dYdX Perpetual Audit

OPENZEPPELIN SECURITY | APRIL 19, 2020

Security Audits

Perpetual is a new perpetual contract market by dYdX that enables users to use native Ethereum-based tokens as collateral to trade derivatives with non-Ethereum underlying assets.

Scope

Contracts

We audited commit `c5e2b0e58aaf532d2c8b1f658d1df2f6a3385318` of the `dydxprotocol/perpetual` repository.

All contracts in the `contracts/protocol` directory were in-scope except for `P1Orders` and any files in the `contracts/protocol/v1/oracles` directory (which was created in a later commit).

Price oracles

Perpetual uses third-party price oracles. These price oracles were out of scope (see the “Security Assumptions” section for more information).

Mechanism design

Perpetual implements incentive mechanisms to nudge user behavior. For example there is a floating interest rate paid between longs and shorts which is intended to incentivize users to take

Security assumptions

Price oracles

Perpetual requires price oracles to report the prices of assets in terms of the token being used as collateral. It is important to understand that a malicious or compromised price oracle would have the ability to steal funds from any Perpetual market that uses it. Furthermore, a price oracle that fails to report accurate prices in a timely manner (for example, if it were DoSed) could also result in a loss of funds for honest users.

We did not audit the third-party price oracles. For the purposes of this audit, we assume that these oracles are honest, uncompromised, and will remain so while reporting timely and accurate price information indefinitely.

Administrators

Perpetual has a privileged admin role that can make arbitrary changes to the market contract, set critical market parameters, set the price oracle, and enable final settlement. The admin can also add/remove global operators, which have the ability to make arbitrary trades on behalf of all users. These privileges would allow a malicious or compromised admin and/or global operator to trivially steal funds from the market.

For the purposes of this audit, we assume the admin and all global operators will remain honest and uncompromised indefinitely.

Choice of collateral tokens

Stable coins make a natural choice for the tokens used as collateral in perpetual markets. One of the most popular stable coins, USDT is capable of extracting a fee from the recipient of a `transfer` or `transferFrom` (see line 131 of the USDT contract code). While USDT is not doing this at the time of writing, it will remain *capable* of doing so unless the `owner` of the USDT contract provably burns their ownership. The audited code does not take such a fee into consideration, and may behave unexpectedly if this fee is ever extracted.

Recommendations

Here we present our findings and recommendations.

Critical

None. 🙂

High

None. 🙂

Medium

Unsafe transparent proxy pattern

OpenZeppelin's `AdminUpgradeabiltyProxy` contract is used to implement the unstructured storage proxy pattern by having the `PerpetualProxy` contract inherit from the former and managing `delegatecall`s to the `PerpetualV1` contract.

One of the features of the proxy model implemented in the library is the transparent proxy pattern, which prevents collisions between function signatures of the proxy and the implementation contracts by not allowing the admin of the proxy to call the implementation contract. This is an important security feature. However, the `_willFallback` function in `PerpetualProxy.sol` has removed this check.

The motivation behind removing the check is to allow for the same address to be the admin of both the `PerpetualProxy` contract *and* the `PerpetualV1` contract. However, this convenience introduces the risk of function signature collisions that could make functions on the implementation contract unreachable by the admin.

The dYdX team is aware of this and intends to mitigate the risk via extensive testing before deploying any new implementation contracts. But the risk could be removed entirely by separating the concerns — having one admin for upgrading the proxy, and different one for calling access-controlled functions on the implementation contract.

have the `onlyAdmin` modifier.

Update: Unchanged. The development team responds, “We have comprehensive tests to ensure that there is no function signature collision.”

Low

Missing NatSpec docstrings

While the code is well commented and easy to follow, most of the public and external functions in the code base lack NatSpec documentation. This may hinder some reviewers’ understanding of the code’s intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings may improve readability and ease maintenance. They should explicitly explain the purpose or intention of the functions, the scenarios under which they can fail, the roles allowed to call them, the values returned and the events emitted.

Consider documenting all functions (and their parameters) that are part of the contracts’ public API using the Ethereum Natural Specification Format.

Update: Fixed via PRs #150 and #168.

Not using latest stable version of OpenZeppelin Contracts

An old version of OpenZeppelin Contracts is being used in the project. Since version 2.5.0 has been recently released with improvements such as gas optimizations in the ReentrancyGuard contract, consider bumping the library to its latest stable version.

Update: Fixed via PR #132. Note that a custom reentrancy guard is being used for compatibility with the upgradable proxy pattern.

Notes

_verifyAccountsFinalBalances could be made easier to verify

`initialBalance.position`, `finalBalance.margin`, and `finalBalance.position` take on values that are positive, negative, or zero. There are 81 combinations of these values (`3^4`). Though many of these combinations can be ruled out quickly, it is still time consuming to verify the correctness of this code.

Consider whether this function can be refactored for easier verification.

Update: Fixed by PR #160.

Typos in comments

There are a few typos in the the code comments. We list them here.

- On line 30 of `PerpetualProxy.sol`, `conntract`, should be `contract`.
- On line 225 of `P1Trade.sol`, `finalPosition` should be `finalBalance.position`.
- On line 244 of `P1Trade.sol`, `[0,+, -/+]` should be `[0/+, -/+]`.

Consider fixing these typos before deploying to production.

Update: Fixed via PR #131.

Conclusion

No critical or high severity issues were found. Some small recommendations were made to improve the project's overall quality and robustness. Overall we found the code to be very clean, well-organized, and easy to follow.

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits