Perpetual Protocol

Smart Contract Security Assessment

04.03.2022

ABSTRACT

Dedaub has performed the first incremental security assessment of Perpetual protocol. The Perpetual protocol has been previously deployed in Q4 2021, and was audited by Dedaub. The audit report of this audit can be found here. Since then, incremental security updates and extra functionality has been added, such as the ability to perform emergency shutdown on markets and the addition of further price Oracles. Dedaub will continue performing security assessments throughout 2022, working alongside the Perpetual protocol team. This audit report covers commit hash `bac1ae0b6dd633275b175e06169c5cb02896b8e5` but also the BandPriceFeed and related contracts. Two auditors worked over the codebase over two weeks.

For a detailed background on the protocol, please refer to the original audit report.

Centralization Aspects

As is common in complex protocols, the owner of the smart contracts yields considerable power over the protocol. Since the last audit, one further mild centralization element has been noticed:

- the ability of the Owner to pause and completely close a market. Upon closing, the owner may set an arbitrary price (`closedPrice`) value that is to be considered in actions which perform accounting calculations originally using the mark price (i.e. action `quitMarket` and calculating the positionNotional value). The reasoning behind this is to protect the system from closing a market with a noticeably big deviation between mark price and index price, which would result in considerable losses for the protocol.

Security Opinion

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") was a secondary consideration, however

intensive efforts were made to check the correct application of the mathematical formulae in the reviewed code. 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. A key limitation of this audit, and generally, any exercise intended to establish the security of Perp V2 is that Perp is, admittedly, one of the most complex protocols in DeFi. Although the auditing team understands the principles employed in the accounting logic and financial design of the protocol, there are several exceptions to these principles, and exceptions to these exceptions which are not documented enough.

No Critical or High Severity vulnerabilities were found, which denotes an overall healthy project and codebase. The protocol is extremely complex and the Perp team provided detailed documentation material which was vital for the audit. However, the core codebase seems to include many implementation parts that serve to "patch" against specific attack vectors. This makes it difficult to follow the accounting flow in several parts, although one may have a very good understanding of the protocol design and the overall accounting logic. One such example is the use of the index price for calculating a user's `accountValue` regarding `unrealizedPnL` when it comes to liquidation margin ratios while in other parts the mark price is used instead (as it is the common case). This differentiation is a mitigation for the Bad Debt Attack vector. Furthermore, this solution comes with other attack vectors, as described in Index Price Spread Attack, which in turn bring further exceptions in some accounting formulas. There seems to be several other parts in the codebase that are difficult to follow in full due to similar reasons. This fact somewhat decreases our level of confidence in the audited core codebase. We strongly suggest that the team write a full documentation for the accounting formulas (and exceptions to these), accompanied by the corresponding design decisions behind them, at least for the cases where any kind of exception is made.

SECURITY REVIEWS CONDUCTED

This section describes the reviews performed by Dedaub in this incremental audit. The items include “New Features”, “Security Updates”, and “Other Modified Contracts”.

NEW FEATURES:

ID	Description	STATUS
N1	Emergency Shutdown of a Market	**COVERED**
Emergency Shutdown shall be triggered in the case of emergency, such as the termination of an oracle price feed or any misconfiguration right after the contracts have been deployed.		

ID	Description	STATUS
N2	Deposit collateral to any account	**COVERED**
Until this version of the protocol, collateral deposits considered `msg.sender` to decide the account to which the deposited amount would be accounted for. Vault::depositFor enables depositing collateral to an account attached to an address A by any other address.		

SECURITY UPDATES:

1. Intentional Bad Debt attack

In order to mitigate this attack two measures have been taken:

A. Bad debt restriction: Traders are not allowed to close or reduce a position if any bad debt occurs.

B. Bad debt liquidation restriction: Only whitelisted actors called “Backstop Liquidity Providers” are allowed to liquidate positions resulting in bad debt.

2. Liquidation Sandwich Slippage attack

In order to mitigate this attack two measures have been taken:

A. Bad debt liquidation restriction: Whitelisted “Backstop Liquidity Providers”, as described before. The attacker has now no reason to manipulate the mark price in such a way that the victim position is liquidated with bad debt.
B. Optional Liquidation slippage protection: Liquidators can protect their liquidation transactions from being sandwiched by providing a maximum slippage threshold.

In this way, both the protocol and honest liquidators are effectively protected.

However, the traders having liquidatable positions still remain exposed to being liquidated by an attacker at a disadvantageous price.

OTHER MODIFIED CONTRACTS:

A number of other contracts were refactored, however,

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”, by the client, or “resolved”, per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

M1	Possible bad debt for makers	OPEN

The way in which free collateral is measured in makers is, at worst and even when unrealized PnL is not taken into consideration, through the total position value. This is proportional to their liquidity in the base token in their AMM calculated multiplied by the reference (index) price.

Because the vAMM operates using a constant product, if a large short position is placed, the maker's liquidity will be swapped to the base token at a rate which assigns more of the base token compared to what the rate of index price would have awarded.

Since the swap doesn't lower the index price, it is possible that the collateral ratio will be inflated. The maker can now take an undercollateralized position. Here's a possible attack that may ultimately drain some funds from the vault:

Assume we start with mark price close to the index price, as in usual operation.
1) Alice opens a maker order. vAMM pool now looks like the picture below, Alice's liquidity is depicted by '*', other user's as '+'

```
               current mark price
                       |
                       |
          ************
          ************++++++++++++++++++++
0.....................................................................10
0
```

2) Bob opens a short taker position, converts all of Alice's liquidity
3) Alice now has a *higher* collateralization ratio due to step 2
4) Alice opens the largest short taker position she can with her increased collateral

	5) Bob closes his position. 6) Bob can now close Alice's excess order and should be able to liquidate her, making a profit. Alice and Bob have made a combined profit.	
M2	Mark price can be manipulated in a single transaction, despite checks	**OPEN**
	Say Alice wants to open a very large position (say N vUSD) to manipulate the mark price, defeating the `isOverPriceLimit` gets in the way. This is how she can still do it: 1. Alice opens a long position without crossing price tick limit 2. Alice 2 opens a short position without crossing price tick limit 3. Alice increases the long position without crossing price tick limit 4. Alice 2 increases the short position without crossing price tick limit 5. Repeat steps 3 and 4 until Alice and Alice 2 have positions sizes that are **each** `N / partialCloseRatio` 6. Alice 2 closes the short position (partial close ratio kicks in)	

LOW SEVERITY:

ID	Description	STATUS
L1	BandPriceFeed may return not-so-accurate prices	**OPEN**

Band price feed provides token price information by maintaining a number of price observations and calculating a cumulative TWAP upon them. A user may interact in two ways with the contract, either by calling update(), which fetches a fresh price value and stores it as a new observation, or by calling getPrice(). While getPrice() also fetches a fresh price value which is also considered in the final returned price value, this fresh value is not stored in the observations' list.

```
/// CumulativeTwap.sol
function _getPrice(
        uint256 interval,
        uint256 latestPrice,
        uint256 latestUpdatedTimestamp
    ) internal view returns (uint256) {
    Observation memory lastestObservation = observations[currentObservationIndex];
    // ...

    uint256 currentTimestamp = _blockTimestamp();
    uint256 targetTimestamp = currentTimestamp.sub(interval);
    (Observation memory beforeOrAt, Observation memory atOrAfter) =
_getSurroundingObservations(targetTimestamp);
    // Dedaub: why not first do an _update?
    // Dedaub: if interval1 = latestUpdatedTimestamp - latestObservation.timestamp
    // Dedaub: and interval2 = currentTimestamp - latestUpdatedTimestamp then:
    // Dedaub: currentCumPrice = latestObservation.priceCum +
(latestObservation.price * interval1) + (latestPrice * interval2)
        uint256 currentCumulativePrice =
    lastestObservation.priceCumulative.add(
              (lastestObservation.price
              .mul(latestUpdatedTimestamp.sub(lastestObservation.timestamp)))
              .add(latestPrice.mul(currentTimestamp.sub(latestUpdatedTimestamp)))
    );
    // ...
}
```

A design in which CumulativeTwap::_getPrice() calls cumulativeTwap::_update seems reasonable, as users of the service would update the observations upon each

price request. Users are currently not directly incentivized to keep the observations up-to-date, possibly resulting in some loss of accuracy. In parallel, such a design would simplify the code in _getPrice calculation of currentCumulativePrice, because the part of the price that considers interval1 (see in our comments above) would be taken care of in _update(). Of course, in the case of multiple requests in a single block, the call to _update can simply be omitted.

L2	Accounts may be liquidated at a disadvantageous price	**OPEN**

In order to mitigate the Liquidation Sandwich Slippage Attack attack two measures have been taken:

1. Bad debt liquidation restriction: Whitelisted “Backstop Liquidity Providers”, as described before. The attacker has now no reason to manipulate the mark price in such a way that the victim position is liquidated with bad debt.
2. Liquidation slippage protection: Honest liquidators can protect their liquidation transactions from being sandwiched by providing a maximum slippage threshold.

In this way, both the protocol and honest liquidators are effectively protected.
However, the traders having liquidatable positions still remain exposed to being liquidated by an attacker at a disadvantageous price.

L3	Inconsistent use of closed price and last known index price	**OPEN**

When the market is closed, most functions such as funding payments are still ongoing. There is an inconsistent treatment between prices. For instance, `Exchange::_getFundingGrowthGlobalAndTwaps` uses the last known index price, whereas account balances use the closed price, which would be different.

If one of the reasons why a market is closed is due to a sudden movement in price (e.g., collapse), and the mark price significantly deviates from the last known index price, users risk getting their positions liquidated due to excessive funding payments.

OTHER/ ADVISORY ISSUES:

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

ID	Description	STATUS
A1	Typographic error	**OPEN**

In BandPriceFeed.sol there is a typo throughout the contract:

```
// Dedaub: typo lastestObservation -> latestObservation
Observation memory lastestObservation = ... ;
```

We recommend fixing this typo for readability.

ID	Description	STATUS
A2	Code style	**OPEN**

In `ClearingHouse::closePosition`, the following code can be shortened:

```
bool isBaseToQuote = response.exchangedPositionSize < 0 ? true : false;
```

ID	Description	STATUS
A3	Compiler known issues	**INFO**

The contracts were compiled with the Solidity compiler v0.7.6 which, at the time of writing, have some known bugs. We inspected the bugs listed for this version and concluded that the subject code is unaffected.

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, as well as a public bug bounty program.

ABOUT DEDAUB

Dedaub offers technology and auditing services for smart contract security. The founders, Neville Grech and Yannis Smaragdakis, are top researchers in program analysis. Dedaub's smart contract technology is demonstrated in the contract-library.com service, which decompiles and performs security analyses on the full Ethereum blockchain.