PUBLIC

Code Assessment

of the Hybrid Vaults Smart Contracts

March 26th, 2026

Produced for

by

Contents

1 Executive Summary

Dear Yield Basis,

Thank you for trusting us to help Yield Basis with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Hybrid Vaults according to Scope to support you in forming an opinion on their security risks.

Hybrid Vaults extends the existing Yield Basis protocol. Users who deposit crvUSD to a HybridVault get higher stablecoin allocation caps. The vault holds LT tokens and crvUSD, requiring users to back at least 55% of their LT value with crvUSD.

The most critical subjects covered in our audit are asset solvency, functional correctness, and access control. Security regarding all aforementioned subjects is high.

The general subjects covered are code complexity, upgradeability, and error handling. Security regarding all aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	2
• Code Corrected	2
Medium-Severity Findings	2
• Code Corrected	2
Low-Severity Findings	6
• Code Corrected	6

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The following files were in scope:

```
contracts/HybridVault.vy
contracts/HybridVaultFactory.vy
contracts/HybridFactoryOwner.vy
```

The assessment was performed on the source code files inside the Yield Basis repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	12 January 2026	409d816da01f8305353b420b12b591354fface95	Initial Version
2	25 January 2026	aed043a46e71108fdf932a3430ff4e7e5f8d34b3	Version with fixes
3	17 February 2026	5bb002d49c1f9af09f4473291f958e41aae53088	Version with fixes
4	18 March 2026	d50a7383069a52b8c75b9779d28d4786b1d59efd	Version with fixes
5	19 March 2026	ad40eff3b4dbfb3f68af57e49821adb47f9e1ca1	Version with fixes
6	24 March 2026	faadf56967ea1b34097c55c082c3e7df8ac1eca8	Version with fixes

For the Vyper smart contracts, the compiler version `0.4.3` was chosen.

2.1.1 Excluded from scope

All other contracts not mentioned in Scope were not in scope of this assessment. Third-party libraries were not in scope.

2.2 System Overview

This system overview describes the current version (Version 2) of the contracts as defined in the Assessment Overview.

At the end of this report section, we have added subsections for each of the changes according to the versions.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Yield Basis Hybrid Vaults extends the existing Yield Basis protocol. Users who deposit crvUSD to a HybridVault get higher stablecoin allocation caps. The vault holds LT tokens and crvUSD (stored in an ERC-4626 vault like scrvUSD). Users must back at least 55% of their LT value with crvUSD. LT tokens can be staked for YB rewards.

Hybrid Vaults has 3 contracts: `HybridVaultFactory`, `HybridFactoryOwner`, and `HybridVault`.

2.2.1 HybridVaultFactory

`HybridVaultFactory` creates and tracks `HybridVault` for each user. Each user can have one vault.

The admin can set:

- `vault_impl` - implementation for new vaults. Does not affect existing vaults.
- `stablecoin_fraction` - required crvUSD backing ratio (default 55%). Higher value means users must deposit more crvUSD per LT value.
- `pool_limits` - max value per pool. Limits how much can be deposited to each YB market.
- `allowed_crvusd_vaults` - whitelist of ERC-4626 vaults (e.g., scrvUSD). Users can only use allowed vaults to deposit crvUSD.

Users call `create_vault()` and pick a crvUSD vault. They can switch to a different crvUSD vault later.

2.2.2 HybridFactoryOwner

`HybridFactoryOwner` owns the Yield Basis `Factory` contract. It acts as a proxy for admin commands. The admin can:

- Call `transfer_ownership_back()` to reclaim direct ownership of the YB Factory.
- Set rates, fees, price aggregator address, flash lender implementation address, add allocator addresses, and fill staker vpools via pass-through functions to the Yield Basis `Factory`.
- kill LTs and invoke `distribute_borrower_fees()` with privileges.
- Add or remove `limit_setters` who can call `LT.allocate_stablecoins()`.

The `HybridVaultFactory` is expected to be a limit setter.

2.2.3 HybridVault

`HybridVault` is the vault implementation. The `HybridVaultFactory` deploys it as a minimal proxy. Each vault has one owner. The owner cannot be changed.

crvUSD backing:

The vault holds crvUSD in an ERC-4626 vault (e.g., scrvUSD). This backing is required before depositing to YB.

- `deposit_crvusd()` - pull crvUSD from caller, deposit to scrvUSD. Anyone can call (adds backing to the vault).
- `deposit_scrvusd()` - transfer scrvUSD shares directly to the vault. Anyone can call.
- `redeem_crvusd()` - withdraw crvUSD from scrvUSD. Owner only. Reverts if backing falls below required ratio.
- `withdraw_scrvusd()` - withdraw scrvUSD shares. Owner only. Same backing check.
- `set_crvusd_vault()` - switch to a different crvUSD vault. Owner only. Old vault must be empty.

YB deposits (LT tokens):

The vault deposits assets to YB markets and holds the resulting LT tokens.

- `deposit()` - deposit assets to a YB pool. Pulls assets from owner, deposits to YB, receives LT tokens. The vault must have enough crvUSD backing (55% of LT value). Can optionally pull crvUSD automatically. Can optionally stake LT tokens immediately.
- `withdraw()` - burn LT tokens, receive assets back. Can optionally unstake first. Can optionally return excess crvUSD to owner.
- `stake()` - stake LT tokens in the pool's gauge for YB rewards.

- `unstake()` - unstake from gauge, receive LT tokens back (to `HybridVault`).
- `claim_reward()` - claim staking rewards from all pools, send to owner.

Other:

- `recover_tokens()` - rescue accidentally sent tokens. Cannot recover LT, staker, or crvUSD vault tokens.

2.3 Trust Model

ADMIN (fully trusted)

The same address is admin for both `HybridVaultFactory` and `HybridFactoryOwner`. Set at deployment from the YB Factory admin. Cannot be changed.

- Can set vault implementation, stablecoin fraction, pool limits, allowed crvUSD vaults.
- Can kill LT markets, set rates and fees, transfer YB Factory ownership.
- Can add/remove limit_setters.
- Can set personal limits per vault.
- Worst case: can brick the system by setting bad parameters, kill all markets, or drain value by manipulating rates.

Vault owner (untrusted)

Each `HybridVault` has one owner, set at creation. Cannot be changed.

- Can only affect their own vault.
- Can deposit/withdraw assets, manage crvUSD backing, stake/unstake LT tokens.
- Cannot affect other users or system parameters.

limit_setters (partially trusted)

Addresses that can call `LT.allocate_stablecoins()` through `HybridFactoryOwner`.

- `HybridVaultFactory` is expected to be a limit_setter so vaults can adjust allocation during deposits.
- Worst case: can manipulate stablecoin allocation limits for LT markets.

Emergency admin (partially trusted)

- Can kill LTs and trigger emergency withdrawal on user positions.
- Gains admin powers if the admin calls `transfer_ownership_back()`.

Anyone (untrusted)

- Can create a vault for themselves (one per address).
- Can deposit crvUSD or scrvUSD to any vault. This only benefits the vault owner.

External dependencies:

- Yield Basis contracts (`Factory`, `LT`, `AMM`) - must function correctly. LT tokens and staking rewards come from these.
- ERC-4626 crvUSD vaults (e.g., scrvUSD) - must be non-malicious. The vault holds user crvUSD in these.
- Price oracles - used by LT/AMM for valuations. Bad oracle data could cause incorrect backing calculations.

Upgradeability:

- `HybridVault` is deployed as a minimal proxy. The admin can change `vault_impl` but this only affects new vaults. Existing vaults keep their original implementation.
- `HybridVaultFactory` and `HybridFactoryOwner` are not upgradeable.

2.4 Differences between versions

2.4.1 Version 1 → 2

- The crvUSD vault changed from a shared immutable to a per-vault setting. Users pick their crvUSD vault at creation and can switch later (`set_crvusd_vault()`).
- Added per-vault personal pool limits (`set_personal_limit()`), summed with global limits.
- Added deposit_stablecoins and withdraw_stablecoins options to `deposit()` and `withdraw()` for automatic crvUSD management.
- Added `recover_tokens()` to rescue accidentally sent tokens.
- Added crvUSD vault allowlist in the factory (allowed_crvusd_vaults).

2.4.2 Version 2 → 3

- stablecoin_allocation changed from global to per-pool tracking.
- Default stablecoin_fraction changed from 40% to 55%.
- `_required_crvusd()` no longer reverts when a pool's oracle fails — returns `max_value(uint256)` instead. Added division-by-zero guards.
- `_required_crvusd_for()` now accounts for admin fees in the value calculation.
- Added `emergency_withdraw()` for killed or stuck markets, with optional crvusd_from_wallet to fund debt repayment directly.
- Added `safe_to_deposit()` check to keep deposits within safe AMM debt/collateral bounds.
- `withdraw()` now supports `max_value(uint256)` as shares (withdraw all) and automatically removes fully exited pools.
- `claim_reward()` now skips stakers that don't support the reward token instead of reverting.

2.4.3 Version 3 → 4

- Added per-crvUSD-vault deposit limits. The admin can cap the total required crvUSD across all `HybridVaults` that use a given crvUSD vault (e.g., limit total scrvUSD exposure). Configured via `set_crvusd_vault_limit()` or as part of `set_allowed_crvusd_vault()`.
- `HybridVaults` now report their required crvUSD to the factory after each `deposit()`, `withdraw()`, and `emergency_withdraw()`. The factory tracks per-vault and per-crvUSD-vault totals. Deposits that would exceed the vault limit revert.
- `set_crvusd_vault()` now resets the vault's tracked requirement to zero when switching vaults.
- `withdraw()` and `emergency_withdraw()` cache the stablecoin_fraction at deposit time and reuse it during oracle-failure withdrawals
- `withdraw()` now handles oracle failures: when `value_oracle()` reverts for any pool, the vault falls back to per-pool crvUSD tracking or proportional allocation reduction instead of reverting.
- Added `pool_downscale_factor` caching (`_downscale_w()`). Stablecoin fractions are cached locally by each `HybridVault`.

2.4.4 Version 4 → 5

- Removed `pool_downscale_factor` caching. All stablecoin fraction scaling now call the factory to query the live `stablecoin_fraction` value.
- Simplified `withdraw()` oracle failure handling: when the target pool's oracle fails (`pool_crvusd_before` or `pool_crvusd_after` is `max_value`), the function now reverts with "Oracle is broken" instead of falling back to proportional allocation reduction. `withdraw_stablecoins` is blocked in all oracle-failure paths.
- `emergency_withdraw()` oracle failure path now computes allocation reduction from actual crvUSD consumption (`crvusd_before - remaining_crvusd`, scaled by `stablecoin_fraction`) instead of proportional `previous_allocation * lt_shares / lt_supply`.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	2
Medium-Severity Findings	2
Low-Severity Findings	6
Informational Findings	4

- Missing Array Length Validation in Constructor Code Corrected
- deposit() Allows Specifying an Arbitrary Recipient Code Corrected
- _required_crvusd_for() Overestimates Required crvUSD Code Corrected
- stablecoin_allocation Is Shared Across Pools Code Corrected
- Missing ERC-20 Rescue Functionality Code Corrected
- Missing Owner Functionality Code Corrected
- Unused _remove_from_used() Function Code Corrected
- HybridVault Lacks emergency_withdraw() Support Code Corrected
- stablecoin_allocation May Retain Residual Value After Full Exit Code Corrected
- withdraw() Does Not Use receiver Parameter Code Corrected
- _add_to_used() Not Called for Debt-Only Deposits Code Corrected
- Imprecise Stablecoin Allocation Reduction on Withdrawal Code Corrected
- crvUSD Approval Not Revoked When Changing Vault Code Corrected
- Function Naming Code Corrected

6.1 Missing Array Length Validation in Constructor

Correctness High Version 1 Code Corrected

CS-YBHYVA-013

The `__init__()` constructor in `HybridVaultFactory.vy` iterates over `pool_ids` using a bounded loop. The break condition allows `i` to reach `len(pool_ids)`, causing an out-of-bounds access on the `pool_ids` array and reverting deployment.

Code corrected:

Break condition was adjusted. Now `len(pool_ids)` is unreachable.

6.2 `deposit()` Allows Specifying an Arbitrary Recipient

Design | High | Version 1 | Code Corrected

CS-YBHYVA-014

The `deposit()` function in `HybridVault.vy` accepts a `receiver` parameter that controls where LT shares (or staker shares when `stake` is true) are sent. The vault owner can set `receiver` to an external address, causing the minted shares to leave the vault.

The `_required_crvusd()` function calculates crvUSD backing requirements based on `pool.lt.balanceOf(self)`. When LT shares are sent to an external address, the vault no longer accounts for them, and the crvUSD backing requirement does not increase accordingly.

This allows the vault owner to repeatedly deposit and direct LT shares to an external wallet while the vault's crvUSD requirement remains low. The owner can effectively mint an unbounded amount of LT tokens while only maintaining crvUSD backing for the shares that remain inside the vault.

Code corrected:

`receiver` parameter was removed. LT tokens cannot be directly accessed by the `HybridVault` owner now.

6.3 `_required_crvusd_for()` Overestimates Required crvUSD

Correctness | Medium | Version 1 | Code Corrected

CS-YBHYVA-001

The `_required_crvusd_for()` function in `HybridVault.vy` computes the crvUSD value of a potential LT deposit as `value_in_amm * lt_shares // lt_supply`. Two factors cause this calculation to overestimate the result:

- `lt_supply` is read via `totalSupply()` before `LT.deposit()` triggers `_calculate_values()`, which adjusts the supply based on staking yield and admin fees. The stale `totalSupply` can differ from the post-recalculation value.
- Unlike `_required_crvusd()`, which adjusts for `liquidity.admin` by computing `value * (liquidity.total - max(liquidity.admin, 0)) // liquidity.total`, `_required_crvusd_for()` uses the raw `value_in_amm` without subtracting the admin fee portion. When `liquidity.admin > 0`, this includes value that is not claimable by LT holders.

Both factors inflate the `additional_crvusd` estimate used in `deposit()`. The overestimated value propagates into `stablecoin_allocation`, which is set to `previous_allocation + 2 * additional_crvusd` after the deposit. This leaves a larger allocation headroom than the actual position requires, allowing the LT pool to take on more stablecoin debt than intended.

A vault owner could exploit this by using a flash loan to deposit assets (inflating the allocation via the overestimated `additional_crvusd`), then immediately withdrawing. The withdrawal reduces `stablecoin_allocation` by `2 * (required_before - required_after)`, which is based on actual `balanceOf` changes rather than the inflated estimate. The net effect is a residual surplus in

`stablecoin_allocation` that persists after the position is closed, expanding the borrowing capacity of the `LT` pool beyond its intended limit.

Code corrected:

In Version 3, `market.staker.deposit()` call is performed in `deposit()` function, before `_required_crvusd()` query. This updates the `LT` values. Additionally, just like `_required_crvusd()`, admin fees are excluded from the token values.

6.4 `stablecoin_allocation` Is Shared Across Pools

Design | Medium | Version 1 | Code Corrected

CS-YBHYVA-002

`stablecoin_allocation` in `HybridVault.vy` is a single global counter shared across all pools, but reductions during `withdraw()` are applied to a specific pool's `stablecoin_allocation`. The global counter does not track which pool contributed what amount.

For example:

1. A `HybridVault` deposits into pool A and pool B, each adding 100 to the global `stablecoin_allocation` (total: 200).
2. User sends `LT` tokens of pool A directly into the `HybridVault`.
3. When withdrawing from pool A, `2 * (required_before - required_after)` can evaluate to full 200, reducing pool A's allocation by 200.
4. When withdrawing from pool B, the delta is 0.

The global `stablecoin_allocation` is reduced to 0 in total (200 + 0 = 200), but only pool A's allocation was reduced. This is effectively due to 2 problems: `LT` tokens can be sent directly and `stablecoin_allocation` is shared across pools.

However, even without step 2, over time, the price in pool A can rise and the price in pool B drop. This will create a similar situation.

Code corrected:

In Version 3, `stablecoin_allocation` is a mapping that tracks allocations for all pools separately. As a result, step 3 would only reduce A's allocation by 100 and step 4 would correctly reduce B's allocation by 100.

6.5 Missing ERC-20 Rescue Functionality

Design | Low | Version 1 | Code Corrected

CS-YBHYVA-015

There is no functionality allowing to access unexpected ERC-20 tokens held in a vault contract. In particular as a result, if the emergency admin were to perform a withdraw operation on a vault, the asset tokens would end up stuck in the vault.

Code corrected:

In Version 2, the function `recover_tokens()` was added to the vault contract: it allows sweeping any ERC-20 token other than the current crvUSD vault, and any yieldbasis LT or staked LT.

6.6 Missing Owner Functionality

Design Low Version 1 Code Corrected

CS-YBHYVA-004

Since an instance of `HybridVaultOwner` takes over the role of admin for the YB `Factory`, it needs to pass through all important admin functionality or they will become unavailable. `LT` allows the admin to call `distribute_borrower_fees()` with an arbitrarily high discount. The owner contract does not take this into account, as a result it is not possible to use the functionality after deploying Hybrid Vaults without resorting to the emergency admin.

Code corrected:

A function `lt_distribute_borrower_fees()` has been added to `HybridVaultOwner` in Version 3, along with `fill_staker_vpool()`, `set_allocator()`, `set_agg()`, and `set_flash()`.

6.7 Unused `_remove_from_used()` Function

Design Low Version 1 Code Corrected

CS-YBHYVA-005

In `HybridVault.vy`, the internal function `_remove_from_used()` is defined but never called. The counterpart function `_add_to_used()` is called during deposits (`HybridVault.vy`), but pools are never removed from `used_vaults` even after the user fully withdraws their position.

Thus, `used_vaults` array only grows. Since `used_vaults` is bounded by `MAX_VAULTS` (16), a user who deposits to 16 different pools and later fully withdraws from some of them would be unable to deposit to any new pools.

In Version 3, `_remove_from_used()` is called whenever a vault withdraws all its holdings from a given pool.

6.8 `HybridVault` Lacks `emergency_withdraw()` Support

Design Low Version 1 Code Corrected

CS-YBHYVA-006

`HybridVault.vy` does not expose an `emergency_withdraw()` function. When the AMM is killed, the emergency admin must call `LT.emergency_withdraw()` directly on behalf of the vault.

`LT.emergency_withdraw()` can have a negative `stables_to_return` value, meaning the caller must provide stablecoins to cover the debt. Since the emergency admin is the `msg.sender` in this scenario, the stablecoins are pulled from the emergency admin's own balance rather than from the `HybridVault`.

This creates an awkward operational flow where the emergency admin must fund the debt repayment out of pocket to rescue the vault's position, despite the vault itself potentially holding sufficient crvUSD to cover it.

`HybridVault.emergency_withdraw()` has been added in Version 3.

6.9 `stablecoin_allocation` May Retain Residual Value After Full Exit

Design Low Version 1 Code Corrected

CS-YBHYVA-007

`HybridVault.vy` tracks a single `stablecoin_allocation` variable to cap allocation reductions during withdrawals. `stablecoin_allocation` is incremented during `deposit()` by `2 * additional_crvusd` (derived from `_required_crvusd_for()`) and decremented during `withdraw()` by `min(2 * (required_before - required_after), stablecoin_allocation)`, where the delta comes from `_required_crvusd()`. `stablecoin_allocation` effectively prevents over-reduction of the allocation after withdrawal.

These two functions use different calculation methods: `_required_crvusd_for()` is a pre-deposit estimate, while `_required_crvusd()` reflects actual post-withdrawal changes. Price movements, admin fee accrual, and supply adjustments via `_calculate_values()` cause the two values to diverge. A full deposit followed by a full withdrawal can therefore leave `stablecoin_allocation` at a non-zero value.

Code corrected:

`_remove_from_used()` is executed for the YB LT pools that no longer have balance or staked balance. If `self.stablecoin_allocation[pool_id]` still remains, `_remove_from_used()` will try to deallocate it from the LT. `self.stablecoin_allocation[pool_id]` will be zeroed out after that.

6.10 `withdraw()` Does Not Use `receiver` Parameter

Correctness Low Version 1 Code Corrected

CS-YBHYVA-016

The `withdraw()` function always sends the assets to the caller, despite the `receiver` parameter.

Code corrected:

In Version 2, the function sends the funds to the receiver.

6.11 `_add_to_used()` Not Called for Debt-Only Deposits

Informational | Version 5 | Code Corrected

CS-YBHYVA-018

In `HybridVault.deposit()`, `_add_to_used(pool_id)` is only called when `assets > 0`. A deposit with `assets = 0` and `debt != 0` would increase the stablecoin allocation and receive LT shares, but the pool would not be tracked in `used_vaults`. As a result, `_required_crvusd()` would not account for the position, and the pool would not be properly managed on withdrawal or vault switching.

In practice, edge-case arbitrage or cryptopool imbalance could allow single-sided stablecoin deposits that produce non-zero LT shares while bypassing pool tracking.

Code corrected:

In Version 6, an `assert lt_shares > 0` check was added after `lt.deposit()`. Combined with the `assert debt <= 11 * additional_crvusd // 10` constraint, a deposit with `assets = 0` cannot produce meaningful LT shares without violating the debt limit, effectively preventing untracked positions.

6.12 Imprecise Stablecoin Allocation Reduction on Withdrawal

Informational | Version 3 | Code Corrected

CS-YBHYVA-019

The `withdraw()` function in `HybridVault.vy` computes the stablecoin allocation reduction as `2 * (required_before - required_after)`, where both values are obtained by calling `_required_crvusd()`. This function iterates over all pools in `used_vaults` and queries each pool's `amm.value_oracle()`.

Between the two `_required_crvusd()` calls, `market.lt.withdraw()` executes. After this, `value_oracle()` start reverting for the withdrawn pool.

Consider the scenario where all pool oracles succeed before the withdrawal but the target pool's oracle starts failing after `lt.withdraw()` changes AMM state. If `_required_crvusd()` handles this by returning `max_value(uint256)` as a sentinel for the failing pool, `required_after` and the per-pool `pool_crvusd_after` become unusable. The proportional fallback in this case would be `previous_allocation * lt_shares // lt_supply`, which is based on the stablecoin allocation rather than the actual oracle-derived backing.

A more precise `required_after` estimate would be: `required_before - pool_crvusd_before * lt_shares // lt_supply`. This estimate will also take into account other pool requirements that are not handles in `previous_allocation` proportional withdrawal.

Resolved:

In Version 5, the proportional fallback (`previous_allocation * lt_shares // lt_supply`) was removed from `withdraw()`. When the target pool's oracle fails after `lt.withdraw()`, the function now reverts with "Oracle is broken" instead of using an imprecise fallback. Users must use

`emergency_withdraw()` in this case, which computes allocation reduction from actual crvUSD consumption rather than proportional allocation.

6.13 crvUSD Approval Not Revoked When Changing Vault

Informational **Version 2** **Code Corrected**

CS-YBHYVA-009

`set_crvusd_vault()` in `HybridVault.vy` approves the new crvUSD vault to spend `CRVUSD` but does not revoke the approval for the previous vault. The old vault retains an unlimited `crvUSD` allowance from the `HybridVault` after it is replaced. If the old vault is later compromised or behaves unexpectedly, it can still pull crvUSD from the `HybridVault`.

Resolved:

The `set_crvusd_vault()` function now revokes the old vault's crvUSD approval by calling `CRVUSD.approve(old_vault.address, 0)` before setting and approving the new vault.

6.14 Function Naming

Informational **Version 1** **Code Corrected**

CS-YBHYVA-012

The function `HybridFactoryOwner.lt_set_amm_rate()` actually sets the fee by calling `LT.set_amm_fee()`, hence its name is surprising.

Code corrected:

In **Version 2**, the function was renamed to `lt_set_amm_fee()`.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 `emergency_withdraw()` Pulls Full Debt From the User

Informational **Version 3**

CS-YBHYVA-017

The `emergency_withdraw()` function in `HybridVault.vy` allows specifying `crvusd_from_wallet=True` to cover debt during emergency withdrawal. In this mode, the function pre-computes `max_needed` and calls `CRVUSD.transferFrom(msg.sender, self, max_needed)` to pull crvUSD from the owner's wallet before calling `lt.emergency_withdraw(lt_shares, self, self)`.

However, `max_needed` is a full portion of the debt that user holds. If `stables_to_return` is negative, `LT.emergency_withdraw()` will pull absolute value of it from the user:

```
stables_to_return: int256 = convert(withdrawn_cswap[0], int256) - convert(withdrawn_levamm.debt, int256)
```

In general, this will be a fraction of `max_needed`. Full debt cover might only be needed if `Cryptopool` is manipulated. However, this is expensive manipulation and benefits the owner of `HybridVault`.

7.2 Cannot Change crvUSD Vault Without Fully Exiting

Informational **Version 2** **Acknowledged**

CS-YBHYVA-008

`HybridVault.set_crvusd_vault()` requires the hybrid vault to hold zero crvUSD shares. The only way to reach this state is to fully withdraw from all YB markets to reset the required crvUSD balance to zero, and then withdraw from the current crvUSD vault. This means it is not possible to change the crvUSD vault on the fly without unwinding, even though this would not bring the vault into an invalid state.

7.3 Mismatched Array Lengths in Constructor

Informational **Version 1** **Acknowledged**

CS-YBHYVA-003

The `__init__()` constructor in `HybridVaultFactory.vy` iterates over `pool_ids` and `pool_limits` in parallel but does not verify that both arrays have the same length. If `pool_limits` contains fewer elements than `pool_ids`, the loop will attempt an out-of-bounds access on `pool_limits`, causing the deployment to revert.

Acknowledged:

Yield Basis considers this intended behavior.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Imprecise `_required_crvusd()` Evaluation

Note Version 1

`_required_crvusd()` in `HybridVault.vy` computes the total crvUSD backing requirement by iterating over all used pools. For each pool, it reads `lt.totalSupply()`, `amm.value_oracle()`, and `lt.liquidity()`. These values may be stale at the time of the call:

- `totalSupply()` may not reflect pending yield accrual or admin fee adjustments until `_calculate_values()` is triggered by the next `LT` interaction.
- `value_oracle()` returns a time-smoothed oracle value rather than the current spot price.

As a result, `_required_crvusd()` is an estimate rather than a precise accounting identity. This is acceptable by design. The backing requirement (configured via `stablecoin_fraction`, default 55%) acts as a safety buffer, not a strict invariant. The `_downscale()` multiplier provides additional margin. Furthermore, the `LT` contract independently enforces that actual debt stays below 50% of the allocation, serving as a second safety layer.

Consequently, the usage of `_required_crvusd()` to calculate stablecoin allocation reduction/increase is subject to imprecision. In general, the exact allocation required is price sensitive and depends on trades in AMM and Cryptopool. Thus, imprecise nature of `_required_crvusd()` does not pose a risk.

This behaviour was acknowledged by Yield Basis and part of the design.

8.2 Locked Stablecoins De-Correlated From Allocation

Note Version 1

Users must lock a configurable fraction (default 40%) of the value of their YB deposit in crvUSD. 40 crvUSD deposited by the user into `HybridVault` immediately allow increase in allocation for 200 crvUSD (50% - safety margin for the price increase). This backing requirement is enforced at deposit time and remains static afterward. The `LT` contract enforces that actual debt stays below 50% of the allocation (below 100 crvUSD), providing a safety margin for price increases.

However, the stablecoin debt requirement moves dynamically with price changes. If prices rise, the debt requirement increases (via arbitrage) while the locked crvUSD stays constant. As a result, the locked stablecoins in `HybridVault` may represent a smaller fraction of the actual allocation over time.

8.3 Oracle Failure Handling

Note Version 4

`HybridVault` relies on `AMM.value_oracle()`. However, this call to it can fail and this is handled by assuming `max_value(uint256)` as a requirement. One `HybridVault` can be invested into multiple YB pools. A single broken oracle in any pool blocks all deposits to all pools. For withdrawals - both `withdraw()` and `emergency_withdraw()` functions are supported for non-broken pools. However,

the crvUSD reduction computed from the withdrawal computed depending of exact scenario. Lets say User has Pools A and B. User tries to exit pool B.

Oracle State Truth Table for `withdraw()`

#	A	B before	B after	Path	Behavior
1	OK	OK	OK	**Happy path**	Allocation reduction is 2x `required_before - required_after` delta. Full delta for crvUSD withdrawal with solvency check. `update_vault_required` called with real value.
2	OK	OK	FAIL	**Revert**	`pool_crvusd_after` is `max_value(uint256)`. Reverts with "Oracle is broken". User must use `emergency_withdraw()` instead.
3	FAIL	OK	OK	**Delta path**	reduction = `2 * (pool_crvusd_before - pool_crvusd_after)`. `withdraw_stablecoins` blocked. `update_vault_required` skipped.
4	FAIL	OK	FAIL	**Revert**	`pool_crvusd_after` is `max_value(uint256)`. Reverts with "Oracle is broken". User must use `emergency_withdraw()` instead.

For pools where `value_oracle()` is broken from the start - only `emergency_withdraw()` works. `emergency_withdraw()` handles oracle failure for the target pool by computing allocation reduction from actual crvUSD consumption (`crvusd_before - remaining_crvusd`, scaled by `stablecoin_fraction`), capped by `stablecoin_allocation[pool_id]`.

8.4 Pool Limits Mechanism

Note **Version 2**

The deposit check compares total pool value (after deposit) against the user's effective limit. A user's effective limit is `global_pool_limit + personal_limit`. The check is: `pool_value + deposit_value <= effective_limit`.

Personal limits raise the ceiling for a specific user but do not reserve capacity. Example: global limit is 1000, pool value is at 1000. Admin grants a 100 personal limit to users A and B. User A deposits 100 (pool now at 1100). User B cannot deposit 100 because the pool value (1200) would exceed B's effective limit (1100).

8.5 Redundant Pool Iteration During Withdrawal

Note **Version 3**

The `withdraw()` function in `HybridVault.vy` calls `_required_crvusd()` both before and after the `lt.withdraw()` call to compute the allocation reduction as `2 * (required_before - required_after)`. Each invocation iterates over all pools in `used_vaults`, querying `amm.value_oracle()`, `lt.liquidity()`, `lt.totalSupply()`, and staker balances for every pool.

Realistically, only the pool being withdrawn from changes state during `lt.withdraw()`. All other pools' contributions to `_required_crvusd()` remain identical between the two calls and cancel out in the

`required_before - required_after` difference. The allocation reduction and the stablecoin withdrawal amount could be derived from the target pool's values alone, avoiding redundant queries that scale linearly with the number of active pools.

The only place where a global all-pools check is genuinely needed is inside `_withdraw_crvusd()`, where the backing invariant assert requires the total `_required_crvusd()` across all pools to verify sufficient scrvUSD remains.