PUBLIC

Code Assessment

of the PT Oracle

Smart Contracts

27 January, 2026

Produced for

Curve

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Curve Team,

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

Curve implements an oracle for pricing Pendle Principal Tokens (PT) by applying a time-based linear discount to the underlying asset prices.

The most critical subjects covered in our audit are access control, functional correctness, and precision of arithmetic operations. The previously reported Incorrect Discount Parameters Check when updating the discount parameters has been fixed, and security regarding these subjects is high.

Other general subjects covered include gas efficiency and integration in Curve. Security regarding all the aforementioned subjects is high.

The note Oracle Usage in Curve LlamaLend Markets describes important economic implications of using this oracle in a Curve LlamaLend AMM that are out of scope for this review. We recommend that the Curve team carefully consider them before deploying this oracle in production.

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	0
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	0

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 assessment was performed on the source code files inside the PT Oracle 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	19 September 2025	5e0615e9af72d675ac840d2c729adfed14676027	Initial Version
2	17 December 2025	a4c92f9f3189cd2d2487b355c396bcf14f6910b0	Fixes

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

The only file that was in scope for this review is:

- `contract/PTOracle.vy`

2.1.1 *Excluded from scope*

The underlying oracle, Pendle PT token, tests, deployment scripts, and documentation files were excluded from the scope of this review.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

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

Curve offers a discounted price feed for a Pendle Principal Token (PT) relative to its underlying asset. The contract computes the discounted PT price based on a linear discount model parameterized by a `slope` and `intercept`, with respect to the PT's time to maturity. It integrates with an external underlying price oracle and enforces access-controlled updates to its discount parameters.

The `PTOracle` contract is the sole contract in scope. It is non-reentrant by default and inherits role-based access control functionality from the imported snekmate's `access_control` module.

Its primary purpose is to provide a PT price that reflects both the underlying asset's spot price and a time-decaying linear discount, suitable for use in Curve LlamaLend when needing a fair valuation of a non-matured PT.

Price Computation

- The oracle queries the configured underlying oracle for the current price of the underlying asset.
- If the PT has expired (`block.timestamp >= pt_expiry`), the oracle simply returns the underlying price.
- Otherwise, it applies a linear discount model defined as:

$$discount = \frac{slope \times time\_to\_maturity\_years}{10^{18}} + intercept$$

The discount reduces linearly with time to maturity and is bounded to ensure it does not exceed full price.

- The discounted price is then:

$$price = \frac{underlying\_price \times (10^{18} - discount)}{10^{18}}$$

The oracle exposes two pricing functions, according to the interface expected by LlamaLend's `AMM`:

- `price()` a view function that calculates and returns the current price on demand.
- `price_w()` a state-changing function that updates the cached price and timestamp to improve gas efficiency.

Discount Parameter Management

- The linear discount parameters, `slope` and `intercept`, can be updated by accounts holding the **MANAGER_ROLE**.
- Updates are rate-limited by `min_update_interval` to mitigate rapid parameter manipulation.
- To guard against abrupt market shifts, the contract enforces maximum per-update change limits: `max_slope_change` and `max_intercept_change`.
- A helper function, `set_slope_from_apy()`, allows managers to derive the slope from a desired annualized yield (APY), automatically setting the intercept to zero.

Administration

All privileged operations are controlled by roles granted at deployment:

- *ADMIN_ROLE*: overall administrative authority; the role admin for all subordinate roles.
- *MANAGER_ROLE*: permitted to adjust discount parameters subject to rate limits.
- *PARAMETER_ADMIN_ROLE*: permitted to adjust update intervals and per-update change limits.

2.2.1 Changelog

In Version 2, only fixes to issues raised in Version 1 were implemented. No new features or changes to existing functionality were introduced.

2.3 Trust Model

The contract is not upgradeable by design.

Economic Assumptions

- This system assumes that a linear relationship between time to maturity and discount is a reasonable approximation for the PT's fair value. This assumption, along with the specific parameter values and the accuracy of the price provided by the oracle, is considered out of scope for this review.
- This oracle is intended to be used as a collateral price oracle for Curve LlamaLend markets. The note Oracle Usage in Curve LlamaLend Markets describes important economic implications of this design choice that are out of scope for this review.

External Dependencies

- **Underlying Oracle**: Assumed to provide an accurate and up-to-date spot price for the underlying asset, expected to be manipulation-resistant and never reverting.
- **Pendle PT**: Assumed to comply with the interface and return a fixed, immutable expiry timestamp.

Roles

The contract uses a strict role-based access control scheme:

- **ADMIN_ROLE** (fully trusted):
    - Can grant or revoke the other two roles.
    - Risk: A compromised admin can replace managers or parameter admins and manipulate discount parameters maliciously.
- **MANAGER_ROLE** (fully trusted):
    - Can adjust discount parameters within the configured limits and intervals.
    - Risk: Misconfiguration or collusion may lead to unrealistic pricing or price jumps, potentially harming integrators relying on the oracle.
- **PARAMETER_ADMIN_ROLE** (fully trusted):
    - Can adjust operational safety limits such as rate limits and change caps.
    - Risk: Setting limits too high or too low may disrupt timely parameter updates or allow excessive market manipulation.
- **End Users and Integrators**
    - Consumers of the oracle price (e.g., lending protocols, AMMs) are assumed to trust the oracle's parameter configuration and the underlying oracle.
    - End users have no special privileges, they can call public view functions.

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:

- 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	0
Medium-Severity Findings	1

- Incorrect Discount Parameters Check Code Corrected

Low-Severity Findings	0
Informational Findings	5

- Code Duplication Code Corrected
- Gas Savings Code Corrected
- Incorrect Comments Code Corrected
- Indexed Event Code Corrected
- Missing NatSpec Comments Code Corrected

6.1 Incorrect Discount Parameters Check

Correctness Medium Version 1 Code Corrected

CS-CURVE-PT-ORACLE-001

In `_update_discount_params()`, it is essential that the new slope and intercept values are set in such a way that `discount` is always smaller than or equal to `1e18`, as otherwise, `_calculate_price()` would revert due to an underflow when computing `DISCOUNT_PRECISION - discount`.

In practice, because `time_to_maturity_years` is always decreasing over time, it is sufficient to ensure that the initial discount at the time of calling `_update_discount_params()` is less than or equal to `1e18`:

```
# Linear discount with 1e18 precision: discount = (slope * time_to_maturity_years) / 1e18 + intercept
discount: uint256 = (
    self.slope * time_to_maturity_years
) // DISCOUNT_PRECISION + self.intercept
assert discount <= DISCOUNT_PRECISION, "discount exceeds precision"
```

However, this check relies on `self.slope` and `self.intercept` that have not yet been updated to the new values. Therefore, it is possible for a malicious or careless Manager to set new discount parameters that would make the oracle consistently revert, effectively making it unusable.

Code corrected:

The check has been updated to use the new slope and intercept values instead of the current ones, ensuring that the discount will always be valid after the update.

6.2 Code Duplication

Informational | Version 1 | Code Corrected

CS-CURVE-PT-ORACLE-002

`_update_discount_params()` and `_calculate_price()` both contain the same logic to compute the discount, which is currently duplicated. This logic could be refactored into a separate internal function to reduce code duplication and improve maintainability.

Code corrected:

Both functions have been refactored to use a new internal function `_calculate_discount()` that encapsulates the shared logic for computing the discount.

6.3 Gas Savings

Informational | Version 1 | Code Corrected

CS-CURVE-PT-ORACLE-003

The following gas optimizations were identified in the codebase:

1. `pt` could be made immutable as it is set only once in the constructor.
2. `underlying_oracle` could be made immutable as it is set only once in the constructor.
3. In `__init__()`, when logging the `PriceUpdated`, the storage load of `self.last_price` could be avoided as the value was stored just before.
4. In `_update_discount_params`, `self.slope` and `self.intercept` are loaded from storage multiple times. Caching them in local variables would save gas.

Code corrected:

All suggested gas optimizations have been implemented.

6.4 Incorrect Comments

Informational | Version 1 | Code Corrected

CS-CURVE-PT-ORACLE-004

1. The following comment is incorrect as `max_slope_change` and `max_intercept_change` prevent any change to their respective parameters if set to zero:

```
# Limit variables for slope and intercept changes (0 = no limit)
```

2. The following comment is misleading as this variable is also used in `set_slope_from_apy()`:

```
# Rate limiting for set_linear_discount
last_discount_update: public(uint256)
```

Code corrected:

Both comments have been updated to accurately reflect the behavior of the code.

6.5 Indexed Event

Informational Version 1 Code Corrected

CS-CURVE-PT-ORACLE-005

The `PriceUpdated` event indexes the new price. This is uncommon as the price is quasi-continuous and not discrete.

Code corrected:

The event was removed from the codebase.

6.6 Missing NatSpec Comments

Informational Version 1 Code Corrected

CS-CURVE-PT-ORACLE-006

Except for `set_slope_from_apy()`, all external functions lack NatSpec comments describing their purpose, parameters, and return values. While the function names are mostly self-explanatory, adding NatSpec comments would enhance code readability and maintainability, especially for new developers or auditors unfamiliar with the codebase.

Code corrected:

NatSpec comments have been added to all external functions.

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 Out of Date Price in `price_w` Function

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

CS-CURVE-PT-ORACLE-009

While the function `price` always returns up-to-date prices with respect to the underlying price oracle, `price_w` might return a cached price if both of the following are true:

- Expiry has not been reached yet
- The last call to `price_w` was in the same block as the current call

In that case, the function will return `self.last_price`. This could be problematic if the underlying price oracle can have multiple updates in the same block, as the PT oracle would not reflect the latest of those updates.

Acknowledged:

This finding was acknowledged, and the following explanation was provided:

```
Same-block caching is a deliberate gas optimization.

Underlying Oracle Considerations:

* Chainlink: Cannot have multiple updates per block - no staleness concern
* Custom oracle: May have multiple updates, but practical impact is minimal within a single block

If fresh price is required: Use the price() view function which always computes current price.
```

7.2 Price Bump

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

CS-CURVE-PT-ORACLE-007

Given a fixed underlying price, the PT price returned by this oracle might have sudden jumps at certain times:

1. Right after the PT expiry, if `intercept` is non-zero, the price jumps from a discounted value to the full underlying price
2. When the `slope` and `intercept` are set

The Manager of the contract should ensure that the parameters are set in a way that avoids unexpected price jumps that could hurt users of the oracle.

Acknowledged:

This finding was acknowledged, and the following explanations were provided:

```
Price jumps at expiry (if intercept != 0) and on parameter updates are inherent to the
linear discount model. Mitigated by:

* max_slope_change and max_intercept_change limits constrain update magnitudes
* Managers are expected to reduce intercept gradually as expiry approaches
```

Moreover, parameter management best practices will be documented.

7.3 Stale `last_price` After Expiry

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

CS-CURVE-PT-ORACLE-008

After the PT expiry, the price returned by `price_w()` is no longer stored in `last_price`. Integrators relying on the stored price may be affected if they do not switch to calling the view function `price()` instead.

Acknowledged:

This finding was acknowledged, and the following explanation was provided:

```
Intentional behavior. Post-expiry, both price() and price_w() return
the underlying oracle price directly.
The last_price storage is not updated to save gas since caching provides
no benefit after expiry.
```

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 Non-Zero `intercept`

Note Version 1

The following are explanations from the developers regarding the rationale for allowing a non-zero `intercept` in the linear discount model:

The intercept effectively shifts the oracle price along the y-axis. Since lowering the intercept would directly reduce prices and could prematurely trigger liquidations, we do not anticipate adjusting the intercept downward. However, there are practical scenarios where increasing the intercept may be necessary.

This can be required when the observed market price deviates materially from the oracle price, for example due to:

1. **Supply-demand imbalances in the Pendle AMM**, which can cause persistent deviations from fair value.
2. **Fundamental changes in the underlying APY**, driven by structural or economic factors rather than short-term volatility.

If there is a structural shift in the underlying earnings and the discount model is systematically underpricing the PT, we may choose to adjust both the intercept and the slope so that the oracle better reflects the true underlying value.

Adjusting only the slope can lead to undesirable behavior, such as the discounted price reaching parity with the underlying asset too early or too late relative to maturity. For instance, if the slope is increased too aggressively, parity may be reached well before the intended maturity window. In such cases, jointly adjusting both the slope and the intercept allows parity to be reached at the appropriate time while preserving a more reasonable and gradual discount price path.

Additionally, during market initialization and near maturity, PT markets tend to exhibit low liquidity and elevated volatility. In these periods, markets may be paused and only activated once sufficient liquidity and stability are present. Upon activation, the oracle price must accurately reflect the underlying asset, and having the flexibility to set a non-zero intercept allows the model to align with observed market conditions from the outset.

8.2 Oracle Usage in Curve LlamaLend Markets

Note Version 1

This oracle is intended to be used as a collateral price oracle for Curve LlamaLend markets. By design, the oracle price is not expected to match the PT market price at all times: it is an estimate based on a deterministic linear discount model.

This design choice has non-trivial economic implications in a Curve LlamaLend AMM, since the oracle price can diverge materially from the market price. In turn, this may create unexpected liquidation behavior and/or trading opportunities.

In particular, we identified potential borrower loss scenarios when PT market prices trade above the oracle price, especially when the oracle follows a predictable upward path toward maturity. In such cases, arbitrageurs may be able to extract value from the AMM via:

- Active arbitrage exploiting oracle/market divergence.
- Buy-and-hold strategies benefiting from deterministic oracle convergence.

We also note that large discrete parameter updates can create unnecessary one-off arbitrage windfalls.

These economic implications are out of scope for this review. We recommend the Curve team carefully consider them prior to production deployment, and simulate a range of market scenarios to better understand the risks and incentives for borrowers, liquidators, and arbitrageurs.

This note was acknowledged with the following response:

```
The deterministic linear model may diverge from market prices.

The core reason we use a linear discount model is that PT markets are mainly used by loopers
with very high LTVs. This makes these positions sensitive to small price fluctuations.
The market price is too volatile and would result in a lot of unnecessary liquidations.
We can leverage the fixed-yield property of PT markets to build an oracle with a much more
stable price and offer a safer experience for loopers. Also, to scale LlamaLend markets
safely, we would like to move away from being dependent on single-pool oracles in general.

We expect divergence with market price especially in long-dated PT markets. This is the
reason we introduce the ability to update parameters. The model will be updated if a significant
deviation is detected. The bounds for parameter updates need to be determined separately for each
asset. The model will be less risky in the sense that there will be less liquidations for the user
compared to using a market oracle. We also introduce safety bounds so that an incorrect
parameter update wouldn't result in a big catastrophe.
```

8.3 `ADMIN_ROLE` Cannot Be Granted or Revoked

Note Version 1

The `ADMIN_ROLE` is assigned at deployment; however, as no role admin is given to that role, it cannot be granted or revoked later, and only the current holder can renounce it.

This has been confirmed to be intentional by the developers:

```
The admin can renounce but not transfer the role. For admin rotation, deploy a new oracle.

Admin rotation is a last resort risk mitigation action that the DAO can leverage
to intervene in case the current admin is performing poorly or maliciously.
```