DEFISAFETY

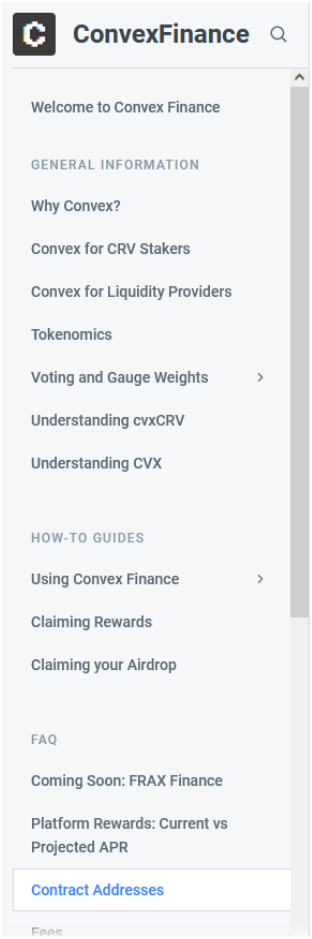
Score Improvement Guide

DeFiSafety wants all protocols to score as highly as possible. To increase the speed at which this occurs, we have produced this guide. Use this as your first port of call when improving your score. An FAQ is included at the end.

Table of Contents

Q1: Are the smart contract addresses easy to find?(%)

Clearly list your protocol's active smart contracts in your documentation. Update these frequently. See right for an example. Create a specific gitbook page entitled contracts for ease of navigation. If it takes us more than 10 seconds to find it, it won't be getting 100%. Make it obvious.



Contract Addresses

System Contracts (Curve System)

- Booster(main deposit contract): 0xF403C135812408BFbE8713b5A23a04b3D48AAE31
- Voter Proxy(whitelist contract): 0x989AEb4d175e16225E39E87d0D97A3360524AD80
- Multisig: 0xa3C5A1e09150B75ff251c1a7815A07182c3de2FB
- Deployer: 0x947B7742C403f20e5FaCcDAc5E092C943E7D0277
- CVX: 0x4e3FBD56CD56c3e72c1403e103b45Db9da5B9D2B
- cvxCRV: 0x62B9c7356A2Dc64a1969e19C23e4f579F9810Aa7
- CRV Depositor: 0x8014595F2AB54cD7c604B00E9fb932176fDc86Ae
- Reward Factory: 0xEdCCB35798fae4925718A43cc608aE136208aa8D
- Token Factory: 0x3c995e43E6ddD551E226F4c5544C77BfeD147aB9
- Stash Factory: 0x877288c4e6EbA4f635bA7428706447353B47De75
- CVX Rewards: 0xCF50b810E57Ac33B91dCF525C6ddd9881B139332
- cvxCRV Rewards: 0x3Fe65692bfCD0e6CF84cB1E7d24108E434A7587e
- Pool Manager: 0x8a849F4074726179f95Ee08C59cAA8F6f21B1E83
- Pool Manager Proxy: 0x5F47010F230cE1568BeA53a06eBAF528D05c5c1B
- Arbitrator Vault: 0x25E12482a25CF36EC70fDA2A09C1ED077Fc21616
- Convex MasterChef: 0x5F465e9fcfFc217c5849906216581a657cd60605
- Vested Escrow: 0xe98984aD858075813AdA4261aF47e68A64E28fCC
- Airdrop Factory: 0xa1Bc2Cf69D474b39B91665e24E7f2606Ed142991
- Airdrop: 0x2E088A0A19dda628B4304301d1EA70b114e4AcCd
- Claim Zap: 0x92Cf9E5e4D1Dfbf7dA0d2BB3e884a68416a65070
- CVX/ETH SLP: 0x05767d9EF41dC40689678fFca0608878fb3dE906

Q2: How active is the primary contract? (%)

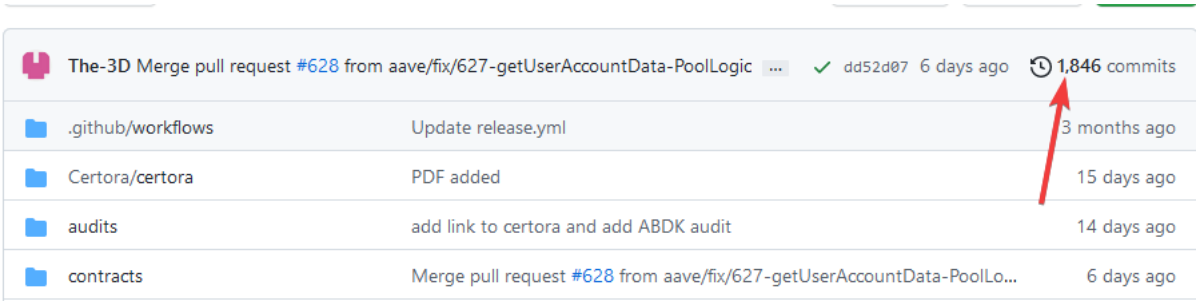
Consider deploying to a chain with lower fees so that you may get more interactions.
We do not consider token addresses in this question. DeFiSafety analyses generally avoid tokens, though this is up to the discretion of the reviewer. MasterChefs, Factory etc. are what we want here.

Q3: Does the protocol have a public software repository? (Y/N)

Use GitHub. Make the repository public. If needed, have a private and public repo.

Q4: Is there a development history visible? (%)

Regularly push updates to the GitHub contract repository. the number of commits will rise. document each part of your development history, as this provides users a good idea into how your protocol develops. See the red arrow for an example. 100+ commits = 100%



Q5: Is the team public (not anonymous)? (Y/N)

State at least two public team members in your documentation and then get them to cross confirm this information in their own personal socials. This can be twitter, linkedin, github ...

If you don't feel comfortable with this, then so be it. We stick to our guidance.

Q6: Is there a whitepaper? (Y/N)

Create a gitbook or equivalent with protocol related documentation. This can be a whitepaper, a PDF or any other link. For this question, there doesn't need to be much in the whitepaper. Existence = yes. From this point on Gitbook will reference your equivalent, if you chose something else.

Q7: Is the protocol's software architecture documented? (Y/N)

Include lots of pretty words and pictures with many arrows explaining how the contracts & functions interact with each other. The clearer, the better. See below for an example.

Reusable credit accounts

Reusable Credit Accounts is an innovative technology that makes Gearbox gas-efficient by keeping user balances with minimal overhead. Users "rent" deployed credit account smart contracts from protocol to save deployment costs.

Each time, when a user opens a credit account in Gearbox protocol, it takes a pre-deployed credit account contract from the AccountFactory and when the user closes the credit account returns it.

If account factory has no pre-deployed contracts, it clones it using https://eips.ethereum.org/EIPS/eip-1167

Q8: Does the software documentation fully cover the deployed contracts' source code? (%)

Cover each contract with software function documentation. Use natspec crawler if you want. Document it into your Gitbook. You don't need to be verbose, just cover each function in each contract for 100%.

Configure natspec crawler for this. It might seem daunting but it doesn't take too long with a little prep.

Events

AddressSet

```
event AddressSet(bytes32 indexed service, address newAddress);
```

Emits each time when a new address is set

State-Changing Functions

setPoolRegistry

```
function setPoolRegistry(address _address)
external
```

Sets address of PoolRegistry

setACL

```
function setACL(address _address)
external
```

Sets address of ACL contract

Q9: Is it possible to trace the documented software to its implementation in the protocol's source code? (%)

Make the software function explicitly traceable. Create a hyperlink to each contract in the Gitbook for each respective contract software function documentation. The link should go to a github repo for the contract you're referencing.

In this example, AddressProvider.sol hyperlinks to the location in the repo.

If your natspec crawler regeneration disagrees with this, create a separate page in your Gitbook that links to the contracts for users to quickly trace origin. Once again, link it to the relevant contract location in GitHub.

In the below example, each word links to an exact location in a repository – you may choose to centralise the traceability and that is fine.

🍡 Common Types & Utils

This is a collection of common data types and queriers which are commonly used in Astroport contracts

---

Relevant files:

- Token
- Querier
- Mock Querier
- Lib
- Common
- Asset

---

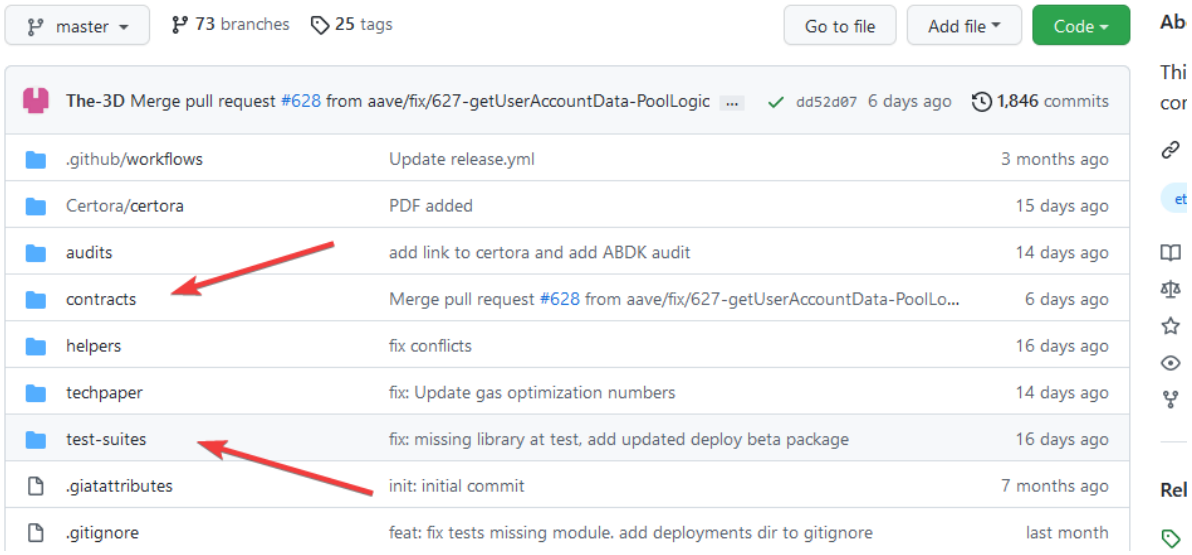
1. Overview

The Astroport codebase reuses several data types, queries and functions across the whole codebase. This section provides a general overview of these common utils. Each Astroport contract may have its own specific types and utils which are detailed on each specific contract docs page.

Cover all contracts with traceable links for 100%

Q10: Has the protocol tested their deployed code? (%)

Document in your GitHub repo at least 1.2x as many lines of test code as code in deployed contracts. In short, we divide lines of test code by lines of contract code to get this. 120%+ gets you 100%. More is better.



Test suite lines of code are divided by contract lines of code

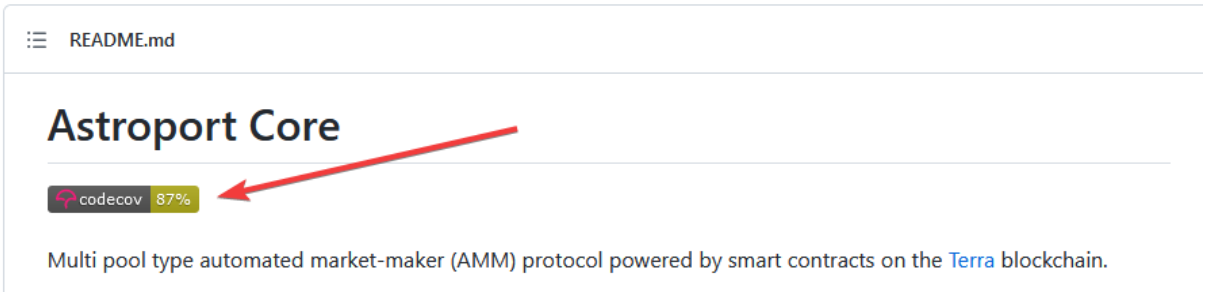
Sometimes we ignore:

- Mocks
- Utils
- Interfaces
- Other protocol code
- Libraries

It's up to the discretion of the reviewer. Document all of your testing in any way to maximise your score. Higher is better: Aave has 3300% test to code (TtC). See if you can beat it.

Q11: How covered is the protocol's code? (%)

Document code coverage testing and integrate the result into your protocol's readme.md. 100% code coverage = 100%.



Q12: Does the protocol provide scripts and instructions to run their tests? (Y/N)

show me which scripts i can run your readme.md and link to a location / provide immediate instructions for my own verification pleasure. Alternatively, make a separate folder and load all scripts into it. Always include instructions.

Build local env

```
npm install
npm start
```

Deploy on `testnet`

Build contract:

```
npm run build-artifacts
```

Create `.env`:

```
WALLET="mnemonic"
LCD_CLIENT_URL=https://bombay-lcd.terra.dev
CHAIN_ID=bombay-12

TOKEN_INITIAL_AMOUNT="1000000000000000"
```

Deploy the contracts:

```
npm run build-app
```

Q13 Is there a detailed report of the protocol's test results?(%)

Document the test results and share them in your readme.md / test script file. Explain what it means. This can be essentially any testing methodology so long as you explain it. Our ideal report follows the format below.

Test Coverage

It was not possible to run standard coverage tests on this code, since those rely on standard `require` statements with revert reasons. To conserve bytecode, we replaced this with custom assembly-coded `_require` function calls that return codes instead of standard revert strings.

```
➤ [@balancer-labs/v2-deployments]: Running normal tests...
➤ [@balancer-labs/v2-deployments]:
➤ [@balancer-labs/v2-deployments]:
➤ [@balancer-labs/v2-deployments]:   StablePoolFactory
➤ [@balancer-labs/v2-deployments]:     with no previous deploy
➤ [@balancer-labs/v2-deployments]:       when forced
➤ [@balancer-labs/v2-deployments]:         ✓ deploys a stable pool factory (7489ms)
➤ [@balancer-labs/v2-deployments]:       when not forced
➤ [@balancer-labs/v2-deployments]:         ✓ deploys a stable pool factory (558ms)
➤ [@balancer-labs/v2-deployments]:     with a previous deploy
➤ [@balancer-labs/v2-deployments]:       when forced
➤ [@balancer-labs/v2-deployments]:         ✓ re-deploys the stable pool factory (1246ms)
➤ [@balancer-labs/v2-deployments]:       when not forced
➤ [@balancer-labs/v2-deployments]:         ✓ does not re-deploys the stable pool factory
```

Q14: Has the protocol undergone Formal Verification? (Y/N)

This is an expensive and time-consuming test. Don't worry unless you're going for 100%. Not worth a great deal in the weighting of scores as such. We know Certora offers them but have not encountered many protocols that undergo this test.

Q15: Were the smart contracts deployed to a testnet? (Y/N)

Show which testnet you deployed to. It could be Kovan smart contract addresses or whichever relevant testnet you chose. Doesn't really matter. If you used Ganache, forked mainnet or some equivalent, state this.

Q16: Is the protocol sufficiently audited? (%)

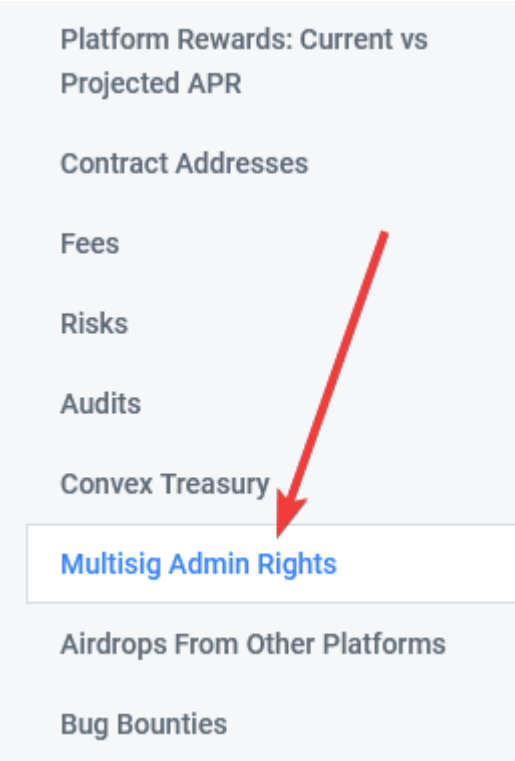
Get an audit before you deploy. Two is better. Document when you deploy / launch.

Q17: Is the bounty value acceptably high (%)

Use immunefi / hats.finance / equivalent. Make sure that it is still live – we do not award points for bug bounty programs that are not active. Document it.

Q18: Is the protocol's admin control information easy to find?

Make an access control / multisig / governance section in your Gitbook. State that it relates to user funds and include the word Admin so users know immediately



Q19-21: Admin control information

See the following for quick information on how contracts should be labelled. Identify:

- Which contracts are upgradeable or immutable
- Who owns them (governance, multisig, onlyowner, Defined Roles etc.)
- How the owners can change the contracts e.g. variables

Example 1

Access Control:

The Craftsman Contract is controlled by CraftsmanAdmin, both contracts are under OnlyOwner protocol.

All contracts are deployed by the same address `0xfCAFD61dF30886B8fBEc3195118e1EDF1f30F545.`

The keys to this address are held by chan@vvs.finance No deployed contracts can be upgraded.

OnlyOwner does not have the ability to change any aspects of the contracts such as pausing the contract.

Owner of the contracts will be restricted to perform only the following actions:

Example 2 - all contracts are immutable - no more elaboration is needed

Upgradeability

All smart contracts that are deployed to mainnet as part of the v1.0.0 release are immutable, meaning their smart contract logic cannot be altered in any way.

Example 3 – a pretty table clearly explaining all of these points in as few words as possible

Type	Mission	Locked?	Owner
Exchange Contract	Manage the Automated Market Maker functions; Swap and Provide Liquidity	Yes	ref-finance.sputnidao.near
Farming Contract	Manage liquidity incentives	No (In Progress)	ref-dev-team.near
Staking Contract	Mint and burn xREF, and Distribute time-based rewards	Yes	ref-finance.sputnidao.near
Sputnik DAO Contract	Ensure the success of Ref by taking strategic decisions (incl. smart contract amendments)	Yes	Multisig (Link)
Sputnik DAO Contract	Manage and allocate funds to specific community contributors	Yes	Multisig (Link)
Sputnik DAO Contract	Execute the Strategy and Roadmap	Yes	Multisig (Link)
Vesting Contract	Manage REF vesting contracts of Dev DAO members	Yes	dao.ref-dev-team

Q22: Is the protocol's admin control information easy to understand? (%)

Write it in simple language so that even the most unfamiliar can understand how you're handling their deposits. If you can't, don't worry - this question isn't very highly weighted.

Q23: Is there sufficient Pause Control documentation? (%)

Identify + explain if you use some form of pause control. If you don't have one, identify that and explain why you chose not to have one - literally any reason is fine.

Run a test on assembling necessary multi-sig signers at short notice and document the response time. Explain that this was a fire drill (do not actually pause your protocol) and document that for 100%.

Q24: Is there sufficient Timelock documentation? (%)

Identify what timelocks you use and why you chose them. State which contracts are subject to this timelock. If you don't have one, state this. Explain why.

Proxy contracts with 48 time-lock

Tetu has two types of proxy contracts:

- TetuProxyControlled - upgrade can be done only after announcing in Announcer and 48h time-lock period expired
- TetuProxyGov - proxy contracts without time-lock, governance can upgrade it at any time. We are using for the most noncritical contracts (contracts with view functions and governance infrastructure)

List of time-locked proxy contracts:

- Announcer
- Bookkeeper
- Controller
- FeeRewardForwarder
- FundKeeper
- VaultController
- MintHelper
- All SmartVault instances
- TetuSwapFactory

Proxy without time-lock

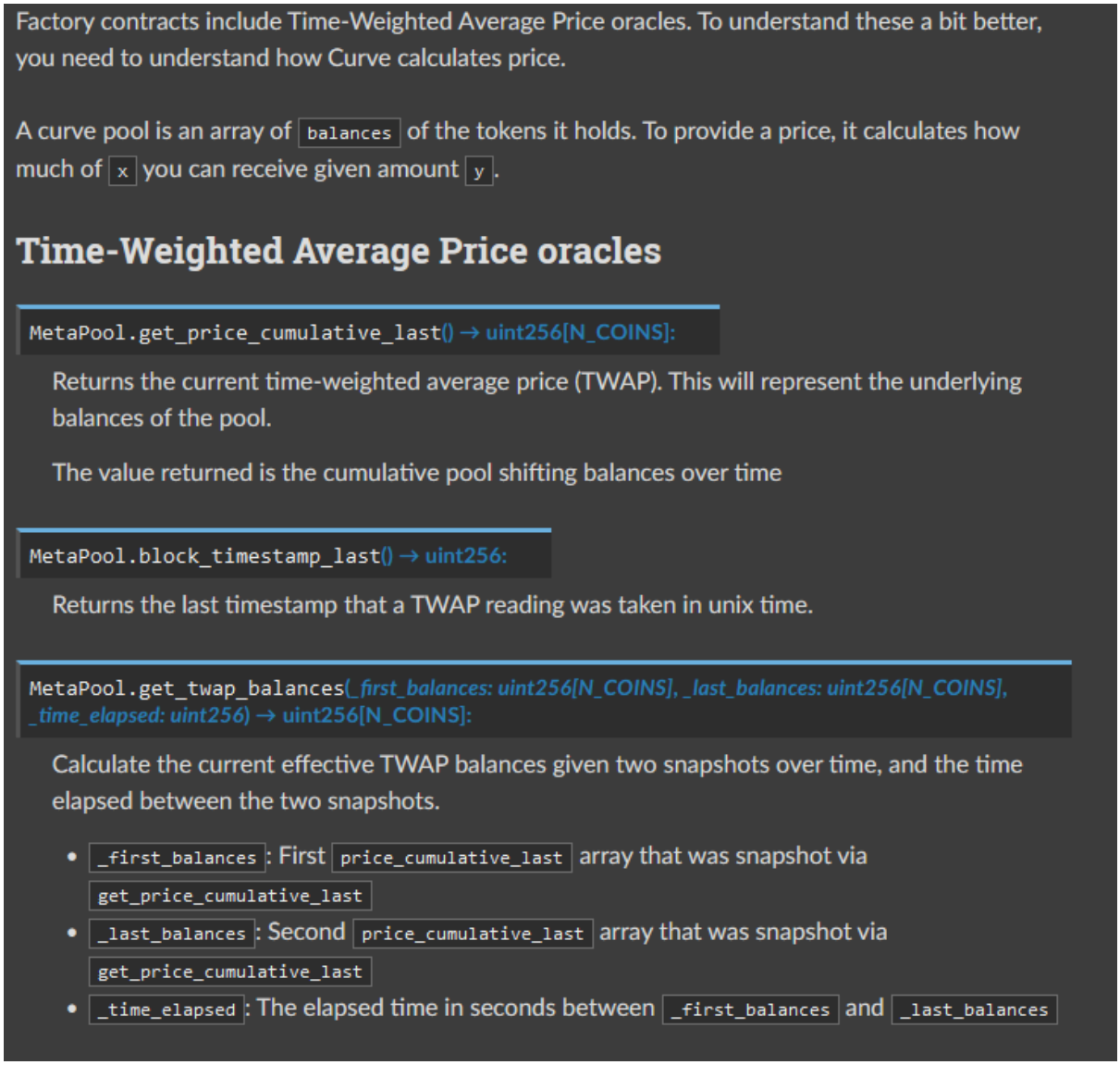
- AutoRewarder (reward distribution processing under full control of governance)
- RewardCalculator (only view functions)
- PayrollClerk (governance infrastructure)
- ContractReader (only view functions)

Q25: Is the Timelock of an adequate length? (Y/N)

Make the timelock between 48h to 1 week or explain why you chose a different length (or none at all).

Q26: Is the protocol's Oracle sufficiently documented? (%)

State which oracle your protocol uses (chainlink, TWAP + many more), provide software function documentation for it and then explain why you chose this data feed.



Factory contracts include Time-Weighted Average Price oracles. To understand these a bit better, you need to understand how Curve calculates price.

A curve pool is an array of `balances` of the tokens it holds. To provide a price, it calculates how much of `x` you can receive given amount `y`.

Time-Weighted Average Price oracles

`MetaPool.get_price_cumulative_last() → uint256[N_COINS]:`

Returns the current time-weighted average price (TWAP). This will represent the underlying balances of the pool.

The value returned is the cumulative pool shifting balances over time

`MetaPool.block_timestamp_last() → uint256:`

Returns the last timestamp that a TWAP reading was taken in unix time.

`MetaPool.get_twap_balances(_first_balances: uint256[N_COINS], _last_balances: uint256[N_COINS], _time_elapsed: uint256) → uint256[N_COINS]:`

Calculate the current effective TWAP balances given two snapshots over time, and the time elapsed between the two snapshots.

- `_first_balances`: First `price_cumulative_last` array that was snapshot via `get_price_cumulative_last`
- `_last_balances`: Second `price_cumulative_last` array that was snapshot via `get_price_cumulative_last`
- `_time_elapsed`: The elapsed time in seconds between `_first_balances` and `_last_balances`

If your protocol doesn't use an oracle, state this. Explain why.

> Convex is price agnostic and thus does not require oracles. Flash loans or any other type of price manipulation do not affect the Convex system.

Q27: Is front running mitigated by this protocol? (Y/N)

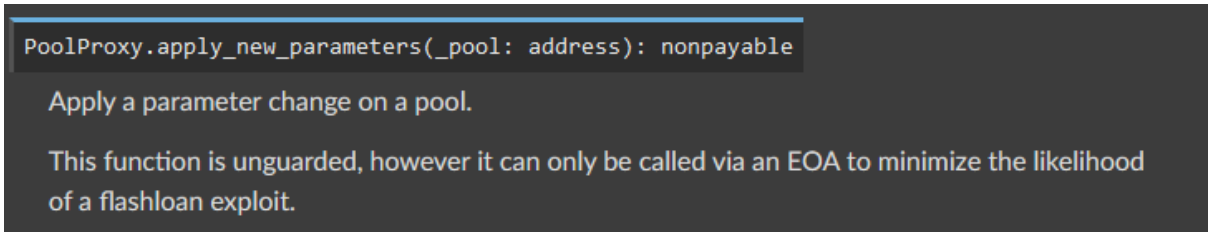
Explain if you have implemented counter-frontrunning measures. Can be anti-sandwich. Can be specified in bug bounty. Can be anything.

```
+ ### Profit distribution
+
+ Once a strategy is harvested, the gains are distributed during 6 hours.
+ This is to prevent sandwich attacks of an account depositing tokens right before the harvest and withdrawing right after and getting the profits without staying in the vault.
```

Explain if you haven't. State this.

Q28: Can flashloan attacks be applied to the protocol, and if so, are those flashloan attack risks mitigated? (Y/N)

Document if you are subject to flashloan attacks, and what countermeasures you have implemented to deal with it.



If your protocol is not vulnerable to flashloan attack, state this. Users like reassurance that developers think of this.

DeFiSafety hopes this was helpful. Please reach out to your assigned analyst at any moment for clarification. See the following commonly asked questions too.

DEFISAFETY

Score Improvement Guide

Commonly Asked Questions

How does DeFiSafety make money?

You have likely been told that the reviewing process as a whole never incurs an exchange of funds, meaning that we effectively make no money off of our reviews. Well then, how do we make revenue? We make revenue via our revenue-generating products; Contract Scores and Chain Scores, as well as the use/integration of our APIs. In addition, we also have concluded a seed funding round in the summer of 2021, and plan to have our Series A soon. Series B will ensue in the distant future, so on and so forth. We are a private company, and therefore these products will cater towards institutions, traders, and high net worth individuals that want access to our enterprise-level risk data.

As such, this is how we can keep our reviews devoid of any charges. This is our public good to the DeFi community and developers.

Who asked for this review?

There are three different ways that your review could have been requested to us:

- A community member requested it through our Telegram, Discord, or e-mail
- One of our enterprise clients requested it
- Self-request, meaning that one of our analysts saw that you were either very popular or have a high TVL and decided to do a review

This is NOT an audit

Auditors review and test the quality of your software, DeFiSafety reviews the transparency and process quality of your documentation as well as your development process. We are fundamentally different by nature and should not be put in the same league as security auditors.

Hack/Exploit/Incident Definitions

Hack: a deliberate attack by externa; malicious party on smart contract code that resulted in funds lost. Funds lost could be either token value or principal lost.
This could include a flash loan attack (or others) where the smart contract acts properly but the result is not expected and results in unexpected losses.

Exploit: a deliberate attack by a malicious party on a protocol or an individual, but not on the smart contract code that resulted in funds lost. Funds lost could be either token value or principal lost.
This could include front end attacks, computer virus or phishing scam.

Incident: an internal mistake or nefarious activity by the core team that results in loss of user funds.
This could include rug pulls, liquidations due to unreplenished reserves, renegade treasury managers, and more.

Question 17: Bug Bounty

By our definitions, there are two different kinds of bug bounty programs:
Active/dynamic – your bug bounty program actively drives hackers to your software via a provider like Immunefi, Hats Finance, and Code Arena

Inactive/static – you mention a bug bounty in your docs that does not actively employ/drive hackers to your software

Why do you need a reward of $1m+ to get 100% for this question? Because the more money you offer as a reward, the bigger the incentive for hackers or whitehats to actively try to find bugs in your software.

Question 10: Testing

We only review test files (JS, TS, Soldity, Rust, Vyper, etc) that are publicly available in your GitHub repositories. Meaning, you cannot send them to us privately for reviewing – they must be viewable by anyone.

Question 13: Test Results

There are two different test result reports we are looking for:
Code coverage report such as in Codecov/coveralls, this will award you 70% for this question

Detailed test reports of your test runs. This can be an exported file of an entire unit test run, or it can be the test runs natively found in the “Actions” tab of the GitHub. This would ideally include tests like lint, fuzz, mythx, CI, unit tests, etc.

Question 19: Immutability/Not upgradeable

There are three different ways with which we defined Immutability and non-upgradeability:
Our primary definition of Immutability is that nothing can be changed about your software. Unchangeable variables, parameters, logic/implementation, and no proxies.

Our primary definition of non-upgradeability is, simply put, certain variables can be changed but the logic/implementation contracts that handle user funds cannot be changed by admin

privileges. In essence, if no entity has access to parameters that directly control user funds, we consider it non-upgradeable.

Immutable: nothing can be changed
Non-upgradeable: logic cannot be changed, arbitrary values/variables can change (fees, emissions,
Upgradeable: anything can be changed

Questions 24 and 25: Timelocks

There are two different entities that we define as corresponding to a Timelock:
A timelock contract that delays the execution of an on-chain transaction, and in which you can see the transaction that would be executed.

In the event of a DAO/Governance structure, we would consider an implementation period as synonymous to a Timelock. For example, if after a successful vote passes there is a delay between the passed vote and the execution of its proposed contents on the blockchain, this corresponds to the core functionality of a Timelock. However, contents of what will be executed must be visible to the community.