Manipulating Uniswap v3 TWAP Oracles

Michael Bentley

August 8, 2022

1 Introduction

To manipulate a geometric mean time-weighted average price (TWAP) on Uniswap v3, an attacker needs to manipulate the spot price for at least 1 block. The scale of the manipulation depends on the deviation they can achieve between the wider-market spot price and the manipulated spot price, and how many blocks they can hold the price manipulation for.

The cost of the attack depends on how much existing liquidity sits in between the current spot price and the manipulated spot price, the extent to which arbitrageurs can revert the attacker's price manipulation, the ability of the attacker to benefit from their own price manipulation, and the ability of the attacker to back-run their own manipulation, among other complex factors.

The potential cost of a manipulation can be estimated with a high degree of precision if we assume that the cost is proportional to the value of the arbitrage opportunity created by the manipulation each block, and there is at least some liquidity in a Uni v3 pool distributed across the entire price range. This assumption allows us to use the constant-product formula from Uni v2 to derive exact solutions for the cost of a manipulated price on each block.

Here, we use this approach to provide estimates for the cost of various types of manipulation of a DAI-USDC TWAP over a window of 30 minutes. We consider single-block and multi-block attacks. We show that even a little liquidity in the range can push the cost of a single-block attack to extraordinary levels. We find that a multi-block is potentially much less costly to implement, but leaves the would-be attacker with no clear path to profit. Practical considerations show that in real-life circumstances an attack on a Uni v3 TWAP for a liquid asset would be infeasible.

1.1 How to Manipulate a Uni v3 TWAP Oracle

To manipulate a geometric mean TWAP requires an attacker to manipulate the spot price on at least one block within the TWAP window. And because TWAP oracles are updated on the first transaction of block i using the last trade price on block $i-1$, an attacker is required to expose their price manipulation efforts to arbitrageurs for at least one block. Let us explore how far an attacker needs to move the spot price in order to impact the TWAP to see what kind of arbitrage opportunity they will expose.

Consider the DAI-USDC price. The spot price at block i is denoted p_i. The geometric mean TWAP can be calculated over n blocks as the nth root of the product of the spot price on each block:

$$\text{TWAP} = \left(\prod_{i=1}^{n} p_i \right)^{\frac{1}{n}}. \tag{1}$$

Now consider the effect of an attacker on the TWAP. Suppose the spot price on each block is manipulated to some target q, for a period of m blocks, and is otherwise some constant spot price p for the remaining $n-m$ blocks. Then the TWAP is calculated as

$$\text{TWAP} = \left(p^{(n-m)} \cdot q^m\right)^{\frac{1}{n}}. \tag{2}$$

An attacker wanting to manipulate the TWAP to some particular oracle price over m blocks will need to know what spot price q they need to move the normal spot price p to in each of those blocks. We can calculate this by simply rearranging equation (2). We obtain

$$q = \sqrt[m]{\frac{\text{TWAP}^n}{p^{(n-m)}}}. \tag{3}$$

This equation shows that it is surprisingly difficult to move the geometric mean TWAP from the wider market spot price when manipulated blocks are few in number relative to unmanipulated blocks. That is, the spot price must be moved a significant distance from its wider market price in order to have even a modest impact on the TWAP.

An example or two will make this clear. Let us assume that the TWAP is calculated over $n = 144$ blocks (roughly a 30 minute window) and that the initial DAI-USDC spot price is $p = 1$ USDC/DAI.

The attacker can only really begin to gain an advantage of any kind on a lending protocol if they can manipulate the price higher (lower) than the maximum loan-to-value % (LTV). Let us assume that the LTV is 0.75, so that the attacker must move the price up (down) by at least 35% to be able to borrow even slightly more than they should ordinarily be able to. In other words, the attacker wants to achieve a manipulated TWAP = 1.35 USDC/DAI. Note that the attack requirements and costs are symmetric with respect to upwards and downward price movements, so we only need to consider single-sided cases as examples.

Most attackers would prefer to carry out their activities within a single atomic transaction, since this will usually enable them to ensure a risk-free return. Here, however, an attacker needs to take on some risk because their attack will span a minimum of two transactions. In general, the more blocks required to drive a manipulation, the harder it becomes for an attacker to ensure a return.

Let us therefore first consider a situation in which an attacker wants to limit their exposure by manipulating the spot price on a single block, so that $m = 1$. We find that the spot price they need to achieve to manipulate the oracle price to \$1.35 is

$$q = \sqrt[1]{\frac{(1.35 \text{ USDC/DAI})^{144}}{(1 \text{ USDC/DAI})^{(144-1)}}} \approx 5,862,227,430,474,700,000 \text{ USDC/DAI}. \tag{4}$$

Clearly, this is not feasible. Even if the attacker had access to significant mining power, they could not perform a selfish mining attack, because there would usually simply be insufficient assets available for them to do so. Infinite flash minting would not help, because the attacker needs to perform this attack over two blocks.

The attacker might therefore try to spread the capital requirements over a larger number of blocks. Let us suppose that they can spread the attack over of $m = 10$ blocks. Then the spot price they need to achieve to manipulate the oracle price to\$1.35 is

$$q = \sqrt[10]{\frac{(1.35 \text{ USDC/DAI})^{144}}{(1 \text{ USDC/DAI})^{(144-10)}}} \approx 75.30 \text{ USDC/DAI}. \tag{5}$$

Compared to our last answer, this is a 'mere' 75x price manipulation of DAI-USDC for 10 blocks. As we increase the number of blocks an attacker feels capable of manipulating, the price per block that they need to achieve the target TWAP price gets lower and lower. The equivalent over 20 blocks

requires an 8.5x spot price manipulation, and if they can manipulate the price for 15 minutes (approx 72 blocks), the requirement is around 2x spot price manipulation.

So the question now is how costly is it to carry out a multi-block attack and manipulate the spot price for m number of blocks in this way? This depends on how much liquidity is already in the pool, how effective arbitrageurs will be in returning the attackers spot price manipulation back to the wider-market level, and the ability of the attacker to back-run their own manipulation in order limit their exposure to arbitrage.

1.2 Cost of Manipulating a Uni v3 TWAP Oracle

On Uniswap v3 most liquidity providers tend to deposit liquidity in a concentrated fashion around the current price. This helps them to maximise the fees they make from market making. However, it also potentially lowers the cost of price manipulation attacks.

If liquidity is tightly concentrated in the range $(p-\epsilon, p+\epsilon)$ for some small ϵ, then the attacker can wipe out half of the concentrated liquidity at a very low slippage cost, allowing the price to become very sensitive to further swaps.

One way to combat this problem is for users and protocols that rely on Uni v3 TWAPs to deposit their own liquidity to the pool spread across the entire range. Even small amounts of liquidity deposited this way can dramatically increase the cost to an attacker of moving the price in the pool.

We can illustrate this using the mathematics of a Uni v2 pool. Consider the DAI-USDC pool from the previous section. The current DAI-USDC spot price is now defined as $p = x_{\text{USDC}}/x_{\text{DAI}}$, where x_{USDC} is the total amount of USDC in the pool, and x_{DAI} is the total amount of DAI in the pool.

The constant-product formula defines liquidity in the pool:

$$x_{\text{USDC}} \cdot x_{\text{DAI}} = k \tag{6}$$

An attacker moves the spot price by swapping against the pool. The swap an attacker needs to make depends on whether or not they want to manipulate the price up or down.

We begin by assuming that they want to manipulate the spot price up, so that $q > p$. This requires them to swap USDC into the pool to withdraw DAI. How much USDC do they need to swap to achieve a price of q given the current price p and the liquidity, k?

The manipulation swap requires them to trade an amount Δx_{USDC} into the pool to receive an amount Δx_{DAI} out of the pool. Swaps maintain liquidity in the pool. This means that after the swap, the new liquidity is

$$k = (x_{\text{USDC}} + \Delta x_{\text{USDC}})(x_{\text{DAI}} - \Delta x_{\text{DAI}}) = x_{\text{USDC}} \cdot x_{\text{DAI}}. \tag{7}$$

Rearranging, we find that the change $-\Delta x_{\text{DAI}}$ induced by a swap of Δx_{USDC} is

$$\begin{aligned} -\Delta x_{\text{DAI}} &= \frac{x_{\text{USDC}} \cdot x_{\text{DAI}}}{x_{\text{USDC}} + \Delta x_{\text{USDC}}} - x_{\text{DAI}} \\ &= x_{\text{DAI}} \left(\frac{x_{\text{USDC}}}{x_{\text{USDC}} + \Delta x_{\text{USDC}}} - 1 \right) \\ &= -x_{\text{DAI}} \cdot \frac{\Delta x_{\text{USDC}}}{x_{\text{USDC}} + \Delta x_{\text{USDC}}}. \end{aligned} \tag{8}$$

Now, recalling that the target price for the attacker after the swap is $q > p$, we have

$$q = \frac{x_{\text{USDC}} + \Delta x_{\text{USDC}}}{x_{\text{DAI}} - \Delta x_{\text{DAI}}}. \tag{9}$$

Substituting in to this equation for $-\Delta x_{\text{DAI}}$ from equation (8), we have

$$\begin{aligned} q &= \frac{x_{\text{USDC}} + \Delta x_{\text{USDC}}}{x_{\text{DAI}} - x_{\text{DAI}} \cdot \frac{\Delta x_{\text{USDC}}}{x_{\text{USDC}} + \Delta x_{\text{USDC}}}} \\ &= \frac{x_{\text{USDC}} + \Delta x_{\text{USDC}}}{x_{\text{DAI}} \left(1 - \frac{\Delta x_{\text{USDC}}}{x_{\text{USDC}} + \Delta x_{\text{USDC}}}\right)} \\ &= \frac{x_{\text{USDC}} + \Delta x_{\text{USDC}}}{x_{\text{DAI}} \cdot \frac{x_{\text{USDC}}}{x_{\text{USDC}} + \Delta x_{\text{USDC}}}} \\ &= \frac{(x_{\text{USDC}} + \Delta x_{\text{USDC}})^2}{k}. \end{aligned} \tag{10}$$

Solving for Δx_{USDC}, the amount of USDC needed to induce a change in spot price from p to q, we have

$$\Delta x_{\text{USDC}} = \sqrt{x_{\text{DAI}} \cdot x_{\text{USDC}} \cdot q} - x_{\text{USDC}}. \tag{11}$$

This is the amount of USDC a user needs to swap in to move the spot price of DAI from p to q on a single block. The amount of DAI they receive out is given by equation (8).

Now let us consider the scenario in which an attacker wants to manipulate the spot price down, so that $q < p$. This requires them to swap DAI into the pool to withdraw USDC. Again, we can ask how much DAI do they need to swap into the pool to achieve a price of q given the current price p and the liquidity already in the pool, k?

The manipulation swap requires them to trade an amount Δx_{DAI} into the pool to receive an amount Δx_{USDC} out of the pool. After the swap, the new liquidity is

$$k = (x_{\text{USDC}} - \Delta x_{\text{USDC}})(x_{\text{DAI}} + \Delta x_{\text{DAI}}) = x_{\text{USDC}} \cdot x_{\text{DAI}}. \tag{12}$$

Rearranging, we find that the change $-\Delta x_{\text{USDC}}$ induced by a swap of Δx_{DAI} is

$$-\Delta x_{\text{USDC}} = -x_{\text{USDC}} \cdot \frac{\Delta x_{\text{DAI}}}{x_{\text{DAI}} + \Delta x_{\text{DAI}}}. \tag{13}$$

Now, recalling that the target price for the attacker after the swap is $q < p$, we have

$$q = \frac{x_{\text{USDC}} - \Delta x_{\text{USDC}}}{x_{\text{DAI}} + \Delta x_{\text{DAI}}}. \tag{14}$$

Substituting in to this equation for $-\Delta x_{\text{USDC}}$ from equation (13), we have

$$
\begin{aligned}
q &= \frac{x_{\text{USDC}} - x_{\text{USDC}} \cdot \frac{\Delta x_{\text{DAI}}}{x_{\text{DAI}}+\Delta x_{\text{DAI}}}}{x_{\text{DAI}} + \Delta x_{\text{DAI}}} \\
&= \frac{x_{\text{USDC}} \left(1 - \frac{\Delta x_{\text{DAI}}}{x_{\text{DAI}}+\Delta x_{\text{DAI}}}\right)}{x_{\text{DAI}} + \Delta x_{\text{DAI}}} \\
&= \frac{x_{\text{USDC}} \cdot \frac{x_{\text{DAI}}}{x_{\text{DAI}}+\Delta x_{\text{DAI}}}}{x_{\text{DAI}} + \Delta x_{\text{DAI}}} \\
&= \frac{k}{(x_{\text{DAI}} + \Delta x_{\text{DAI}})^2}
\end{aligned}
\tag{15}
$$

Solving for Δx_{DAI}, the amount of DAI needed to induce a change in spot price from p to q, we have

$$
\Delta x_{\text{DAI}} = \sqrt{\frac{x_{\text{DAI}} \cdot x_{\text{USDC}}}{q}} - x_{\text{DAI}}. \tag{16}
$$

This is the amount of DAI a user needs to swap in to move the spot price of DAI from p to q on a single block. The amount of USDC they receive out is given by equation (13).

Whether the attack is manipulating the spot price up or down, the difference between Δx_{DAI} and Δx_{USDC} gives us an upper limit for the cost of the attack, c. The cost of the attack is roughly the slippage between the unmanipulated price and the manipulated one, under the assumption that arbitrageurs will profit from the attacker by returning the manipulated price back to normal after the attack. The cost is an upper limit, because sophisticated attackers may be able to recoup some of the slippage costs by arbitraging their own transaction after the attack (see below). The upper-limit slippage cost, per manipulated block, is

$$
c = \Delta x_{\text{USDC}} - \Delta x_{\text{DAI}} \cdot p, \tag{17}
$$

where we have multiplied Δx_{DAI} by the unmanipulated price p in order to put the cost of the attack on the same price scale (in terms of USDC). Note that the upper limit on the overall cost of an attack is $m \cdot c$, because the cost c must be paid on m manipulated blocks.

Let us now return to our earlier examples to find out how much they would cost. Let us assume that before the attack the reserves of DAI and USDC are equal, so that $x_{\text{USDC}} = 50,000$ USDC, and $x_{\text{DAI}} = 50,000$ DAI.

In the first example, recall that the attacker manipulated the spot price for a single block. The amount of USDC they would need to swap into the pool to achieve the manipulated price for this attack is

$$
\begin{aligned}
\Delta x_{\text{USDC}} &= \sqrt{50,000 \text{ USDC} \cdot 50,000 \text{ DAI} \cdot 5,862,227,430,474,700,000 \text{ USDC / DAI}} - 50,000 \text{ USDC} \\
&\approx 121,060,185,709,756 \text{ USDC}.
\end{aligned}
\tag{18}
$$

This is quite a lot more USDC than there currently is in existence. The upper limit for the final cost is $C = c \approx 121,060,185,659,756$ USDC. Such an attack does not seem feasible. Remember, even flash minting is not an option here, because this attack is carried out across multiple blocks.

In the second example, recall that the attacker attempted to manipulate the spot price for $m = 10$ blocks. The amount of USDC they must swap into the pool to achieve this price, on each of $m = 10$ manipulated blocks, is

$$\begin{aligned}\Delta x_{\text{USDC}} &= \sqrt{50,000 \text{ USDC} \cdot 50,000 \text{ DAI} \cdot 75.30 \text{ USDC / DAI}} - 50,000 \text{ USDC} \\ &\approx 383,884 \text{ USDC}.\end{aligned} \tag{19}$$

The upper limit for the final cost per block is $c \approx 339,645$ USDC. If this is paid for $m = 10$ blocks, then the overall cost is $C = 3,396,454$ USDC. These are seemingly much more feasible numbers than we obtained from the single block price manipulation. Could a sufficiently motivated attacker therefore pull off such an attack? There are several important practical reasons we have neglected to consider so far that make the answer probably no.

First, we have assumed an unrealistic hypothetical case in which there is zero other liquidity in the pool, concentrated or otherwise. We have also looked at a stable-coin pairing, which is a particularly troublesome pair to use because it encourages extreme levels of liquidity concentration. Nevertheless, in reality, there is often a lot of liquidity in Uni v3 pools, both concentrated and otherwise, even for stable coin pairs. For example, let us look at what is required to carry out this attack on the DAI-USDC pair on Uni v3 today, without any further addition of distributed liquidity. As of today (12/11/21), a 100,000,000 USDC swap incurs slippage of just -0.344%. An attacker could remove this concentrated liquidity pretty 'cheaply', as we discussed above, because they could swap USDC into the pool and they would receive approximately the same value in DAI back (i.e. $\approx 100,000,000$ DAI). If there were no further liquidity in the pool, they could then swap a small amount of extra USDC and move the DAI-USDC spot price arbitrarily. In practice, however, there is still enough liquidity in the tail to make this infeasible: to get to the target attack price of \$75 would require the attacker to deposit 3,050,000,000 USDC in order to receive just 40,746,500 DAI. This represents an arbitrage opportunity of gigantic proportions. An attacker might hope to back run their own price manipulation here, but remember that they need to do this every block for 10 blocks. It is extremely unlikely they would be able to do this, even with access to services like flashbots or access to significant amounts of mining power.

Second, it is important to remember that in this kind of attack the price manipulator has no way of ensuring that they are the beneficiaries of the attack. A manipulated TWAP emerging over a period of half an hour would alert a wide variety of opportunists to the opportunity to profit, meaning that the attacker's final share of the spoils from the attack would more than likely be a small fraction (if anything) of the total profits available. This contrasts sharply to most attacks in decentralised finance, which are usually carried out in a risk-free atomic transaction environment. Ultimately, therefore, an attacker would realistically need to put an enormous amount of funds on the line with no real way to guarantee any kind of return.

Third, an attack of this nature would require the attacker to acquire a large initial deposit in an anonymous fashion, and be able to cash out the proceeds in an anonymous fashion too. The risk of being caught is much greater for this kind of attack than the flash loan-based attacks more typically seen in attacks on decentralised finance protocols. An attacker could feasibly find themselves facing prosecution, having lost all of their principal, and having made no profit.

1.3 Flashbots

One might imagine that an attacker could use a service like flashbots to hide their manipulation transaction, back run their own manipulation transaction to alleviate slippage, or fill up blocks to prevent arbitrage. However, as we highlighted above, the ordering of the TWAP update means that all spot price manipulation attacks will be publicly exposed for at least one block, presenting a potentially large arbitrage opportunity to flashbots searchers and other arbitrage bots. The efficient block space market created by flashbots is therefore expected to require the attacker to bid up the bundle cost of each block to a level that approaches the slippage cost, c. Rather than make TWAP manipulation attacks more likely, one could even argue that they make them less likely. However, this is a complex and continuously evolving topic that will require a lot more research.

2 Liveness

Uniswap v3 TWAP oracles may fail in more subtle ways than manipulation. When calling observe, the oracle specifies a secondsAgo which is a parameter that sets the length of the TWAP. Observations is an array that contains cumulative values of the TWAP. If you call observe with secondsAgo = $[3600, 0]$ you should get a 1 hour TWAP right? Well as long as sufficinetly many entires of the observations array have been initialized so that the earliest tick cumulative was added at least an hour ago, you would be right. If the earliest tick cumulative is more recent than an hour ago, you are not actually getting a 1 hour TWAP you are getting something shorter.

So how do we know when we have sufficiently many observations initialized? The answer requires an intricate understanding of how the TWAP is calculated. The first time a uniswap pool is called within each block, it updates one of its observations with the data from that block. So depending on how often the pool is called, it could be as often as every block. Indeed, we should probably just assume that it gets called every block since an attacker could simply interact with the pool every block if it was advantageous to do so. So what if we wanted to compute a 1 hour TWAP, we are going to call observe(secondsAgo = [3600,0]). Then we need enough observations so that we will consistenty have an observation from an hour ago in storage. The average blocktime is 13 seconds on ethereum so can we just initialize $3600/13 = 277$ observations. But then what we will be getting is an observation which is sometimes a 1 hour TWAP and sometimes a 55 minute TWAP because the number of blocks mined in 1 hour is a random variable.

In proof of work systems, we model block arrival as a poisson point process. Consider the following argument for why. When a miner checks a nonce, the random variable S which encodes whether the hash produced is sufficiently small the meet the current difficulty requirement is bernouli(p) where p is very small. So the number of valid blocks produced in a given time interval is $P = H \cdot S$ where H is the total hashrate over that window. So the number of blocks mined in a window P is binomial with large n and small p. The poisson point process is defined as the limit of a binomial as $n \to \infty$ and $p \to 0$ so this model is quite accurate.

Given that block arrival in a 1 hour window on Ethereum is ∼Poisson(3600/13), we can see that there is a high probability that more than 280 blocks get mined in a 1 hour period.

Of course, even if we aren't getting a 1 hour TWAP, we are probably getting a 55 minute TWAP or around there so this isn't neccearily a bad thing. In fact it might actually be a good thing if we play our cards right. When should we expect a ton of calls to a the pool and therefore heavy use of the oracle slots in a short period? Precicely when volatility is highest, which also happens to be when we want our oracles to be the most live. So maybee it makes sense to have an oracle which becomes more live when the pool is under heavy load. We could do this by calling observe(secondsAgo = [a,0]) in which case we will get a standard length TWAP unless the network is under heavy enough load that the oldest observations have been overwritten. The problem with this aproach is that anyone can increase the number of initialized observations for whatever reason so we may not always have this addaptive liveness property. If you want to have adaptive liveness no matter what, you can query the observations directly so you would call the observation 50 slots before the current one for example[1].

The key insight in this discussion is that the number of initialized observations on a univ3 pool decides its TWAP behavior as much or more than secondsAgo in the observe call. The actionable result is to monitor this input when calculating oracle quality.

3 Conclusion

Manipulation of the Uniswap v3 geometric mean TWAP is technically possible, but only under restrictive conditions. To carry out a single-block manipulation attack, there must be negligible liquidity in a

[1] Be careful implementing this because the implemenation wraps around the observations array, overwriting previous observations.

pool, otherwise the scale of the spot price manipulation needed will often require a swap of more assets than an attacker can feasibly access. To carry out a multi-block attack, an attacker must have access to significant initial funds and must be prepared to lose all of them, with no guarantee of making a profit. The pool must also have very little liquidity distributed over the entire range. An attacker may attempt to use flashbots to limit the impact of arbitrage on their spot price manipulation. However, flashbots creates an efficient market for block space, and it is more than likely that searchers will bid up the bundle cost for blocks to the marginal price of the arbitrage available. Ultimately, multi-block attacks are unlike any attack we have see yet in decentralised finance, because they do not offer risk-free profit opportunities to attackers of the kind usually used to attack protocols in the context of an atomic transaction.

4 Acknowledgements

Thanks to Doug Hoyte, banteg, Mudit Gupta, Robert Miller and Noah Zinsmeister for reading earlier drafts of this article and providing helpful feedback.

5 Collaboration

Got thoughts on this analysis? Let's work together! There are many other technical and practical considerations that I have neglected. I welcome feedback from the decentralised finance and Ethereum security communities to help improve this research.

6 Disclaimer

This paper is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. It should not be relied upon for accounting, legal or tax advice or investment recommendations. This paper reflects current opinions of the author only. The opinions reflected herein are subject to change without being updated.