CavalRe AMM

Smart Contract Design Advisory

Sep. 10, 2022

ABSTRACT

Dedaub was commissioned to audit the CavalRe protocol implementation, at https://github.com/CavalRe/amm. The protocol implements a novel automated market maker (AMM) design, based on the principles of the technical paper "A Family of Multi-Asset Automated Market Makers". The result is an AMM that supports seamlessly multiple-pool assets, as well as unifies staking/unstaking and swapping, since the pool liquidity token is just another asset in the pool, with a negative weight (i.e., counted as a "liability").

SETTING AND CAVEATS

The audit report covers commit hash 63d8e19ad14da0a1cc28dd509cdb48c3d4dcc2c9. Two auditors and a mathematician worked on the codebase over 4 days. The scope of the audit is limited to a single file (Pool.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 of most aspects (e.g., 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. Importantly, thorough integration testing in the setting of final use is also an aspect that is not effectively covered by human auditing and remains the responsibility of the development team.

Economic considerations are unavoidable when attempting to understand the protocol in depth. However, the protocol creators are explicitly "*not expecting [Dedaub] to audit the economics. Just the contract security.*" Our comments on the financial model are listed below, for advisory purposes only. The present document does not constitute the security audit report, which is issued separately.

PROTOCOL-LEVEL CONSIDERATIONS

The main threats for the protocol will be economic threats. The model described in the technical paper admits an infinite number of solutions, and the code currently implements just one of them, materializing many design choices. The design choices should be assessed with thorough studies, through simulation, and with clear argumentation for why the protocol behaves desirably. One should be concerned about the possibility of high price impact (colloquially, "slippage"), the impact of fees on weight adjustment, and more.

Specifically, the code implements a specialization of the model in the technical paper for the parameter $k = 1$. However, many more design decisions are implicitly encoded. These design decisions should be made explicit in a whitepaper, with careful consideration of their impact and validation that the final AMM behaves desirably (possibly in extensive simulations). The current test suite is not adequate for such validation. It is diminutive in size, covering little in terms of both usual and extreme scenarios, and has no explicit automatic checks of financial soundness: numbers are output for human inspection instead of being cross-checked automatically with expected or admissible behavior. The protocol is still at an early stage of development, but emphasis should certainly be placed on these aspects.

Below are design decisions that we identified in the code, specializing the mathematical model of the technical paper.

For this AMM ($k = 1$) the liquidity curve constraint can be expressed as: $\sum_{i=0}^{n} w^t_i / g^t_i = 0$, where w^t_i is the weight of an asset i and g^t_i is its "growth" during a swap/stake/unstake action, i.e., the ratio a^t_i / a^{t-1}_i, for asset holdings a^t_i at time t. (This statement of the formula sets $w_0 = -1$. It is a slightly different formula than the one we see in documentation, but more compact. It also makes explicit that the whole liquidity curve is a canceling sum.) The code, in addition, does the following:

- It uses w^{t-1}_i in the calculations, instead of w^t_i. It is not clear whether this is sound. Although the weights of assets are expected to remain relatively stable, they are certainly not entirely stable: the code itself adjusts the weight, right after a swap/stake/unstake action.
- The sum $\sum_{i=0}^{n} w^t_i(1 - 1/g^t_i)$ of the positive-growth terms (of $\sum_{i=0}^{n} w^t_i/g^t_i$) is called the "value" (`fracValueIn`) in the code. This is not the same as the concept of "value" in the technical paper. To get unique solutions for the rest of the g_i terms, the code uses an input `allocation` array, splitting the value to tokens that are output from a swap/stake/unstake action. It is not clear to us (i.e., although possibly true, it needs to be validated in simulation or formal argument) that this allocation truly reflects value. If a user specifies an allocation of her output tokens, expecting, e.g., that she will receive 40% of the value in token A, and the result is an arbitrary number, 40% over an obscure mathematical quantity, this would be concerning for usability/attractiveness.
- The update of weights is also a very significant design choice. The choice is both in what updates *are* being made, and in what updates *are not* being made.
    - Weights are currently updated based on fees paid over the token, by updating both the numerator (scale of asset) and the denominator (scale of all assets).

      ```
      delta = _assets[payToken].fee.dmul(amount);
      _assets[payToken].scale += delta;
      _scale += delta;
      ```

      This is an arbitrary decision and it is far from clear what is its financial impact. It is, for instance, plausible (though perhaps unlikely, given that the adjustment is small) that a winning swap strategy will include a previous wash trade that just changes weights so that the desired swap is performed under better terms. Also, it is not clear how this adjustment interacts with the fees available to liquidity providers: there is no enforced

guarantee in the code that all fees get to liquidity providers. Instead, this is expected to follow as a side-effect of the financial model, and the update of weights has an impact on that.

- As mentioned earlier, the solution of the liquidity formula assumes that weights are being kept stable, using w_i^{t-1}, instead of w_i^t. However, it is not clear that a stake operation should have the same slippage/weight-adjustment as a swap operation. For instance, it might be reasonable to expect that staking will incur less slippage and more weight update than a swap: when an investor stakes token A in an AMM, the investor wants to keep token A, whereas when they are swapping it, they are trying to get rid of it, which makes a price-loss (slippage) more reasonable. (Of course, it can be argued that the investor will know that they will incur slippage, so their depositing *only* token A, without matching deposits in other pool assets, signifies "getting rid" of A.)

Although the above concerns may end up being dismissable, they hopefully illustrate why extensive study is required to argue that an AMM's behavior is practically desirable.

Additionally, it is not clear what the protocol will display to users as the "price" of a token. There are many different definitions of "price" in an AMM (e.g., differential price vs. average price of most recent transaction vs. ratio of asset quantities) and the relevant computation is not in the current codebase.

A final comment is on the concept of "self-financing transactions". The principle of "self-financing" is described as "there should be no change of value due to price movement". However, this is fairly arbitrary, although simplifying for the math. There is no reason why the term $a(\Delta P)$ (asset-quantity times price-delta) is "change due to price movement". The change in P could well be due to the change of assets in the pool, since P and a are interrelated and changing quantities results in a change of price. Therefore, although "self-financing" is an excellent idea, yielding simple liquidity curves, it is certainly not an unavoidable mathematical law of AMMs.