n-MVTL Attack: Optimal Transaction Reordering Attack on DeFi

Jianhuan Wang[1], Jichen Li[2], Zecheng Li[1], Xiaotie Deng[2], and Bin Xiao[1]

[1] The Hong Kong Polytechnic University, Hong Kong.
[2] Peking University, China.

Abstract. Decentralized finance (DeFi) is a global and open financial system built on the blockchain technology, typically using Ethereum smart contracts. Decentralized exchanges (DEXs) are very important sectors in the DeFi ecosystem, with billions of USD trading volume daily. Unfortunately, the transparency of pending pools can be exploited by attackers and DEXs are vulnerable to *transaction reordering attacks*, allowing attackers to gain miner extracted value (MEV). Previous transaction reordering attacks aim at exploiting the vulnerability of a single victim transaction, such as sandwich attack and dagwood sandwich attack.
In this paper, we propose a novel transaction reordering attack named n-multiple-victim-transaction-layer (n-MVTL) attack to exploit the overall vulnerability among multiple victim transactions. Such advanced design can significantly expand the victim transaction search space and bring more profits to attackers. Given a set of ordered victim transactions, we propose an optimal algorithm to identify the optimal solution for n-MVTL attacks, which aims to maximize the profit of the attack strategy. This algorithm supports a trade-off between time efficiency and attack profit, making the attack algorithm more practical. Our simulations show that the n-MVTL attack can yield an average extra daily profit of 940 USD from the top 2 most popular liquidity pools in Uniswap V2 from Mar. 2021 to Apr. 2023, compared with the sandwich attack.

Keywords: Decentralized Finance (DeFi) · Miner Extractable Value (MEV) · Decentralized Exchange (DEX) · DeFi Attack · Blockchain

1 Introduction

In recent years, decentralized finance (DeFi) as a supplement to traditional finance has become an enormous ecosystem with a total locked value of 47 billion USD in May 2023 [5]. Within the DeFi ecosystem, *Automated Market Makers (AMMs)* play a crucial role by providing real-time asset pricing for user transactions in DeFi. The AMMs-based exchange platforms (e.g., Uniswap [9] and Pancakeswap [8]) handle swap transactions with a total volume of several billions of USD per day [4].

However, the feature of Ethereum prioritizing transaction ordering based on gas fees rather than time sequence makes AMMs susceptible to *transaction reordering attacks*. These attacks are defined as the manipulation of transaction order within blocks by miners, with the aim of extracting *miner/maximum extracted value (MEV)* [3]. One of the most common transaction reordering attacks is *sandwich attack*, which was formalized and quantitatively analyzed for attack

profitability by Zhou *et al.* [13]. As shown in Fig. 1(a), the sandwich attack strategy involves the execution of a malicious front-running transaction and a malicious back-running transaction aimed at a victim transaction. Then, the attacker profits from the discrepancy between the execution prices of the front and back-running transactions. One extension work of the sandwich attacks is the dagwood sandwich attack [1] which targets multiple victim transactions simultaneously by utilizing front-running attacks on each victim transaction separately as shown in Fig. 1(b). The sandwich attack has a single-victim-transaction layer and the dagwood sandwich attack has several single-victim-transaction layers. We refer to these attacks as n-single-victim-transaction-layer (n-SVTL) attacks.

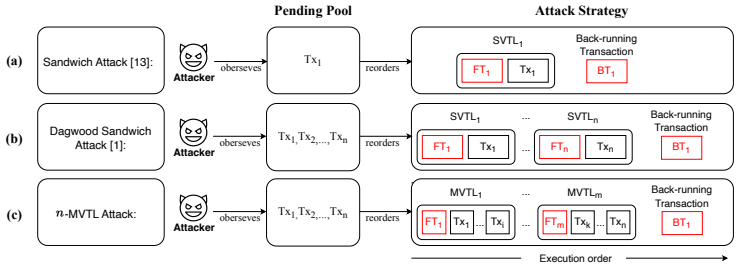

Fig. 1: Visualization of n-SVTL attacks and n-MVTL attack.

Previous studies in the field of defense mechanisms of transaction reordering attacks focus on limiting transaction parameters [13, 7, 12, 14]. Due to the simple structure of n-SVTL attack, the attack profit gain from each layer is strongly influenced by the trading amount and tolerated price slippage of the victim transaction. DeFi users can prevent n-SVTL efficiently by two defenses (i.e., limited trading volume [13, 12, 14] and limited slippage [13, 7]). We refer to transactions that utilize these two defenses as *un-sandwichable transactions*. In light of the aforementioned advancements in defense mechanisms against n-SVTL attacks, an important question arises:

> *Do the existing defenses offer adequate security for transactions of DeFi users? Are there other kinds of transaction reordering attacks that can bypass these defenses?*

To address this question, we propose the utilization of a more flexible structure termed n-MVTL (details provided in Section 4). Unlike the approach of employing n front-running transactions to attack n victim transactions in n-SVTL attack, we split the n victim transactions into m ($m \leq n$) different layers, with only one front-running transaction used to attack the victim transactions in each layer, as illustrated in Fig. 1(c). Note that the n-SVTL structure is a typical sub-class of the n-MVTL structure. By adopting the n-MVTL structure instead of the n-SVTL structure, we can explore additional potential attack strategies. Furthermore, we observe that the transaction reordering attack employing the n-MVTL structure, hereafter referred to as n-MVTL attack, can detect the overall vulnerability among multiple un-sandwichable transactions, enabling attacks on un-sandwichable transactions. To comprehensively evaluate the severity of the n-MVTL attack, we provide a formalized description of this novel attack and

quantify its associated risks. To the best of our knowledge, we are the first to explore the feasibility and profitability of transaction reordering attack that exploit the overall risk of multiple transactions. The contributions of our research can be summarized as follows:

- **Novel Transaction Reordering Attack.** We propose the n-MVTL attack, a novel transaction reordering attack to compromise the traditional defense mechanisms in DeFi. Compared to the dagwood sandwich attack, the n-MVTL attack can exploit multiple swap transactions simultaneously in a general market environment that has no assumption about the real price of tokens. In addition, our algorithm considers an important cost factor, *AMM swap fees*, which is not included in most existing attacks.
- **Optimal Algorithms for Attack.** We propose a **Transaction Selecting** algorithm, which aims to identify the largest subset of transactions that can be attacked based on the current state of the liquidity pool and the list of vulnerable transactions in the pending pool. Furthermore, for these selected transactions, we propose an optimal algorithm to find an optimal strategy to attack them and to maximize the attacker's profit. Details as discussed in Section 4.
- **Implementation.** We implement a prototype of our proposed n-MVTL attack to discover the optimal attack strategies. We also test the time efficiency of our attack algorithms in Section 6. Experimental results show that our algorithms are efficient and practical for generating attack strategies against constant product market markers (CPMMs) even with a personal computer (e.g., Macbook Pro).
- **Validation of Attacks on Historical Transactions.** We validate the n-MVTL attack strategies on a simulation system that implements the swap formula of Uniswap V2. We find that n-MVTL attack yields an extra profit of 656,972 USD compared to the sandwich attack from block height 12,000,000 to 17,000,000 in Ethereum. We demonstrate that n-MVTL attack can spot more attack opportunities than sandwich attacks.

1.1 Paper Organization

The remainder of this paper is organized as follows. Section 2 reviews related literature. We describe how to encode AMM protocols into state transition models in 3. We propose n-MVTL attack in Section 4. We conduct a comprehensive analysis of the revenue of the optimal attack algorithm in Section 5. We evaluate our attack algorithms and validate our attack strategies in Section 6. Section 7 concludes our paper.

2 Background and Related Work

In this section, we would like to introduce the necessary background and discuss closely related works on transaction reordering attacks.

2.1 Reordering Transactions

There are three types of reordering transactions that are used to launch transaction reordering attacks: (I) **Front-running transaction (FT)**: If a transaction

runs before a victim transaction and its parameters are carefully set by an attacker to prevent the victim transaction from failing to swap the tokens, we classify this transaction as an FT. This type of malicious transaction is required in many transaction reordering attacks (e.g., sandwich attack [13]); (II) **Fatal front-running transaction (FFT)**: If a transaction runs before a victim transaction and its parameters are maliciously set by the attacker to cause the failure of the victim transaction's execution, we classify this transaction as an FFT. This type of malicious transaction results in more losses for DeFi users as they must pay gas fees even if their transactions fail to swap the tokens; (III) **Back-running transaction (BT)**: If a transaction runs after a victim transaction, we classify this transaction as a BT. It is frequently used in conjunction with other malicious activities (e.g., front-running in sandwich attack [13]).

2.2 Sandwich Attack

The most common transaction reordering attack is the sandwich attack, formalized by Zhou *et al.* [13]. Due to the transparency of DeFi, transaction information in the pending pool can be obtained by attackers. We consider a victim transaction TX_1, whose information is observed by an attacker. Then, the attacker emits a front-running transaction FT_1 and a back-running transaction BT_1 to launch a sandwich attack as shown in Fig. 1(a). These malicious transactions aim to trigger the largest price slippage of TX_1. The gas prices of malicious transactions are carefully set to ensure that the execution order will be: FT_1, TX_1, BT_1. Then, the attacker gains attack profit from the difference of execution prices of FT_1 and BT_1. Zhou *et al.* demonstrated that the sandwich attack yields a total profit of 174.34M USD from block 6,803,256 to 12,965,000 in Ethereum.

To further strengthen the attack capability, Bartoletti *et al.* [1] proposed a multi-layer dagwood sandwich attack which can attack several victim transactions simultaneously. The idea of this attack is to repetitively launch a front-running attack on each transaction, targeting every transaction individually. This approach, however, disregards the overall vulnerability among victim transactions, making it susceptible to resistance from conventional defense mechanisms [7, 14, 11]. Furthermore, the dagwood sandwich attack relies on two unrealistic assumptions: (I) *Stable price assumption.* The author employs the stable price assumption for facilitating the calculation of attack profit. However, this assumption is hard to guarantee in the real market, attackers may incur losses due to fluctuations in real prices of tokens. (II) *No AMM swap fees.* The authors introduce the assumption that no AMM swap fees are charged by the liquidity pool. This assumption enables the use of a remarkably convenient property (i.e., the supply of the liquidity pool is always constant) to determine the maximum loss state for each victim transaction. However, almost all of the DEXs (e.g., Uniswap [9] and Pancakeswap [8]) charge AMM swap fees. In summary, the two assumptions of the dagwood sandwich attack make it difficult to apply it to the real DeFi market.

2.3 Defense Strategies

Many researchers have focused on defense strategies against sandwich attack [7, 14, 11]. Zhou *et al.* [13] proposed two primary protection possibilities that

could be adopted for DeFi users to prevent sandwich attacks: Limit Slippage and Limit Trading Volume.

Limit Slippage. A DeFi user can set her transaction slippage as small as possible. Thus, the attacker cannot benefit from a sandwich attack. To extend the defense method of limiting slippage, Heimbach *et al.* [7] introduced the sandwich game to analyze sandwich attack analytically and provided traders with a simple and extremely effective algorithm for setting the valid slippage tolerance. However, a transaction with small slippage is still likely to suffer from unexpected slippage. Meanwhile, our n-MVTL attack can break this kind of protection(cf. Example 2 in Appendix B).

Limit Trading Volume. A DeFi user can also set the trading amount of her transaction below a minimum profitable victim input, so it is unprofitable for attackers to launch a sandwich attack. To extend this defense method, Züst [14] proposed a valid mitigation strategy that splits a sandwichable transaction into several small transactions with limited trading volumes. Although DeFi users lose extra transaction fees when using this strategy, they can protect their transactions from sandwich attack (or dagwood sandwich attack). Unfortunately, this defense strategy also can not defend against our n-MVTL attack (cf. Example 1 in Appendix B).

3 Model

In this section, we formally describe a state transition model for AMM protocols. Then we describe the profit and strategy space of our n-MVTL attack.

3.1 AMM Model

AMM is a protocol that enables automated trading tokens for DeFi. Given a pair of tokens, an AMM will create a liquidity pool (LP), which state will change after executing transactions, and it can set prices automatically based on a specific rule. In this paper, we focus on the CPMM [10], which is the most widely adopted subclass of AMMs and utilizes the constant product pricing rule.

Liquidity Pool State. Given a liquidity pool LP with two types of tokens, τ_x and τ_y, the state $S = (X \in \mathbb{N}^+, Y \in \mathbb{N}^+)$ denotes the state of the LP, where X and Y denote the amount of τ_x and τ_y token in this LP respectively. The total liquidity of a LP can be derived as $K = X \cdot Y$.

Transactions. In this paper, we only focus on *swap* transactions. We use $TX^d : swap(ax \in \mathbb{N}_0^+, ay \in \mathbb{N}_0^+)$ to denote the collection of swap transactions, where $d \in \{x \to y, y \to x\}$ is the swap direction. For example, when $d = x \to y$, then this transaction means that the user wants to swap ax of token τ_x for at least ay of τ_y, and vice versa.

Constant Product Pricing Rule. Given a transaction $TX^{x \to y} : swap(ax, ay)$ and LP with a state (X_0, Y_0), the execution price p of this transaction can be calculated by Formula 1, where f indicates the AMM swap fee rate set by LP (e.g., 0.3% in Uniswap [9]). The end (post-execution) state of $\mathcal{LP}$ after executing $TX^{x \to y}$ can be derived as $(X_0 + \Delta x,\ Y_0 - \Delta y)$. When this transaction is

successfully executed, the user should pay $\Delta x \cdot f$ of token X as a transaction swap fee to LP, which will add some liquidity to the pool.

$$\Delta x = ax,\ \ \Delta y = \lfloor Y_0 - \frac{K_0}{X_0 + (1-f)\cdot \Delta x} \rfloor,\ \ p = \frac{\Delta x}{\Delta y} \tag{1}$$

In the following attack modeling, we assume that all victim transactions' swap directions are $x \rightarrow y$. To simplify notation, we write $TX(ax, ay)$ for $Tx^{x\rightarrow y}(ax, ay)$. In addition, we use $FT \subset TX^{x\rightarrow y}$ and $BT \subset TX^{y\rightarrow x}$ to denote the collections of malicious front-running transactions and back-running transactions, respectively. Since we assume the attackers have the power to manipulate the order of transactions, they need not care about the tolerated slippage of FT and BT. To simplify notation, we write $FT(ax)$ and $BT(ay)$ for $FT^{x\rightarrow y}(ax, 0)$ and $BT^{y\rightarrow x}(0, ay)$, respectively.

AMM Transition Functions. We define a swap transition function as $\mathcal{F}((X_i, Y_i), TX_{i+1}, f) \rightarrow S_{i+1}$, which outputs the next state S_{i+1} after executing TX_{i+1} on the state S_i, where f is the transaction fee rate set by the LP. We also define a swap amount calculating function based on Formula 1 as $\mathcal{F}_{\mathcal{A}}((X_i, Y_i), TX_{i+1}, f) \rightarrow y_{i+1}$, which output the amount of swapped token τ_y after executing transaction TX_{i+1}. We represent the state change of $\mathcal{LP}$ upon the execution of TX_{i+1} as $(X_i, Y_i) \xrightarrow{TX_{i+1}} (X_{i+1}, Y_{i+1})$.

Slope Point. We introduce a crucial auxiliary concept, called the *slope point*, which plays a significant role in sandwich attack and dagwood sandwich attack. Given an initial state of a liquidity pool (X_0, Y_0) and a transaction $TX(ax, ay)$, we aim to identify a transition $(X_{i-1}, Y_{i-1}) \xrightarrow{TX_i} (X_i, Y_i)$ (where $X_i \cdot Y_i = X_0 \cdot Y_0$) that maximizes the execution price of the transaction (i.e., $TX_i.p$), up to its maximum tolerated prices (i.e., $\frac{ax_i}{ay_i}$). We can calculate (X_i, Y_i) by Formula 2. We define $sp_i = X_i$ as the *slope point* of TX_i.

$$sp_i = X_i = \frac{ax_i + \sqrt[2]{ax_i^2 + 4 \cdot X_0 \cdot Y_0 \cdot \frac{ax_i}{ay_i} \cdot (1-f)}}{2}, Y_i = \frac{X_0 \cdot Y_0}{X_i} \tag{2}$$

3.2 Attack Model

Attack Structure. Given an LP (τ_x/τ_y) and multiple victim transactions $\{TX_i\}_{i=1}^{N}$, the attacker can devise an attack strategy consisting of a series of malicious and victim transactions. We assume that the attacker has the authority to alter the execution order of the victim transactions in order to maximize the attack profit. In addition, as shown in Fig. 1(c), the attacker inserts m FTs, denoted as $\{FT_i\}_{i=1}^{m}$, into these victim transactions, where $m \leq n$. These FTs ensure that the victim transactions are executed in the state envisioned by the attacker. By employing these FTs, the attacker can swap τ_x for τ_y at relatively lower prices. Towards the end of the attack, the attacker emits a back-running transaction BT_1 to convert all swapped τ_y back to τ_x. Note that the execution price of this BT is relatively higher, allowing the attacker to gain profits.

Transaction Selection. We prioritize the selection of transactions to attack based on their slope points. Transactions with higher slope points are given higher precedence for n-MVTL attack. This is because transactions with larger

slope points usually show greater vulnerability as they allow BT emitted by the attacker to swap back τ_x at higher prices. Furthermore, preserving transactions with larger slope points allows for more flexibility in subsequent optimization algorithms, providing additional room for adjustments.

Attack Profit. According to the structure of n-MVTL attack strategy, the attack profit of a n-MVTL attack strategy can be calculated by Formula 3. This means that the attacker's profit is R of τ_x. By ensuring that R is positive, the attacker can determine the profitability of the attack strategy, without the need for the stable price assumption.

$$R = \mathcal{F}_{\mathcal{A}}(S_n, BT, f) - \Sigma_{i=1}^{m}\ FT_i.ax \tag{3}$$

where S_n is the post-execution state of LP after executing the last victim transaction; BT and $\{FT_i\}_{i=1}^{m}$ are the malicious transactions emitted by the attacker.

4 *n*-MVTL Attack

Given an initial state (X_0, Y_0) of an LP and a list of victim transactions $\{TX_i\}_{i=1}^{N}$ with the same swap direction $x \to y$ in the pending pool, we find the optimal attack strategy using two algorithms: **Transaction Selecting** algorithm and **Optimal Attack** algorithm. We first provide a transaction selecting an algorithm to identify the largest subset of vulnerable transactions $\{TX_i\}_{i=1}^{n}$. Then, we provide an optimal attack algorithm to calculate an approximate optimal attack strategy, thereby yielding maximum attack revenue. Our attack process can be represented as follows:

$$\{Tx_i\}_{i=1}^{n} \leftarrow \texttt{TransactionSelecting}((X_0, Y_0), \{TX_i\}_{i=1}^{N}), \tag{4}$$

$$\mathcal{ST} \leftarrow \texttt{OptimalAttack}((X_0, Y_0), \{Tx_i\}_{i=1}^{n}) \tag{5}$$

4.1 Transaction Selecting

We present an approximation iterative algorithm to select victim transactions. In this approximation iterative algorithm, the parameters (i.e., trading amount) of malicious transactions generated by the algorithm are closer to their precise values (i.e., the values that let the last executed transactions in all layers trigger their maximum tolerated prices) as the number of iterations increases. Our algorithm considers *AMM swap fees*, which are not considered in most existing attack designs. In practice, an LP charges AMM swap fees from DeFi users for conducting their swap transactions. Therefore, the supply of the LP will increase after each transition. We use $EK_i \in \mathbb{N}^+$ to represent the estimated supply of the LP after executing TX_i. As illustrated in Fig. 2, **Transaction Selecting** involves three phases: (1) *Initial Phase*; (2) *Iteration Phase*; and (3) *Final Phase*.

Initial Phase. We initialize the parameter required for the iterative algorithm. We assign a value of K_0 for all EK_i, $i \in [N]$. These parameters will be continuously updated throughout the iterations.

Iteration Phase. We execute two processes for each iteration in the iteration phase: transaction generating and transaction executing. We first use the transaction generating function to generate the attack transactions FTs between victim transactions $\{TX_i\}_{i=1}^{N}$ in the LP (in Step 1-3). Then, by transaction executing, we can calculate the post-execution states $\{(X_i, Y_i)\}_{i=1}^{N}$ for $\{TX_i\}_{i=1}^{N}$ (in Step 4), and then use $\{(X_i, Y_i)\}_{i=1}^{N}$ to update EK_i (in Step 5).

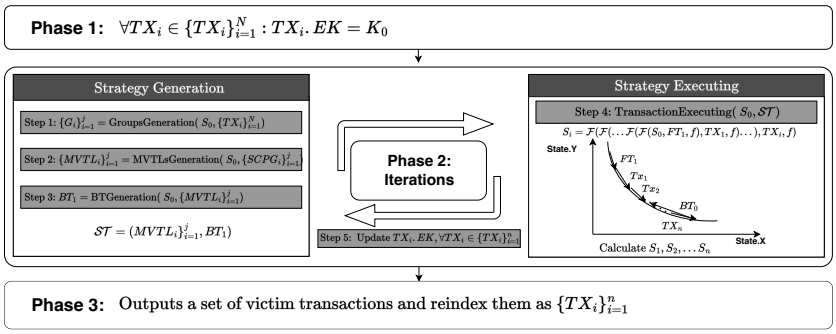

Fig. 2: Overview of Algorithm **Transaction Selecting**.

Step 1 Transaction Group Generation. A transaction group G is defined as a sequential of victim transactions that can be successfully executed consecutively, where the last victim transaction meets its slope point. To facilitate the description of the following steps, we define SS_j and ES_j as the expected start and end states of transaction group G_j, respectively. We represent the state change of the LP upon the execution of G_j as $SS_j \xrightarrow{G_j} ES_j$.

We split the victim transactions $\{TX_i\}_{i=1}^{N}$ into several transaction groups by **Group Generating Algorithm**. In each group, the end state of the last victim transaction triggers at its slope point corresponding to its maximum tolerated prices, while the end state of each other victim transaction is the start state of its next victim transaction.

Building Block: Group Generating Algorithm. Given an input $\{(X_0, Y_0), \{TX_i\}_{i=1}^{N}\}$, the algorithm output a list of $\{G_j\}_{j=1}^{k}$. In this algorithm, for each TX_i, we first use its estimated attribute EK_i instead of $X_0 \cdot Y_0$ and the transaction swap fee rate f to calculate its estimated slope point by Formula 2. Then, we traverse all victim transactions by their estimated slope point in reverse order and put transactions into groups as follows:

1 For the first transaction TX_N, which has the largest slope point, we create the first group and put this transaction into it. We set $SS.X = sp_N - ax_N$ and $ES.X = sp_N$, respectively (where $SS.X$ and $ES.X$ are the numbers of τ_x in the start and end states, respectively).
2 For other $TX_i \in \{TX_i\}_{i=1}^{N-1}$, if $SS.X \leq sp_i$, we insert TX_i to the front of the current G_j, since it can be executed successively and successfully with the subsequent victim transactions (i.e., will not trigger to its slope point) in this group. Then we update $SS.X = SS.X - ax_i$. Otherwise, if $SS.X > sp_i$, we finish the transaction group G_j and create a new G_{j+1}. Then we put TX_i into this group and reset $SS.X$ and $ES.X$ to $sp_i - ax_i$ and sp_i, respectively.

Note that when attempting to place TX_i into an G_j, if $SS.X - ax_i \leq X_0$, the operation of adding TX_i to G_j will be aborted, and the next iteration will proceed. Ultimately, only a subset of the victim transactions may be placed into groups. We reverse the index the transaction group as $\{G_j\}_{i=1}^{k}$, and denote the selected transaction subset as $\{TX_i\}_{i=1}^{n}$.

Step 2 Multiple-Victim-Transactions Layers (MVTLs) Generation. An MVTL $MVTL_j$ is defined as a combination of a front-running transaction FT_j and a

transaction group G_j (cf. Fig. 1(c)). FT_j is used to let the last victim transaction in G_j trigger at its slope point, thereby maximizing the victims' loss.

After we split the victim transactions into $\{G_j\}_{j=1}^{k}$, there are some state change gaps among them. For each G_j, we generate FT_j to fill the state change gap before G_j by Formula 6 and then combine it with G_j into $MVTL_j$. The process of creating FT_i is:

$$FT_i = \begin{cases} FT(SS_j.X - X_0), & j = 1 \\ FT(SS_j.X - ES_j.X), & 1 < j \leq k \end{cases} \tag{6}$$

Step 3 Back-Running Transaction Generation. After generating $\{MVTL_j\}_{j=1}^{k}$, we construct BT that is used to swap all the τ_y swapped by $\{FT_j\}_{j=1}^{k}$ for τ_x. This transaction can be generated as $BT(\mathcal{F}_{\mathcal{A}}(S_0, FT_1, f) + \Sigma_{1<j\leq k}\ \mathcal{F}_{\mathcal{A}}(SS_j, FT_j, f))$.

Step 4 Strategy Executing. We combine the outputs of steps 2 and 3 into a strategy as $\mathcal{ST}' := (\{MVTL_j\}_{j=1}^{k}, BT)$. Then, we can input $\mathcal{ST}'$ and S_0 to Function `Transaction Executing` to calculate the states S_i after executing $TX_i \in \{TX_i\}_{i=1}^{n}$, respectively, as illustrated in Fig. 2.

Step 5 Parameters Updating. Based on the calculated states $\{S_i\}_{i=1}^{n}$ in Step 4, we update the estimated supply ES_i for each transaction $TX_i \in \{TX_i\}_{i=1}^{n}$ by the formula: $EK_i = X_i \cdot Y_i$ and go to the next iteration.

Final Phase. After several iterations in the iteration phase, the algorithm finds a set of victim transactions $\{TX_i\}_{i=1}^{n}$ in which the parameters of all transactions are approximated to the precise values.

4.2 Optimal Attack Algorithm

Our optimal attack algorithm is based on a simple observation: as transactions are executed, the price of token τ_x continuously decreases. Therefore, inserting the front-running attack as early as possible can maximize the attacker's revenue. This observation forms the basis of our optimal attack algorithm, which utilizes two critical building blocks: the **Front-running algorithm** and the **Backward algorithm**.

Building Block: Front-Running Algorithm. Given the current state of the LP and the set of victim transactions, this algorithm calculates an optimal front-running attack, assuming that victim transactions don't have slope points.

- Input: The algorithm gets $\{(X, Y), \{TX_i\}_{i=1}^{n}, s_y\}$ as inputs, where (X, Y) is the current state of the liquidity pool, $\{TX_i\}_{i=1}^{n}$ is the set of victim transactions, and s_y is the number of tokens τ_y hold by the attacker.
- Output: The algorithm output an optimal front-running attack $FT(\Delta x)$.

Although directly calculating Δx is difficult, we can obtain the range of Δx based on Theorem 1. After inserting an attack transaction $FT(\Delta x)$, the attacker's will have $s'_y = s_y + (Y - \frac{X \cdot Y}{X + (1-f)\Delta x})$ number of token τ_y, and the current state (X_1, Y_1) of the LP becomes $X_1 = X + \Delta x$, $Y_1 = \frac{X \cdot Y}{X + (1-f)\Delta x}$.

To simplify the notation, let's denote the amount of tokens in $\{TX_i\}_{i=1}^{n}$ by $\{x_i\}_{i=1}^{n}$, $V_x = \sum_{i=1}^{n} x_i$ and $t = 1 - f$. Then the new state (X_2, Y_2) of the LP after executing all victim transactions satisfied:

$$X_2 = X + \Delta x + V_x,\ \ Y_2^{min} \le Y_2 \le Y_2^{max}.$$
$$Y_2^{min} = \frac{X_1 \cdot Y_1}{X_1 + tV_x} = \frac{(X+\Delta x)(X \cdot Y)}{(X + t\Delta x)(X + \Delta x + tV_x)}.$$
$$Y_2^{max} = \frac{(X+\Delta x)(X \cdot Y)}{t^{n-1}(X + t\Delta x)(X + \Delta x + V_x)}.$$

where Y_2^{min} and Y_2^{max} come from Theorem 1. The attacker's revenue after the back-running transaction is

$$R_x = X_2 - \frac{X_2 \cdot Y_2}{Y_2 + (1-f)s'_y} - \Delta x \tag{7}$$

According to Equation 7, we find that R_x and $\frac{dR_x}{d\Delta x}$ are strictly decreasing when Y_2 is increasing. Therefore, we can obtain the Δx range by taking the derivative $\frac{dR_x}{d\Delta x} = 0$. The result of $\Delta x \in [\Delta x^{min}, \Delta x^{max}]$ is:

$$\Delta x^{max} = \frac{X(s_y(-1+t)tX - XY + t^2(V_x+X)Y) + \sqrt{B^{max}}}{(-1+t)\left(s_y t(tV_x - X) + (t^2V_x - X - tX)Y\right)}$$
$$B^{max} = t^2V_x^2X(s_y(-1+t)+tY)$$
$$\times \left(s_y(-1+t)t(-X + t(V_x+X)) + (X - t^2(V_x+X) + t^3(V_x+X))Y\right)$$
$$\Delta x^{min} = \frac{X(s_y(-1+t)t^nX - XY + t^{1+n}(V_x+X)Y) + \sqrt{B^{min}}}{(-1+t)s_y t^n(tV_x - X) + (t^{2+n}V_x + X - t^{1+n}(V_x+X))Y}$$
$$B^{min} = t^{1+n}V_x^2X(s_y(-1+t)+tY)$$
$$\times \left(s_y(-1+t)t^n(-X + t(V_x+X)) + (X - t^{1+n}(V_x+X) + t^{2+n}(V_x+X))Y\right)$$

Building Block: Backward Algorithm. Given the current state of LP and the set of victim transactions, this algorithm calculates a maximized FT attack while ensuring the last victim transaction can be successfully executed.

- The algorithm gets $\{X, Y, \{TX_i\}_{i=1}^k\}$ as input, where (X, Y) is the current state of the liquidity pool, $\{TX_i\}_{i=1}^k$ is the set of victim transactions.
- Output: The algorithm output a maximized front-running attack $FT(\Delta x)$.

We also calculate the range of Δx based on Theorem 1. To simplify the notation, we denote the number of tokens in $\{TX_i\}_{i=1}^k$ by $\{x_i\}_{i=1}^k$, $V_x = \sum_{i=1}^{k-1} x_i$ and $t = 1 - f$. Then the post-executing state after FT is $X_0 = X + \Delta x$, $Y_0 = \frac{XY}{X+(1-f)\Delta x}$. The state $\{X_{k-1}, Y_{k-1}\}$after executing $\{TX_i\}_{i=1}^{k-1}$ is:

$$X_{k-1} = X_0 + \Delta x + V_x,\ \ Y_{k-1}^{min} \le Y_{k-1} \le Y_{k-1}^{max}$$
$$Y_{k-1}^{min} = \frac{X_0 \cdot Y_0}{X_0 + tV_x} = \frac{(X+\Delta x)(X \cdot Y)}{(X + t\Delta x)(X + \Delta x + tV_x)}.$$
$$Y_{k-1}^{max} = \frac{(X+\Delta x)(X \cdot Y)}{t^{n-1}(X + t\Delta x)(X + \Delta x + V_x)}.$$

Then TX_k can swap y_k number of token:

$$y_k = Y_{k-1} - \frac{X_{k-1} \cdot Y_{k-1}}{X_{k-1} + tx_k} = \frac{t \cdot Y_{k-1}x_k}{X_{k-1} + tx_k} \tag{8}$$

According to Equation 8, we know that y_k is strictly decreasing when Y_{k-1} is increasing. Therefore we can obtain the range of $\Delta x \in [\Delta x^{min}, \Delta x^{max}]$ by solve the Equation 8:

$$\Delta x^{min} = (-\frac{1+t}{2})V_x - X - \frac{tx_k}{2} + \sqrt{\frac{tXx_kY}{yk} + \frac{((1-t)(V_x - x_k) + x_k)^2}{4}}$$

$$\Delta x^{max} = t^{1-k}\Delta x^{min}$$

Attack Algorithm. Given a start state (X_0, Y_0) of the LP and a sequence of sorted victim transactions $\{TX_i\}_{i=1}^n$, the attack algorithm outputs an attack strategy ST, which maximizes the attacker's revenue.

The algorithm runs in n rounds. In each round $k \in [n]$, the algorithm attack transactions $\{TX_i\}_{i=k}^n$. At the beginning of each round, the algorithm initializes parameters $s_y = 0$, $X = X_0$, $Y = Y_0$, and $l = k$, where s_y is the amount of token τ_y the attacker have got, and l is the index of victim transaction. In each round, the algorithm runs in two steps:

1 The algorithm run **Front-running Algorithm**$(X, Y, \{TX_i\}_i^n, sy = 0)$, receive an interval $\{\Delta x^{min}, \Delta x^{max}\}$ containing the optimal front-running attack. Then the algorithm runs **Binary Search** d times to search an FT approximated with the optimal attack transaction.

2 The algorithm executes FT from state (X, Y), then try to execute victim transactions $\{TX_i\}_{i=l}^n$.
- All transactions are executed successfully: The algorithm should record the attack strategy and moves to the next round.
- When executing transactions, a transaction TX_j reach its slope point: Now the algorithm should run **Backward Algorithm** $(X, Y, \{TX_i\}_{i=l}^j)$, receive and interval $\{\Delta x^{min}, \Delta x^{max}\}$. Then using **Binary Search** d times, the algorithm can find an approximate maximum FT attack x which swaps y number of tokens τ_y. Assume the final state after executing transactions FT and $\{TX_i\}_{i=l}^j)$ is (X_j, Y_j) Finally, the algorithm set $s_y = s_y + y$, $X = X_j$, $Y = Y_j$, $l = j + 1$ and continue running step 1.

Finally, the algorithm compares all the revenues in n rounds and chooses the attack strategy $\mathcal{ST}$ with the largest revenue.

Each step of the algorithm runs binary search d times, in which the algorithm should execute at most n transactions. Hence the algorithm runs at most $O(nd)$ time in each step. As the algorithm runs each step at most n times in each round, and there are n rounds, the time complexity of the optimal attack algorithm is $O(n^3 d)$.

4.3 Implementing Attacks on Blockchain

We consider an Ethereum-like blockchain where many DeFi users initiate their transactions on the pending pool (cf. Fig. 3). We consider a rational attacker $\mathcal{A}$ who observes the pending pool's transactions in real-time. When $\mathcal{A}$ detects one or more victim transactions that interact with the same LP's smart contract and have the same swap direction, $\mathcal{A}$ can use n-MVTL attack algorithms to find an optimal attack strategy $\mathcal{ST}$ to attack these victim transactions. After that, $\mathcal{A}$

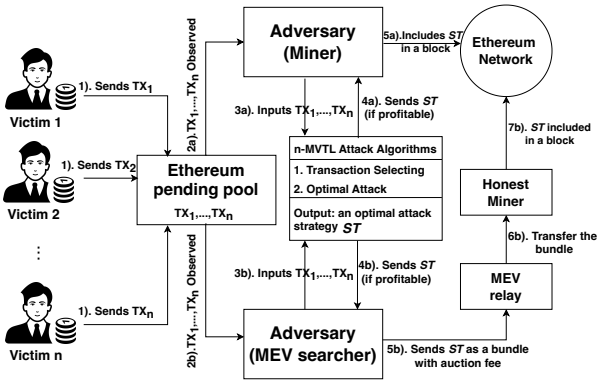

Fig. 3: n-MVTL attack system.

can include $\mathcal{ST}$ in a new block and broadcast it to Ethereum on his own if $\mathcal{A}$ is a miner (cf. 5a of Fig. 3). Otherwise, $\mathcal{A}$ can send $\mathcal{ST}$ as a bundle with an auction fee to an honest miner via an MEV relay (e.g., flashbots [3], Eden Network [6]). If $\mathcal{A}$ wins the auction, the honest miner will then include $\mathcal{ST}$ in a new block and broadcast it to Ethereum (cf. 5b-7b of Fig. 3).

The n-MVTL attack surpasses prior works by overcoming three limitations: (I) *No stable price assumption.* The attack profit obtained by the n-MVTL attack are positive amount of Token X, hence ensuring positive gains without the stable price assumption. (II) *Consider AMM swap fees.* The supply of a liquidity pool is not constant when considering AMM swap fees. After each swap transaction is executed, a portion of the tokens is collected as AMM swap fees and added to the liquidity pool. In our algorithm, we take this factor into account to generate attack strategies. This increases the computational complexity, but it makes our generated attack strategies more practical. (III) *Hard to defend.* We employ a structure n-MVTL to exploit the vulnerability among multiple victim transactions, making it difficult for DeFi users to resist n-MVTL attack using conventional methods. We show how the novel structure works in Appendix B.

5 Analysis

In this section, we first proved the upper and lower bounds of the pool state after executing re-ordered transactions. Then we provide a comprehensive analysis of the revenue of the optimal attack algorithm.

5.1 Post-execution State Analysis

The greatest difficulty when analyzing the revenue of the algorithm is determining the quantity of token τ_y in the LP, because it is according to the transaction order. To estimate the quantity of Y in the pool after n trades, we prove the following theorem:

Theorem 1. *Given a start state* (X_0, Y_0) *of the LP and a set of transactions* $\{TX_i\}_{i=1}^n$ *in which the swap tokens are* $\{x_i\}_{i=1}^n$. *Then post-executing state* (X_n, Y_n) *satisfied:*

$$X_n = X_0 + V_x$$

$$\frac{X_0 Y_0}{X_0 + (1-f)V_x} < Y_n < \frac{X_0 Y_0}{X_0 + (1-f)V_x} * (1-f)^{1-n}$$

where $V_x = \sum_{i=1}^{n} x_i$.

The proof is in Appendix A.1.

With Theorem 1, we can estimate the algorithm output in **Front-running Algorithm** and **Backward Algorithm**.

Corollary 1. *In Front-running algorithm (or Backward algorithm) with d times of Binary search, the swap amount Δx in the output attack transaction and the swap amount Δx_{opt} for the optimal attack transaction are satisfied:*

$$\frac{\Delta x_{opt} - \Delta x}{\Delta x_{opt}} \leq \frac{(1-f)^{1-n} - 1}{2^d} \tag{9}$$

where n is the number of input victim transactions in the algorithm.

Proof. Because $\Delta x_{max} \leq (1-f)^{1-n} \Delta x_{min}$, thus we have:

$$\frac{\Delta x_{opt} - \Delta x}{\Delta x_{opt}} \leq \frac{\Delta x_{max} - \Delta x_{min}}{2^d \Delta x_{min}} = \frac{(1-f)^{1-n} - 1}{2^d} \tag{10}$$

5.2 Revenue Analysis

Meanwhile, we notice that when the total number of τ_x in the FT is fixed, the attacker can get more revenue by inserting as many front-running transactions as possible in the front. This statement can be written into the following lemma.

Lemma 1. *Given a starting state (X_0, Y_0), a victim transactions TX_1 and two attacking transaction $FT_1 = FT(\Delta x_1)$ and $FT_2 = FT(\Delta x_2)$, the attack sequence $\{FT(\Delta x_1 + \Delta x_2), TX_1\}$ can earn more token τ_y than sequence $\{FT_1, TX_1, FT_2\}$, if TX_1 do not reach its slope point.*

The proof is straightforward as the price of token τ_y is a monotone increase while executing transactions. Then according to Corollary 1 and Lemma 1, we have the following theorem:

Theorem 2. *Given a set of ordered victim transactions $\{TX_i\}_{i=1}^{n}$, assume the revenue of our attack is P_{n-MVTL}, the maximum revenue by any attack strategy is P_{max}, we have the following lower-bound:*

$$P_{n-MVTL} > (1 - \frac{(1-f)^{1-n} - 1}{2^d})^2 P_{max} \tag{11}$$

The proof is in Appendix A.2. This theorem shows that for any ϵ, the algorithm with parameter $d > \log_2 \frac{(1-f)^{1-n}-1}{\epsilon}$ can receive at least $(1-\epsilon)$ of the maximum revenue. Therefore, the attack strategy of the algorithm is an approximate optimal attack strategy.

6 Evaluation

We first implement a prototype of an attack system in Python 3.8.0. We rigorously test the implementation of the critical method $\mathcal{F}$ in our attack system to ensure that our system does not deviate from the real smart contract due to different implementations of rounding methods. We use real-world data to test the accuracy of our calculations, where test data is extracted from the `Sync event`[3] initiated by the Uniswap V2 smart contract to obtain the state of the

[3] Topic0 of Sync events: 0x1c411e9a96e071241c2f21f7726b17ae89e3cab4c78be50e062b03a9fffbbad1

liquidity pool before and after each transaction, and the `Swap event`[4] initiated by the Uniswap V2 smart contract to obtain the amount of τ_x spent by the user in the transaction and the amount of τ_y obtained. We test 10,000 historical transactions by executing them based on their respective states before execution, and the execution states calculated by our attack system are consistent with the execution states in history.

Then, we conduct the evaluation on a machine with Quad-Core Intel Core i5 (1.4 GHz) and 16 GB memory. We conduct a series of experiments aimed at (a) evaluating the convergence rate of **Transaction Selecting**, (b) evaluating the time efficiency of attack algorithms, (c) validating the profit capability of attack strategies, and (d) conducting the trade-off analysis for **Optimal Attack**.

6.1 Convergence Rate of the Iterative Algorithm

We define the loss of an attack strategy $\mathcal{ST}$ as follows:

$$strategy\_loss = \text{mean}(\Sigma_{i=1}^{\text{Size}(\mathcal{ST}.MVTLs)} \ |\hat{FT}_i.ax - FT_i.ax|)$$

where $\hat{FT}_i$ denotes the i^{th} FT in $\mathcal{ST}$ with estimated parameters generated by **Transaction Selecting**, while FT_i denotes the i^{th} FT with precise parameters generated by **Transaction Selecting** and using binary search algorithm. We evaluate the loss of $\mathcal{ST}$ in 10 numbers of iterations based on the different sizes of LPs in Fig. 4. We can see the loss shows a logarithmic decrease in all cases as the number of iterations increases. When the number of iterations is 8, losses in all cases are smaller than 1,000.

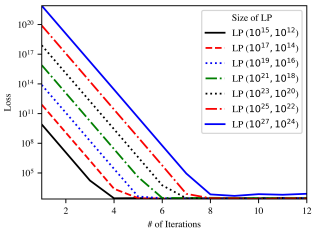

Fig. 4: Loss of Transaction Selecting Algorithm

6.2 Time Complexity

To evaluate the time efficiency of our algorithms, we record the time taken to perform each step in our algorithms. We set the state at $(10^{27}, 10^{24})$ in the experiments since it exceeds the current size of any liquidity pool. Victim transactions were randomly generated in the experiments as the veracity of the transactions will not influence the time efficiency of our algorithms. We set the number of iterations utilized in **Transaction Selecting** to 10. We do not limit the number of iterations of the BSA utilized in **Optimal Attack**. As we can see from Table 1, the time cost of **Transaction Selecting** algorithms rises nearly linearly in all cases as the number of random transactions increases. The time cost of **Optimal Attack** has cubic growth. In total, it takes around 4 seconds to build an optimized attack strategy when the number of random transactions is 100. Note that the average number of executed transactions per block is 154.8, and the average block generation time is 12.2 seconds on April 20, 2023, in Ethereum [2]. The results show that our algorithms are feasible in the current Ethereum.

[4] Topic0 of Swap events: 0xd78ad95fa46c994b6551d0da85fc275fe613ce37657fb8d5e3d130840159d822

Table 1: Time Cost of Attack Algorithms.

Algorithms \ Transactions	10	20	50	100
Strategy finding	0.014s	0.024s	0.510s	0.120s
Strategy optimization	0.042s	0.171s	0.888s	4.018s
All	0.560s	0.195s	1.398s	4.138s

6.3 Attack Strategy Validation

We evaluate the profitability of n-MVTL attack on historical blockchain data from block 120,000,000 to block 170,000,000 over a total of 699 days. We locally deploy the Uniswap V2 Router02 smart contract[5] using Foundry[6], a popular EVM development platform, and interact with the deployed smart contracts to execute all malicious and victim transactions. We compute the actual profits based on the transaction execution results in Foundry. We focus on the transactions of the top 2 most popular liquidity pools in Uniswap v2 (i.e., ETH/USDC and ETH/USDT). Our attack is solely targeted at successfully executed transactions recorded on the blockchain. Each attack is specifically directed towards transactions with the same swap direction that are present within a single block. We set the number of iterations utilized in FindStrategy to 10. We do not limit the number of iterations of the BSA utilized in OptimizeStrategy. To measure the performance of n-MVTL attack on extracting extra profits compared to sandwich attacks, we establish a comparison group as a baseline by launching sandwich attacks on victim transactions. As shown in Table 2, n-MVTL attack can extract extra profits in all cases compared to the baseline. In total, n-MVTL attack can yield 656,976 (i.e., 295,608 + 361,368) USD of extra attack profit from 5,006 (i.e., 2,498 + 2,508) profitable victim transactions since n-MVTL attack can only attack victim transactions in one swap direction at the same time.

Table 2: Estimated attack profit of SVTL attack and MVTL attack.

LP and swap direction		Profitable TXs / total TXs	Attack profit of 1-SVTL attack	Attack profit of n-MVTL attack	# of n-MVTL attacks	Extra profit (token)	Extra profit (USD)
ETH/USDC (Uniswap V2)	ETH ->USDC	1,965/1,481,222	562.24 ETH	616.40 ETH	592	54.16 ETH	103,066 USD
	USDC ->ETH	2,498/1,373,080	3,555,817 USDC	3,851,425 USDC	790	295,608 USDC	295,608 USD
ETH/USDT (Uniswap V2)	ETH ->USDT	2,564/1,387,482	677.27 ETH	718.76 ETH	782	41.49 ETH	78,955 USD
	USDT ->ETH	2,508/1,472,498	1,230,739 USDT	1,592,419 USDT	798	361,368 USDT	361,368 USD

To eliminate the influence of private pending pools on n-MVTL attacks, we did not attack against failed and pending transactions. Our attacks are restricted to transactions that were successfully executed within the same block in the historical blockchain. These transactions were inevitably witnessed by one miner, regardless of whether these transactions originated from private pools. Thus, if the miner possesses malicious intent, he can launch an n-MVTL attack against these transactions. As the number of transactions attacked in this validation is less than the number of real attackable transactions in history. Therefore, the profit statics only represent lower bounds on the severity of n-MVTL attack.

[5] https://github.com/Uniswap/v2-periphery/blob/master/contracts
[6] https://github.com/foundry-rs/foundry

6.4 Trade-off Analysis of Time Cost and Profit

We employ historical transaction data of LP (ETH/USDT) for trade-off analysis. We re-run the attack on transactions in the USDT->ETH swap direction by varying the limits on the number of iterations of the binary search algorithm (BSA) utilized in OptimzieStrategy. Fig. 5 illustrates the interplay between time cost, attack profit, and the corresponding iteration limits in BSA. As the iteration limits increase, the time cost exhibits a linear growth pattern, while the attack profit demonstrates exponential growth when the number of iterations is below 15. Remarkably, the profit nearly converges to that of attacks with no limit when the number of iterations reaches 15. In a real-world setting, the attacks can adjust the iteration limits within BSA to strike a trade-off between time cost and profitability.

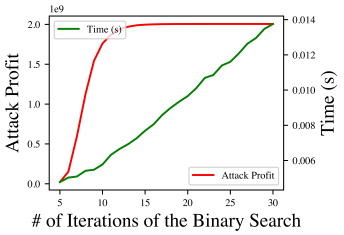

Fig. 5: Average time cost and attack profit of attacks against LP (ETH/USDT)

7 Conclusion

In this paper, we propose a novel transaction reordering attack, n-MVTL attack, to attack against multiple transactions in DeFi. Unlike traditional transaction reordering attacks, n-MVTL attacks enable attacks on un-sandwichable transactions and consider *AMM swap fees*. In addition, we provide an optimal algorithm to generate an optimal n-MVTL attack strategy with maximum attack profit. This algorithm strikes a balance between time efficiency and attack profit, enhancing the practicality of the attack algorithm. We also validate the attack strategies on historical blockchain data. The result shows that the n-MVTL attack can generate an average daily more profit of 940 USD compared to the sandwich attack. Our new attack can offer attackers more profit and thus cause more loss to normal users in DeFi. Compared with the sandwich attack, n-MVTL attack is more difficult to defend against and harmful to DeFi users. We hope our research raises awareness of this unresolved MEV risk and engenders future work on defense mechanisms against MEV.

References

1. Bartoletti, M., Chiang, J.H.y., Lluch-Lafuente, A.: Maximizing Extractable Value from Automated Market Makers. 2022 International Conference on Financial Cryptography and Data Security (FC) (2022)
2. Bitinfocharts, https://bitinfocharts.com/
3. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 910–927. IEEE (2020)
4. DeFi Tracker, https://defiprime.com/dex-volume
5. DeFi Llama, https://defillama.com/
6. Eden Network, https://www.edennetwork.io/

7. Heimbach, L., Wattenhofer, R.: Eliminating Sandwich Attacks with the Help of Game Theory. In: Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security (ACM ASIACCS 2022). pp. 153–167 (2022)
8. Pancakeswap, https://pancakeswap.finance/
9. Uniswap, https://www.uniswap.org
10. Uniswap v1, https://docs.uniswap.org/protocol/V1/introduction
11. Wang, Y., Zuest, P., Yao, Y., Lu, Z., Wattenhofer, R.: Impact and User Perception of Sandwich Attacks in the DeFi Ecosystem. In: CHI Conference on Human Factors in Computing Systems. pp. 1–15 (2022)
12. Zhou, L., Qin, K., Gervais, A.: A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. arXiv preprint arXiv:2106.07371 (2021)
13. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-Frequency Trading on Decentralized On-Chain Exchanges. In: 2021 IEEE Symposium on Security and Privacy (SP). pp. 428–445. IEEE (2021)
14. Züst, P.: Analyzing and Preventing Sandwich Attacks in Ethereum. https://pub.tik.ee.ethz.ch/students/2021-FS/BA-2021-07.pdf (2021)

A Proof

A.1 Proof for State Analysis

The proof of the lower bound for Y_n comes from the following claim.

Claim. Given two transactions TX_1, TX_2 with swap tokens x_1 and x_2 and a transaction TX with swap tokens $x = x_1 + x_2$, the number of τ_y in the pool after running transaction TX is lesser than running transactions TX_1 and TX_2.

Proof. We denote the number of tokens τ_y in the pool after executing transaction TX as Y_1, the number after executing TX_1, and TX_2 as Y_2. Assume the state of the LP is X_0, Y_0:

$$Y_1 = \frac{X_0 Y_0}{X_0 + (1-f)(x_1 + x_2)},$$

$$Y_2 = \frac{(X_0 + x_1) X_0 Y_0}{(X_0 + (1-f)x_1)(X_0 + x_1 + (1-f)x_2)},$$

$$\frac{Y_1}{Y_2} = \frac{(X_0 + (1-f)x_1)(X_0 + x_1 + (1-f)x_2)}{(X_0 + x_1)(X_0 + (1-f)(x_1 + x_2))}.$$

Notice that $(X_0 + x_1) + (X_0 + (1-f)(x_1 + x_2)) = (X_0 + (1-f)x_1) + (X_0 + x_1 + (1-f)x_2)$ and $(X_0 + (1-f)x_1) < (X_0 + x_1)$, $(X_0 + (1-f)(x_1 + x_2)) < (X_0 + x_1 + (1-f)x_2)$. With average inequality, we have $\frac{Y_1}{Y_2} < 1$, thus $Y_1 < Y_2$.

With this claim, we can get the lower bound by merging all the victim transactions into one transaction TX, in which $V_x = \sum_{i=1}^{n} x_i$. Thus we have:

$$Y_n > \frac{X_0 Y_0}{X_0 + (1-f)V_x}.$$

The proof of the upper bound for Y_n comes from the following claim.

Claim. Given the start state X_0, Y_0, two transactions TX_1 and TX_2 such that the swap tokens satisfied $x_1 + x_2 = T$. Then when $x_1 = \sqrt{X_0 T} - X_0$, the pool has a maximum number of tokens τ_y after executing two transactions.

Proof. Assume the number of tokens τ_y after executing is Y, we have:

$$Y = \frac{(X_0 + x_1)(X_0 + T)X_0Y_0}{(X_0 + (1-f)x_1)(X_0 + x_1 + (1-f)(T - x_1))},$$
$$\frac{dY}{dx_1} = \frac{f(1-f)X_0Y_0(X_0 + T)(X_0T - 2X_0x_1 - x_1^2)}{(X_0 + (1-f)x_1)^2(X_0 + x_1 + (1-f)(T-x))^2}.$$

Notice that $\frac{dY}{dx_1} > 0$ when $x_1 < \sqrt{X_0T} - X_0$, and $\frac{dY}{dx_1} < 0$ when $x_1 > \sqrt{X_0T} - X_0$. Thus Y is maximum when $x_1 = \sqrt{X_0T} - X_0$.

With this claim, we know that the maximum Y_n when V_n is given is when $X_i^2 = X_{i-1} \cdot X_{i+1}$ for each $i \in [n-1]$. Therefore, we have:

Thus the maximum Y_n is:

$$\begin{aligned}
Y_n &\le \frac{X_0Y_0}{X_n} \prod_{i=1}^{n} \frac{X_i}{fX_{i-1} + (1-f)X_i} \\
&= \frac{X_0Y_0}{X_n} \prod_{i=1}^{n} \frac{X_0^{\frac{n-i}{n}}(X_0 + V_x)^{\frac{i}{n}}}{fX_0^{\frac{n-i+1}{n}}(X_0 + V_x)^{\frac{i-1}{n}} + (1-f)X_0^{\frac{n-i}{n}}(X_0 + V_x)^{\frac{i}{n}}} \\
&= \frac{X_0Y_0}{X_n} \prod_{i=1}^{n} \frac{X_0^{\frac{n-i}{n}}(X_0 + V_x)^{\frac{i}{n}}}{X_0^{\frac{n-i}{n}}(X_0 + V_x)^{\frac{i-1}{n}}\left(fX_0^{\frac{1}{n}} + (1-f)(X_0 + V_x)^{\frac{1}{n}}\right)} \\
&= \frac{X_0Y_0}{X_n} \prod_{i=1}^{n} \frac{(X_0 + V_x)^{\frac{1}{n}}}{\left(fX_0^{\frac{1}{n}} + (1-f)(X_0 + V_x)^{\frac{1}{n}}\right)} \\
&= \frac{X_0Y_0}{X_n} \frac{(X_0 + V_x)}{\left(fX_0^{\frac{1}{n}} + (1-f)(X_0 + V_x)^{\frac{1}{n}}\right)^n} \\
&= \frac{X_0Y_0}{X_n} \frac{(1-f)^{-n}}{(1 + \frac{fX_0^{1/n}}{(1-f)(X_0+V_x)^{1/n}})^n} \\
&\le \frac{X_0Y_0}{(1-f)^{n-1}(fX_0 + (1-f)(X_0 + V_x))} \\
&= \frac{X_0Y_0}{X_0 + (1-f)V_x} \cdot (1-f)^{1-n}
\end{aligned}$$

Thus the upper bound is proved.

A.2 Proof for Profit

Due to the page limit, we only give a proof of sketch. Firstly, if the output of the Front-running and Backward algorithms is the optimal solution, we prove that our algorithm can get the maximum attack profit.

Proof. Assume there has an optimal attack strategy that executes transactions as $\{FT_i, G_i\}_{i=1}^{k}$. According to Lemma 1, the last transaction of G_i, $1 \le i \le k-1$ must reach its slope point. (Otherwise, remove a part of FT in the front can get more revenue) This is exactly what our Backward algorithm is working for. Also, for the last FT transaction FT_k, it should consider BT to maximize its profit, which is exactly the result of our Front-running algorithm. Thus our algorithm can get the maximum profit.

However, our Front-running and Backward algorithms have a little loss in the output, which can be bounded according to Corollary 1. We now calculate the total loss in the attack.

Assume there are k numbers of front-running transactions $\{FT_i\}_{i=1}^{k}$ in our attack. By Section 4, we know that the first $k-1$ FT is calculated by Backward algorithm. Assume the swap tokens τ_x in the transaction FT_i is Δx_i, and $V_{FT} = \sum_{i=1}^{k-1} \Delta x_i$, by Corollary 1 we have:

$$\frac{V_{FT}^{opt} - V_{FT}}{V_{FT}^{opt}} \leq \frac{(1-f)^{1-n} - 1}{2^d}.$$

As the price of τ_y is monotone increasing, the profit of Backward algorithm P_{BW} satisfied:

$$P_{BW} \geq (1 - \frac{(1-f)^{1-n} - 1}{2^d}) P_{BW}^{opt}$$

Then we calculate the loss of FT_k. Because of backward algorithm loss, the input state Y_n is bigger than the exactly Y_n^{max} and satisfied:

$$\frac{Y_n}{Y_n^{opt}} \leq \frac{V_{FT}^{opt}}{V_{FT}} \leq \frac{1}{1 - \frac{(1-f)^{1-n}-1}{2^d}}$$

And according to Corollary 1, the gap between the output Δx of the algorithm and the optimal output Δx_{max} in FT_k also satisfied:

$$\frac{\Delta x_{opt} - \Delta x}{\Delta x_{opt}} \leq \frac{(1-f)^{1-n} - 1}{2^d}$$

So the profit of Front-running algorithm P_{FR} satisfied:

$$P_{FR} \geq (1 - \frac{(1-f)^{1-n} - 1}{2^d}) P_{FR}^{opt}$$

As there are only two types of loss in the algorithm, we finished the proof of Theorem 2.

B Examples

B.1 Example 1 for A Typical 1-MVTL Attack.

As shown in Fig. 6, we assume that a user $\mathcal{U}$ wants to swap 120,000 τ_x for at least 800 τ_y. If $\mathcal{U}$ initiates a transaction with 120,000 token X, this transaction is prone to sandwich attack (cf. (1) of Fig. 6). Suppose $\mathcal{U}$ uses the limiting volume defense strategy [14] that splits her transaction into four small transactions to defend against sandwich attack (cf. (2a) of Fig. 6). Then, each small transaction only has a small trading volume (30,000 τ_x) so that none of the split transactions can be attacked by the sandwich attack (cf. (2b) of Fig. 6). In contrast, the n-MVTL attack can identify the overall vulnerability among the victim transactions. In this case, the large state change provided by these split transactions is one form of overall vulnerability, which are prone to n-MVTL attack. As shown in (3) of Fig. 6, the attack profit of the n-MVTL attack is 27,621 of τ_x.

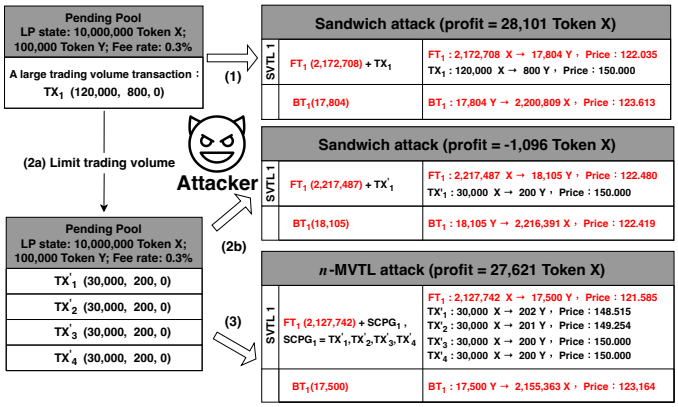

Fig. 6: Example 1. A 1-MVTL attack strategy. We highlight the malicious transactions (in red) initiated by $\mathcal{A}$.

B.2 Example 2 for n-MVTL Attack with Optimization.

When the real price of a cryptocurrency increases or decreases dramatically, there might be a large number of arbitrage transactions in the pending pool with the same swap direction. We assume that the current state of an LP is (10,000,000, 1,000,000), and the real price of this token pair is 11.0. The pending pool has ten arbitrage transactions, as illustrated in Fig. 7. To attack these victim transactions, we use **Transaction selecting** (cf. Section 4.1) to find the largest set of victim transactions $\{TX_i\}_{i=2}^{11}$ that can be attacked together, and these transactions can be grouped into five MVTLs. In each MVTL, there exists one FT and one or more victim transactions. Then, we optimize the attack strategy by **Optimal Attack**. The algorithm's results indicate that we can maximize the attack profit when we only attack against $\{TX_i\}_{i=5}^{11}$. The strategy optimization increases the attack profit from 4,621 of τ_x to 7,042 of τ_x.

We observe that TX_1 and TX_2 have the ability to defend against sandwich attack since they are set with a small slippage (only 1%). However, they still face the risk of n-MVTL attack. In the optimal n-MVTL attack strategy, TX_1 and TX_2 are not executed intentionally by $\mathcal{A}$. We can regard that TX_1 and TX_2 suffer a fatal front-running attack that makes the users fail to swap their tokens.

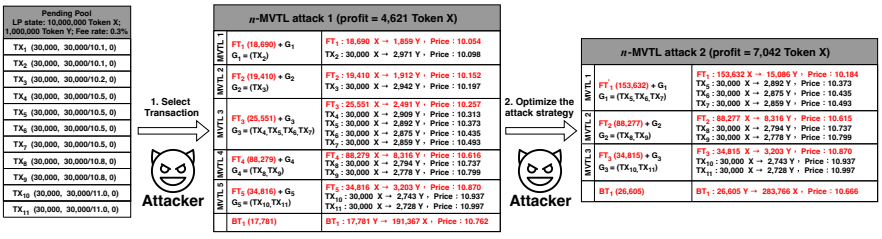

Fig. 7: Example 2. An optimized n-MVTL attack strategy. We highlight the malicious transactions (in red) initiated by $\mathcal{A}$.