Quantstamp

Yield Basis

Executive Summary

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

Type	DeFi AMM/Leveraged Liquidity
Timeline	2025-04-01 through 2025-04-16
Language	Vyper
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Leveraged Liquidity Paper
Source Code	• yield-basis/yb-core #16b1780
Auditors	• Cameron Biniamow Auditing Engineer • Jonathan Mevs Auditing Engineer • Gereon Mendler Auditing Engineer

Documentation quality	Low
Test quality	Medium
Total Findings	8 Fixed: 6 Acknowledged: 2
High severity findings	3 Fixed: 3
Medium severity findings	2 Fixed: 2
Low severity findings	1 Fixed: 1
Undetermined severity findings	0
Informational findings	2 Acknowledged: 2

Summary of Findings

Yield Basis is a decentralized finance system that facilitates leveraged liquidity provision and automated market making (AMM), with a focus on mitigating impermanent loss. Built on Curve's AMM framework, the core innovation is a 2x leverage mechanism that transforms the price behavior of liquidity positions to match that of the underlying tokens, effectively eliminating impermanent loss while preserving fee generation capabilities.

The protocol enables an admin to create a Yield Basis market for a specific Curve pool that includes crvUSD and a cryptocurrency, such as WBTC. Users can deposit cryptocurrency into Yield Basis and specify the amount of crvUSD they want the protocol to take on as debt to create a leveraged LP position in the Curve pool. Typically, the debt would be close in value to the deposited cryptocurrency. Once users deposit crypto, they are issued `yb` tokens, representing their share of Curve LP tokens in the Yield Basis protocol. Users can burn their `yb` tokens during a withdrawal to claim the crypto they initially deposited. Holders of `yb` tokens will earn fees from deposits and withdrawals. Optionally, users can stake in `LiquidityGauge`, a simple ERC4626 contract, and earn rewards based on the rate of supplying the LP tokens directly to the Curve pool.

Quantstamp was tasked with auditing the Yield Basis contracts to identify potential vulnerabilities and verify that the contracts operate as expected. Specifically, the `AMM`, `CryptopoolLPOracle`, `Factory`, `LT`, and `VirtualPool` contracts were in scope. The `LiquidityGauge` contract and all external contracts, such as the Curve contracts, were considered out of scope. The Yield Basis codebase consisted of Vyper contracts, a technical paper detailing the mechanisms and mathematical equations used in the protocol, and a test suite with moderate coverage.

The auditing process for the Yield Basis contracts has revealed several vulnerabilities that need to be addressed to ensure the security and functionality of the protocol. Key vulnerabilities identified include:

- Updating the `staker` address does not transfer the staker's balance to the new staker, resulting in incorrect calculations regarding the staked balance of `yb` tokens.
- The incomplete integration of flashloan functionality in the `VirtualPool` contract, which lacks critical validations required by ERC3156 and has an inaccessible function, as it is not marked as `external`.
- The `LT` contract does not update the staker address correctly after a market is created, which could lead to operational issues if the two contracts are out of sync.

The audit team recommends implementing robust validations to enforce safe operations, such as ensuring that addresses passed to critical functions are not zero and applying a two-step ownership transfer pattern for administrative privileges to prevent potential misconfigurations. It is also essential to enhance the test coverage for the `VirtualPool` contract to catch any erroneous functions and to fully document the logic pertaining to functions like `distribute_borrower_fees()`, ensuring their intent and functionality are clear.

Fix Review: The Yield Basis team has successfully addressed several vulnerabilities and suggestions in their system at commit `8b05ee9dec073941e7406cf8469e0e11797a436d`. Vulnerabilities YIELD-1 through YIELD-6 are confirmed as fixed. Issues YIELD-7 and YIELD-8 were acknowledged and deemed acceptable as per the client's design decisions. All auditor suggestions were fixed except for S7, which the client stated is desired behavior.

Beyond the fixes for vulnerabilities and suggestions listed in this report, the Yield Basis team made the following changes to the codebase:

1. Added functionality for emergency withdrawals to the `LT` contract.
2. Added functionality in the `AMM` and `LT` contracts to pause and unpause the contracts, altering the ability for users to deposit, withdraw, and swap.
3. The `VirtualPool` contract was updated to include comments indicating that some logic may not yet be implemented.
4. DAO contracts were added to the codebase during the fix review, but remained out of scope.

ID	DESCRIPTION	SEVERITY	STATUS
YIELD-1	**Staker Address Update Does Not Transfer Staker's Balance**	High	Fixed
YIELD-2	**No Fee Enforcement in `LT` Contract**	High	Fixed
YIELD-3	**Incorrect Flashloan Integration**	High	Fixed
YIELD-4	**Overleveraging After Allocator's Stablecoins Are Reclaimed**	Medium	Fixed
YIELD-5	**Staker Is Not Updated in Some Cases**	Medium	Fixed
YIELD-6	**Missing Setter Function for `staker_impl`**	Low	Fixed
YIELD-7	**Curve Cryptopool `donate()` Function Is Poorly Defined**	Informational	Acknowledged
YIELD-8	**`LT` Does Not Expose Burn Functionality**	Informational	Acknowledged

Assessment Breakdown

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

Disclaimer

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

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

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

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.

2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Scope

Files Included

Repo: `https://github.com/yield-basis/yb-core`

Included Paths: `contracts`

Files Excluded

- `contracts/LiquidityGauge.vy`

Operational Considerations

1. The Yield Basis contracts heavily depend on external contracts that were out of scope for this audit. While integrations of external contracts are reviewed, the external contracts may contain vulnerabilities that could lead to potential loss of funds in the Yeild Basis contracts or prevent functionality from working as expected. Specifically, the Yield Basis contracts rely on the following external contracts:
    - Curve `Twocrypto` pools: Used for supplied liquidity from users to create LP shares, which will be held by the Yield Basis protocol in the `AMM` contracts.
    - Curve `crvUSD ControllerFactory` : Set as the `mint_factory` in the `Factory.vy` contract. The `mint_factory` contract supplies crvUSD to the `Factory` contract and can reclaim its crvUSD for contract migration.
    - Curve `AggregateStablePrice2` oracles: Used to obtain the USD price of `STABLECOIN` in the Curve `Twocrypto` pools.
    - Curve `FlashLender` : Used by the `VirtualPool` contract to obtain a flash loan for asset exchanges in the `AMM` contract.
2. There are inherent front-running risks when trading on this platform. Users should always specify a tolerable amount of slippage.
3. `AMM` contracts should never be assigned as the `Factory` allocators. The YB admin should take special care when assigning these addresses. If an `AMM` were assigned, debt ratio accounting could be disrupted by supporting an arbitrary transfer.
4. The `VirtualPool` contract relies on a crvUSD flashloan provider to amplify price corrections through arbitrage. This only works with a sufficiently large flashloan provider without a lending fee.
5. The contracts rely on a provider of crvUSD as debt for leverage trading. Allocation is a multi-step process. It requires the allocator to give transfer approval to the factory, which is then processed by the admin calling `set_allocator()` in the factory to distribute them to individual markets using their `allocate_stablecoins()` function. Deallocation follows a similar reversed flow.
6. The `LT.withdraw_admin_fees()` function requires the `admin` to be a contract, specifically the `Factory` contract. However, the `LT` contract allows updates to the `admin` address, which can be updated to an EOA. Therefore, if the `admin` is updated to an EOA, the function `withdraw_admin_fees()` will fail during execution.

Key Actors And Their Capabilities

- `AMM`
    - `DEPOSITOR` (`LT` contract)
        - Can set the interest rate of the `AMM` .
        - Can adjust internal accounting for LP token deposits in the `AMM` contract, resulting from a user adding liquidity to a Curve pool via `LT.deposit()` .
        - Can adjust internal accounting for LP token withdrawals from the `AMM` contract, which result from a user removing liquidity from a Curve pool via `LT.withdraw()` .
        - Can collect interest fees generated in the `AMM` contract.
        - Can kill (pause) the `AMM` contract, preventing swaps, deposits, and fee collection.
        - Can unpause the `AMM` contract, allowing swaps, deposits, and fee collection.
        - Can update the exchange fee to a value of 10% or less.
    - Users
        - Can exchange Curve pool LP tokens and stablecoins (crvUSD).
        - Can initiate fee collection for the `DEPOSITOR` (`LT` contract).
- `Factory`
    - `admin`
        - Can call `add_market()` to create a new market for a Curve pool. The `admin` will set the market's `fee` , `rate` , and `debt_ceiling` .
        - Can deploy a `VirtualPool` and a `staker` contract for a previously created market via `fill_staker_vpool()` .
        - Can set the `mint_factory` contract address once. After the `mint_factory` is set, unlimited approval of `STABLECOIN` is granted from the `Factory` contract to the `mint_factory` . Therefore, the `mint_factory` can transfer unlimited `STABLECOIN` out of the `Factory` contract. The `Factory` contract cannot revoke the unlimited approval to the `mint_factory` .

        - Can set an `allocator` and their allocation by executing `set_allocator()`. During the execution of `set_allocator()`, `STABLECOIN` is transferred from the `allocator` to the `Factory` contract up to the `allocator`'s allocation. If the `allocator` has allocated more `STABLECOIN` than their allocation, the `allocator` is granted approval to transfer the difference between the allocated amount and the `allocator`'s allocation. The `allocator` must claim the difference by transferring the `STABLECOIN` from the `Factory` contract in a separate transaction. The `admin` can only transfer an allocation amount from the `allocator` if the `allocator` has previously granted approval to the `Factory` contract.
        - Can update the `agg` contract address, which is referenced by future deployments of the `CryptopoolLPOracle` contract, created in the `add_market()` function.
        - Can update the `flash` contract address, which is referenced by future deployments of the `VirtualPool` contract, created in the `add_market()` function.
        - Can update the `admin` address by transferring `admin` privileges to a new address.
        - Can update the `fee_receiver` address to a new address.
        - Can update the `emergency_admin` address, which grants access to pause or unpause the `AMM` and `LT` contracts.
        - Can update the minimum admin fee.
- `LT`
    - `admin` and `Factory.admin()`
        - Can set the `amm` contract address once.
        - Can update the `admin` address by transferring `admin` privileges to a new address.
        - Can set the interest rate of the `amm` contract.
        - Can set an allocator's allocation and transfer `STABLECOIN` from the `admin` (initially set as the `Factory` contract) to the `LT` contract when the allocator's allocation exceeds their allocated amount. If the allocator's allocated amount exceeds their allocation, the allocator is refunded the difference.
        - Can distribute borrower fees.
        - Can withdraw admin fees from the `LT` contract. Since admin fees are `yb` tokens, the `LT` contract mints `yb` tokens to the `fee_receiver` of the `Factory` contract.
        - Can initially set the `staker` address to a new address. If the address has a non-zero `yb` token balance, the balance is transferred to the `Factory.fee_receiver()`.
        - Can update the exchange fee in the `AMM` contract.
        - Can pause the `LT` and `AMM` contracts, preventing swaps, deposits, withdrawals, and fee collection.
        - Can unpause the `LT` and `AMM` contracts, allowing swaps, deposits, withdrawals, and fee collection.
    - Users
        - Can deposit the Curve pool deposit token and receive `yb` shares in return.
        - Can burn their `yb` shares to receive the respective Curve pool deposit tokens.
        - Can initiate the distribution of borrower fees, which collects crvUSD fees from the `AMM` contract and donates them to the Curve pool.
        - Can transfer their `yb` shares to another address.
        - Can emergency withdraw when the `LT` contract is paused.
- `VirtualPool`
    - Users
        - Can exchange Curve pool LP tokens and stablecoins (crvUSD).

Findings

YIELD-1 Staker Address Update Does Not Transfer Staker's Balance

High Fixed

Update

The client fixed the issue in commit `f797e3043714ac28f58aed6bdb059f3398368f91` and provided the following explanation:

```
This was already fixed on Apr 4 by not allowing setting staker when the staker was already set
```

File(s) affected: `contracts/LT.vy`

Description: If the `set_staker()` function is called and the `staker` address is updated without transferring the previous staker's `balanceOf` to the new staker, the following issues could arise:

1. The previous staker's balance would remain with the old staker's address, leaving the new staker with a zero balance. This could lead to a situation where the new staker cannot perform any operations that require a balance, such as withdrawing staked tokens.
2. Incorrect calculations of staked tokens would result in the `_calculate_values()` function producing different values than expected.

Recommendation: When the `set_staker()` function is executed, transfer the previous staker's balance to the new staker. Otherwise, consider implementing functionality to pause the protocol to prevent users from depositing or withdrawing while the staker is being updated and the old staker's balance is transferred to the new staker.

YIELD-2 No Fee Enforcement in `LT` Contract

High Fixed

Update

The client fixed the issue in commit `30bf611d68363724044c281ec60afda151aeb4ca`.
and provided the following explanation:

```
This was already fixed on Apr 5 by setting min_admin_fee in factory
```

File(s) affected: `contracts/LT.vy`

Description: `LT.min_admin_fee` is never assigned, preventing any admin fees to be charged from the LT contract.

Recommendation: Assign `min_admin_fee` during `LT` contract deployment. Optionally, allow the admin role to change its value. Further, test this for correctness.

YIELD-3 Incorrect Flashloan Integration

High · Fixed

Update
The client fixed the issue in commit `0fd7b45e0e8323a437804bcd9e6d0c373d0d0896` and provided the following explanation:

```
Mentioned issues were addressed in this commit, but this only can be considered fixed when VirtualPool
is actually tested (not as of commit time)
```

File(s) affected: `contracts/VirtualPool.vy`

Description: The `VirtualPool` contract makes use of a crvUSD flashloan provider to amplify user arbitrage for price correction. However, the flashloan integration is incomplete.

1. The `onFlashLoan()` function needs to be external according to ERC3156.
2. The `onFlashLoan()` function should verify the `initiator` and `token` addresses.
3. The `exchange()` function calls a `FLASH.ceiling()` function that appears to be undefined based on the intended integrated flash contract, Curve Flashlender.
4. The `exchange()` function should be `@nonreentrant`.

Recommendation: Make sure that the flashloan integration is complete and secure. Consider adding a test suite to further check the correctness of the `VirtualPool` contract.

YIELD-4
Overleveraging After Allocator's Stablecoins Are Reclaimed

Medium · Fixed

Update
The client fixed the issue in commit `4515defffff5b816ea0069ab842b71794ea1d398b` and provided the following explanation:

```
Added a condition in allocate_stablecoins() when deallocating
```

File(s) affected: `contracts/AMM.vy`, `contracts/LT.vy`

Description: If the `Factory` contract `admin` were to reclaim the `Factory`'s allocated crvUSD from the `AMM` contract via `LT.allocate_stablecoins()`, the `LT` contract debt could become higher than half of the available crvUSD. In such a case, the protocol could become insolvent, resulting in halted deposits and imbalanced reserves, potentially preventing the execution of the `AMM.exchange()` function.

Recommendation: Consider adding a check in the `LT.allocate_stablecoins()` function that verifies the maximum debt will not be reached after stablecoins are unallocated from the `AMM` contract.

YIELD-5 Staker Is Not Updated in Some Cases

Medium · Fixed

Update
The client fixed the issue in commit `f797e3043714ac28f58aed6bdb059f3398368f91` and provided the following explanation:

```
Already fixed in this commit on Apr 4
```

File(s) affected: `contracts/Factory.vy`

Description: If the `Factory` creates a market using `add_market()` before a `staker_impl` is set, a staker can be added later by calling the `fill_staker_vpool()` function. However, this function does not call the `LT.setStaker()` function to update the staker in the `LT` contract.

Recommendation: Add the missing external call to mirror the behavior in `add_market()`.

YIELD-6 Missing Setter Function for `staker_impl`

Low | Fixed

Update

The client fixed the issue in commit `d9b8eebf84b2bca4b761a86080fe381ffff6a0ba` and provided the following explanation:

```
Already fixed in this commit
```

File(s) affected: `contracts/Factory.vy`

Description: In the function `fill_staker_vpool()`, a `staker` contract is deployed for the given `market` if the `staker_impl` contract address is set and if the `staker` contract was not already deployed from executing `add_market()`. However, the `staker_impl` contract address is not updatable in the `Factory` contract. Therefore, if the `staker_impl` is not set during deployment, the `staker` contract will not be deployed when `fill_staker_vpool()` is called as the following code block will never execute:

```
if market.staker == empty(address) and self.staker_impl != empty(address):
    market.staker = create_from_blueprint(
        self.staker_impl,
        market.lt)
```

Recommendation: Create a setter function for the `staker_impl` address.

YIELD-7 Curve Cryptopool `donate()` Function Is Poorly Defined

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

```
min_amount is minimal amount of LP token which could have been minted in the donation (since donation
is not symmetric). This will be rechecked once Curve cryptopool implementation will be fully done.
```

File(s) affected: `contracts/LT.vy`

Description: Using the `distrubute_borrower_fees()` function, anyone can force the `LT` contract to collect the fees from the releveraging AMM and donate them to the underlying curve liquidity pool. It is unclear how this function works in detail, as only a stub is provided. Therefore the purpose of the `min_amount` parameter is also unclear.

Recommendation: Make sure that this integration is correct.

YIELD-8 `LT` Does Not Expose Burn Functionality

Informational | Acknowledged

Update

The client acknowledged the issue and provided the following explanation:

```
Proper token burns are disallowed to, for example, limit LP token inflation attacks. However, indeed,
people are free to transfer the token to an address which cannot be accessed
```

File(s) affected: `contracts/LT.vy`

Description: The `LT` contract also serves as a ERC20 liquidity token. The `_transfer` function attempts to stop token burns by disallowing transfers to the zero address or the token contract, but no other burn function is exposed to properly handle this. Users may still choose to burn tokens by transferring them to irretrievable addresses instead, making lively tokens harder to track.

Recommendation: Expose a `burn()` function to properly handle this.

Auditor Suggestions

S1 Missing Test Suite for `VirtualPool` Contract

Fixed

Update

The client fixed the suggestion in commit `7ed1469d75a1922171e5fbd0ae3834839a7afd96` and provided the following explanation:

```
7ed1469d75a1922171e5fbd0ae3834839a7afd96
  6ca46c5205d4b204bdf2000a56f0905aa91cc14b
  ca95f0bc942d633b2bf4e7b5c2ad78e7f15ab293
  5fb6c13d4fdabf274acb450da7828041d63adc4e
  Test suite is written and few bugs are fixed. In addition, a possibility to replace VirtualPool
implementation if it is different on a live pool is inclided
```

File(s) affected: `contracts/VirtualPool.vy`

Description: The codebase contains a test suite for most major contracts, but not for the `VirtualPool`.

Recommendation: We highly recommend adding a comprehensive test suite to cover the `VirtualPool` integrations and calculations.

S2 Insufficient Input Validation

Fixed

Update

The client fixed the suggestion in commit `3e3a542167b28adb35dd39d4715d14dcb700b473` and provided the following explanation:

```
Fixed in this and few other commits. Few comments:
    • fee_receiver can be 0x0 – that's a legitimate situation where claim of admin fees won't be
      allowed (code adjusted accordingly)
    • flash lender can be reset to 0x0 – also ok to keep this possibility
```

File(s) affected: `contracts/Factory.vy`, `contracts/AMM.vy`, `contracts/LT.vy`

Description: The following contract functions lack sufficient input validation, which can lead to incorrect contract configuration. Since these functions are all permissioned, there is no risk of users altering the contract state to undesired values. However, insufficient validation could result in the Yield Basis team having to redeploy contracts with the correct configuration.

1. `Factory.__init__()`:
    - `STABLECOIN` is not checked against the zero address.
    - `amm_impl` is not checked against the zero address.
    - `lt_impl` is not checked against the zero address.
    - `price_oracle_impl` is not checked against the zero address.
    - `agg` is not checked against the zero address.
    - `fee_receiver` is not checked against the zero address.
    - `admin` is not checked against the zero address.
2. `Factory.set_agg()`:
    - `agg` is not checked against the zero address.
3. `Factory.set_mint_factory()`:
    - `mint_factory` is not checked against the zero address.
4. `Factory.set_admin()`:
    - `new_admin` is not checked against the zero address.
5. `LT.set_staker()`:
    - `staker` is not checked against the zero address.
6. `AMM.set_rate()`:
    - Consider enforcing bounds for the rate that can be assigned.

Recommendation: We recommend adding the relevant checks.

S3 Inconsistent Variable Naming

Fixed

Update

The client fixed the suggestion in commit `c5c88f7d5d1f15bd205c193ebb2edac61f4506e2` and provided the following explanation:

```
1. c5c88f7d5d1f15bd205c193ebb2edac61f4506e2 Rename to ASSET_TOKEN
```

2. e418b87e491f16c5f5f9d4de35ec4aacb150599e Rename to cryptopool only in LT contract because AMM can potentially work with plain assets (universal)
3. Admin can be either EOA or factory which has admin, contained in the same storage varaible admin . This is by design
4. 80def7a6acd947368b5f496295ecd3ca29586835, 440e04b60a9ca3099f43619f3df6ba06a3ea7040

File(s) affected: `contracts/Factory.vy`, `contracts/LT.vy`, `contracts/AMM.vy`, `contracts/VirtualPool.vy`

Description: Throughout the codebase, numerous instances exist where the variable naming of the same contract differs, leading to decreased readability. The following contract variables use different names for the same address:

1. In the `Factory` contract, `market.collateral_token` is set as the deposit token of the Curve pool. In the `LT` contract, the same deposit token is set as `DEPOSITED_TOKEN`. In the `VirtualPool` contract, the deposit token is set as `CRYPTO`. Consider changing `LT.DEPOSITED_TOKEN` and `VirtualPool.CRYPTO` to `COLLATERAL_TOKEN`.
2. In the `Factory` contract, `market.cryptopool` is set as the address of the Curve pool. In the `LT` and `AMM` contract, the same Curve pool is set as `COLLATERAL`. Consider changing `COLLATERAL` to `CRYPTOPOOL` in the `LT` and `AMM` contracts.
3. In the `AMM` contract, the `DEPOSITOR` is set as the `LT` contract address upon deployment. Consider changing `DEPOSITOR` to `LT_CONTRACT`.

Recommendation: Implement the mentioned variable name changes for improved readability.

S4 Application Monitoring Can Be Improved by Emitting More Events

Fixed

Update

The client fixed the suggestion in commit `8b05ee9dec073941e7406cf8469e0e11797a436d` and provided the following explanation:

```
f3bb88a3df2ffc16894b186654ebebbaaca918f5
  762c11438a4a399e3c20de23abd49423ee0b6d5b
  d70b7f600c5e8ff66cac3b7976cca21c73afcaba
  dfba913a7049c893f4ee41e6a6f9040e36f447b7
```

File(s) affected: `contracts/Factory.vy`, `contracts/LT.vy`, `contracts/AMM.vy`

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

1. `Factory.fill_staker_vpool()`: Emit an event containing the `market.virtual_pool` and `market_staker` addresses if those contracts are deployed.
2. `LT.set_amm()`: Emit an event containing the `amm` and `agg` addresses.
3. `LT.allocate_stablecoins()`: Emit an event containing the `allocator`, `stablecoin_allocation`, and `stablecoin_allocated`.
4. `LT.distribute_borrower_fees()`: Emit an event containing the `discount` and `amount`.
5. `AMM.set_killed()`: Emit an event containing the status of `is_killed`.

Recommendation: Consider emitting the events.

S5 Precision Loss

Fixed

Update

The client fixed the suggestion in commit `9b1ecc37a55f50e5b712a5375fc564a1396a375f` and provided the following explanation:

```
Re-arranged calculating constants in init: exactly the same result but reads better.
  Other cases for precision loss are NOT fixed because fixing could potentially cause overflows. In
general, I am trying to avoid O(WAD)**3 appearing anywhere during computations due to this reason.
```

File(s) affected: `contracts/AMM.vy`

Description: Precision loss occurs when multiplying after division. The following functions may experience precision loss during calculations:

1. `AMM.__init__()` when calculating `LEV_RATIO`, `MIN_SAFE_DEBT`, and `MAX_SAFE_DEBT`
2. `AMM.get_x0()` when calculating the `D` value
3. `AMM.withdraw()` when calculating the `withdrawn` value
4. `AMM._calculate()` when calculating `D`.

Recommendation: Refactor the calculations to minimize precision loss by avoiding multiplying values after dividing them.

S6 Code Improvements

Fixed

Update

The client fixed the suggestion in commit `6194bdce2970c29dcec689bc611d86758e416dc4` and provided the following explanation:

```
6194bdce2970c29dcec689bc611d86758e416dc4 instead of immutable, fee should have a setter!
  6113adeb5b20d47b467ae56b61af5d23c5a587ad LEVERAGE made public
  DEPOSITED_TOKEN_PRECISION already removed
```

File(s) affected: `contracts/AMM.vy`, `contracts/Factory.vy`, `contracts/LT.vy`

Description:

1. `AMM.fee` can be made immutable.
2. In `Factory.vy`, make `LEVERAGE` public.
3. `DEPOSITED_TOKEN_PRECISION` in `LT.vy` is not public and never used.

Recommendation: Consider including the suggestions.

S7 Critical Role Transfer Not Following 2-Step Pattern

Acknowledged

Update

The client acknowledged the suggestion and provided the following explanation:

```
In prod, it will be the DAO which is the admin, not EOA. Having a timelock on top of the DAO creates a
need to vote for every change twice -> not good. Therefore removing that.
```

File(s) affected: `contracts/Factory.vy`, `contracts/LT.vy`

Description: Consider reassigning the admin in `Factory.set_admin()` and `LT.set_admin()` in a two-step process. The pending admin is stored in the contract, and the admin is only transferred once the new admin claims the role. This prevents the contract from getting bricked by accidentally assigning an address that is not controlled by the team.

Recommendation: Implement a two-step ownership transfer.

S8 Unverified Aggregator Price Asset

Fixed

Update

The client fixed the suggestion in commit `fd0b7d906b886e40f6480d30f1fc4630a63b7fb1` and provided the following explanation:

```
Included rough validation by price
```

File(s) affected: `contracts/CryptopoolLPOracle.vy`

Description: In the `CryptopoolLPOracle` contract, the price aggregator (`AGG`) contract address is set during deployment. However, no validation ensures the `AGG` contract returns the USD price for the correct asset. Therefore, if the wrong price aggregator contract is set, the functions `price()` and `price_w()` would return incorrect data.

Recommendation: During deployment of the `CryptopoolLPOracle` contract, validate that the `AGG.STABLECOIN()` matches `POOL.coins(0)`.

S9 Disconnected Configuration Updates

Fixed

Update

The client fixed the suggestion in commit `5fb6c13d4fdabf274acb450da7828041d63adc4e` and provided the following explanation:

```
Flash can indeed be changed. That affects only VirtualPools, so added ability to replace those (which
can also receive further improvenents by themselves)
  As for oracles, it is by design that they cannot be changed. DAO must not be able to rug existing
oracles
```

File(s) affected: `contracts/CryptopoolLPOracle.vy`, `contracts/Factory.vy`, `contracts/VirtualPool.vy`

Description: The functions `set_agg()` and `set_flash()` in the `Factory` contract update the `agg` and `flash` contract addresses, respectively. After calling either function and updating the contract address, future deployments of the `CryptopoolLPOracle` and the `VirtualPool` contracts via `add_market()` and `fill_staker_vpool()` will reference the updated `agg` and `flash` contracts. However, any previously deployed `CryptopoolLPOracle` or `VirtualPool` contracts will still reference the `agg` or `flash` contracts set at the time of deployment. Thus, if a vulnerability is found in the `agg` or `flash` contracts, requiring the contracts to be updated via `set_agg()` or `set_flash()`, all previously deployed `CryptopoolLPOracle` and `VirtualPool` contracts will continue to interact with the vulnerable `agg` or `flash` contracts.

Recommendation: Consider refactoring the `CryptopoolLPOracle` and `VirtualPool` contracts so that they get the `agg` and `flash` contract addresses through external calls to the `Factory` contract. When the `agg` or `flash` contract addresses are updated, all deployments of the `CryptopoolLPOracle` and `VirtualPool` contracts reference the correct `agg` and `flash` contracts.

S10 Enforce Decimals of `STABLECOIN` and `COLLATERAL`

Fixed

Update
The client fixed the suggestion in commit `d3aace71ade4278ff6f5462eae04f9b4af048b1a`.

File(s) affected: `contracts/Factory.vy`

Description: While it is clear that the stablecoin and crypto pools used in the system are intended to be crvUSD or Curve Cryptopools, accounting will be severely affected if tokens or LP tokens used are different than 18 decimals. Therefore, this should be enforced at the Factory level.

Recommendation: In `Factory.__init__()` ensure that the `STABLECOIN` assigned has 18 decimals, or that it is the expected `crvUSD` contract. In `Factory.add_market()`, ensure the LP token has 18 decimals.

S11 `LT.withdraw_admin_fees()` Will Revert in the Case of Negative `v.admin`

Fixed

Update
The client fixed the suggestion in commit `63270ea0854f751fc50c847cc2bb6868354b047a` and provided the following explanation:

```
Custom error added
```

File(s) affected: `contracts/LT.vy`

Description: `LT.withdraw_admin_fees()` will revert if `v.admin` is negative, as that value is attempted to be converted to a uint256. Instead of reverting due to an integer issue, revert earlier with a custom error.

Recommendation: Assert that `v.admin` is non-negative before converting to `uint256`.

Definitions

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

Appendix

File Signatures

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

Files

Repo: `https://github.com/yield-basis/yb-core`

- `a3a...1a5 ./contracts/AMM.vy`
- `dd1...783 ./contracts/CryptopoolLPOracle.vy`
- `541...50b ./contracts/Factory.vy`
- `85a...92c ./contracts/LT.vy`
- `5ec...95d ./contracts/VirtualPool.vy`

Test Suite Results

The test suite included 15 test cases. All test cases were run successfully with the following commands.

Fix Review Update: Additional tests were added to the test suite. All 19 test cases were run successfully.

```
> pipx install virtualenv
> virtualenv -p python3 .venv
> source .venv/bin/activate
> pip3 install poetry
> poetry add git+https://github.com/vyperlang/titanoboa.git#ee4cb70a5e41713dd863f4fa1e0b6d8c53180cd2
> poetry install
> pytest -n <num_of_cores> -vv
```

```
============================================================================================= test
session starts
==============================================================================================
platform linux -- Python 3.12.3, pytest-8.3.5, pluggy-1.5.0 -- /home/appuser/workspace/projects/AT-
2627/yield_basis-yb-core-master-github~full/.venv/bin/python
cachedir: .pytest_cache
hypothesis profile 'default' -> deadline=timedelta(milliseconds=1000000),
database=DirectoryBasedExampleDatabase(PosixPath('/home/appuser/workspace/projects/AT-2627/yield_basis-
yb-core-master-github~full/.hypothesis/examples'))
rootdir: /home/appuser/workspace/projects/AT-2627/yield_basis-yb-core-master-github~full
configfile: pyproject.toml
plugins: titanoboa-0.2.5, cov-6.0.0, anyio-4.9.0, hypothesis-6.129.1, forked-1.6.0, xdist-3.6.1
8 workers [15 items]
scheduling tests via LoadScheduling

tests/amm/test_unitary.py::test_view_methods
tests/lt/test_factory.py::test_create_market
tests/amm/test_unitary.py::test_exchange
tests/amm/test_unitary.py::test_deposit_withdraw
tests/lt/test_factory.py::test_factory
tests/amm/test_stateful.py::test_stateful_amm
tests/amm/test_unitary.py::test_set_rate
tests/amm/test_adiabatic_trade.py::test_adiabatic
[gw5] [  6%] PASSED tests/amm/test_unitary.py::test_exchange
tests/lt/test_unitary.py::test_stake
[gw3] [ 13%] PASSED tests/amm/test_unitary.py::test_view_methods
tests/lt/test_unitary.py::test_allocate_stablecoins
[gw2] [ 20%] PASSED tests/amm/test_unitary.py::test_set_rate
tests/lt/test_unitary.py::test_informational
[gw6] [ 26%] PASSED tests/lt/test_factory.py::test_factory
tests/lt/test_unitary.py::test_collect_fees
[gw7] [ 33%] PASSED tests/lt/test_factory.py::test_create_market
[gw6] [ 40%] PASSED tests/lt/test_unitary.py::test_collect_fees
```

```
[gw5] [ 46%] PASSED tests/lt/test_unitary.py::test_stake
[gw3] [ 53%] PASSED tests/lt/test_unitary.py::test_allocate_stablecoins
[gw2] [ 60%] PASSED tests/lt/test_unitary.py::test_informational
[gw4] [ 66%] PASSED tests/amm/test_unitary.py::test_deposit_withdraw
tests/lt/test_unitary.py::test_deposit_withdraw
[gw4] [ 73%] PASSED tests/lt/test_unitary.py::test_deposit_withdraw
[gw1] [ 80%] PASSED tests/amm/test_stateful.py::test_stateful_amm
tests/lt/test_st_staker.py::test_price_return
[gw0] [ 86%] PASSED tests/amm/test_adiabatic_trade.py::test_adiabatic
tests/lt/test_price_return.py::test_price_return
[gw1] [ 93%] PASSED tests/lt/test_st_staker.py::test_price_return
[gw0] [100%] PASSED tests/lt/test_price_return.py::test_price_return

========================================================================= 15 passed in 4663.21s (1:17:43)
========================================================================
```

Changelog

- 2025-04-17 - Initial report
- 2025-04-25 - Final report

About Quantstamp

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

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

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

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

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

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