to the `SolendReserve` type and checking the followings and check that:
  - . the `owner` of the reserve account info is the program id of Solend
  - . `reserve.collateral.supply` matches `solend_accounts.collateral_supply`
  - . `reserve.liquidity.supply` matches `solend_accounts.reserve_liquidity_supply`
  - . `reserve.lending_market` matches `solend_accounts.lending_market`
  - . `reserve.collateral.mint` matches `solend_accounts.reserve_collateral_mint`
  - . `reserve.liquidity.pyth_oracle_pubkey` matches `solend_accounts.reserve_liquidity_pyth_oracle`
  - . `reserve.liquidity.switchboard_oracle_pubkey` matches `solend_accounts.reserve_liquidity_switchboard_oracle`
- In `solend_without_lm.rs::SolendWithLMHandler::withdraw`, we recommend deserializing the `reserve` to the `SolendReserve` type and check that:
  - . the `owner` of the reserve account info is the program id of Solend
  - . `reserve.collateral.supply` matches `solend_accounts.collateral_supply`
  - . `reserve.liquidity.supply` matches `solend_accounts.reserve_liquidity_supply`
  - . `reserve.lending_market` matches `solend_accounts.lending_market`
  - . `reserve.collateral.mint` matches `solend_accounts.reserve_collateral_mint`
  - . `reserve.liquidity.pyth_oracle_pubkey` matches `solend_accounts.reserve_liquidity_pyth_oracle`
  - . `reserve.liquidity.switchboard_oracle_pubkey` matches `solend_accounts.reserve_liquidity_switchboard_oracle`

Recommendation: Add the validations pointed out in the description section.

Update: The Mercurial Finance team have implemented some validations for this issue in the following PR. However, it worth noting that the following issues have not been addressed:

- `solend_without_lm.rs::SolendWithoutLMHandler::init_strategy`
  - . `reserve(:SolendReserve).collateral.mint_pubkey` should be the same as `_collateral_mint.key()`.
  - . `_bumps` should be an empty array.
- `solend_with_lm.rs::SolendWithLMHandler::init_strategy`
  - . `_bumps` should have exactly two items.

-
  `.reserve(:SolendReserve).collateral.mint_pubkey` should be the same as `_collateral_mint.key()`.

Quantstamp determines that this is a non-critical issue towards Mercurial's security.

QSP-15 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

Description: The vault has both an administrator and an operator with special privileges to make modifications to the program.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the program allows to the admin and the operator.

Update: The Mercurial Finance team has acknowledged this issue.

QSP-16 No Error for Reaching Max Strategies on `add_strategy`

Severity: *Informational*

Status: Fixed

File(s) affected: `programs/vault/src/state.rs`

Description: `L166` contains a `TODO` comment about checking whether the maximum number of strategies have been reached.

Recommendation: Validate that `strategies` can accept an additional strategy and throw an error if that is not the case.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-17 Consider Enforcing Rewards Collection on Strategy Removal

Severity: *Informational*

Status: Acknowledged

File(s) affected: `programs/vault/src/lib.rs`

Description: The function `disable_strategy`, which removes a strategy from the vault, requires the admin to manually collect the rewards before calling this function. This can easily lead to lost rewards for anyone not intimately familiar with the code.

Recommendation: Consider enforcing in some way that rewards have recently been collected or always collect the rewards (implemented as no-op on strategies that don't support this).

Update: The Mercurial Finance team has acknowledged this issue with the following statement: "Due to transaction size limit, we can only ensure it from an off-chain service. We also keep the rewards, so users will not be affected in case the service doesn't claim rewards before disabling a strategy".

QSP-18 Estimating Amounts in `StrategyWithdraw`

Severity: *Informational*

Status: Fixed

File(s) affected: `programs/vault/src/lib.rs`

Description: The function `disable_strategy` logs a `StrategyWithdraw` event with the amount estimated by converting from the collateral amount to the liquidity amount (L205). Ideally, such events should log the exact amounts.

Recommendation: By computing the delta between the token balance before and after the call to withdraw, the amount should be accurate on all withdrawals.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-19 Risk of Setting the Same Strategy Twice to the Same Vault

Severity: *Informational*

Status: Fixed

File(s) affected: `programs/vault/src/lib.rs`, `programs/vault/src/state.rs`

Description: The instruction `libs.rs::vault::set_strategy` does not verify if the vault has the same strategy already or not. In `libs.rs::L123` it calls `vault.add_strategy` and it will override the first slot in the `vault.strategies` with empty/default `Pubkey`. The lack of validation potentially allows the same strategy to be set twice to the vault, with the risk of breaking the formula of `total_amount = liquidity_in_vault + sum(liquidity_in_strategies)` as the implementation can count the same strategy twice.
Fortunately, the `libs.rs::vault::set_strategy` instruction has another context field: `collateral_vault` which relies on `strategy.Key()` to be part of the seed of the PDA and `collateral_vault` is with the `init` account constraint. The constraint will check if the account is empty or not and prevent the same strategy from being set twice as `collateral_vault` will not be empty after the first call of `set_strategy`.

Recommendation: Add validation that the same strategy does not exist in `vault.strategies` before adding to the array in the `set_strategy` instruction.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-20 Profit Lockup

Severity: *Undetermined*

Status: Fixed

File(s) affected: `programs/vault/src/libs.rs`

Description: In the instruction `libs.rs::vault::update_locked_profit_degradation`, there is no validation that the `locked_profit_degradation` value is larger than zero. Once the admin sets it to zero, none of the locked funds will ever be unlocked and cause the user not to be able to withdraw profit.

Recommendation: Add validation that `locked_profit_degradation` cannot be zero.

Update: The Mercurial Finance team has implemented the recommendation for this issue in the following PR.

QSP-21 Effectiveness of Sandwich Attack Prevention

Severity: *Undetermined*

Status: Mitigated

Description: The `state.rs::LockedProfitTracker` is configured to drip the yield profit every hour. This prevents an immediate sandwich attack. However, the user can now simply monitor a huge profit rebalance event and then immediately deposit for an hour to take such profit away.

Recommendation: One way to mitigate this is to ensure the strategy is rebalanced regularly in a short period. For instance, if the profit is added every 10 minutes, then the cost of locking the fund for an hour might not seem worth doing for a specific rebalance event. Also, the gain from the strategy will be smoothed out by a shorter period of profit collection time. Note that this does not necessarily require the operator to withdraw and re-deposit the strategy. The team can consider adding a simple rebalance instruction that re-calculates the gain from the strategy and adds it to the `vault.total_amount`.

Update: The Mercurial Finance team has mitigated the issue and given the following statement: "We have increased the drip time to 6 hours to mitigate the issue. In the future we will try to implement dynamic dripping time". This aforementioned implementation for this issue is in the following PR.

QSP-22 Lack of Protection Against Black Swan Events

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `programs/vault/src/lib.rs`

Description: It is not impossible to have a loss when there is a black swan event on the lending platform used in the strategy. It can potentially happen when the liquidate mechanism is not working or the volatility of a token is exceptionally high. Also, if a bug occurs on the lending platform, it can lead to a loss in profit. However, `lib.rs::vault::withdraw` does not call rebalance crank to reflect the loss, and `lib.rs::vault::withdraw_directly_from_strategy` reaches the rebalance crank only after `lib.rs::L550-552` calculated the `out_amount`.

Exploit Scenario: Alice tries to immediately call `withdraw_directly_from_strategy` from the Mercurial vault after a loss on the third-party platform. Alice can take away all the funds without reflecting on the flop.

Recommendation: Quantstamp does not have an elegant way to solve this aside from rebalancing all strategies of a vault before allowing withdrawal. However, if that is not feasible due to computation limit constraints, we recommend doing the following:

- Have a function that simply rebalances the strategy. And the team ensures the keeper will continuously rebalance all strategies within a reasonable short time frame. This helps sync the amount of liquidity between the vault and the strategies, as well as reducing the possibility of the user withdrawing these affected funds.
- Prepare and document operation plans in the case of 3rd party issues.

Update: The Mercurial Finance team has acknowledged this issue and given the following statement: "Our program will log about the loss, and we are also talking with lending protocols about an internal insurance fund".

Automated Analyses

Cargo Audit

```
Scanning Cargo.lock for vulnerabilities (556 crate dependencies)
1 vulnerability found!
Crate:         regex
*Version:       1.5.4
Title:         Regexes with large repetitions on empty sub-expressions take a very long time to parse
Date:          2022-03-08
ID:            RUSTSEC-2022-0013
URL:           https://rustsec.org/advisories/RUSTSEC-2022-0013
Solution:      Upgrade to >=1.5.5
```

Rust-Clippy

```
error: using `clone` on a double-reference; this will copy the reference of type `&[anchor_lang::prelude::AccountInfo]` instead of cloning the inner type
   --> programs/vault/src/lib.rs:167:34
    |
167 |         let remaining_accounts = ctx.remaining_accounts.clone();
    |                                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = note: `#[deny(clippy::clone_double_ref)]` on by default
    = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#clone_double_ref

error: using `clone` on a double-reference; this will copy the reference of type `&[anchor_lang::prelude::AccountInfo]` instead of cloning the inner type
   --> programs/vault/src/lib.rs:258:34
    |
258 |         let remaining_accounts = ctx.remaining_accounts.clone();
    |                                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#clone_double_ref

error: using `clone` on a double-reference; this will copy the reference of type `&[anchor_lang::prelude::AccountInfo]` instead of cloning the inner type
   --> programs/vault/src/lib.rs:323:34
    |
323 |         let remaining_accounts = ctx.remaining_accounts.clone();
    |                                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#clone_double_ref

error: using `clone` on a double-reference; this will copy the reference of type `&[anchor_lang::prelude::AccountInfo]` instead of cloning the inner type
   --> programs/vault/src/lib.rs:575:38
    |
575 |             let remaining_accounts = ctx.remaining_accounts.clone();
    |                                      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    |
    = help: for further information visit https://rust-lang.github.io/rust-clippy/master/index.html#clone_double_ref
```

Soteria

Soteria was run against the mercurial_vault. It founds 7 arithmetic-related issues, we have removed the false positives and list the remaining findings in our issues.

Adherence to Specification

There is sufficient documentation on how the program is supposed to work in general and the relations between the various components. Of note is the documentation provided for the locked profit implementation, which was well written. However there is much more to be covered, such as documentation of all the main state structs.

Code Documentation

Although some critical parts of the code are commented and the spec provide adequate context there are many functions which lack comments completely. Some particular issues are:

- Several traits are defined in strategy/base.rs::StrategyHandler without any documents or comments on what each function and each parameter should do. Since this is a critical part of the strategies, we suggest providing some description.
- None of the claim_rewards methods of the strategy handlers are implemented. It is either no-op or simply returning an error. There should be some comments/documents on the plan.
- Various accounts being passed in throughout the program are unchecked and as such need to have an explanation as to why that is the case.

We suggest mandatory rustdoc annotations for all methods so documentation can be generated automatically.

Adherence to Best Practices

1. Use `UncheckedAccount` instead of `AccountInfo`. This is recommended from the anchor document: link. Also, should provide a reason in the comments why the team decide to not use checked account types. Update: **Fixed**
   - `context.rs::L28`: `base` of `Initialize`. However, this can be as `Signer` instead (doc).
   - `context.rs::L43`: `admin` of `Initialize`
   - `context.rs::L64`: `rent` of `Initialize`
   - `context.rs::L83`: `operator` of `SetOperator`
   - `context.rs::L106`: `new_admin` of `TransferAdmin`
   - `context.rs::L139`: `reserve` of `SetStrategy`
   - `context.rs::L184`: `reserve` of `DisableStrategy`
   - `context.rs::L245`: `reserve` of `RebalanceStrategy`
   - `context.rs::L303`: `reserve` of `WithdrawDirectlyFromStrategy`
2. Use `Program` for program-related accounts instead of `AccountInfo.` Update: **Fixed**
   - `context.rs::L66`: `token_program` of `Initialize`
   - `context.rs::L68`: `system_program` of `Initialize`
   - `strategy/solend_with_lm.rs::L333`: `token_program` of `InitObligation`
   - `strategy/solend_with_lm.rs::L408`: `token_program` of `DepositReserveLiquidityAndObligationCollateral`
   - `strategy/solend_with_lm.rs::L465`: `token_program` of `WithdrawObligationCollateral`
   - `strategy/solend_with_lm.rs::L513`: `token_program` of `RedeemReserveCollateral`
   - `strategy/solend_without_lm.rs::L273`: `token_program` of `DepositReserveLiquidity`
   - `strategy/solend_without_lm.rs::L299`: `token_program` of `RedeemReserveCollateral`
3. Use `Sysvar<'info, Clock>` as the type for the `clock` field for better readability and to leverage the auto account validations from the anchor when calling `try_accounts`. Update: **Fixed**
   - `strategy/port_finance_without_lm.rs::L206`: under `DepositReserveLiquidity`
   - `strategy/port_finance_without_lm.rs::L221`: under `PortRemaining`
   - `strategy/port_finance_without_lm.rs::L269`: under `Redeem`.
   - `strategy/solend_with_lm.rs::L511`: under `RedeemReserveCollateral`
   - `strategy/solend_with_lm.rs::L246`: under `SolendRemaining`.
   - `strategy/solend_with_lm.rs::L270`: under `DepositReserveLiquidity`
   - `strategy/solend_with_lm.rs::L296`: under `RedeemReserveCollateral`
4. Remove unnecessary `#[account(mut)]` constraint on `admin` context field. Update: **Fixed**
   - `context.rs::L197`
   - `context.rs::L212`
5. In `context.rs::L64`, remove `rent` from the `Initialize` structs for simplicity. `pub rent: Sysvar<'info, Rent>` is no longer needed together with `init` account constraint. See the change log of 2021-08-08 (link). Update: **The recommended fix results in a build error**
6. Remove unused struct fields of `context.rs::StrategyBumps`: `collateral_vault_bump` and `strategy_bump`. Update: **Fixed**
7. In `strategy/solend_with_lm.rs::L63`, remove unnecessary `system_program.clone()` in `invoke_signed`. Update: **Fixed**
8. Use `&Account<'info, TokenAccount>` as the type for `collateral_vault` instead of `&AccountInfo<'info'>` in the trait method `get_collateral_amount` of the struct `strategy/base.rs::StrategyHandler`. This removes the need to deserialize again inside the implementation. Update: **Fixed**
9. In `strategy/solend_with_lm.rs::L287`, it is better to get the `obligation_collateral` by `obligation.find_collateral_in_deposits(_reserve.key())` instead of assuming it to be at index 0. Update: **Fixed**
10. Consider returning an error if the strategy does not exist during the removal in `state.rs::Vault::remove_strategy`. Update: **Fixed**
11. Consider to call `withdraw_obligation_collateral_and_redeem_reserve_liquidity` (instruction link) instead of having two instructions in

`solend_with_lm.rs::SolendWithLMHandler::withdraw` method. Update: **Fixed**

12. In `lib.rs` the vault_seeds are constructed inline several times, sometimes even twice in a single function. Consider encapsulating that logic in a function that takes a vault parameter to avoid code duplication. Update: **Fixed**
13. In `lib.rs` `set_`, `enable_` and `disable_strategy` are inaccurate names. Consider renaming them to `initialize_`, `add_` and `remove_strategy` respectively. Update: **Fixed**
14. In `lib.rs, L827`: Consider renaming `StakingRewards` to something more general, since rewards don't necessarily have to come from staking. Update: **Fixed**
15. On `state.rs, L10` there is a typo in `LOCKED_PROFIT_DEGRATION_DENUMERATOR`. Should probably be corrected to `..._DEGRADATION_DENOMINATOR`. Update: **Fixed**

Test Results

Test Suite Results

All tests passing.

```
running 17 tests
test state::tests::test_deposit_liquidity_overflow ... ok
test state::tests::test_deposit_liquidity_success ... ok
test state::tests::test_get_amount_by_share_overflow ... ok
test state::tests::test_get_unmint_amount_overflow ... ok
test state::tests::test_get_amount_by_share_success ... ok
test state::tests::test_get_unmint_amount_success ... ok
test state::tests::test_add_strategy_max ... ok
test state::tests::test_profit_drip ... ok
test state::tests::test_remove_liquidity_overflow ... ok
test state::tests::test_remove_liquidity_success ... ok
test state::tests::test_sandwich_attack ... ok
test state::tests::test_update_total_liquidity_overflow ... ok
test test_id ... ok
test state::tests::test_vault_total_amount_override ... ok
test utils::tests::test_performance_fee ... ok
test state::tests::test_valid_deposits ... ok
test state::tests::test_valid_withdrawals ... ok

test result: ok. 17 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.01s


running 3 tests
test test_transfer_fee_vault_invalid_treasury ... ok
test test_transfer_fee_vault_invalid_token_mint ... ok
test test_transfer_fee_vault ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.15s


running 1 test
test test_mango_strategy ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.99s


running 1 test
test test_vault_field_strategy_account_migration ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.54s


running 1 test
test test_port_finance_strategy ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.29s


running 4 tests
test test_solend_with_lm_strategy_counterfeit_obligation_owner ... ok
test test_solend_with_lm_strategy_counterfeit_obligation ... ok
test test_solend_without_lm_strategy ... ok
test test_solend_with_lm_strategy ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.77s


running 3 tests
test test_port_finance_without_lm_strategy ... ok
test test_solend_without_lm_strategy ... ok
test test_token_locked_in_vault ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 3.80s


running 1 test
test test_transfer_admin ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.01s


running 2 tests
test test_update_zero_locked_profit_degradation ... ok
test test_update_locked_profit_degradation ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.61s


running 4 tests
test test_vault_disable_strategy_after_set_strategy ... ok
test test_vault_with_counterfeit_strategy ... ok
test test_vault_disable_strategy_solend_without_lm ... ok
test test_vault_disable_strategy_port_without_lm ... ok

test result: ok. 4 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.70s


running 1 test
test test_vault_is_disable ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.97s


running 3 tests
test test_vault_total_amount_after_deposit ... ok
test test_vault_total_amount_after_withdraw ... ok
test test_vault_total_amount_after_withdraw_directly_from_strategy ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 1.45s


running 1 test
test test_vault_set_operator ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.70s
```

Code Coverage

A coverage report could not be obtained for this audit. Quantstamp usually recommends developers to get their branch coverage to and above a `90%` before a project goes live, in order to avoid hidden functional bugs that might not be easy to spot during the development phase.

Appendix

File Signatures

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

Contracts

```
2c6788f73c88bb43d51b977235c873befa160a194572f6f77a391617aeb53576 ./src/lib.rs
03db67c9cd7708645aa08b6287dfe01190b26bda0979ea18fc2f20f87f500f9e ./src/state.rs
514206b3fdc0da2cc7114bffc22c68eb87d254be2469fc951dfb2903348d54a8 ./src/macros.rs
2a701be7f5a8edc0af4841cb23e71e0cdd44121ba406aa6d452529311ca5cf42 ./src/utils.rs
d29ae8d9c011d269bb97716512fb694c2e9788c7fade32a7b29905a4313f4feb ./src/context.rs
a4f4f42b207bed03ebfbd6ba8f607f89166bffe28f39e561cc98de317ddfacd3 ./src/seed.rs
33677e42273a84beeaff62d8f070b2ebd590cd14c490fbee115c97cdb23e06b7 ./src/strategy/solend_with_lm.rs
cf89006f73c0a1a2f9188bef1d09bdc2bd92315a297d9c6c197fcdda7829ee58 ./src/strategy/base.rs
d003a534ee43b25b0c2945d9eabc83b7979618dfc0f3198b72a0254fb4a71806 ./src/strategy/mod.rs
27dfe2fa5793bef0bc1e03bae50cc4fd44d0359014ff46f56849da8645cf3553 ./src/strategy/solend_adaptor.rs
18581119ac59a45a2e0d653592db5c2998970e0651057db33722f0e3f1c3d504 ./src/strategy/solend_without_lm.rs
```

Tests

```
d974d3d5aec93cd778d3689bc9e0a00f35d1bf941f60d40195cca87298714974 ./tests/test_migration.rs
6e6c78205de0964d23c5547c3b58e3ac2c4d6582f009d117907cb41971882cb1 ./tests/test_vault_disable_strategies.rs
22f52c1dcd8aff3a59fb4d6aef1c7cfe9e446cda1d6b6d9d44aceaafa46aef6e ./tests/test_fee_vault.rs
2fe95a3c9cd8985dbe1de9eb6f991c918bf7962894c52811cd8c2d004f479c7d ./tests/test_vault_is_disabled.rs
bc89de009e39fbf62e1028ad3f98791828c7dff438fb6b9de61662dfb05d5199 ./tests/test_strategies.rs
a3b884444148cb0baf3404b940164a7f969280cf89cd151cee6f43eab336166e ./tests/test_update_locked_profit_degradation.rs
b024cba4ee508f72fe813271a1494bea2354d802c1b009019f849d750356bcca ./tests/test_solend_strategy.rs
9b10dc4fe8bd5e5813bf9dece44eea274a9b6876060eb6c39d7ef8edd542954b ./tests/test_vault_transfer_operator.rs
4375fabc984228aa9a64e12f7389297d230fdf4e6f4e4a27d6f6b14879ef69e7 ./tests/test_transfer_admin.rs
00904be482443addd9321bed1b36a9f2456d91b05d368044bfbafaaeae1b3c74 ./tests/test_vault_total.rs
31f78b5bb80f9d572c213410b3099c761d168875f00c51254c705e5a0961db19 ./tests/helpers/mod.rs
069743f13392151beebf538bca148510860a9eedd324537be8f9ff5f86d2074c ./tests/helpers/solend.rs
2e21758102ce659db8d3c8694d60917587f2002869f7d1b1621fab219e08611e ./tests/helpers/test_transfer_admin.rs
c9d8a471c51f46306703ff14aa41e2cc12ca7e152ca8991f2bd44ace552bcaa0 ./tests/helpers/ids.rs
ceca0e19574b69343cdd17c41209113fd2e36639a79479d815a4a017123f25ac ./tests/helpers/vault.rs
8403d38637ed99e5842993abd6a5ce4ac79658732ee31bd17692294878d5df1f ./tests/helpers/utils.rs
8e27fb9285e8feff05f50bdecf339d08cea0d79b03026db56f011a5139aac72f ./tests/helpers/token_swap.rs
92103d83740762a4854d80873d36b63314539b33194bac8224eaceabc47c7b78 ./tests/helpers/strategy_handler/solend_with_lm.rs
9aa93a4233cf84a2999055271325da3108d932e140ea231649727ce2260fa6df ./tests/helpers/strategy_handler/base.rs
97a9d98a95ffd1d06b2e6ebe107f12e2397503d9d1c02d0db723f5a6a0d1d030 ./tests/helpers/strategy_handler/mod.rs
33f234c3ab84bee7c70c25d8167033ef5ea8b4ef871864bf72a8a777845a83ff ./tests/helpers/strategy_handler/solend_without_lm.rs
```

Changelog

- 2022-05-19 - Initial report
- 2022-06-04 - Re-Audit report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.