DELPHI LABS

Presents.

Attack Cost and Profit from Manipulating Constant Product Market Maker TWAP Oracles in DeFi Protocols.

Attack Cost and Profit from Manipulating Constant Product Market Maker TWAP Oracles in DeFi Protocols.

Ayana T. Aspembitova, Jonathan Erlich

January 2022

1 Introduction

Blockchain-based smart contracts have been successfully growing and their use cases are quite innovative and have attracted lots of interest valued in the billions of dollars. However, there is a fundamental limitation of decentralized applications - they execute in a closed environment and some bridge service (oracle) is needed when obtaining information outside of the blockchain. As decentralized applications evolve and mature, oracles play an increasingly prominent role in ensuring the safety across smart-contracts.

Generally speaking, there is some *ground truth* information that resides outside of smart contracts and smart contracts need it for the proper performance. To obtain such ground truth, smart contracts need reliable *data sources* - any entity that stores the ground truth information (databases, sensors or other smart contracts). Then *data feeders* report off-chain data to an on-chain system. There can be one or few data feeders and there can be various mechanisms for how to select and aggregate data feeders. *Aggregation* is a very important step and the final output quality depends on both the data feed selection and the proper aggregation method. Overall, the process of obtaining the true information for a smart contract is shown in the flow chart in Figure 1.

As can be seen from Figure 1, there are few steps from processing the real data to the output signal to the smart contract. At each step information can be distorted or manipulated by attackers willing to benefit from the protocol. In this paper we do not aim to discuss all possible attack vectors, instead we focus on one attack vector arising from a popular way of obtaining ground truth - using another smart contract as a data source and using statistical measures to aggregate the data and produce the output. Most of the time the information smart contracts need is the price information and *Automated Money Market* protocols are often used as data feeders for an oracle. The *Time Weighted Average Price* aggregation mechanism is widely used to produce the output information.

It is possible to get an approximate estimation of an attack cost knowing the data source and aggregation method used as an oracle. To estimate the profit from an attack, we need to understand the type of protocol that could be a target for an attack, how it works and evaluate all possible vectors for the attacker to profit. Overall, oracles are needed for a wide variety of decentralized applications - money markets, yield farming protocols, synthetic assets, derivatives, prediction markets etc. In this paper we discuss two possible attack vectors - attack on liquidation mechanism (when attacker targets to false-liquidate the position and get bonuses by pumping the value of debt asset) and attack on collateral-backed positions (by pumping the value of their collateral asset and falsely being able to borrow more assets).

Overall, in this paper we aim to provide a comprehensive understanding of how oracles can be manipulated in smart contracts and how to ensure the safety by monitoring key parameters. High economic security in smart contracts when using AMM-based DEX as oracles happens when the financial resources needed to compromise the price are in a way that financial benefits from compromising are not higher than the cost needed for manipulation. Also, the user experience

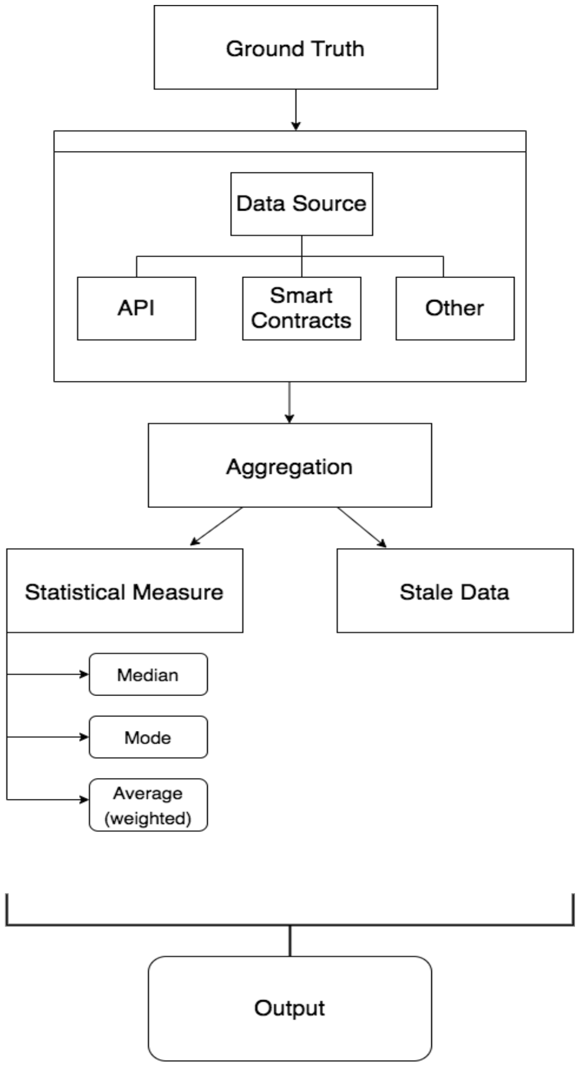

Figure 1: Flow chart how the ground truth information is delivered to the smart contract.

is also important and while minimising the potential attacker's profit and maximising the attack cost, we also want to minimise the deviation between the spot price and the oracle's output price. We develop the comprehensive methodology that assesses step by step the quality of data feeder, checks the price deviation, estimates cost of profit and attack under varying conditions and gives recommendations for safe parameters for both types of protocols.

2 Attacker's Profit from manipulating TWAP

2.1 Profit from falsely increasing health factor

One of the common oracle related attacks happens when an attacker falsely increases their health factor (for example in lending protocols) to borrow more assets than originally intended by the protocol. The expected income from an attack is shown in Equation 1.

$$V = N \times \epsilon \tag{1}$$

where N is the active collateral of an attacker $N = Collateral \times LTV$ and ϵ is the expected deviation from the spot price in percentage:

$$\epsilon = \frac{TWAP_m - TWAP_s}{TWAP_s} \tag{2}$$

where $TWAP_m$ - is the expected TWAP after manipulation, $TWAP_s$ is the spot (actual) TWAP. And the $TWAP_m$ should be at higher than the Loan-to-Value to be able to profit from the attack:

$$TWAP_m > LTV \tag{3}$$

2.2 Profit from falsely decreasing health factor

The profit from decreasing the health factor of certain positions in different protocols can come from the income generated by falsely liquidating positions that are in fact above the liquidation threshold. Let's assume that attacker wants to drive the price of a debt asset higher and cause the health factor of the position to fall below the liquidation threshold. Overall, the manipulated price of a debt asset should be at least as big as shown in Eqaution 4:

$$TWAP_m \geq \frac{TWAP_s \times HF}{LT} \tag{4}$$

where HF is the current health factor of the position, LT is the liquidation threshold. Having $TWAP_m$ equal or more than the value in Equation 4 will ensure that position will fall below the liquidation threshold. Manipulating $TWAP_m$ of the debt asset to the less value than in Equation 4 will not bring any profit to the attacker, since the position would not be liquidated.

The total income from the attack would be as follows:

$$V = A \times TWAP_s \times LB \tag{5}$$

here A is the target position size that the attacker wants to liquidate. His profit is defined by the liquidation bonus LB.

3 Impact of manipulation on TWAP

Some oracles use three core statistics for aggregation - mean, median and mode. To improve the quality of such simple statistics and mitigate the price manipulation, weights can be applied - time weighted or liquidity weighted average prices. Time-weighted average prices (TWAP) are used in Uniswap V2 (weighted arithmetic mean) and other AMM-based DEXs. Uniswap V3 TWAP optimized for more detailed queries - allows users to include the liquidity volume and calculate the geometric mean. In this report we discuss the attack on the arithmetic mean TWAP, (ref) has demonstrated the cost to the Uniswap V3 type of oracles using geometric mean TWAP.

3.1 Arithmetic mean TWAP

The arithmetic mean TWAP over n price updates is calculated as follows:

$$TWAP = \frac{\sum_{i=1}^{n} t_i p_i}{\sum_{i=1}^{n} t_i} \tag{6}$$

where t_i is the time elapsed between the price update i and next price update $i+1$, and p_i is the price during that period. n is the averaging window.

We estimate the effect of manipulation on TWAP price when the attacker wants to consistently manipulate the spot price for m times of price updates within the averaging window n.

$$TWAP_m = \frac{\sum_{i=1}^{n-m} t_i p_i + \sum_{j=n-m+1}^{n} t_j p_j}{\sum_{i=1}^{n} t_i} \tag{7}$$

assuming that attack would happen in the last m blocks. From here, we would like to estimate the p_j - how big should be the manipulated price that attacker should target in order to achieve the desired effect on $TWAP_m$.

$$\sum_{j=0}^{m} t_j p_j = TWAP_m \times \sum_{i=1}^{n} t_i - \sum_{i=1}^{n-m} t_i p_i \tag{8}$$

In case when attacker does not want to be exposed to arbitrageurs and wants to manipulate the price within one block $m = 1$ the manipulated price will be as follows:

$$p_j = \frac{TWAP_m \times \sum_{i=1}^{n} t_i - \sum_{i=1}^{n-m} t_i p_i}{t_j} \tag{9}$$

4 Manipulation Cost

To profit from a certain smart contract, an attacker would need to be familiar with the oracle data feeder and the aggregation mechanism. If the data comes from an AMM-based DEX and the attacker knows the pricing formula, it is possible for him to manipulate the price and get the desired output price in a targeted smart contract. There are different types of pricing equations across AMMs - Uniswap V2 type of equation is so far the most popular (constant product market), Uniswap V3 and Curve formulas. For different pricing equations the attack cost would vary. In the subsection below we derive the attack cost for a constant product market.

4.1 Constant product AMM

In constant product AMMs the liquidity is defined as follows:

$$x \times y = k \tag{10}$$

If an attacker wants to move the spot price of an asset, he would need to swap against the pool, depending on whether he wants to manipulate the price up or down. If we assume that he wants to increase the price of y, he would need to sell some amount of x and receive some y tokens in return (adding x tokens from the pool, an attacker decreases its value). After the swap the liquidity in the pool will be as follows:

$$(x + \Delta x)(y - \Delta y) = k = xy \tag{11}$$

From here, we can express the Δy - how much of token y attacker would receive after making a swap:

$$\Delta y = y(\frac{\Delta x}{x + \Delta x}) \tag{12}$$

Since the attacker wants to increase the price y by adding the Δx tokens and removing Δy tokens, we can express how big would be the change of manipulated price p_j:

$$p_j = \frac{x + \Delta x}{y - \Delta y} = \frac{x + \Delta x}{y - y(\frac{\Delta x}{x + \Delta x})} = \frac{(x + \Delta x)^2}{xy} = \frac{(x + \Delta x)^2}{k} \tag{13}$$

From here, we can express the Δx - how many tokens x are needed to get the target p_j:

$$\Delta x = \sqrt{p_j \times y \times x} - x \tag{14}$$

Now, when we know both how many tokens x will be needed to make a swap and how many tokens y we receive in exchange, we can easily calculate the total attack cost by subtracting $\Delta x - \Delta y$:

$$AC = (\sqrt{p_j \times y \times x} - x) - p_j \times y \frac{\sqrt{p_j \times y \times x} - x}{x + (\sqrt{p_j \times y \times x} - x)} \tag{15}$$

Below we show two examples that guide step by step on how attack can be done on two different types of smart contracts, how costly and how profitable it would be for an attacker.

5 Attack Examples

5.1 Example 1. False collateral value increase attack

Let's assume an attacker has $N = 10000$ USD of the active collateral (after meeting LTV requirements), and he wants to get the profit of $V = 1000$ from the credit protocol (he is able to borrow another asset for 11k worth of collateral instead of 10k, and we assume that LTV is just 10%). We assume he wants to attack the stablecoin pool (USDC-USDT) and the spot price is 1. The total liquidity in a pool is 200k: $x = y = 100000$.

First, we find the ϵ from equation 1 - how much the spot price (in percentage) should deviate in order to get the profit of 1000.

$$\epsilon = \frac{V}{N} = 0.1 \tag{16}$$

From here, we can find the $TWAP_m$ - what should be the target manipulated TWAP value in order to get the desired profit. From the equation 2, we find that:

$$TWAP_m = TWAP_s(\epsilon + 1) = 1.1 \tag{17}$$

It means that attacker should somehow manipulate the TWAP in the way, so instead of the spot price $TWAP_s = 1$ it will show $TWAP_m = 1.1$.

Next, the attacker needs to figure out p_j - how big should be the manipulated price within manipulation time so it has the desired effect on the TWAP. Here the attacker needs to decide within how many blocks he wants to perform the attack - if the attack will happen within one block, there is no risk that their effort will be affected by arbitrageurs. We generate 144 random values (approx number of blocks within 30 minutes in Ethereum blockchain) of times elapsed for each block in range between 5 and 9 (average time of price updates in Uniswap stablecoin pool); and 144 values of prices within the range 0.95-1.05. And the following is the estimated p_j:

$$p_j = \frac{1.1 \times 978 - 981}{5} = 18.96 \tag{18}$$

which means that the p_j - manipulated price in one block should be higher than spot almost 19 times to get the desired TWAP.

How costly is it to move the spot price to 19x? For this we refer to the equations 8 to 11 and calculate how much of tokens x is needed with the given pool liquidity:

$$\Delta x = \sqrt{18.96 \times 100000 \times 100000} - 100000 = 343621.4 \tag{19}$$

is needed to move the spot price to the desired value. The amount of token y attacker gets can be calculated by equation 8:

$$\Delta y = 100000(\frac{343621.4}{100000 + 343621.4}) = 77458.2 \tag{20}$$

The attack cost will be:

$$AC = \Delta x - \Delta y = 266163.2 \tag{21}$$

Given that attacker's profit would be only 1000 USD, it is not rational to perform such an attack for the pool with the given liquidity and averaging window.

5.2 Example 2. False Liquidation Attack.

Let's imagine a situation when an attacker targets to false liquidate the position $A = 100000$ USD. For simplicity, we assume that it is a stablecoin pool position and the $TWAP_s = 1$. We also assume the health factor of the position is slightly above the liquidation threshold $HF = 1.3$, which refers to the debt $D = 76923$. Attacker aims to get the profit of $V = 100000 \times TWAP_m \times 0.05$. The liquidation threshold is $LT = 1.2$ and therefore, the target manipulated TWAP needed to drive the position to false liquidation according to equation 4 is:

$$TWAP_m = \frac{1 \times 1.3}{1.2} \geq 1.08 \tag{22}$$

How to achieve the goal of oracle showing the wrong price of debt asset equal or greater $TWAP_m = 1.08$?

For this, the attacker needs to manipulate the price on DEX within one or few blocks. Before he starts swapping tokens in a pool, he needs to figure out the p_j - how big should be the manipulated price on AMM so it has the desired effect on the TWAP. Here the attacker needs to decide within how many blocks he wants to perform the attack - if the attack is to happen within one block, there is no risk that their effort will be affected by arbitrageurs. We generate random 144 values (approx number of blocks within 30 minutes) of times elapsed for each block in range between 5 and 9; and 144 values of prices within the range 0.95-1.05. And the following is the estimated p_j:

$$p_j = \frac{1.08 \times 978 - 981}{5} = 15.05 \tag{23}$$

If the attacker wants to get the target TWAP within one block, the p_j - price on DEX for the debt asset should be higher than spot price for at least 15 times.

Next question is how expensive would it be to actually make the price 15 times higher than spot? It depends on the liquidity in the pool and using equations 12-15 we can calculate the attack cost. Assuming the total liquidity in the pool is 200k USD $x = y = 100000$:

$$\Delta x = \sqrt{15.05 \times 100000 \times 100000} - 100000 = 287943 \tag{24}$$

is needed to move the spot price to the desired value. The amount of token y attacker gets can be calculated by equation 8:

$$\Delta y = 100000\left(\frac{287943}{100000 + 287943}\right) = 74223 \tag{25}$$

The attack cost will be:

$$AC = \Delta x - \Delta y = 213720 \tag{26}$$

Given that attacker's profit would be only 5000 USD, it is not rational to perform such an attack to the position with the given size for the pool with given liquidity and averaging window.

6 Recommendations for TWAP oracle parameters

The examples above showed that it is quite expensive to perform an economic attack to the TWAP oracle by manipulating spot price. However, these examples do not state that it is totally safe to use TWAP oracles for all types of smart contracts. There can be different scenarios when attack cost could be less than the potential profit. In this section we develop an oracle assessment framework, that takes into account various core parameters to decide on the safety of the pool to be used as price oracle, suggests optimal averaging window and proposes the safe setting of parameters for both types of protocols.

6.1 Safe price oracles and Optimal Window size to update the TWAP

Step 1. Find optimal averaging window size. Attack Cost formula has been derived in Equation 15 and it can be seen that it depends on liquidity x, y in a pool and p_j - how big should be the manipulated price to affect the TWAP to the desired level. Pools with big liquidity are more resistant to the price manipulation, but it is not the only factor. As Equation 9 shows, the p_j value depends on values of t_j - for how long the manipulated price was live and t_i - how long the real price was live. It is clear that if the ratio t_i/t_j is big (meaning that the unmanipulated price was live for much longer than manipulated one), then p_j would also be large, making the attack cost more expensive. In other words, the longer the manipulated price was live, the cheaper the attack would be. Therefore, apart from the liquidity level in a pool, another factor affecting the safety of oracle use is the *frequency and uniformity* of price updates. If the distribution of time of price updates is skewed, it could lead to the situation when attacker luckily for him would change the spot price and it will not get updated for a long time. Together with the short window size of TWAP calculation it would lead to the successful attacks. When deciding on whether to use the certain pool in DEX, the first step should be looking at the time distribution of price updates.

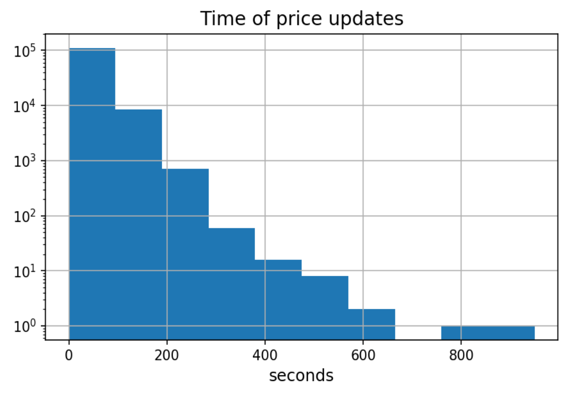

Figure 2: Time distribution of price updates in USDC-ETH pool.

Figure 2 shows how often the price was updated in the USDC-ETH pool in Uniswap V2 (it is two week data in December 2021). It can be seen that there were few times when price was not updated for more than 800 seconds, but most of the time price updates are quite frequent. Having understanding of the frequency of price updates helps us to decide on the optimal averaging window for TWAP - we assume that attack can happen in a tail of time distribution and want to make sure that unmanipulated time was at least twice longer than manipulated price time. From here, we propose the formula on how to decide on the optimal averaging window size:

$$WS \geq TS(99.9\%) \times n \tag{27}$$

where WS is the window size, and $TS(99.9\%)$ is the 99.9th percentile of time series of price updates. The 99.9th percentile was taken instead of 99th or 95th in order to capture the tail of highly skewed distributions (when time distribution is closer to normal, 99th and 99.9th percentiles are very close in value, but when we have fat and long tails, taking 99.9th percentile allows to capture the tail and suggests safer window size). n is multiplication term to ensure that even if attacker gets lucky and manipulated price will be in the tail of time distribution, the unmanipulated time would be at least twice longer:

$$n = 1 + \frac{max(TS)}{TS(99.9\%)} \tag{28}$$

For the example of the USDC-ETH pool in Uniswap V2, the WS should be greater or equal 20 minutes.

Step 2. Ensure the acceptable price deviation. After the safe window size has been defined, the next step is to make sure that the TWAP price does not deviate greatly from the spot price. We aim to have the deviation between TWAP and spot to be less than 3% in 99% of the times.

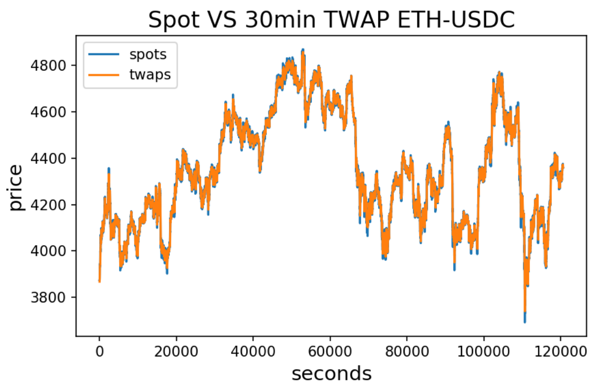

Figure 3: Evolution of spot and TWAP prices in USDC-ETH pool.

Figure 3 shows the evolution of both spot (blue line) and TWAP (orange line) and it can be seen that overall they are very close. To look closer at price deviations, we plot the histogram of errors between spot and TWAP:

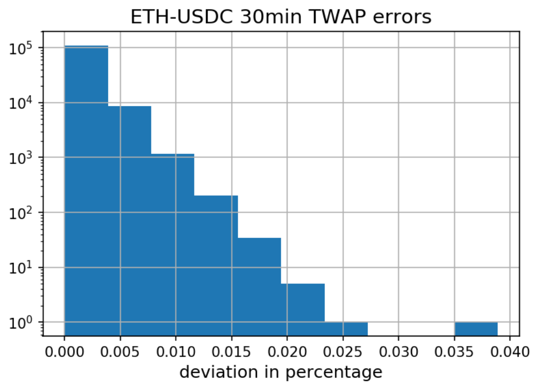

Figure 4: Distribution of price deviations in USDC-ETH pool.

Figure 4 shows that there was only one time when the TWAP deviation from spot was greater than 3% . Box plot below shows the overall statistics for USDC-ETH pool:

The bottom of each box in Figure 5 is the third quantile of data, top - the first quantile. Green line shows the median value, the lowest black line is the minimum value and the upper black line is the maximum value. Dots above the maximum value are outliers. For the USDC-ETH pool the median deviation between spot and TWAP is around 0.1%, with the maximum less than 0.5%. This is quite good results and it means that the USDC-ETH pool in Uniswap V2 is safe to use as a price oracle.

In this subsection we showed how to choose an AMM pool that is safe to use as an oracle in general. However, having safe window size and acceptable price deviation from spot is not the only criteria that we need to take into account. There are also protocol-specific parameters that need to be adjusted right to avoid successful attacks. Next subsections show which parameters need to be monitored in two types of protocol - given that only safe pools are used as price oracles, we recommend the safe setting of protocol-specific parameters.

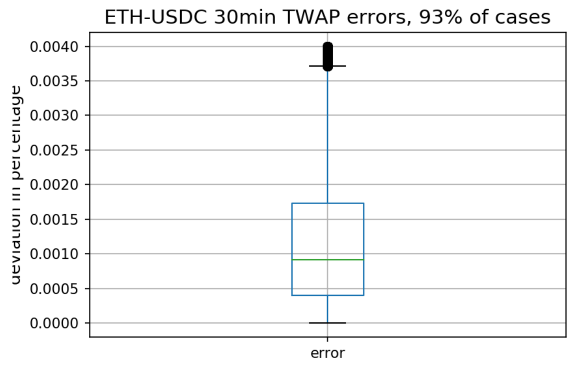

Figure 5: Time distribution of price updates in USDC-ETH pool.

Overall, there is a linear dependence between the attack cost, liquidity in the pool and window size - the bigger the window size and higher the liquidity, the more expensive would be the attack. Figure 6 shows the general relations between these three parameters.

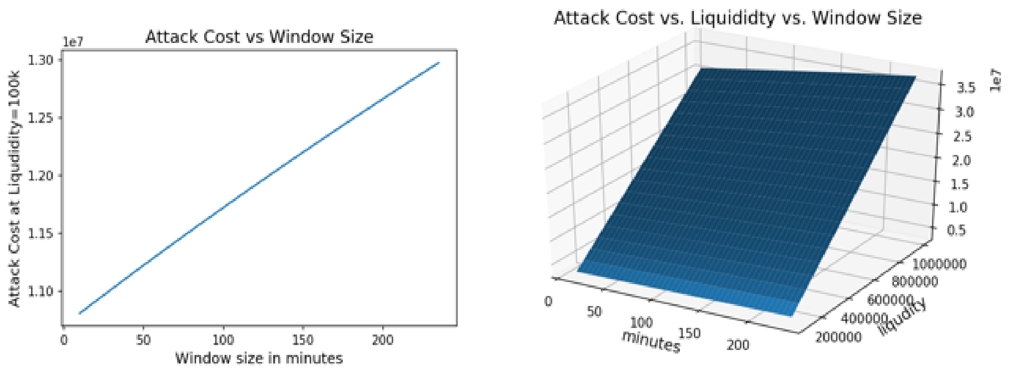

Figure 6: Relations between the Window Size, Attack Cost and Pool Liquidity

To decide what should be the minimum requirement for the liquidity in the pool to use as an oracle, we suggest the following metrics:

$$(\sqrt{p_j \times y \times x} - x) - y(\frac{\Delta x}{x + \Delta x}) > A \times TWAP_s \times LB \tag{29}$$

The minimum p_j we suggest to be as follows:

$$p_j = \frac{0.01 \times TWAP_s \times 2n - n \times TWAP_s}{n} \tag{30}$$

where n is the maximum time of price updates, $0.01 \times TWAP_s$ was taken as the minimum deviation from the spot (1% deviation from the spot) to make the estimation more conservative. For the right side of equation, A is the value of assets in a position. Also, it should be noted that here we assume the total liquidation - whenever the position falls below the liquidation threshold, it gets fully liquidated. This is very conservative scenario while most of the protocols implement partial liquidation - liquidator only repays the amount of a user needed to meet the liquidation threshold. In this case, the profit of potential attacker is even less, unless they would be able to liquidate many positions at the same time.

6.2 Parameters for protocols prone to attacks of false collateral value increase

When an attacker wants to falsely increase the collateral value, the attack cost depends on two factors - liquidity in the pool (x and y), and ϵ - how much they want to increase their collateral. Profit of the attacker will be the ϵ minus the attack cost. Same as in example before, we assume the riskiest scenario - that attack happened during the longest time of price update (USDC-ETH pool, Uniswap V2). To make the example more general, we change the pool liquidity and attacker's profit in terms of their collateral (how many times it is larger than initial collateral value).

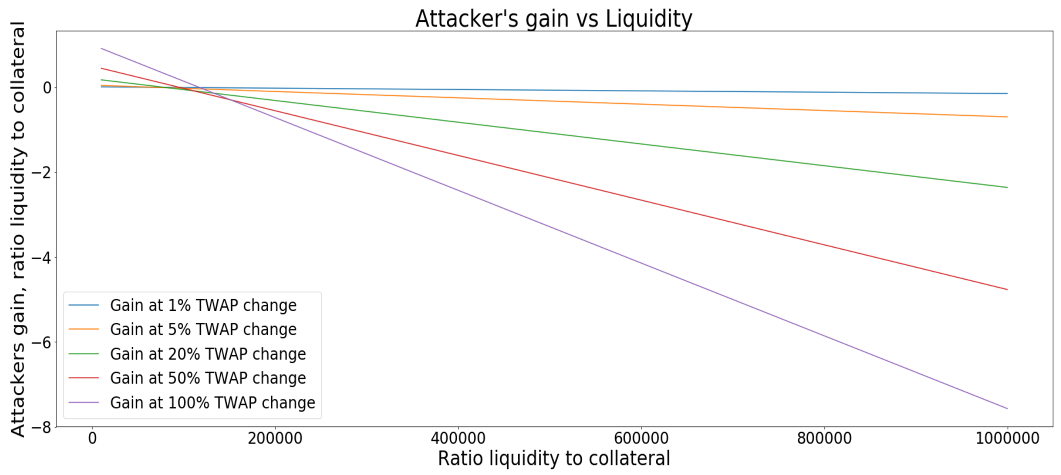

Figure 7: Recommendation for the safe setting of the collateral size and pool liquidity.

As shown in Example 1, we calculate the $TWAP_m$, p_j and Attack Cost for different values of the desired TWAP change (attacker wants to get profit from 1% to 100% of their collateral value) across different liquidity ranges (from very low liquidity pools when the total liquidity in the pool is just 0.1 of the assumed attacker's collateral to higher liquidity pools - 10 times more than attacker's collateral). Figure 7 shows that an attacker can obtain profit only in low liquidity in the pool - when liquidity in pool is larger than the attacker's collateral at least 2 times, the attack becomes not profitable for any range of TWAP manipulation.

6.3 Parameters for protocols prone to liquidation based attacks.

As we showed above, the attack cost linearly depends on the liquidity in the pool - the higher the liquidity, the more costly is the attack. Also, attack cost depends on the initial health factor of the target position - the higher the health factor, the bigger should be the price deviation between spot and manipulated TWAPs and therefore, the more costly is the attack. We have also shown that for the protocols prone to liquidation based attacks, the profit depends on the target position size - it is not reasonable to target small positions, since the attacker's profit is only 5% of the liquidated position. Here we assume the most conservative scenario - when there is a full liquidation of the target position and attacker gets the bonus from the entire value of the position. It needs to be noted, that it is not a common practice and most of the protocols prone to the liquidation-based attacks implement the partial liquidation - when attacker gets bonus from the amount liquidated to meet the threshold.

With everything in mind, we simulate various scenarios of attacks - we change the health factors, position sizes and liquidity size in AMM-based DEX pool. Note that A from the equation 4 will stay fixed at $A = 100000$ for all scenarios. From the fixed A we change D in order to get the varying values of the position's health factor. The liquidation threshold is assumed to be $LT = 1.2$ for all cases.

As shown in Example 2 above, we calculate the $TWAP_m$, p_j and Attack Cost AC for every health factor across all liquidity ranges from 10k to 1mln USD. We assume the riskiest scenario - that attack happened during the longest time of price update (USDC-ETH pool, Uniswap V2) and calculate the attack cost. *Safe Position Size* is then the maximum size of the position 5% of which is higher than the attack cost. Only when condition $SafePositionSize \times 0.05 > AC$ is met, the attack would be reasonable.

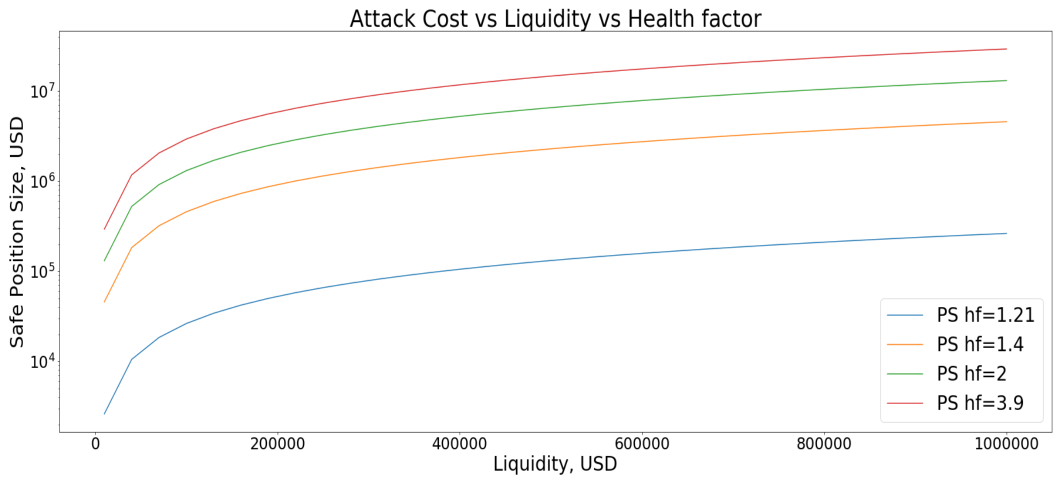

Figure 8: Recommendation for the safe setting of the position size and pool liquidity.

Figure 8 demonstrates the maximum safe position sizes for given health factors and liquidity size - the threshold above which the attack is *profitable* for an attacker. Overall, the risky scenarios are when we have small liquidity pools in AMM and when we have large positions with the small health factor in our protocol. As it can be seen from Figure 8, high health factor increases position requirement and therefore, the safety of protocol drastically.

To take a closer look at liquidity and safe position size requirements, we plot the most risky scenario separately - when the health factor is slightly above the liquidation threshold. From Figure 9 it can be seen that attack cost (orange line) can be quite cheap when we have pools with small liquidity and therefore, having positions of large size can be quite dangerous.

7 Conclusions

Obtaining the price information for smart contracts is a challenging task and every step of this process can be prone to various attacks. One of the most popular ways to obtain price information is to use AMM-based DEXs as a data feeder and aggregate the output using the statistical measure - weighted average. In this report we looked at how to safely adjust the window size for TWAP aggregation to reduce the risk of price manipulation attacks. We derived the exact attack cost when a constant product AMM-based DEX is used as an oracle and estimated the profits of an attacker in two different types of protocols.

Overall, when using AMM-based DEX as an oracle, there is a chance of successful price manipulation. However, understanding the mechanics and cost behind it, it is possible for protocols to adjust key parameters (TWAP window size, position size, liquidity requirements) in order to safely use the oracle.

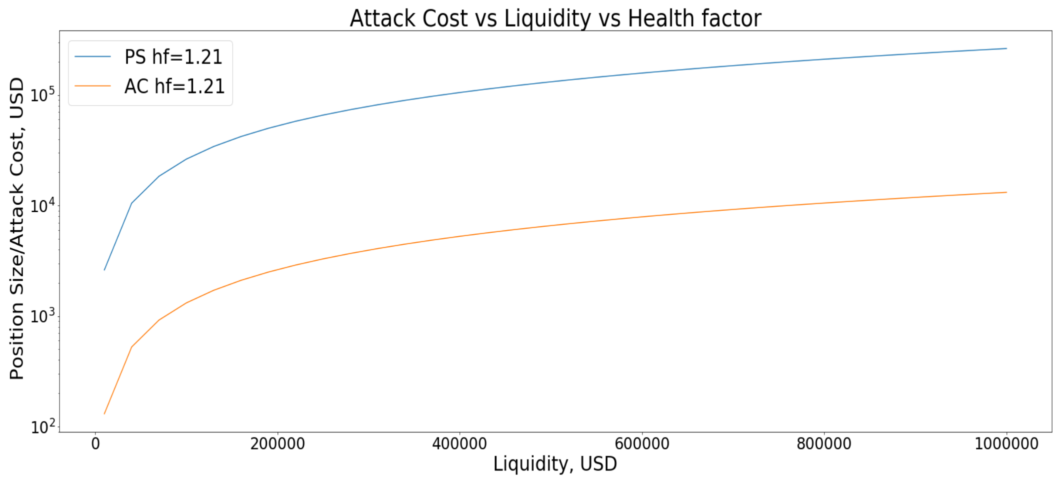

Figure 9: Recommendation for the safe setting of the position size and pool liquidity.

References

[1] M. Bentley *Manipulating Uniswap V3 TWAP Oracles*. November 2021

[2] S. Eskandari, M. Salehi, W. C. Gu and J. Clark *SoK: Oracles from the Ground Truth to Market Manipulation*. CoRR 2021

[3] G. Caldarelli, J. Ellul *The Blockchain Oracle Problem in Decentralized Finance—A Multivocal Approach*. Applied Sciences 2021

[4] H. Adams, N. Zinsmeister, M. Salem, R. Keefer and D. Robinson *Uniswap V3 Core*. March 2021

[5] J. Xu, N. Vavryk, K. Paruch, S. Cousaert *SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) protocols*. ar-xiv preprint 2021