PUBLIC

Code Assessment

of the Tricrypto Smart Contracts

September 29, 2021

Produced for

Curve

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Curve Team,

First and foremost we would like to thank you for giving us the opportunity to assess the current state of your Tricrypto system. This document outlines the findings, limitations, and methodology of our assessment.

We hope that this assessment provides valuable findings. We are happy to receive questions and feedback to improve our service and are highly committed to further support your project.

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	9
• Code Corrected	6
• Acknowledged	3

2 Assessment Overview

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

2.1 Scope

The assessment was performed on the three contracts `CurveCryptoMath3`, `CurveCryptoSwap`, `CurveCryptoViews3`. These are vyper code files inside the project repository. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	08 Apr 2021	f3b5a9233a45981237dccadfcfb9b498cf3d22d0	Initial Version
2	29 May 2021	0316911f5339ad10ced59aafce243e5eb6c51012	Second Version
3	27 Aug 2021	8c5aa63c6a4e08021dcd87a0d179ce792099e789	Third Version
4	26 Sep 2021	dfc1166a3eabbf03b6640cf91d8cfd02bb6bbb5d	Fourth Version

For the vyper smart contracts **Version 1**, the compiler version `0.2.11` was chosen. For the vyper smart contracts **Version 2**, the compiler version `0.2.12` was chosen. For the vyper smart contracts **Version 3** and subsequent versions, the compiler version `0.2.15` was chosen.

2.1.1 Excluded from scope

We tried to assess the project as detailed as possible as well as the underlying economical and mathematical concepts. Still, the project's complexity makes it hard to finally conclude that there are no unexpected corner cases or black swan events that will break these concepts.

2.1.2 Excluded from this report

During the engagement shortcomings inside the smart contracts were discovered by the Curve team. These shortcomings are not listed in this report.

2.2 System Overview

Curve.finance extends their exchanges to swap (n=3) coins instantly, where the coins no longer need to be equivalent in value. The project consists of three relevant smart contracts written in the Vyper programming language (`CurveCryptoMath3`, `CurveCryptoSwap`, `CurveCryptoViews3`). Generally, Curve is a variant of a decentralized exchange (DEX) that relies on automated market making (AMM). Curve and similar AMM projects build upon the concept of liquidity pools and an invariant to determine the ratio/price to swap one coin vs another. A liquidity pool consists of multiple tokens. The tokens are added to the pool by so called liquidity providers. In return, liquidity providers receive a token that represents a share of the funds they own of the pool. Providing liquidity is incentivized by trading fees that the liquidity provider will receive when users trade (the fees are paid out indirectly by increasing the pool's value). By having a certain amount of tokens, trades can be executed immediately in one transaction. The execution can be done immediately because no counter-party is needed.

Curve modified their function compared to e.g. Uniswap in a way that the price is more robust by introducing a modified invariant. This is achieved by flattening the curve around the equilibrium and shifting the curve given certain conditions are met. This new version aims to protect liquidity providers

better, increase their profit and increase liquidity. The main invention of the new invariant is that the prices are included into the invariant. Additionally, conditional price updates are performed to shift the curve if desired.

2.2.1 The Pool

Even though the contracts are written for the general scenario of n tokens in a pool, the contracts were only assessed for the case of n=3. Hence, a pool has three different tokens. The pool has size restrictions described later in the subsection Miscellaneous.

2.2.2 The Curve

A pool always tracks the balances and prices for each of the three tokens (adhering to the restrictions. Prices are quoted relative to the first token (asset zero).

An invariant with the following parameters defines a curve which is used to determine the prices for trading. The parameters are `D` (invariant value), `A` (amplification factor), and `gamma` (which controls the size of the flat curve area). The invariant is fully defined in Curve's documentation.

2.2.3 Profit and Conditional Price Recalculations

As in the previous Curve version there is a virtual price to track the development of liquidity shares. The virtual price is determined by the value of the pool in the equilibrium. Changes of this value are used as a profit/loss indicators. Changes to the virtual price determine whether a potential price change is accepted or not.

Many curve actions will trigger the check whether a price update should be performed. This check will evaluate whether the currently used prices differ significantly from the internal price oracle. Before accepting a price update, the resulting theoretical gain/loss is calculated by comparing the new updated prices and the resulting value of the pool with the accumulated profits (interest-bearing) the pool has made. The formula is defined in Curve's documentation in detail.

If a price update results in a loss for the pool (by the definition mentioned before) which exceeds half the accumulated profits, the transaction would not update the prices. As a result the curve would not be shifted but instead, a movement on the curve would happen. Hence, the exchange still works, but the flat area of the curve is not being utilized until the price update becomes possible or the prices shift back to the previous values.

2.2.4 The Fee Model

There are two kinds of fees, admin fees and dynamic fees. Admin fees occur only when the liquidity pool accumulates funds (measured as `xcp_profit`). Admin fees are paid by minting new liquidity provider token to an admin account.

Dynamic fees are paid when depositing, exchanging and withdrawing liquidity in one coin. The fee remains in the pool, hence, increasing the value of the liquidity tokens which is the incentive to provide liquidity. These fees depend on how close the current balances are to the equilibrium point of the curve.

2.2.5 Administration

The only role in the system is the owner. The role can transfer the ownership to another account by calling the functions `commit_transfer_ownership` and after a waiting period `apply_transfer_ownership`. The transfer can be reverted by calling `revert_transfer_ownership`.

The curve parameters `A` and `gamma` can be changed by calling `ramp_A_gamma`. The change will take place gradually (over a defined period e.g. 24h) and not at once. The change process can be stopped by calling `stop_ramp_A_gamma`.

Fees and fee related parameters, the adjustment step, the moving average half-time parameter for the price oracle and the allowed extra profit can be adjusted by calling `commit_new_parameters` which allows to call `apply_new_parameters` after a fixed waiting period (3 days). The changes are immediate after `apply_new_parameters` has been called. Alternatively, if `revert_new_parameters` the proposed changes are reset.

The contract can be paused and unpaused (called `killed`) in the first two months which prevents calling `add_liquidity`, `remove_liquidity_one_coin` and `exchange`.

2.2.6 Liquidity

Initially, for the exchange to work, liquidity needs to be provided. A future liquidity provider can call `add_liquidity` to do so. If the contract is not paused, the function pulls the funds into the exchange contract with a `transferFrom`. Based on the new balances and existing prices in the pool, the curve parameter `D` is calculated. `D` is needed to determine the amount of tokens representing the share the liquidity provider now owns from all deposited funds in the exchange pool (called liquidity tokens). A fee is deducted and the liquidity tokens are minted to the liquidity provider.

If `D > 0` (should be the case if it is not the first deposit), the function conditionally updates the price information and profit calculation. The condition for updating the price information and, hence, changing the curve is that no more than half of the accumulated historic exchange profit can be lost with price updates. The definition of profit and loss is provided in Curve's documentation. When liquidity has been added successfully, `add_liquidity` emits the event `AddLiquidity`.

To withdraw provided tokens, a liquidity provider can call `remove_liquidity` or `remove_liquidity_one_coin`. The functions burn the provided amount of pool liquidity tokens, calculate the corresponding amount of tokens and transfer the tokens to the function callee. `remove_liquidity` will transfer tokens from each coin in the pool's current ratio. `remove_liquidity_one_coin` will payout an equivalent amount in one token. Both functions will update `D`. Additionally, `remove_liquidity_one_coin` will deduct a fee and conditionally update the price information.

2.2.7 Trading

Users that want to exchange two tokens can call `exchange`. The user needs to provide the information about which tokens shall be exchanged, provide the amount to be exchanged and specify a minimum amount of tokens to be received. The exchange function pulls the funds to be exchanged into the exchange contracts via a `transferFrom`. Then, it calculates how many tokens the user will receive, deducts the fees and transfers the resulting amount to the user.

As described above, the function conditionally updates the price information and profit calculation. When the trade was successful, the event `TokenExchange` is emitted.

2.2.8 Miscellaneous

The code has multiple checks for unsafe parameters. These unsafe parameters were obtained by fuzzing and might not be sufficiently tight. As an example the convergence limit for the approximation of determining a new pool token balance is defined as:

```
convergence_limit: uint256 = max(max(x_sorted[0] / 10**14, D / 10**14), 100)
```

Further conditions exist for the size of pool contributions (measured in value and hence as price times amount) for a single token as well as the relative size of different pool contributions.

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 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 associated with the items defined in the engagement letter on whether it is used in accordance with its specifications by the user meeting the criteria predefined in the business specification. 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 the facts that over the life cycle of any software product changes to the product itself or to its environment, in which it is operated, can have an impact leading to operational behaviours other than 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 severities. 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 Findings

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

- Design: Architectural shortcomings and design inefficiencies

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	3

5.1 Asymmetrical Norm in Price Update Threshold

Design Low Version 3 Acknowledged

In the `tweak_price` function the `norm` value is calculated to determine whether the distance between `price_oracle` and `price_scale` is sufficiently large so that a price update should be tried. To calculate the norm, the ratios between the `price_scale` and `price_oracle` are used. However, the ratios aren't treated symmetrically, i.e. if `price_oracle = price_scale * 1.1` then the value 0.1^2 is added to the norm, but if the ratio is reversed, `price_oracle * 1.1 = price_scale`, then the value 0.09^2 is added. This means the price update is more sensitive to changes where the price oracle is too high, than when it is too low.

Acknowledged:

As typical differences between `price_scale` and `price_oracle` are between zero and five percent, the effect is not as large. Hence, Swiss Stake GmbH decided to not make any more changes at the moment

5.2 Missing Boundary or Sanity Checks When Initializing

Design Low Version 1 Acknowledged

Most variables have implicitly or explicitly enforced minimal and maximal values or should not take certain values like address zero. These are enforced when changing the values or given the ownership through a claiming scheme. However, there are no sanity checks or any checks at all when initializing the contract. Mistakes can happen and silently set one of the values to an obviously incorrect value.

Acknowledged

Swiss Stake GmbH is aware of the issue and confident no deployment errors will happen. In case of a factory contract the issue needs to be reconsidered.

5.3 Potential Gas Savings in `tweak_price`

Design | Low | Version 1 | Acknowledged

The following code is present in the `tweak_price` function:

```
xp: uint256[N_COINS] = empty(uint256[N_COINS])
xp[0] = D_unadjusted / N_COINS
for k in range(N_COINS-1):
    xp[k+1] = D_unadjusted * 10**18 / (N_COINS * price_scale[k])
xcp_profit: uint256 = 10**18
virtual_price: uint256 = 10**18
```

Most of these variables (except `xcp_profit`) are only used when the condition `old_virtual_price > 0` is true. Hence, these variables could be moved inside of the condition to save gas in case the condition evaluates to `false`.

Acknowledged

Swiss Stake GmbH acknowledges the issue with the reasoning that gas savings are not significant enough to make a change.

6 Resolved Findings

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

- Certain Token Combination Cause Numerical Errors Code Corrected

Low-Severity Findings	6

- Mismatched Bounds Code Corrected
- Event Information Missing Code Corrected
- Parameter Check Missing Code Corrected
- Admin Fees Can Be Claimed Retroactively Code Corrected
- Packed Getters Can Be More Restrictive Code Corrected
- Slight Code Simplification Code Corrected

6.1 Certain Token Combination Cause Numerical Errors

Design Medium Version 1 Code Corrected

If one of the tokens has very few decimals, e.g. Gemini USD which has 2 decimals, and another token either has more than 18 decimals or a fairly low token value, severe numerical errors can arise.

Example:

- Token 0: Gemini USD (GUSD), 2 Decimals
- Token 1: Another Token (AT), 18 Decimals, 1 AT = 0.005 USD

An exchange of 10,000 GUSD to 2,000,000 AT takes place. Note that the amounts don't matter as the error will occur based on the ratio. The price is computed as:

- `p = dx * 10**18 / dy`
- with `dx = 10,000 * 10** 2`
- and `dy = 2,000,000 * 10**18`
- hence, `p = 0`

In this case the calculated price is zero, which triggers no special checks or fallbacks.

This situation is even more likely due to the packing of calculated prices and the used `PRICE_PRECISION_MUL` of `10**8`.

Another Example:

- Token 0: USDC, 6 Decimals
- Token 1: Another Token (AT), 18 Decimals, 1 AT = 1.00 USD

An exchange of 10,000 USDC to 10,000 AT takes place. Note that the amounts don't matter as the error will occur based on the ratio. The price is computed as:

- `p = dx * 10**18 / dy`
- with `dx = 10,000 * 10** 6`
- and `dy = 10,000 * 10**18`
- hence, `p = 10 ** 6`

However, during packing this price will be divided by `10**8` and hence become `0`.

Overall, the project has a good test suite, but it would benefit from tests containing token contracts with different decimals.

Code corrected:

The price calculation was refactored and changed. The token amounts are now scaled to 18 decimals always instead of relative to the other token.

6.2 Mismatched Bounds

Design | Low | Version 3 | Code Corrected

The minimum and maximum values for `A` in the swap contract and the math contract do not match. In fact, the bounds in the math contract are more restrictive, meaning it's possible to ramp to a new value such that the math contract will revert all calls to `newton_y` and `newton_D`, basically locking the system until a valid value for `A` is set. Additionally, the maximum value for `gamma` also is mismatched, but the bounds are more restrictive in the swap contract, which does not cause any issues.

Code corrected:

The code was corrected to make sure that the bounds of the math contract and the swap contract match.

6.3 Event Information Missing

Design | Low | Version 2 | Code Corrected

The `CurveCryptoSwap` contract emits different events. When parameter ramping starts, a `RampAgamma` event is emitted. However, contrary to expectations based on the name, it contains no information about `gamma`.

Code corrected:

The relevant information was added to the `RampAgamma` event.

6.4 Parameter Check Missing

Design | Low | Version 2 | Code Corrected

System parameters can be updated by the administrators of the system. When being updated the new values are checked. The `price_threshold` and the `mid_fee` can both be updated, however, the fact that:

```
assert new_price_threshold > new_mid_fee
```

will only be checked when the `price_threshold` is updated and not when `mid_fee` is updated.

Code corrected:

The issue was resolved through refactoring.

6.5 Admin Fees Can Be Claimed Retroactively

Design | Low | Version 1 | Code Corrected

When users see the `admin_fee` variable of the pool being `0` they will probably assume that no admin fees are being charged at the moment. However, this is not correct. Let's consider the following scenario:

Time	Action
0	the pool is started with `admin_fee = 0`
10	numerous swap have occurred and `xcp_profit` has grown
11	the admin fee is set to 1% using `commit_new_parameters` and `apply_new_parameters`
12	the function `claim_admin_fees` is called

Users might expect that the admin fees will only be claimed for the time period of 11-12. However, admin fees will be claimed for the time period 0-12. This is because `xcp_profit_a` hasn't been updated in the meantime.

Code corrected

When changing the admin fee, the admin fees are claimed for the period until the change. Admin fees are only paid for the period beginning at the last time they were claimed. Hence, the issue is resolved.

6.6 Packed Getters Can Be More Restrictive

Design | Low | Version 1 | Code Corrected

There are three functions to access packed values: `price_oracle`, `price_scale`, and `last_prices`. All three functions take an integer as input and retrieve the value at the respective offset. They make sure that the provided integer is:

```
assert k < N_COINS
```

However, `k == N_COINS - 1` is not a valid input for any of these functions and could also be blocked.

Code corrected

A check to validate `k < N_COINS - 1` has been added.

6.7 Slight Code Simplification

Design Low Version 1 Code Corrected

Within the `claim_admin_fees` function there is the following code:

```
frac: uint256 = vprice * 10**18 / (vprice - fees) - 10**18
total_supply: uint256 = CurveToken(token).totalSupply()
claimed: uint256 = CurveToken(token).mint_relative(owner, frac)
total_supply += claimed
```

During `mint_relative` the `totalSupply` will be updated. Hence, it could also be queried once after the call to `mint_relative` instead of querying it before and then updating it later.

Code corrected

The first total supply query was removed and the total supply is queried after the update.

7 Notes

We leverage this section to highlight minor findings that should be noted and considered for further development, but don't necessarily require an immediate code change.

7.1 Possible Price Manipulations

Note **Version 1**

We see the following price manipulations as possible:

1. Pushing `price_scale` towards `price_oracle`. In case a user wants to perform a larger exchange and the price inside the `price_oracle` is significantly better for that exchange than the price inside `price_scale`, then the user can push `price_scale` towards `price_oracle` using small trades. This works as the update for `price_scale` only depends on its distance to `price_oracle` not on previous actions within the same block.
2. The `price_oracle` is only affected by the last price seen in each block. Hence, big exchanges can be "hidden" from the `price_oracle` if they are followed by other exchanges with a different rate. Note that these trailing exchanges can be way smaller. Such trailing exchanges, if reliably inserted, allow full control over the `price_oracle` and thereby (as mentioned in the previous comment) also over `price_scale`.

7.2 Potential Gas Saving for Balanced Liquidity Additions

Note **Version 1**

In case `last_prices` did not change because liquidity was added in the current pool ratios, the following code part could be skipped.

```
__xp: uint256[N_COINS] = _xp
    dx_price: uint256 = __xp[0] / 10**6
    __xp[0] += dx_price
    for k in range(N_COINS-1):
        self.last_prices[k] = price_scale[k] * dx_price / ...
```

However, it is unclear whether this is a worthwhile addition as a perfectly balanced liqudity addition will be a rare case unless the UI encourages it to save gas costs.

7.3 Splitting up Exchanges

Note **Version 1**

For users it can be beneficial to split up a larger exchange into multiple smaller exchanges in order to save fees. Depending on the price constellation and the gamma value they can remain in the "flat" area of the curve and hence save fees. This is of course detrimental to the liquidity providers. The usefulness of the split depends on the gas costs and the curve parameters.

7.4 Supported Tokens

Note Version 1

There is are variety of different token implementations on the Ethereum blockchain. Using tokens with unusual behavior will lead to unexpected changes of the curve or put the smart contracts into a bad state. In particular, the following token types will not work:

- rebasing tokens, where balances can change without transfers. These tokens will lead to incorrect accounting.
- tokens with transfer fees. These tokens will lead to incorrect accounting.
- tokens with incorrect ERC20 implementations.
- tokens with more than 18 decimals
- tokens with more than one address

7.5 The General Case of n Token Versus A

Note Version 1

The audit was scoped for the case n=3 tokens. Nonetheless, we like to highlight our concerns for a bigger n. The contracts are written very generic for the case of n tokens. However, the n cannot be simply increased. As an example, with larger n space for the packed variables becomes smaller. Hence, such cases need to be tested carefully.

7.6 Variable Naming

Note Version 1

Naming variables in a clear and understandable way supports the understanding of complex projects like this. Most variables have self explaining names. But some are confusing or used inconsistently like the use of `x` and `xp`. The value (product of price and amount) of a pool token is usually denoted with `xp`. However, the `CurveCryptoMath3` contract often does not follow this naming convention consistently and `x` is used. Furthermore, `A` which is actually `ANN * A_MULTIPLIER` or `gamma` in `reduction_coefficient` which should be `fee_gamma`.

7.7 Vyper Is Still Beta

Note Version 1

Even though Vyper is used heavily in the latest DeFi projects (especially AMMs), the Vyper language is still Beta software and should be used with care as bugs might arise. Nevertheless, Curve and other AMMs have used recent Vyper versions successfully.