Mitigation of Transaction Manipulation Attacks in UniSwap

Ana Tatabitovska[1], Oğuzhan Ersoy[1], and Zekiraya Erkin[1]

[1]Cyber Security Group, Department of Intelligent Systems, Delft University of Technology

Abstract

Front-running is the illegal practice of obtaining information unavailable to the general public with regards to upcoming transactions and performing actions based on this knowledge as to gain profit. This type of attack has been an issue since the introduction of the first stock market and it is not a surprise it has spread into blockchain technologies. With the invention of Decentralized Exchange Systems, this type of exploit has been an even more common occurrence that puts both the user and system at a disadvantage.

This paper aims to explore the circumstances that enable front-running in UniSwap, one of the most used DeFi Systems as of the writing of this paper. Factors such as lack of privacy, slippage, as well as a miner's role in the exploit are analysed in a broad and UniSwap-centered context to provide more insight to the problem.

Moreover, this paper analyses a potential adjustment of a commit-reveal scheme, a time lock scheme, and off-chain slippage limit as possible solutions and an overall analysis of the proposed solutions. Through comparison and adjustments of different solutions, this paper strives to expose where the root of the problem lies - the Ethereum blockchain itself.

1 Introduction

Since the emergence of Bitcoin in 2008, decentralized structures for asset management have gained popularity [1]. Decentralized Finance (DeFi) Systems have taken it a step further by building structures on top of blockchain protocols that enable users to loan, borrow, trade, and in general manipulate and maintain assets without a centralized body as a middleman [2]. However, these systems are still new and prone to issues. UniSwap, the currently most popular DeFi System in terms of volume of asset exchange per 24 hours, is vulnerable to such problems. By mitigating these issues, UniSwap can offer its users a more secure environment than the current one.

One such threat to the safety of UniSwap are transaction manipulation attacks, in which malicious agents seek to exploit the system for monetary gain. This paper aims to answer the question: **How can UniSwap V3 protect itself against transaction ordering attacks and to what extent does a possible solution affect its modus operandi?** To answer this question, a thorough literature analysis of the issue is presented, as well as

an analysis of the impact and cost of solutions. The first goal of this paper is to analyze the factors that contribute to transaction manipulation attacks, with a focus on these attacks in UniSwap. The second goal is to look at solutions on the DeFi System level which UniSwap can undertake to protect itself, such as a commitment scheme, based on the Submarine Commits scheme, and a time lock scheme, as suggested by the 2020 UniSwap audit [3] [4].

Relevant Work Though DeFi Systems are a recent invention, research has already been put into discovering and mitigating potential vulnerabilities. Most of the research is focused on identifying problems and offering theoretical solutions as well as peer-reviewed work that focuses on measuring how impactful certain attacks are.

Several different papers, in the form of Systematization of Knowledge, give an overview of how DeFi systems operate, as well as how they deal with transaction manipulation attacks [5] [6] [7]. These papers present a basis to understanding the circumstances within a DeFi system that enable transaction manipulation and why UniSwap specifically is prone to them. On the other hand, research papers that present findings with regards to transaction manipulation serve to highlight how severe the issue is [8] [9]. Though the problem of transaction manipulation has been analysed from several different angles, few mitigations exist and even fewer are implemented, such as the Submarine Commit solution which is designed for bug bounty withholding attacks [3].

Contributions of the Paper The contribution of this paper can be divided into two parts. The first is the analysis of the causes of front-running in UniSwap, why UniSwap is prone to such attacks, as well as the damage done by these attacks. The second comes in the form of an algorithm adjustment that could prevent front-running in UniSwap, based on Submarine Commits. An analysis of how such a solution works and its costs, as well as a comparison between different solutions in UniSwap, are the main contributions of the second part. The conclusion of this paper aims to offer a recommendation on how to handle transaction manipulation attacks from UniSwap's perspective.

Structure of Paper The following section, Section 2, gives a background of the concepts needed to understand transaction manipulation attacks in UniSwap. The focus of Section 3 is to analyse the factors that enable transaction manipulation attacks. Section 4 looks at the possible solutions offered to mitigate the issue, a commit-reveal scheme and time lock scheme specifically. Following is Section 5, which delves into the ethics and integrity of the research. Section 6 aims to elaborate more on the shortcomings of the mitigations discussed in Section 4, as well as to suggest what should be researched further. Section 7 is the final section of this paper that concludes the findings.

2 Background

This section aims to introduce several important concepts such as Ethereum, transaction manipulation attacks, and UniSwap in order to reason why transaction manipulation attacks should be protected against.

2.1 Ethereum Background

Ethereum is a decentralized blockchain protocol [10]. Decentralization as a concept in Computer Science can be traced back to David Chaum's Mix Network, first introduced in his

1981 paper "Untraceable electronic mail, return addresses, and digital pseudonyms" [11]. Peer-to-peer systems, on the other hand, were popularized by a music sharing platform known as Napster in 1999 [12]. These two concepts are the foundations for what is known today as the blockchain protocol, a decentralized, peer-to-peer system where no central body or central structure controls the flow of execution. As of May 2021, Ethereum is the protocol with the second largest cryptocurrency by market capitalization [10] and is the current most popular choice for Decentralized Finance Systems [13].

In order to talk about transaction manipulation attacks, an overview of how Ethereum works should be given. Ethereum, as of the writing of this paper, uses a consensus algorithm known as proof-of-work to record transactions [14]. As seen in Figure 1, a user transaction is submitted as the first step in the blockchain protocol. This transaction is then broadcast to all nodes in the network and is able to be "mined" by any miner and added to the blockchain, as showcased in Figure 1, steps 2 and 3. Miners are actors in the network who have to sacrifice electrical energy to compute the hash of any transaction to be able to put said transaction in a block [15]. After sufficient transactions have been found by a miner, a block can be formed and added to the blockchain, which makes the block unchangeable from that point on [15].

Figure 1: Simplified illustration of how the Ethereum blockchain works [14]

Though this overview of how the Ethereum blockchain might be simplistic, it is enough to cover the general idea of the workflow. However, every Ethereum transaction has a special attribute which is important when discussing transaction manipulation attacks and should therefore be explored. This characteristic is Gas, an Ethereum native unit of measurement regarding how much it costs to put a transaction on the blockchain [15]. Gas is important because every block in Ethereum has a limited maximum gas price and different transactions have different costs [15]. This means that there is no fixed number of transactions per block and miners have to strategically order transactions to fill up the block as much as they can whilst also making a profit off their work. Gas is crucial to the problem of transaction manipulation attacks because Gas price can be used as a "bribe" for miners to prioritize one transaction over another [6].

2.2 Decentralized Finance Systems

Decentralized Finance Systems, at their core, are a set of smart contracts built on top of blockchain technology that allow for asset exchange through borrowing, loaning, funding, and more [5]. The goal of these systems is to give the power back to the users by removing the middleman that exists in centralized systems and letting the individuals maintain their own assets.

There are four ideal principles that make a Decentralized Finance System according to Werner et al. [5]. The first is non-custodial, meaning the user's assets are always in their

possession and no one has "custody" over them like in centralized systems [5]. The second ideal is permissionless; a system should not block any user from interacting with its services [5]. The third is openly auditable so that anyone at any point can verify the system's correctness [5]. The fourth and final ideal is composable which refers to the ability of the actions within a DeFi System to be combined in new ways [5].

In conclusion, DeFi Systems can be thought of as the counter-response to centralized structures, such as banks, that use a distributed ledger as their foundation. These systems have a set of rules written in code that users must follow and said users hold the power over their own assets and have a say in the system's rules [5].

2.3 UniSwap

An instance of the previously described Decentralized Finance Systems is UniSwap. Introduced in 2018, UniSwap is a DeFi System boasting an average volume of 700 billion dollars a day [16]. The popularity of UniSwap is unsurprising as it offers a lot of lucrative characteristics such as liquidity pools, support for a variety of tokens, and a welcoming community of developers.

One of the main attractions of UniSwap, which introduces a new functionality on top of Ethereum, is liquidity pools. A liquidity pool is simply a smart contract that holds two different currencies and allows funds to be deposited there by users [17]. The users who do so are known as liquidity providers - they provide the pool with liquidity which can then be loaned or swapped. Moreover, liquidity pools have a special role to play in UniSwap, in that they serve as oracles, tools for providing current asset value, for the prices of the pair they hold [17]. In fact, this manner of providing asset value is a protocol known as Automated Market Maker (AMM). An AMM provides the price for an asset through a mathematical formula, and, in UniSwap, this mathematical formula uses the exact reserves in a given pool [18]. UniSwap's AMM is also known as a CFMM, Constant Function Market Maker, because the result of the formula must remain constant [17].

$$x * y = k$$

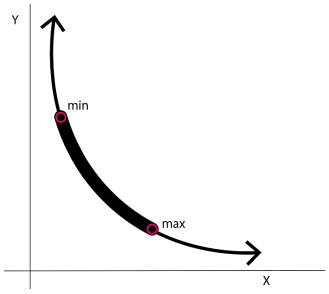

Figure 2: UniSwap's Concentrated Liquidity and AMM [19]

In UniSwap's case, k is the product that must remain constant, and x and y represent the amounts of each of the tokens currently in the pool. Trades in which k ends up changing open arbitraging opportunities for other users to even out the imbalance [17]. This function can be observed in Figure 2, to better understand how the curve works and the relation between x and y.

As of UniSwap V3, a new aspect beneficial to Liquidity Providers is Concentrated Liquidity [19]. This perk allows Liquidity Providers to specify the range in which they would like to provide liquidity which, if utilized properly, would have a much higher earnings percentage for Liquidity Providers. As observed in Figure 2, this feature allows liquidity providers to specify a *minimum* and *maximum* ratio of two tokens in a pool between which they are willing to provide liquidity in said pool [19]. Whilst UniSwap has good intentions behind the introduction of this new attribute, there is a level of danger with regards to manipulation by utilizing Concentrated Liquidity.

Namely, if observable by an attacker, the bounds of a certain user can be used to render their liquidity useless, which can theoretically be done with a transaction manipulation attack. Though V3 was too recently rolled out to gather any data regarding this concern, a future audit should reveal if Concentrated Liquidity abuse by an attacker leads to a new UniSwap specific transaction manipulation attack.

2.4 Transaction Manipulation Background

In centralized trade markets and finance systems, front-running is the illegal practice in which an actor profits off of privileged information about an upcoming transaction [6]. In Decentralized Finance systems, transaction ordering attacks are the family of all problems that exploit slippage caused by a transaction and can be classified as front-running, back-running, and sandwich attacks [20]. The fundamental difference between centralized and decentralized transaction-based attacks is the manner in which attacks are carried out. Moreover, the terminology used from here on is based on blockchain concepts, meaning front-running refers to a specific type of transaction attack, while transaction manipulation attacks is the umbrella term for any attack that manipulates the order of transaction execution.

In DeFi systems the transaction manipulation attacks effectively take place in the underlying blockchain protocol [6]. Namely, the attacker can find out a certain transaction is currently in some transaction pool, waiting to be added to the blockchain. The attacker then manipulates the gas price of their transaction to be higher or lower than the targeted transaction, thus ensuring their transaction will be placed in the correct order to exploit the targeted transaction [6].

2.4.1 Types of Transaction Manipulation Attacks

There are several different ways an attacker can exploit a targeted transaction by manipulating the ordering of the transactions. However, in DeFi systems, the relevant attacks are front-running, back-running, and sandwich attacks.

Front-running attacks are transaction manipulation attacks where the attacker aims to have their transaction executed before a targeted transaction [5]. This type of attack is done by making the gas price of the malicious transaction higher than the gas price of the targeted transaction. This attack can be used to render a user's transaction useless, such as front-running a user's bug report transaction. It can also be used to cause more slippage to the targeted transaction and thus reduce the user's profit [5].

On the other hand, **back-running** are transaction manipulation attacks where the attacker's transaction is executed after the targeted transaction [5]. This attack is done by assuring the gas price of the malicious attack has a lower gas price than the targeted transaction [7]. This attack aims to benefit off of the slippage caused by the targeted transaction. Unlike front-running, back-running requires several low-gas transactions to exploit the transaction, making this attack detrimental to the network and not the user [7].

Finally, **sandwich** attacks are transaction manipulation attacks which combine front-running and back-running to effectively "sandwich" the targeted transaction [5]. The goal of this attack is to exploit the targeted transaction by causing unfavourable currency fluctuation before the transaction and returning the currency to its original position after it [5]. This type of attack is the most frequent on DeFi Systems, with 180,185 measured attacks of this nature on UniSwap V2 by Torres et al. in less than a year since its launch [8]. As seen in Figure 3, UniSwap V1 and V2 suffer from the most transaction manipulation attacks out of all DeFi systems used in the study, as measured by the 6 heuristics [8]. Due to this

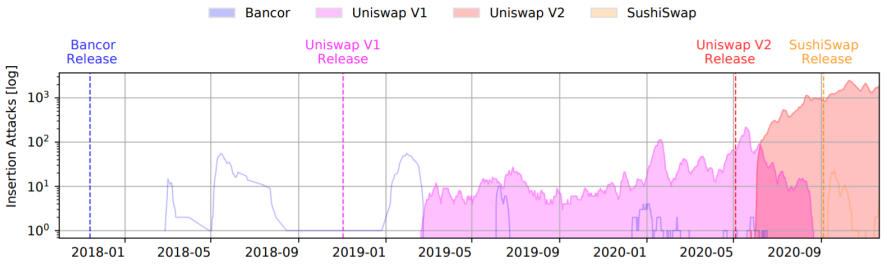

Figure 3: Number of measured insertion attacks in different DeFi systems [8]

frequency of attacks, the mitigation of transaction manipulation is vital to keeping UniSwap a safe and secure DeFi System [8].

3 Transaction Manipulation Attacks Analysis

The aim of this section is to analyze the aspects that contribute to transaction manipulation attacks as well as how these attacks happen in UniSwap. The analysis is the core of this research, as understanding the major factors that play a role in transaction manipulation attacks allows for different types of mitigations to be discussed.

3.1 Conditions that Enable Transaction Manipulation

Because of a lack of a central governing authority with the ability to punish for transaction manipulation or enforce any type of regulations, the current DeFi System and blockchain environment is the perfect breeding ground for these types of attacks [6]. To be able to devise a solution, the first step is to analyse the aspects of UniSwap and the underlying Ethereum blockchain, which enable transaction manipulation. The first aspect identified is slippage, the second is the miners' ability to order transactions, and the final is the lack of privacy on the blockchain with regards to transaction information.

Slippage In the world of trading, both centralized and decentralized, slippage is defined as the "difference between the expected price of a trade and the price at which the trade is executed" [21]. Slippage in and of itself is a neutral phenomenon and is a naturally occurring effect of any trade market. Any volatile currency experiences slippage extensively, because, as transactions are executed, the value of the cryptocurrency fluctuates. There is no way to predict the slippage that a transaction will incur, seeing as the slippage depends on the order of the transaction within a block. However, even though slippage and currency fluctuations are not inherently harmful, transaction manipulation attacks bring profit to the attacker by exploiting these occurrences. The goal in these attacks is to order transactions in such a way that the currency fluctuation caused by the targeted transaction provides positive slippage for the adversary.

Slippage cannot be solved or removed from any market completely, as there is no way to get rid of fluctuations [7]. Currency fluctuations are not only normal, but expected and encouraged in any trading market seeing as that is how any trader earns revenue. However, both slippage and fluctuation can be minimized through careful transaction ordering. In an

ideal scenario, all transactions would be ordered in such a way to minimize slippage. Though the ordering of transactions in and of itself is not a difficult task to fulfill with a sorting algorithm, the enforcement of the algorithm over the Ethereum blockchain is. Moreover, this type of mitigation would provide additional overhead with $n * n$ complexity, where n is the amount of transactions in the block being sorted.

Miners in Validation Protocols Currently, Ethereum functions on a consensus algorithm known as Proof-of-Work, which encapsulates how Ethereum miners find transactions and how they are added to the blockchain [14]. The current system in place hashes all transactions with a publicly known function and it is a miner's job to find the hash of said transaction to be able to put it on the blockchain. However, miners look to make a revenue out of this computationally extensive job and to that extent they have the power, and right, to order transactions within a block for maximum profit. This ordering is usually based on either increasing or decreasing gas fees according to Torres et al. [22], with up to 79% of miners following this strategy [9]. Because of the miners' predictable behavior, attackers can adjust the transaction price to be executed before or after the targeted transaction [9]. Miners themselves suffer no loss if a transaction manipulation attack occurs within one of their blocks, meaning there is little incentive for them to change their strategy. In fact, they themselves can profit off of other transaction manipulation attacks, which makes them less likely to change their behaviour [8].

Unlike slippage, this behavior from miners can be remedied by disallowing miners to order transactions in their benefit. The Byzantine Consensus is a solution in which the nodes act as "bookkeepers" and communicate with each other the order in which transactions are received and enforces an order-fairness consensus [23]. This mitigation would be implemented at the blockchain level and would protect against transaction manipulation at the DeFi level, as well seeing as miners would not be ordering transactions in an exploitable manner [23]. However, this implementation is complex and costly, seeing as nodes would have to keep records and communicate with each other to verify them [23].

Lack of Privacy By design, Ethereum has been made to favour transparency over privacy [15]. The practical implication of this decision is an undesirable one in the context of transaction manipulation attacks; anyone can view the transactions - and their details - that are waiting to be added to the blockchain [15]. This is in fact the origin of the issue that is transaction manipulation, seeing as all transaction manipulation attacks rely on being able to gain information from the transparency of the mempool [6]. The attacker can simply wait for exploitable transactions to come into the pool, read the data, and adjust their attack accordingly.

Seeing as this is the root of the problem, it is sensible to have a mitigation here. However, to solve this problem Ethereum as a protocol needs to be overhauled. Namely, the mitigation would be encryption of the mempool, meaning that even if someone were to observe the pending transactions they would have nothing to gain, seeing as all transactions would be encrypted. Moreover, the Gas price could be left visible to allow miners to order in a profitable manner to them, which would not be a cause for concern, seeing as attackers would have no way of knowing which transaction causes what amount of slippage due to the encryption of data.

From the three different aspects of UniSwap and Ethereum that enable transaction manipulation attacks, the lack of transactions encryption can be identified as the most severe. If a mitigation were to be implemented to reduce slippage or to deter miners from order-

ing predictably, the result would at most complicate transaction ordering enough to make most attackers step back. However, if encryption over the mempool were implemented, then transaction manipulation would not be feasible as a concept no matter how hard someone tried to front-run or sandwich attack.

3.2 UniSwap and its Predisposition to Transaction Manipulation

In Constant Function Market Makers, slippage is an even bigger concern when dealing with liquidity pools since any change in x or y in the constant function changes the value of the asset being traded [8]. Since UniSwap precisely uses a CFMM, attention should be given to transaction manipulation attacks.

UniSwap's 2020 audit can be used as a starting point to observe how these attacks happen in the protocol [4]. Because the terminology for transaction manipulation attacks is not universal, UniSwap defines the two attacks experienced in their protocol as "Moving the market against the trader" and "Sandwiching large trades with mint and burn", both of which fall under the sandwich attack terminology discussed in the Background section.

Moving the Market against the Trader The first version of a transaction manipulation attack in UniSwap follows the strategy of the sandwich attack. A malicious user observes a pending transaction that they consider profitable enough to attack and the transaction is sandwiched in their favour [4]. In UniSwap, the smart contract being attacked is a Pair (pool) meaning that the user transaction swaps token A for token B and the attacker's transactions buy and sell one of the tokens to cause slippage in a certain direction [4]. Because UniSwap imposes a fee that must be paid by every Liquidity Provider, this attack usually targets sums large enough for the fee to be worth the attack. Moreover, the attacker has control over the slippage caused by both the sell and buy transactions, which in turn means they have a say in how much profit the attack generates [4].

This type of attack targets and impacts the user and not the system directly. In practice, what this attack results in is a trade between the user and the attacker, where the attacker's buy and sell transactions and the user's transaction dictate how much is traded between the two actors. What is currently in motion to protect against this attack is an off-chain slippage limit which allows users to individually specify the maximum slippage they would accept the trade at. This mitigation is briefly discussed in the following section.

Sandwiching Large Trades with Mint and Burn The second type of transaction manipulation attack encountered in UniSwap is in the sandwich classification as well, but with a different approach. In the first version, the attacker bought and sold one of the tokens in the pool, whilst, in this version, the attacker mints and burns one of the tokens [4]. Mint, in UniSwap, is the creation of currency, while Burn is the deletion of said currency, meaning this attack creates and destroys currency in the process of exploiting the targeted transaction [17].

This attack, although similar to the first, has some impact on the system. When an attack manipulates existing currency, though damaging to the user, does no tangible harm to the pool because the attack is indistinguishable from regular transactions in its end result. However, when minting and burning is involved, the attacker can earn a large portion of the Liquidity Provider fees, which is damaging to the pool as a whole [4]. Currently there is no solution except a proposed time lock [4]. This solution will be further analysed in the Mitigation section.

4 Mitigations in UniSwap

As discussed in the previous section, most solutions to transaction manipulation attacks involve Ethereum rather than UniSwap. However, there are mitigations UniSwap can implement, namely a commit-reveal scheme and a time lock scheme. This section explains and validates these solutions.

4.1 Commit-Reveal Scheme in Uniswap

This mitigation is a commit-reveal scheme, a cryptographic algorithm used to obscure the true value of a variable. This mitigation is an adjustment of LibSubmarine's commitment scheme which was originally designed to prevent bug bounty withholding [3].

Conceptual Model of Submarine Commits As described in the documentation for the Hydra Framework, the proposed Submarine Commit scheme consists of three transactions, defined as the commitment, reveal, and retrieval transactions [3]. Conceptually, the scheme consists of two phases, the commit and reveal, which is how they are presented in this section, due to the mitigation itself being conceptual [3].

The Commitment Phase: In the commit phase, the Submarine Commit hides the recipient address by hashing it using the true value of the recipient address, the sender address, the opcode of the transaction being performed, and a user-selected 256-bit value called a key [3]. This hashed value appears as a novel address in the blockchain, rather than a known contract address, thus deterring front-runners from attacking this transaction [3]. It should be noted that there is a certain price, called deposit, needed to execute the scheme, which is sent along with the commit transaction. This transaction can contain any other data needed for the smart contract it is working with.

The Reveal Phase: Once the commit transaction has been added to the blockchain, the 256-bit key and block number in which the commit transaction was chained in are sent to the actual recipient address in the form of a second transaction. This reveal transaction is sent after a certain number of blocks, to ensure that an attacker cannot see the reveal transaction's information before the commit has been added to the blockchain. To confirm that the reveal transaction matches the commit transaction, a cryptographic proof known as Merkle-Patricia is used [3]. After the reveal, the smart contract retrieves the deposit and any other data, such as the sum being transferred, located at the hashed address and the scheme is completed successfully.

Implementation in UniSwap This model can be implemented in UniSwap as well, with a few adjustments. Namely, this address obscuring would essentially hide which liquidity pool a user is interacting with. Though the sender address would be still visible, due to the hiding of the recipient address, the attacker would not be able to determine what the user is aiming to do with the funds involved in the transaction. If the attacker cannot discern where to buy or sell tokens to exploit a given transaction, then they are not able to exploit said transaction [3].

The important concepts for this model, as defined by LibSubmarine and new ones needed for UniSwap are as follows:

Definitions of Variables
P - Person submitting exchange transaction on UniSwap
T - User Transaction
A - Attacker wanting to exploit T
S - Smart Contract
pr - Price to pay for commit-reveal scheme
$deposit$ - Cost to execute a Submarine Commit
$recAddress$ - Recipient address
$sendAddress$ - Sender's address
$extraData$ - Data needed by UniSwap
$pQueue$ - Priority Queue
r - Amount of blocks A can delay for
q - Amount of blocks A can rewind for
Δ - Number bigger than $r + q$
n - Number of blocks during which attack can happen
n' - $n + \Delta$
$commBlock$ - Block in which commit is sent

As far as the commit stage goes, the conceptual model from the Submarine Commits scheme would remain largely unchanged in terms of how the address is hidden. The core change in UniSwap would be adjusting the pools to have the ability to validate a reveal message and retrieve the data from the submarine address. Namely, a verify() function should be added to verify that the reveal matches the commit, and a onReveal() function should be added to gather the $deposit$, actual sum being transferred, and $extraData$.

Two implementations are explored with regards to how this scheme can be implemented. The first proposes an easier implementation where each user crafts their own commit transaction and sends their own reveal transaction. The second implementation, though more challenging to implement, involves a priority queue on UniSwap's side that would take care of the reveal transaction.

Simple Implementation: P creates transaction T, where T is a transaction tailored for the UniSwap protocol. T is ordered through a smart contract, S, which interacts with the Router smart contract implemented by UniSwap. S can be created as a template by UniSwap which users can fork out of or they can craft their own smart contract S. T has an encrypted $recAddress$, as described in LibSubmarine, where the original address is obtained through the Router. From here, S keeps the information regarding the key needed to unlock the commit, the block number, and which block to send the reveal in. Once the reveal is safe to send, S broadcasts the key and block number to the Router which forwards to the actual Pair. This scenario implements the pure LibSubmarine framework in the sense that every user would have to individually take care of the commit-reveal through their own smart contract S. The benefit of this is added privacy as only the user would know when the reveal is scheduled, however it is an extra burden onto the user.

Complex Implementation: A harder to execute version of the Simple Implementation would be a UniSwap implemented priority queue, $pQueue$, which would have to be off-chain and inaccessible to attackers. This $pQueue$ would appear instead of the smart contract S, to keep the sensitive data, namely key and block number. In effect, $pQueue$ would automate

the reveal process instead of leaving the burden on the user, meaning the user's interaction with the pools would remain unchanged from the current implementation. Each time a user submits a transaction through the Router, it would be "wrapped" in a commit, and UniSwap would add an element to *pQueue* with the proper key and blocks left until reveal. The elements of *pQueue* would be ascending, from least to most blocks left until reveal. The difficulty here would be locking the sensitive data so that an attacker would not be privy to the information. This implementation would also require an event listener to keep track of the latest block added to the blockchain and would add a time and space complexity of m, where m is the number of elements in *pQueue*. The m space complexity would be added because every element in *pQueue* needs to be stored, and the time complexity is m every time a new block is added to the blockchain seeing as all counters need to be updated. The benefit of this implementation is an easier experience for the user seeing as UniSwap would automate the reveal process, however anyone with access to the *pQueue* could cheat this system.

Validation As shown in the Hydra Framework, the proof that this approach works against bug withholding is a game-based approach [3]. This proof can be modified to show that a Submarine Commits approach works against transaction manipulation attacks on DeFi Systems as well. The game in question is known as F-withhold, where two different players, the honest user and the attacker, can only carry out two different moves, commit and reveal, with a delay between the two. An advantage of the attacker is the power to rewind by q blocks, as to insert the attack, and the power to delay a transaction by r blocks [3]. The game is played in n out of n' blocks, where n is defined as a section of n' such that $n = n' - \Delta$, and $\Delta > q + r$. This is set up in such a way to show that even under optimal conditions the attacker cannot win when front-running [3]. Finally, B_1 to B_n denote each block in the range of interest.

The set-up so far is the same as in the Hydra Framework, however the difference is in how A becomes the winner of the game. Firstly, the honest player P chooses uniformly randomly in which block, [1, n], to post the commit, with the reveal being posted at $commBlock + q$. A can only win if it successfully posts two commits that sandwich P's commit, with the reveals coming in later blocks. This is because A needs to cause and repair the slippage. Because the commit message encrypts the true value of $recAddress$, A has no way of knowing that this transaction is of interest to them seeing as the address is not a known UniSwap pool address. Thus the only way for A to definitely win is if the very first commit that sets up for slippage is in B_1 and the second commit is done in B_n. However, though A wins in the game, A has no profit here, because this is not a type of transaction manipulation, rather regular trading. Simply, no transaction has been attacked.

$$Pr = A_(win) * sl - 2 * am$$

As seen in the profit calculation formula, modified to fit the current game and not the bug bounty calculation by LibSubmarine [3], A only profits off of sl, the slippage they cause and restore, if and only if they win, $A_(win) = 1$. Otherwise, they just pay for the two commit transactions that cause sl. It should be noted that unlike *bounty* in the original calculation, sl is not known until all transactions have been executed, however, its maximum is if A successfully performs a transaction manipulation attack.

As for the rewind and delay parameters discussed before, they are used to show that if A learns of a commit at a later time it is of no benefit to them. Specifically, if A sees a reveal transaction in some block, B_j, then A could infer that the commit was in $B_j - q$ as defined

previously. Even if A rewinds by q, its maximum ability, B_j is the last block that was added to the blockchain, and thus unchangeable, and A's commit would go to block $B_j + 1$. This means that unless A had previously committed their slippage-causing transaction in $[B_1, B_(j-q)]$, even this rewinding power is not enough to front-run. This is proof that A, simply by knowing that a reveal is in motion, cannot front-run the commit message and attack P's transaction. Moreover, back-running is only possible starting from block $B_(j-q+1)$ with this forward rewind, however back-running in different blocks is also unprofitable because the slippage P's transaction cause is likely already arbitraged.

The Submarine Commits document has a theorem in place with regards to which numbers should take the place of Δ and $\$deposit$.

Theorem 3. *Let* $\Delta \geq 4$ *and* $\$deposit > \frac{10(\Delta+1)}{9n} \cdot \$bounty$. *Then a pure front-running adversary has* $\mathbb{E}[\$payoff] < 0$.

Figure 4: Hydra Framework, Theorem 3 on values for $\$\Delta$ and $\$bounty$, [3]

The Hydra Framework contains proof for this theorem, however it is outside of the scope of this paper to prove the theorem again. The takeaway for applying this solution in UniSwap is to have a large enough Δ and large enough *deposit* so that A has no payoff. With Δ being 4 or bigger, this means the delay between the commit and reveal transactions should be at least 4 blocks or larger, and *deposit* should be larger than $\frac{50}{9*n} * sl$, where n would be picked with respect to the average blocks per hour added to the Ethereum blockchain. The final variable of the Theorem that should be modified to fit a UniSwap implementation is the *bounty*. For DeFi Systems, bounties do not exist in regular transactions, such as swapping or minting and burning. To that extent, slippage should be used as the variable in the theorem, seeing as, in the context of transaction manipulation attacks, slippage is how an attacker profits [5]. Because the slippage of a transaction is not known until said transaction is executed, a simple implementation would be looking at the current reserves in the UniSwap pool the transaction is being executed in and calculating the slippage between the balance and the transaction. With this estimated slippage, the *deposit* can be calculated as to fulfil the conditions in Theorem 3 and successfully execute a commit-reveal transaction which is protected from transaction manipulation.

4.2 Time Lock for Sandwich with Mint and Burn

The Commit-Reveal strategy obscures transactions in such a way that both attacks in UniSwap are protected against. Nevertheless, due to drawbacks discussed in the following section, it might not be worthwhile for UniSwap. To that extent, a time lock strategy can be used to protect only against the UniSwap mint and burn attack.

Definitions of Variables
T - User Transaction
A - Attacker wanting to exploit T
Z - Sum minted
L - list of wallet address and time until allowed action
X - time in blocks
$currBlock$ - current block
$allowBlock$ - block after which certain restricted transactions are allowed

The adjustment to the current system would include a time lock for any liquidity provider which would forbid them from burning or otherwise removing and swapping liquidity for a certain amount of time, denoted as X. This would be implemented in every single Pair as a dynamically allocated list, L, where the address of the user's wallet would be associated with a block number to when they can next burn.

Implementation: Attacker A wishes to mint/burn a targeted transaction T. Upon minting some sum, Z, a mathematical formula is applied which, based on Z, calculates when the user can burn next. This function would result in a block number at which the user can next perform a transaction, by assuming that a block will be added per 14 seconds on Ethereum as is the current trend. A naive formula could look as follows:

$$currBlock + (Z) * (1/14) + 10 = allowBlock$$

A note on the formula: there is an addition of 10 to make sure that in the case of Z being an amount that begins with 0 there is at least a delay of 10 blocks. After the calculation, the address of A and $allowBlock$ would be added to L in an ascending order. In fact, whatever the formula used, the key to this implementation is having a delay large enough that the attacker is exposed to the risk of being a Liquidity Provider for some period, seeing as that is what would discourage this type of attacks [4]. As long as the delay is long enough to repel attackers yet short enough to still allow for regular traffic on the Pair, this solution could solve this specific version of a sandwich attack in UniSwap.

Validation To affirm this solution works for the mint/burn attack presented in the previous section, a simple validation should be discussed. Since this mitigation was not implemented in code, a conceptual validation will be used as proof that this proposition works.

If A attempts to mint/burn sandwich attack a transaction, then by having a time lock, the minimum time the attacker would have to wait to burn, swap, or sell their liquidity would be determined by the formula. This means that unless the attacker wants to add liquidity, or mint more tokens, they are forbidden from interacting with the targeted pool. Because they can't move liquidity to another pool, they are also blocked from performing the burn at another address. Since the profit of this attack comes from the Liquidity Provider fees whilst avoiding the risks of being a liquidity provider, by elongating the time that the minted tokens have to remain in the pool, the attack becomes unattractive [4].

Formally, if the game-based validation for the Commit-Reveal scheme were implemented here, then A places their mint/burn transactions in block B_1 and B_X per the formula, and the only way they would win is if the only transaction taking place between these two is the targeted transaction T. However, with a lock of length X long enough, and X differing based on Z, this is unlikely.

By disallowing any user who mints to burn, swap, and trade for an appropriate amount of time this mint/burn sandwich attack is nearly impossible to be carried out by attackers. Thus, this mitigation would successfully protect the system against this variant of the attack at the cost of limited transaction from liquidity providers.

4.3 Off-chain Slippage Limit

This mitigation is already implemented in UniSwap, however it should be briefly mentioned in the section. The premise of this solution is that any user can define the most amount of slippage they would allow their transaction to be executed at [4]. However, whilst this

solution seems like a good mitigation in theory, according to Torres et al. it does very little to actually protect the users [8]. This is mainly because it is a very difficult task to determine what the limit should be, seeing as too tight of a limit means the transaction will most likely not go through, whilst a too loose limit is vulnerable to transaction manipulation attacks.

5 Responsible Research

This paper examines a potential mitigation to front-running in the UniSwap environment. Though everything is theoretical and no experiment took place, there is still a need for an ethical research discussion.

The methodology through which the conclusions and information was gathered is a literature study mainly consisting of peer-reviewed articles as well as blogs and whitepapers by UniSwap and Ethereum. The sources, at the time of writing the paper, have been deemed trustworthy by the merit of other papers referencing them.

As for the possible mitigation offered in the paper, a commit-reveal strategy examined at the top-most level, it is an idea rather than experiment. This means that reproducibility is an inapplicable concept. Moreover, this paper was written with the aim of not only exploring a certain solution, but also exploring its effects on UniSwap's ecosystem. With that in mind, integrity was one of the leading values to analysing the possible solution, seeing as ignoring the downsides of a possible solution would render the paper incomplete and even useless. It should be noted though that intent itself is not always enough, and the proof of integrity should be observed in the criticism and analysis presented when talking about the Transaction Ordering Attacks and possible solutions.

6 Discussion of Findings

This paper's aim is to discover how UniSwap can protect its users and system from transaction manipulation attacks. However, in analysing the variables that contribute to transaction manipulation attacks as well as how these attacks play out, it was concluded that the root of the problem lies in Ethereum as a protocol. Though the two theoretical mitigations help to lessen the issue, the only fool-proof way to accomplish this would be the encryption of the transactions [6]. Unfortunately, this would be a costly endeavour as the whole blockchain would have to be reconstructed. Though costly, this is possible and Secret Network is a proof that this can be made in practice [24].

Because solving the problem at the root is not something UniSwap can undertake, two different methods of mitigation at the DeFi system level were analysed in the previous section. However, both of these methods have drawbacks that make them unappealing in practice. Firstly, the Commit-Reveal strategy as proposed by LibSubmarine includes too many moving parts, with the worst offender being the need for 3 transactions per commit-reveal. This means that if there are n transactions per day currently being executed in UniSwap that number would jump to $3*n$. This has the potential to congest the network and make it slower for users. Moreover, users would have to pay a deposit for the commit-reveal scheme which could deter users from using UniSwap. Finally, the actual implementation of the commit-reveal, if done with a $pQueue$, would be complex and would add a centralized element in the protocol, seeing as UniSwap would regulate when the reveal transaction is ready to be sent out.

As for the time lock scheme mitigation, it carries its own issues as well. Firstly, it only solves the issue with mint/burn sandwich attack, which, for the amount of work needed to practically implement this solution, is a small benefit. Moreover, this type of mitigation limits the amount of transactions a user can do on a certain pool if they decide to mint, which could be a reason for users to change DeFi Systems. Finally, this solution introduces a level of centralization as well, seeing as UniSwap would act as the body that dictates who can do what when.

Both of the solutions have their drawbacks and advantages, though realistically neither is the perfect solution. The truth is both have a level of complexity that trumps the benefit. As previously discussed, transaction manipulation is an attack that impacts the user individually rather than the system and because it is a common issues in any DeFi System, UniSwap has little incentive to focus resources on mitigation.

7 Conclusions and Future Work

The goal of this paper was to analyse what UniSwap can do to protect its users from a transaction manipulation attack as well as what the consequences of a given mitigation mean for the ecosystem. To this extent, two possible mitigations were discussed at length that could be implemented by UniSwap and their drawbacks. A commit-reveal scheme augmentation based on LibSubmarine's bug withholding solution is the first of the discussed solutions, with an increase of transactions as its main drawback. The second solution examined is a time-lock based approach, which has the opposite drawback in that it would minimise a user's ability to perform transactions.

In the process of the research and problem analysis, the three main factors that contribute to transaction manipulation were explored, namely slippage, miners in the blockchain, and lack of privacy with regards to transaction details. The deeper conclusion of this paper is in fact that the only definitive way to prevent transaction manipulation is to obscure transaction details. However, this is a much more complex solution that would need its own paper and different options on how to obscure the transaction details.

The ultimate conclusion of this paper is transaction manipulation attacks are difficult to protect against on the DeFi System level, since the problem is in the Ethereum protocol itself. Though mitigations exist, their cost-versus-benefit ratio is not appealing to UniSwap.

References

[1] N. Calabrese, *The growing popularity of blockchain technology*, 2021. [Online]. Available: `https://research.g2.com/insights/growing-popularity-of-blockchain-technology`.

[2] R. Sharma, *Decentralized finance (defi) definition*, 2021. [Online]. Available: `https://www.investopedia.com/decentralized-finance-defi-5113835`.

[3] L. Breidenbach, P. Daian, F. TramÃšr, and A. Juels, *Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts*, Cryptology ePrint Archive, Report 2017/1090, `https://eprint.iacr.org/2017/1090`, 2017.

[4] D. Currin, D. Terry, D. Erfurt, L. Livnev, L. Manacorda, and M. Lundfall, *Uniswap v2 audit report*, 2020. [Online]. Available: `https://uniswap.org/audit.html#orga1aca00`.

[5] S. M. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. J. Knottenbelt, *Sok: Decentralized finance (defi)*, 2021. arXiv: 2101.08778.

[6] S. Eskandari, S. Moosavi, and J. Clark, *Sok: Transparent dishonesty: Front-running attacks on blockchain*, 2019. arXiv: 1902.05164.

[7] J. Xu, N. Vavryk, K. Paruch, and S. Cousaert, *Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols*, 2021. arXiv: 2103.12732.

[8] C. F. Torres, R. Camino, and R. State, *Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain*, 2021. arXiv: 2102.03347.

[9] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, *High-frequency trading on decentralized on-chain exchanges*, 2020. arXiv: 2009.14021.

[10] Coindesk, 2017. [Online]. Available: https://www.coindesk.com/learn/ethereum-101/what-is-ethereum.

[11] D. L. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms*, New York, NY, USA, 1981. DOI: 10.1145/358549.358563. [Online]. Available: https://doi.org/10.1145/358549.358563.

[12] 1999. [Online]. Available: https://us.napster.com/.

[13] [Online]. Available: https://defipulse.com/.

[14] *Ethereum*, 2015. [Online]. Available: https://ethereum.org/en/.

[15] V. Buterin, *Ethereum whitepaper*, 2013. [Online]. Available: https://ethereum.org/en/whitepaper/.

[16] *Top cryptocurrency decentralized exchanges*, 2021. [Online]. Available: https://coinmarketcap.com/rankings/exchanges/dex/.

[17] *Uniswap documentation v2*, 2020. [Online]. Available: https://uniswap.org/docs/v2/.

[18] G. Angeris, H.-T. Kao, R. Chiang, C. Noyes, and T. Chitra, *An analysis of uniswap markets*, 2021. arXiv: 1911.03380.

[19] *Uniswap documentation v3*, 2021. [Online]. Available: https://docs.uniswap.org/.

[20] C. Mitchell, *Front-running*, 2020. [Online]. Available: https://www.investopedia.com/terms/f/frontrunning.asp.

[21] A. Hayes, 2021. [Online]. Available: https://www.investopedia.com/terms/s/slippage.asp.

[22] C. F. Torres, A. K. Iannillo, A. Gervais, and R. State, *The eye of horus: Spotting and analyzing attacks on ethereum smart contracts*, 2021. arXiv: 2101.06204.

[23] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, *Order-fairness for byzantine consensus*, Cryptology ePrint Archive, Report 2020/269, https://eprint.iacr.org/2020/269, 2020.

[24] *Secret network*. [Online]. Available: https://scrt.network/.