Perpetual Protocol

Smart Contract Security Assessment

30.01.2023

ABSTRACT

Dedaub was commissioned to perform a security audit of multiple new releases of the Perpetual V2 protocol and its oracle implementation. This is an audit of changes ("delta audit"), following up on our previous audits of the protocol.

SETTING AND CAVEATS

The scope of the audit focused mainly on the changes introduced in the new versions of the protocol available at https://github.com/perpetual-protocol/perp-curie-contract and the at the time latest version of the Perp oracle contract, which can be browsed at https://github.com/perpetual-protocol/perp-oracle-contract.

The specific versions of the Perpetual V2 protocol whose changes were considered are:

- v2.2.4-rc (a7b0d4cd5d0c749e609c27e6c098476da13289f2)
    - ClearingHouse.sol
- v2.5.0-rc (65e0cffe0af97dd43dbe4631b695c91ba785def4)
    - AccountBalance.sol
    - ClearingHouse.sol
    - ClearingHouseConfig.sol
    - Exchange.sol
    - lib/PerpMath.sol
    - lib/UniswapV3Broker.sol
    - storage/ClearingHouseConfigStorage.sol
- v2.6.0-rc (cde1ddec04d3bf75ac3ada1a41a07a49174bb01c)
    - BaseToken.sol
    - ClearingHouseConfig.sol
    - Exchange.sol
    - Vault

The audit of the oracle contract considered the following contracts up to commit 2c876094d68b1c89229e2595a59afcf20bc034d5:

- ChainlinkPriceFeedV3.sol
- PriceFeedDispatcher.sol
- PriceFeedUpdated.sol
- UniswapV3PriceFeed.sol
- twap/CachedTwap.sol
- twap/CumulativeTwap.sol

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than regular use of the protocol. Functional correctness (i.e., issues in "regular use") is a secondary consideration. Functional correctness relative to low-level calculations (including units, scaling, quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

Intensive efforts were made to check for economic attacks, e.g., price manipulation and protocol bad debt attacks. As an audit of changes, some context may be missing: only the most relevant parts of the whole code base were revisited due to time constraints. As the audit's main focus is security, the quality of the economic model of the protocol and its assumptions were not considered in depth. We believe that the behavior of the system depends on financial assumptions that are hard to assess in the scope of a security audit, as they might require simulations, etc.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: -User or system funds can be lost when third party systems misbehave. -DoS, under specific conditions. -Part of the functionality becomes unusable due to programming error.
LOW	Examples: -Breaking important system invariants, but without apparent consequences. -Buggy functionality for trusted users where a workaround exists. -Security issues which may manifest when the system evolves.

Issue resolution includes “dismissed” or “acknowledged” but no action taken, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

ID	Description	STATUS
M1	Normal protocol function depends on the timely updates of the price feeds and of the associated PriceFeedDispatchers	**OPEN**

The PriceFeedDispatcher contract, which by default uses a Chainlink price feed (contract ChainlinkPriceFeedV3), might dispatch to the internal Uniswap V3 vAMM price feed if the underlying Chainlink price feed has “timed-out”, i.e., has not been successfully updated for a period of time. Thus, the Chainlink price feed and consequently the PriceFeedDispatcher contracts’ methods that update the current price have to be successfully called at regular time intervals not only to effectively keep track of the asset prices but also of the status of the Chainlink price feed used by the PriceFeedDispatcher to timely switch to the fallback Uniswap oracle.

Let’s consider the following concrete scenario for better clarity: a Chainlink oracle that is consulted by a certain Chainlink price feed of the Perpetual protocol is not functioning properly leading to the price getting “timed-out”. However, the PriceFeedDispatcher using the aforementioned Chainlink price feed is not called to update or get the price during the time-out period. In that case the status of the dispatcher never changes to Uniswap and is as if the timeout never happened.

The current design of the Chainlink price feeds and the PriceFeedDispatcher contracts requires the owners/operators and/or users to run off-chain bots that are programmed to call the update/query methods of the aforementioned contracts at the right intervals to ensure the normal function of the protocol. However, such off-chain solutions cannot be trusted even for events that might be rare, such as a Chainlink oracle reporting the wrong price for a prolonged period of time. We believe that there should be a fallback (implemented on-chain) that ensures the normal function of the protocol even if the state of the price feeds and dispatchers is not timely updated, be it due to off-chain bot malfunction or absence of market participants (and therefore sparse contract calls/updates).

M2	Chainlink deviation checks do not take time into account	**OPEN**

ChainlinkPriceFeedV3 ensures that every change in the Chainlink price is not larger by a certain fraction of the old price. However, this check only takes into account the difference in price, and not the time that it took for this difference to occur.

```
function _isOutlier(uint256 price) internal view returns (bool) {
      uint256 diff = _lastValidPrice >= price ? _lastValidPrice - price :
price - _lastValidPrice;
      uint256 deviationRatio =
diff.mul(_ONE_HUNDRED_PERCENT_RATIO).div(_lastValidPrice);
      return deviationRatio >= _maxOutlierDeviationRatio;
}
```

Time is only implicitly taken into account by the frequency of updates: if they are regular then this is not an issue, however, too frequent or infrequent updates could potentially be problematic:

- Assuming that an adversary can control the Chainlink updates and wishes to produce a huge change without triggering the deviation check. This could be achieved by applying a change of the maximum allowed ratio but in every single block. The overall difference can be large without any individual update exceeding the limit.

- Inversely, assume that a token is volatile and its price is rapidly increasing. An adversary might wish to prevent the oracle from registering the increase in price; this could be achieved by preventing the update transactions from executing, using some DoS technique (e.g., DoS on the nodes sending such transactions). If the price gradually increases above the maximum ratio over a period of time, but no update happens during that period, then the next update will trigger the deviation limit, further preventing the price from updating.

For the above reasons we recommend that the deviation checks take time into account.

LOW SEVERITY:

ID	Description	STATUS
L1	ClearingHouse::quitMarket is not guarded against reentrancy	**OPEN**
	The function `ClearingHouse::quitMarket` is not guarded against reentrancy. Even though there is no such threat in the current version of the codebase, we would suggest adding reentrancy guards (as it is done for all functionality that is offered by the ClearingHouse contract) as a precaution measure, in case a reentrancy is made possible by future code changes.	
L2	No StatusUpdated event emitted in PriceFeedDispatcher constructor	**OPEN**

There is no `StatusUpdated` event emitted in the `PriceFeedDispatcher` constructor even though the `_chainlinkPriceFeedV3` storage variable is set. The `status` storage variable is also technically set for the first time.

OTHER/ ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

A1	PriceFeedUpdater does not allow adding/removing feeds	**INFO**
The PriceFeedUpdated contract does not support addition and removal of price feeds, meaning that the whole contract should be redeployed in case this is a need.		
A2	Storage variables can be made immutable	**INFO**
There exist a couple of storage variables that are set in the constructor and cannot be modified afterwards. These variables can be declared immutable: ● `UniswapV3PriceFeed::pool` ● `PriceFeedDispatcher::_chainlinkPriceFeedV3`		
A3	Compiler known issues	**INFO**
The contracts were compiled with the Solidity compiler v0.7.6 which, at the time of writing, has a few known bugs. We inspected the bugs listed for this version and concluded that the subject code is unaffected.		

CENTRALIZATION ASPECTS

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

As already mentioned in issue M1, the normal function of the protocol depends on the timely updates of the Chainlink price feeds and of the associated PriceFeedDispatcher contracts. These updates are performed by off-chain bots, which cannot be fully trusted.

DISCLAIMER

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure the most prominent protocols in the space. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Prominent blockchain protocols hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.