**Blog: Understanding Price Oracle Manipulation**

DeFi strives to revolutionize financial services, one of the main risks facing DeFi is cyberattacks stealing investor's funds from platforms. The [most common](https://coingeek.com/the-defi-hacks-of-2020/) DeFi hack is oracle manipulation, a partial list of platforms that fell victim to this attack are [yEarn](https://blog.trailofbits.com/2020/08/05/accidentally-stepping-on-a-defi-lego/), [Harvest](https://www.rekt.news/harvest-finance-rekt/), [xToken](https://www.rekt.news/xtoken-rekt/), [bZx](https://peckshield.medium.com/bzx-hack-ii-full-disclosure-with-detailed-profit-analysis-8126eecc1360), [Cheese Bank](https://peckshield.medium.com/cheese-bank-incident-root-cause-analysis-d076bf87a1e7), Synthetix.

Smart contracts are great at executing predetermined code but their main weakness is that they are isolated from the outside world and are only exposed to on-chain information.Developers use oracles to solve this problem, oracles are protocols that transmit information from outside the blockchain and add it on-chain so smart contracts are updated. Uploading information on-chain is essential for creating decentralized platforms with real-world use cases. For example, lending platforms need up-to-date asset prices to know if a borrower is under-collateralized and needs to be liquidated (for more information on DeFi lending platforms see [part 2](https://redefine.net/defi-101-part-2/) of the DeFi 101 series). Other use cases include the Augur platform that allows users to bet on the outcomes of real-world activities - like the election results, oracles can upload the results to the smart contract which will settle according to the results.

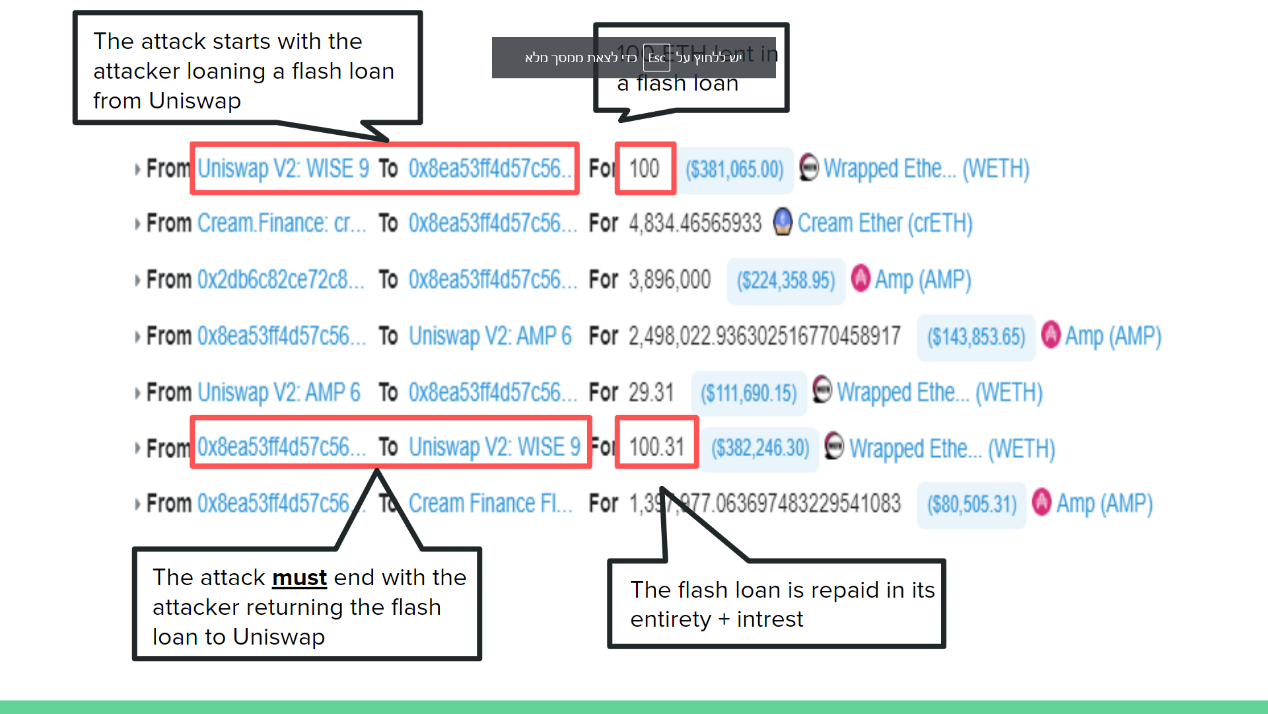
The main challenge with oracles is ensuring that they are sufficiently decentralized to prevent malicious actors from uploading false information. To achieve a satisfactory level of decentralization projects like Chain Link and Band use an incentive structure that incentivizes participants to upload true information and penalizes them for supplying false information. Some projects namely [Maker](https://makerdao.com/en/) put an emphasis on oracle decentralization and aggregate many different price feeds, some of them on-chain and some from off-chain CEX (Centralized Exchanges) and then takes the average between them, price feeds that deviate more than 1% are discarded. This makes an oracle attack on the platform extremely difficult (for more information regarding the [Maker oracle](https://docs.makerdao.com/smart-contract-modules/oracle-module)).

Some DeFi projects don't put as much emphasis on decentralization and rely instead on on-chain information from a single DEX to get a price feed (see [part 2](https://redefine.net/defi-101-part-2/) for more information on DEX). A common method attackers use is to manipulate the price of an asset listed on a DEX using a flash-loan to artificially change its price.

**What is a Flash Loan?**

Flash loan is a financial instrument unique to crypto enabled by the use of smart contracts. Flash loans allow borrowers to loan huge sums of money -virtually an unlimited amount- without collateral for the duration of one block. The flash loan supplier requires the borrower to return the loan inside the same block, if the borrower fails to do this then the transaction reverts and the money returns to the borrower - hence flash loans are risk-free for the lender. Flash loans are beneficial when arbitraging thus have an important part in improving DeFi’s price efficiency. On the flip side, flash loans can also be used by hackers to exploit weakness and inflect huge financial damages in a short time frame. (for more [information](https://decrypt.co/resources/what-are-flash-loans-the-defi-lending-phenomenon-explained)).

Here is an example of a flash loan attack on the CREAM protocol:



**How a oracle attack is executed using a flash loan?**

Lets assume that a attacker wants to steal funds from a lending platform relying on the Uniswap oracle. The lending platform requires borrowers to have a over- collateralization ratio of 150%

DeFi Dex's use the AMM method to determine asset prices based on the amount of assets held by the reserve. A attacker can trade between an asset pair and while

can profit from

but the assets held by the reserve changes as users trade between ETH and USD. What if a malicious user performs a trade before and after taking a loan from your platform?

Before the user takes out a loan, they buy 5,000 ETH for 2,000,000 USD. The Uniswap exchange now calculates the price to be 1 ETH = 1,733.33 USD. Now, their 375 ETH can act as collateral for up to 433,333.33 USD worth of assets, which they borrow. Finally, they trade back the 5,000 ETH for their original 2,000,000 USD, which resets the price. The net result is that your loan platform just allowed the user to borrow an additional 333,333.33 USD without putting up any collateral.

This case study illustrates the most common mistake when using a decentralized exchange as a price oracle - an attacker has almost full control over the price during a transaction and trying to read that price accurately is like reading the weight on a scale before it’s finished settling. You’ll probably get the wrong number and depending on the situation it might cost you a lot of money.

**How Can Platforms Prevent Exposure to Oracle Manipulation?**

**How do I protect myself?**

By now, I hope that you’ve learned to recognize the common thread - it's not always obvious that you're using a price oracle and if you don't follow the proper precautions, an attacker could trick your protocol into sending them all of your money. While there’s no one-size-fits-all fix that can be prescribed, here are a few solutions that have worked for other projects in the past. Maybe one of them will apply to you too.

**Shallow Markets, No Diving**

Like diving into the shallow end of a pool, diving into a shallow market is painful and might result in significant expenses which will change your life forever. Before you even consider the intricacies of the specific price oracle you’re planning to use, consider whether the token is liquid enough to warrant integration with your platform.

**A Bird in the Hand is Worth Two in the Bush**

It may be mesmerizing to see the potential exchange rate on Uniswap, but nothing’s final until you actually click trade and the tokens are sitting in your wallet. Similarly, the best way to know for sure the exchange rate between two assets is to simply swap the assets directly. This approach is great because there’s no take-backs and no what-ifs. However, it may not work for protocols such as lending platforms which are required to hold on to the original asset.

**Almost Decentralized Oracles**

One way to summarize the problem with oracles that rely on on-chain data is that they’re a little too up-to-date. If that’s the case, why not introduce a bit of artificial delay? Write a contract which updates itself with the latest price from a decentralized exchange like Uniswap, but only when requested by a small group of privileged users. Now even if an attacker can manipulate the price, they can’t get your protocol to actually use it.

This approach is really simple to implement and is a quick win, but there are a few drawbacks - in times of chain congestion you might not be able to update the price as quickly as you’d like, and you’re still vulnerable to sandwich attacks. Also, now your users need to trust that you’ll actually keep the price updated.

**Speed Bumps**

Manipulating price oracles is a time-sensitive operation because arbitrageurs are always watching and would love the opportunity to optimize any suboptimal markets. If an attacker wants to minimize risk, they’ll want to do the two trades required to manipulate a price oracle in a single transaction so there’s no chance that an arbitrageur can jump in the middle. As a protocol developer, if your system supports it, it may be enough to simply implement a delay of as short as 1 block between a user entering and exiting your system.

Of course, this might impact composability and miner collaboration with traders is on the rise. In the future, it may be possible for bad actors to perform price oracle manipulation across multiple transactions knowing that the miner they’ve partnered with will guarantee that no one can jump in the middle and take a bite out of their earnings.

**Time-Weighted Average Price (TWAP)**

Uniswap V2 introduced a TWAP oracle for on-chain developers to use. The [documentation](https://uniswap.org/docs/v2/core-concepts/oracles/) goes into more detail on the exact security guarantees that the oracle provides, but in general for large pools over a long period of time with no chain congestion, the TWAP oracle is highly resistant to oracle manipulation attacks. However, due to the nature of its implementation, it may not respond quickly enough to moments of high market volatility and only works for assets for which there is already a liquid token on-chain.

**M-of-N Reporters**

Sometimes they say that if you want something done right, you do it yourself. What if you gather up N trusted friends and ask them to submit what they think is the right price on-chain, and the best M answers becomes the current price?

This approach is used by many large projects today: Maker runs a set of [price feeds](https://developer.makerdao.com/feeds/) operated by trusted entities, Compound created the [Open Oracle](https://medium.com/compound-finance/announcing-compound-open-oracle-development-cff36f06aad3) and features reporters such as [Coinbase](https://blog.coinbase.com/introducing-the-coinbase-price-oracle-6d1ee22c7068), and Chainlink aggregates price data from Chainlink operators and exposes it on-chain. Just keep in mind that if you choose to use one of these solutions, you’ve now delegated trust to a third party and your users will have to do the same. Requiring reporters to manually post updates on-chain also means that during times of high market volatility and chain congestion, price updates may not arrive on time.

*Two articles from Peaster [7] and Tarasov [128], display a list of the most famous attacks in the DeFi space. In accordance with Thompson [4], they show that the most common and rewarding DeFi hack is the Flash Loan attack. Due to that reason, Qureshi [76], commenting on the flash loan, declares that to date, the major outcome of Flash Loans is the enabling of flash loan attacks. The bZx incident constitutes a typical example of the oracle problem in DeFi, and it is quite famous for being implemented multiple times on the same platform with success and even higher turnover. Basically, bZx smart contracts utilized the Kyber Decentralized Exchange price feed as an oracle. This dependency was exploited during the attack. The attack that happened on February 18, 2020, will therefore be described and displayed in Figure 10. An attacker obtained a flash loan on the bZx of 7500 ETH, which was split into three parts (3518, 900, and 3082). The first part of 3518ETH, was used to buy sUSD, a synthetic USD token enabled by the Syntetix protocol and which should always keep the price of $1. The sUSD were actually bought at the price of $1. However, a second part of the loaned ETH (900 ETH) was used to buy sUSD on Kyber, artificially inflating the price over $2. The bZx used Kyber as a price oracle, so on their platform, the sUSD was then priced at $2. The overpriced sUSD were then used as collateral on bZx to borrow ETH. Due to the inflated price of sUSD, the hacker was able to borrow 6796 ETH instead of 3518ETH. The attacker then had a total pot of 6796 ETH plus 3082 ETH from the original loan for a total of 9878 ETH. In order for all these transactions to be validated, the initial loan had to be paid back so that the attacker was obliged to send 7500 ETH back to the smart contract. However, he could run away with the surplus of 2378 ETH, equal to $636000 by that time [116]. The Flash Loan attack is an important issue with oracle in DeFi since the very presence of Oracles causes it. As described, no collusion, bug, or data tampering was exploited, but only the oracle and their refresh rate. All the flash loans are performed more or less the same way as the one described here. They are indeed used a lot, for their simplicity and most of all because they come basically at no cost. If they fail, only the transaction fees have to be paid, and if they succeed, then the turnover is priceless.*