Security assessment and code review

Initial Delivery: May 30, 2022

Prepared for:
Shao-Kang Lee | Perpetual Protocol
Yenwen Feng | Perpetual Protocol

Prepared by:
Thomas Steeger | HashCloak Inc
Karl Yu | HashCloak Inc

Table Of Contents

Executive Summary

Perpetual Protocol is building a protocol for enabling decentralized and trustless perpetual futures contracts on Ethereum. At the core of their protocol is a concept called the virtual automated market maker (vAMM) that enables one to mint virtual tokens for accounting purposes with no value. In v2 of their protocol, they leverage the increased capital efficiency guarantees of Uniswap V3 in order to build a more capital efficient vAMM.

Perpetual Protocol Team engaged HashCloak Inc for an audit of their smart contracts written in Solidity. The codebase was audited by 2 auditors for 2 weeks, from May 16 to May 30, 2022. The scope of the audit was:

- ➢ diff of perp-lushan repository at commit 86b766…a20fb5 to previous audit.
- ➢ periphery contracts at commit d32380…e00f67; only contracts in folders `interface`, `limitOrder` and `storage`
- ➢ oracle contracts at commit e4a64c…105b74; without the contract `EmergencyPriceFeed.sol`

These commits added the following features to Perpetual Protocol:

- ➢ BLP: Liquidations now move positions from trader to liquidator
- ➢ Introduction of limit orders via periphery contracts
- ➢ Caching of TWAP price feeds

We found a variety of issues ranging from critical to informational.

Severity	Number of findings
Critical	0
High	0
Medium	0
Low	0
Informational	1

Findings

Confusing variable naming

Type: Informational
Files affected: CumulativeTwap.sol, CachedTwap.sol

During our audit we were a little bit confused by the variable naming `lastUpdatedTimestamp` or `latestUpdatedTimestamp` in the functions `_update`, `_cacheTwap` and `_getCachedTwap`.

Impact: No impact on protocol security. It just makes it harder for new people to understand the code.
Suggestion: We recommend renaming the variables to something like e.g. `lastTimestamp` or `latestTimestamp`.

General Recommendations

Multiply operations first and then apply division

When dealing with floating/fixed point numbers , we need to do multiplication operations before division in order to reserve as much precision as possible.
Since Perpetual Protocol is backed by SafeMath , we have no worries about overflow problems. As such, we need to minimize loss of precision.

Public vs External Function

Public functions which are solely called from external (other smart contracts or externally owned accounts) should be marked `external` instead of `public`, since it saves gas costs.

Named return statements

The code makes heavy use of named return statements. This feature of solidity is primarily used for documentation purposes in the codebases, since in these cases also explicit `return` statements are used. We want to point out that there is a possibility of variable shadowing, when not using explicit `return` statements. In these cases the

compiler will issue a warning, which should be taken seriously. We give an example of this behavior:

```
function sqrt(uint x) public pure returns (uint y) {
    uint z = (x + 1) / 2;
    uint y = x;
    while (z < y) {
            y = z;
            z = (x / z + z) / 2;
    }
}
```