The AMMazing Frontrunner: Practical Frontrunning on the XRP Ledger Automated Market Maker

Double-Blind Submission

***Abstract*—Often referred to as A Dark Forest, Ethereum is home to predatory trading bots that prey on user transactions. Frontrunning is made simpler on Ethereum as builders & validators are incentivised to process the highest fee transactions first. Therefore, frontrunners can control how their transactions are executed in relation to the victims. One suggested mitigation strategy is to process transactions in a pseudo-random order, preventing frontrunners from predictably affecting transaction execution order.**

XRP Ledger, one of the oldest blockchains to use pseudo-random ordering, is launching an Automated Market Maker that seamlessly integrates with the existing Close Limit Order Book decentralized exchange.

This study investigates the applicability of frontrunning techniques commonly observed in Ethereum Automated Market Makers to the forthcoming XRP Ledger Automated Market Maker. In summary, our findings demonstrate that with minor adjustments, the conventional Sandwich Attack is feasible. Additionally, we unveil a distinctive attack facilitated by the integration with the Close Limit Order Book.

***Index Terms*—XRP Ledger, Frontrunning, Automated Market Maker, Blockchain, Security**

I. Introduction

Blockchain technology is revolutionizing traditional finance. A key player in the financial revolution is the new decentralized finance *DeFi* paradigm. The core idea behind DeFi is that users can perform various financial operations in a trust-less environment without relying on trusted third parties such as banks, brokers and other financial institutions. One member of the DeFi ecosystem, the decentralized exchange (DEX), has attracted exceptional adoption [1]. A DEX is a marketplace that allows users to exchange assets directly without intermediaries. Despite their popularity, DEXes have a crucial flaw: *frontrunning*. Adversaries observe the content of public pending transactions and place competing orders before them to extract wealth from the traders.

The ability to influence transaction execution order is one of the primary components of frontrunning. On Ethereum [2], block producers receive a portion of the transaction fees as a reward for their work. Therefore, the producers are incentivized to prioritize high-fee transactions. Frontrunners take advantage of the incentives. By paying a higher or lower fee than the victim, the frontrunners influence the transaction execution order in relation to the victim. Frontrunning on Ethereum is a well-studied topic in academia [3], [4], [5] and industry [6].

XRP Ledger is one of the oldest well-established blockchains [7]. Unlike the *Proof-of* protocols used in Bitcoin or Ethereum, XRP Ledger relies on a federated byzantine agreement protocol, where a group of validators vote to advance the ledger history. As a result, compared to Ethereum, XRP Ledger offers significantly higher transaction speeds and lower transaction fees. Unlike other blockchains, XRPL operates a Close Limit Order Book (CLOB) based DEX. In a CLOB, users issue buy and sell offers, and the blockchain matches these with existing ones at the best possible price. To protect the DEX against frontrunning, XRP Ledger executes transactions in a pseudo-random order, or so-called Canonical Order [8]. Despite being a recommended mitigation strategy against frontrunning [9], it is not without limitations.

In a recent study, Tumas *et al.* [10] demonstrated that the Canonical Order makes frontrunning attacks probabilistic. I.e. the frontrunner has only a 50% chance of a successful attack. However, due to the low transaction fees of the blockchain, an attacker can perform frontrunning attacks with multiple accounts to increase their odds of success without incurring significant additional costs, bypassing the benefits of the Canonical Order.

The XRP Ledger community released an Automated Market Maker (AMM) based DEX on the XRP Ledger [11]. In contrast to CLOB, AMMs determine the exchange rate algorithmically. However, AMMs are also the most lucrative sources of frontrunning on the Ethereum network. Therefore, in this paper, we examine to what extent the existing frontrunning techniques of Ethereum apply to the upcoming XRP Ledger AMM and whether the unique integration with the existing CLOB exposes new frontrunning opportunities.

Concretely, our contributions are as follows:

- We analyze the feasibility of frontrunning on the XRP Ledger Automated Market Maker.
- We demonstrate that the existing frontrunning techniques are also feasible on XRP Ledger. However, more importantly, we discover a new type of frontrunning attack that takes advantage of the unique AMM/CLOB integration.
- We provide heuristics for detecting these attacks.

We organise the remainder of this paper as follows. In Section II, we describe the relevant components of the XRP Ledger and outline the AMM design. We describe the evaluation methodology in Section III and discuss our results in Section V. Finally, we examine related work and conclude our work in Sections VI and VII.

II. Background

In this section, we overview XRP Ledger and its Automated Market Maker. Furthermore, we discuss the interaction between the new Automated Market Maker and the existing Close Limit Order Book.

A. XRP Ledger

The XRP Ledger is a public blockchain that operates on a distributed decentralized peer-to-peer network. It facilitates fast and cost-effective transfer of the digital native currency XRP.

Fundamentally, XRP Ledger is an append-only distributed list of records. Users change the state of the database by submitting cryptographically signed instructions, known as transactions. Users identify themselves with cryptographic signatures derived from a pair of public/private keys.

Each XRP Ledger server follows the same deterministic rules to process user transactions. The servers use an XRP Ledger Consensus Protocol [7] to agree on a common state of the blockchain.

1) Accounts: An account corresponds to a holder of XRP and a source of transactions. Each account is uniquely identified by *Address* and a *Sequence* number. The *address* is a sequence of 20 bytes in Base58 format derived from the public key of the account. The *sequence* number is a 32-bit unsigned integer used to ensure the account's transaction execution order.

2) Transactions: A transaction is the only way to update the state of the XRP Ledger. Each is identified with a unique *hash*, *sequence* number and the address of the sending account. The sequence number ensures that transactions from the same account execute in their submission order.

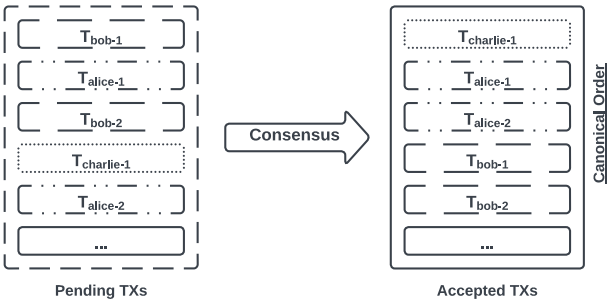


Fig. 1: XRP Ledger Canonical Order.

3) Transaction Order: All XRP Ledger servers have to execute transactions in the same order. They achieve this through a mechanism called *Canonical Order*. The Canonical Order is a set of rules that produce a deterministic, pseudo-randomly ordered list of transactions [8].

The shuffling algorithm works as follows. First, it derives a random salt from the set of transactions accepted by the consensus algorithm. Afterwards, for each transaction, the algorithm computes an *account key* by *XOR*-ing the account address of the transaction and the salt. Finally, the algorithm sorts the transaction by performing a pairwise comparison using three rules:

- **Rule 1:** Order transactions by their *account key*.
- **Rule 2:** Order transactions with an equal *account key* by their *sequence* number.
- **Rule 3:** Finally, order by transaction hash.

The result of the algorithm is a list of transactions where the transactions submitted by the same account are grouped by their submission order. However, the overall order of the transaction groups is pseudo-random (See Figure 1).

4) Tokens: Any user can issue their own digital assets or tokens. Users can freely transfer tokens to other users or trade them on the decentralized exchange. A token is identified by a currency code, for example, *USD*, *BTC*, *ETH*. A token is also associated with the issuing account address. Therefore, multiple tokens with the same currency code but a different issuing account may exist.

B. Automated Market Maker

The XRP Leder DEX combines the Automated Market Maker (AMM) and the Close Limit Order Book (CLOB), enabling traders to access liquidity on both AMM Pools and CLOBs by determining which provides the best exchange rate. AMMs and CLOBs differ in many ways, but mainly by how they determine the token price.

CLOBs utilize a limit order book for trading. Traders place limit orders to trade an asset at a specified or better price. The DEX maintains a list of buy and sell orders in the market and matches incoming trades. In other words, the traders determine the price of the trades, and the DEX attempts to fulfil them.

In contrast, AMM is a decentralized exchange that pools the liquidity of two tokens and makes it available to traders at an algorithmically determined price. In the rest of this section, we will outline some key aspects of the AMM.

1) Traders: A trader is an entity that performs swaps against the AMM Pool Instance. They exchange a token A for a token B or vice versa. Traders pay a fee for each swap they execute against the pool.

2) Liquidity Providers: Liquidity Providers (LPs) provide their assets to the pools for trading. In exchange, they receive *LPTokens* that they can use later to withdraw the liquidity provided plus collected fees.

3) Conservation Function: There are multiple types of protocols to determine the AMM price. However, they all use the same principle mathematical formula called *Conservation Function* to maintain a balance between the token amounts in the pool. Specifically, XRP Ledger AMM uses a weighted geometric mean market maker (GM3):

$$\mathcal{C} = Q_A^{W_A} \times Q_B^{W_B} \tag{1}$$

Where Q is the balance of tokens A and B in the pool, and W is the normalized weight of the token. XRP Ledger AMM uses implicit weights: $W_A = W_B = 0.5$. No matter how the asset amounts in the pool change, C must remain constant. Therefore, when a trade removes some amount of

asset A from the pool, they must put some amount of asset B to preserve $\mathcal{C}$.

4) Spot-Price: The *Spot-Price* is the weighted ratio of the pool instance balances:

$$SP_B^A = \frac{\frac{Q_B}{W_B}}{\frac{Q_A}{W_A}} \times \frac{1}{1 - TFee} \tag{2}$$

The *TFee* is the trading fee for trades executed against the AMM instance.

To exchange (or swap) one asset in the pool for another, we can't rely on the spot price to determine the price. For example, imagine a *XRP* and *TOK* Pool Instance containing 5 or each token. In our hypothetical pool, the constant $\mathcal{C} = 5$. It would be fair to assume that given that these assets are equal in value, one should be able to pay 1 XRP to receive 1 TOK. If the Pool Instance accepted this trade, it would contain 6 XRP and 4 TOK. However, from Equation 1 we can calculate that $6^{0.5} \times 4^{0.5} \approx 4.89$, which is not equal to the original constant $\mathcal{C} = 5$.

5) Swap Out: We introduce a *Swap Out* function to compute the amount of XRP one would have to pay to swap out 1 TOK from the pool.

$$\Delta_B = Q_B \left[\left(\frac{Q_A}{Q_A - \Delta_A} \right)^{\frac{W_A}{W_B}} - 1 \right] \frac{1}{1 - TFee} \tag{3}$$

The equation computes the minimum amount Δ_B of token B we have to put into the pool to receive Δ_A amount of token A.

6) Swap In: With the equation bellow, we can also calculate the maximum amount of Δ_A one would receive for paying Δ_B amount of token B:

$$\Delta_A = Q_A \times \left[1 - \left(\frac{Q_B}{Q_B + \Delta_B \times (1 - TFee)} \right)^{\frac{W_B}{W_A}} \right] \tag{4}$$

With Equation II-B5, we can calculate that in our hypothetical TOK/XRP pool, 1 TOK would cost 1.25 XRP. Or, using Equation 4, we can compute that 1 XRP would yield *at most* ≈ 0.834 TOK.

7) Slippage: Notice that when discussing swapping in or out, we used words *at least* and *at most*. To explain the reason behind this, we introduce one more concept: *slippage*.

Slippage refers to the difference between the expected price of a trade and the actual executed price. It is a fundamental property of AMMs. Slippage occurs due to natural movements in the market. For example, we observe that, at *ledger 1*, we can purchase 1 TOK for 1.25 XRP. However, we missed that some other user placed a trade swapping out 0.1 TOK for ≈ 0.1002 XRP, which executed in *ledger 2*. By the time our transaction executes in *ledger 3*, the cost of 1 TOK moved up to 1.3 XRP. We can calculate the slippage that occurred using the following equation:

$$\%slippage = 100 \times \left(\frac{Price_{actual}}{Price_{expected}} - 1 \right) \tag{5}$$

8) Slippage Tolerance: In our toy example, the slippage equates to $\approx 4\%$. The trader can incorporate *slippage tolerance* into their trade to mitigate slippage. In essence, *slippage tolerance* is the trader's appetite for risk expressed as a fraction of the trade size s, where $0 < s \leq 1$. For example, using Equation II-B5, we computed that for $\Delta_A = 1$ TOK, we have to pay at least $\Delta_B = 1.25$ XRP. However, we are very motivated to receive our token. Therefore, we are willing to pay up to 10% more and thus, our final $\Delta_B = 1.25 + 1.25 * 0.1 = 1.375$ XRP.

Low slippage tolerance may result in the trade failing due to natural market movement. However, a high slippage tolerance exposes the trader to a different type of risk: *frontrunning*.

9) Cross-Currency Payments: A crucial feature of XRP Ledger is allowing *Payment* transactions to deliver a different type of currency than is being sent. For example, assume that Alice wants to send Bob 5 USD and pay for it in XRP. The XRP Ledger Payment Engine will determine whether it is possible to fulfil the transfer by exchanging XRP to USD on the decentralized exchange [12].

C. Frontrunning on XRP Ledger

1) Types: Broadly, there are two categories of frontrunning attacks [13]: *destructive* and *tolerant*.

a) Destructive: A destructive frontrunner assumes (or calculates) that a victim's transaction is profitable. Thus, they copy the victim's transaction and submit it before the victim. If the frontrunner transaction executes first, the victim's transaction will fail, assuming the victim's transaction is executable only once due to market conditions.

b) Tolerant: On the other hand, a tolerant frontrunner exploits the price margin between the victim's trader and the optimal cost of the token. The attacker purchases the token before the victim at an optimal price and sells it back at an increased price. In this work, we only consider tolerant frontrunning.

2) Sources: We differentiate between two sources of frontrunning attacks: *miners* or, in the case of XRP Ledger *validators* and *users*.

a) Validator Frontrunning: One of the strengths of XRP Ledger is that it eliminates *validator* frontrunning. To discuss why, we first outline the XRP Ledger Consensus Protocol. The XRP Ledger Consensus Protocol [7] is a type of federated byzantine agreement (FBA) protocol [14]. Each validator declares a unique node list (UNL) of other validators it trusts. During the consensus process, the validator only considers the votes of the servers within the UNL. A ledger is accepted when at least 80% of UNL validators agree on its content. In other words, there is no single decision-maker in the consensus protocol. Thus, no one validator can propose an alternate transaction order that enables frontrunning.

One might argue that validators as a group could perform frontrunning attacks. While theoretically possible, at least

80% trusted validators would have to behave maliciously. However, this is highly unlikely as most validators use the same UNL[15]. The XRP Ledger Foundation verifies validator identities before including them in the trusted UNL. Therefore, the validators are under social pressure to behave correctly.

b) User Frontrunning: The Canonical Order creates a pseudo-random shuffle of transactions, where transactions sent by the same account are grouped and ordered by their sequence number. As a result, the frontrunner has a 50% chance of success. When the frontrunners' transactions execute before the victim, they win. Otherwise, they lose. For an in-depth analysis of the frontrunning probabilities on XRP Ledger, we refer the reader to Tumas *et al.*[10]. However, the advantages of the low transaction fees also bring a drawback. The frontrunners can employ multiple accounts to attack a victim, allowing them to increase their odds of success without incurring significant additional costs.

III. Methodology

This section describes the frontrunning attacks we found feasible on the new XRP Ledger AMM.

A. Preliminaries

1) Detecting Opportunities: The frontrunner requires a stream of pending transactions submitted by potential victims, existing offers on the Close Limit Order Book and the state of AMM Liquidity Pools. XRP Ledger provides a set of convenient WebSocket APIs [16] that allow anyone to subscribe to a stream of pending and accepted transactions. The API also provides a Close Limit Order Book and Liquidity Pool state. Previous works by Tumas *et al.* [10] and Peduzzi *et al.* [17] demonstrate that the WebSocket API is sufficiently fast to detect frontrunning and arbitrage opportunities in real-time. We can further improve the speed of pending transaction detection by running a highly connected XRP Ledger server. The high number of peers allows the server to quickly acquire new transactions.

2) Frontrunner Model: A frontrunner is an entity listening for incoming pending transactions to launch a frontrunning attack once it detects a profitable opportunity. We assume, the frontrunner is using $A = \{a_1, ..., a_n\}$ accounts to perform the attacks, where $n > 1$.

3) Fees: There are four costs associated with a frontrunning attack.

a) Transaction Fee.: Transaction fee is a small amount of XRP burnt to process the transaction. A fee is non-recoverable and is the only source of loss for the attacker.

b) Reserve.: A *Base Reserve* is an amount of XRP frozen by each account. The base reserve is 10 XRP [16]. These funds are released once the account is deleted. The *Owner Reserve* freezes 2 XRP for each object an account owns on the ledger. The relevant objects for the frontrunner are offers and trustlines. Thus, each attacker's account reserves $10 + 2 * offers + 2 * trustlines$.

c) Liquidity.: Liquidity represents the funds needed to perform an attack. It forms a significant portion of the attacker's capital.

d) AMM Fees.: The AMM charges each trade a small fee. These fees are accumulated in the pool and can be withdrawn by the Liquidity Providers.

B. Frontrunning Opportunities

Insertion attack, commonly known as the *sandwich*, is the most frequent type of frontrunning on Ethereum [3], [6]. In an insertion attack, a frontrunner F observes a profitable transaction from the victim T_V from a victim V and submits two transactions T_{F_1} and T_{F_2} to the network. The attacker sends the transactions in such a way that T_{F_1} executes before T_V and T_{F_2} after it. However, some insertion attacks require that T_{F_1} and T_{F_2} execute before T_V. Due to this difference, we will use the term *Sandwich attack* to identify frontrunning attacks that surround the victim, and *Insertion attack* to identify ones that do not.

C. Terminology

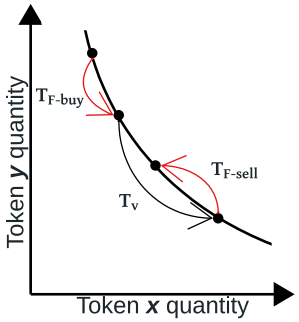
For the reader's convenience, we provide a list of terminology.

- V and F refer to the victim and the frontrunner, respectively.
- $P(Q_x, Q_y)$ refers to a AMM Pool Instance for the token pair (x, y), where Q is the amount of each token in the pool.
- $O(\Delta_x, \Delta_y)$ identifies an offer on the CLOB selling a number of Δ_x tokens for Δ_y tokens.
- Let $T(p, s, r, o, i)$ identify a transaction for the DEX. *p* is the transaction's position in the ledger, *s* and *r* are the sender and receiver account addresses. Finally, *o* is the amount of tokens *s* is buying or swapping out, and *i* is the amount of tokens *s* is willing to pay.
- $swap\_out(P(Q_x, Q_y), T_{\Delta_x}) = \Delta_{y-min}$ is the minimum cost of swapping out T_{Δ_x} tokens given the Pool Instance $P(Q_x, Q_y)$. Swap Out is calculated by applying Formula II-B5.
- $spot\_price(P(Q_x, Q_y))$ is the Spot-Price of the AMM Pool Instance $P(Q_x, Q_y)$, calculated using Formula 2.
- $exchange\_rate(O(Q_x, Q_y))$ is the Exchange Rate of a CLOB offer $O(Q_x, Q_y)$, it is the ratio of the token amounts of the offer.

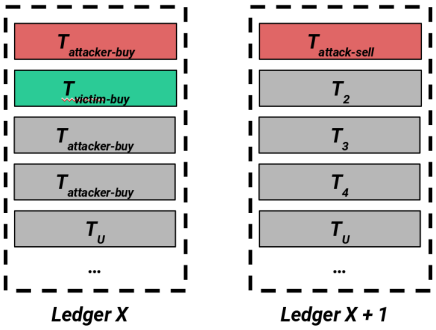
D. AMM Sandwich

A well-known sandwich attack arbitrages the AMM Liquidity Pool, which takes advantage of the fact that an incoming trade will shift the price of the AMM Liquidity Pool. The attacker observes an incoming trade with a profitable *slippage tolerance*, sends a *buy* transaction before the trade and a *sell* transaction after it.

For example, assume an AMM Pool Instance containing 5 TOK and 5 XRP. In this pool, the minimum cost of swapping out 2 TOK is $\approx$ 3.34 XRP. A trader places a trade attempting to buy 2 TOK for up to 4 XRP (See Figure 2a). This poses an opportunity for the frontrunner to capitalize on some of the slippage tolerance. The frontrunner purchases just enough TOK from the AMM Pool Instance to not cause the victim's



(a) Sandwich attack effect on the AMM pools



(b) Frontrunner transactions

Fig. 2: AMM Sandwich Attack

transaction to fail. Then, he sells the TOK after the victim's transaction is processed, capitalizing on the increased asset price.

Let's assume an AMM Liquidity Pool Instance $P(Q_x, Q_y)$ and the victim's transaction T_V. The transaction is worth frontrunning when $T_V(i) - swap\_out(P(Q_x, Q_y), T_V(o)) > bn$, where b is the base transaction fee and n is the number of accounts the frontrunner is using for the attack. In other words, when the difference between the amount the victim is paying and the optimal price is larger than the cost of the frontrunning attack with n accounts.

1) Detecting AMM Sandwich: Torres *et al.* [3] provide heuristics for detecting sandwich attacks on the Ethereum network. However, they do not directly apply to the XRP Ledger. We, therefore, provide a set of rules to identify Sandwich attacks on the XRP Ledger.

The cross-currency payment feature (See Section II-B9) allows the frontrunner, with one transaction, to purchase tokens from the pool and send them to the selling account. Let $\mathcal{T}_{F1} = \{\forall (T_{F_a}, T_{F_b}) \in \mathcal{T} : T_a(p,s) \neq T_b(p,s) \wedge T_a(r,o,i) = T_b(r,o,i)\}$, $|\mathcal{T}_{F1}| = n-1$. It is a set of frontrunner's purchase transactions submitted with $n-1$ accounts, where each transaction buys an identical amount of tokens for the same price and sends the purchased tokens to the seller account r. Let T_{F2} denote the frontrunner's sell transaction, where $\forall T_F \in \mathcal{T} : T_F(r) = T_{F2}(s)$. Let T_V denote the victim's transaction.

We detect sandwich attacks by examining any two adjacent ledgers (L_n, L_{n+1}), and examining whether $\mathcal{T}_{F1}$, T_V and T_{F2} transactions exist for which the following rules hold:

- **Rule 1:** Transactions $\mathcal{T}_{F1}$, T_V and T_{F2} must operate with the same token pair (x, y).
- **Rule 2:** The receiver $\mathcal{T}_{F1}(r)$ is equal to the sender $T_{F2}(s)$.
- **Rule 3:** The transaction set is $\mathcal{T}_{F1} \in L_n$ and the sell transaction $T_{F2} \in L_{n+1}$. In other words, for the attack to be successful, it must span at least two blocks, as illustrated in Figure 2b. The sell transaction must occur in a separate ledger to ensure it executes after the victim. Otherwise, if $T_{F2} \in L_n$, the Canonical Order may order T_{F2} before T_V, which would cause the frontrunning attack to fail.
- **Rule 4:** The token amount bought by $\mathcal{T}_{F1}$ is approximate to the number of tokens sold by T_{F2}. Practically, this should be set with a tolerance threshold to avoid false positives.
- **Rule 5:** $\exists T_F \in \mathcal{T}_{F1} : T_F(p) < T_V(p)$. In other words, at least one attacker's transaction executes before the victim. As long as at least one T_{F1} executes successfully, all other $T_{F1(n-2)}$ will fail.
- **Rule 6:** Let $T_{F1} - win$ denote the frontunners account positioned before the victim. The victim receives $T_V(o)$ from the AMM and sends $T_V(i)$ tokens to the AMM. In addition, $T_{F1} - win(s)$ sends $T_{F1} - win(i)$ tokens to $T_{F2}(s)$ at the cost of $T_{F1} - win(o)$ tokens. Finally, $T_{F2}(i) > T_{F1} - win(o)$.

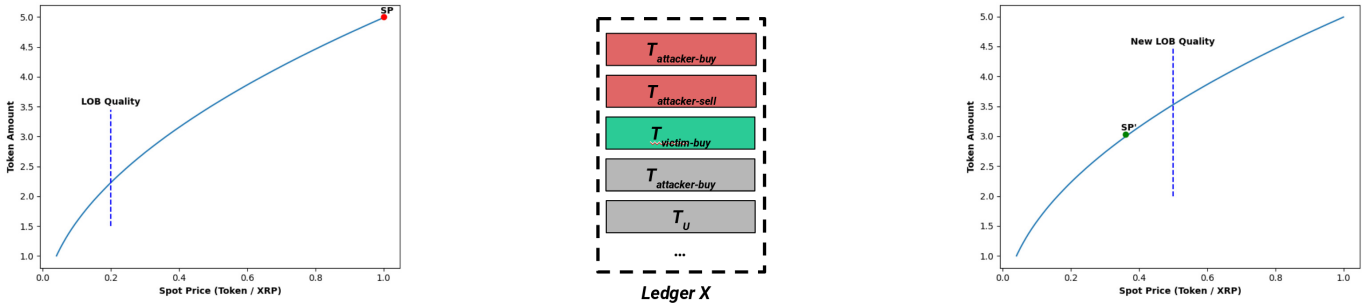
Note that if **Rules 1** through **4** hold, but **Rules 5 & 6** do not, the heuristic detects an attempted, failed AMM Sandwich attack.

E. AMM/CLOB Insertion

The interaction between the new Automated Market Maker and the existing Close Limit Order Book provides frontrunners with a new attack vector.

Similarly to the previous example, assume an AMM Pool Instance containing 5 TOK and 5 XRP and a CLOB sell offer of 10 XRP for 2 TOK. The spot price of the AMM is 1.0 whereas the quality of the CLOB is 0.2 (see Figure 3a).

A trader places an offer to buy 2 TOK for 4 XRP. Without interference from the frontrunner, the purchase would execute against the AMM because it offers a better price. However, the frontrunner can capitalize on the entire victims *slippage tolerance* by forcing them to purchase from the CLOB. The frontrunner purchases the 2 TOK the victim is attempting to buy but at an optimal price. When the purchase executes, it will drive down the AMM spot price. Secondly, the frontrunner to create a sell offer on the CLOB, selling the 2 TOK for 4 XRP, increasing the CLOB quality above the AMM (see Figure 3c). In case the frontrunner transactions execute before



(a) Initial LOB & AMM states (b) Frontrunner transactions (c) Modified AMM & LOB states

Fig. 3: AMM/LOB attack breakdown

the victim, the victim buy offer will execute against the CLOB, as it provides a better exchange rate.

Let's assume an AMM Liquidity Pool Instance $P(Q_x, Q_y)$ and a sell offer on the CLOB $O(\Delta_x, \Delta_y)$, such that the $spot\_price(P(Q_x, Q_y)) > exchange\_rate(O(\Delta_x, \Delta_y))$ (See Figure 3a). Finally, assume victim transaction T_V.

Victim's transaction is worth frontrunning when the $exchange\_rate(O_F(\Delta_x, \Delta_y)) > spot\_price(P'(Q_x, Q_y))$. Where, $O_F(\Delta_x, \Delta_y)$ is the frontrunner's offer on the CLOB such that $O_F(\Delta_x) = T_V(o)$ and $O_F(\Delta_y) = T_V(i)$. The $P'(Q_x, Q_y)$ is the posterior AMM Pool State after executing T_F such that $T_F(o) = T_V(o)$ and $T_F(i) = swap\_out(P(Q_x, Q_y), T_V(o))$.

1) Detecting AMM/CLOB Insertion: We introduce T_{CLOB}, a transaction that crosses the CLOB. The attacker creates a set of transactions $\mathcal{T}_F = \{(T_{F-buy-1}, T_{F-CLOB-1}), ..., (T_{F-buy-n}, T_{F-CLOB-n})\}$, $|\mathcal{T}_F| = n$. The T_{F-buy} purchases tokens from the AMM, and T_{F-CLOB} creates the offer on the CLOB. We use $\mathcal{T}_{F-buy}$ to refer to all purchase transactions and $\mathcal{T}_{F-CLOB}$) to refer to all CLOB transactions of the frontrunner. We detect a hybrid attack by examining transactions of each ledger that hold the following heuristics:

- **Rule 1:** Transactions $\mathcal{T}_F$, T_V must operate with same token pair (x, y).
- **Rule 2:** There must be at least one frontrunner transaction pair that occurs before the victim. $\exists(T_{F-buy}, T_{F-CLOB}) \in \mathcal{T}_F : T_{F-buy}(p) < T_{F-CLOB}(p) < T_V(p)$ (See Figure 3b).
- **Rule 3:** The token amount bought by $\mathcal{T}_F(T_{F-buy})$ is similar to the number of tokens sold by $\mathcal{T}_F(T_{F-CLOB})$.
- **Rule 4:** Transaction T_V executes against the CLOB, and consumes $O_F(\Delta_x, \Delta_y)$, where $O_F(\Delta_x) = \mathcal{T}_{F-CLOB}(o)$ and $O_F(\Delta_y) = \mathcal{T}_{F-CLOB}(i)$.

IV. Evaluation

In this section, we first describe the evaluation methodology and afterwards discuss the results. Frontrunning is an unethical activity. Therefore, we conducted the experiments on the *AMM-Devnet* parallel network specifically created to test the AMM functionality.

A. Methodology

1) Goals: We designed the evaluation with two goals in mind:

- Determine whether the proposed attacks are technically feasible on the XRP Ledger AMM.
- Frontrunning on XRP Ledger is probabilistic [10]. Therefore, we aim to quantify the success probability of these attacks.

2) Setup: To evaluate the attacks, we implemented a frontrunning bot. However, we believe that publically disclosing such a system would pave an easy path for frontrunners to the XRP Ledger. Therefore, we keep it close source. Though, briefly the system works as follows:

a) Initial State: The bot creates a zero-fee AMM Pool for 5 TOK and 5 XRP. TOK is a custom token we issued to not interfere with other users. Increasing the fee would reduce the frontrunner's profits but not affect the success probability. The bot issues a victim transaction attempting to purchase 2 TOK for 4 XRP for both attack types. In the case of the AMM/CLOB Insertion attack, the bot also creates a CLOB offer selling 2 TOK for 10 XRP.

b) Attacker Transactions: The bot computes the frontrunner transactions based on the specific attack type. In the case of the AMM Sandwich, the bot calculates the maximum amount of tokens to purchase without causing the victim's transaction to fail. In the case of the AMM/CLOB Insertion, the bot clones the victim's transaction and creates a sell offer that is guaranteed to appear on the CLOB.

c) Frontrunning: The bot batches attacker and victim transactions and sends them to the *AMM-Devnet*. The batching ensures that all transactions occur in the same ledger. We did not implement a system for acquiring pending transactions in real-time. However, such a system is trivial [10], [17] to implement using the XRP Ledger Subscribe API. Finally, the system determines whether the attack was successful by checking if the attacker's account balances increased.

3) Statistical Significance: Frontrunning on XRP Ledger is probabilistic. Therefore, we used a Monte Carlo simulator to run the bot.

As discussed in Section II, the Canonical Order combines the transaction account address with a random seed using

an XOR operation. Therefore, to add more variance to the ledger data at each iteration, the simulator picked 3 frontrunner accounts and 1 victim account from a pool of 1,000 pre-funded accounts.

4) Results: The outcomes of the experiments indicate the following success probabilities for the attacks:

- AMM Sandwich Attack: Approximately 73% success probability.
- AMM/LOB Insertion Attack: Approximately 79% success probability.

It is important to note that the *AMM-Devnet* has a low transaction volume. Therefore, our transactions dominated the ledgers, potentially skewing the probabilities. However, despite this, the results show that the ever-present AMM Sandwich attack is feasible on the XRP Ledger AMM. Furthermore, the AMM and the CLOB integration creates an opportunity for a new type of frontrunning attack.

V. Discussion

The XRP Ledger Canonical Order guarantees that the transaction execution order, in practical terms, is impossible to influence predictably. Due to the pseudo-random nature of the shuffle, an attacker with a single account has a 50% probability of a successful attack. However, they can use multiple accounts to increase their odds. Our results show that this principle applies to the new XRP Ledger AMM as well.

A. Low delay

The XRP Ledger processes a new ledger every ≈ 4 seconds. Significantly faster than most other blockchains. The fast confirmation has a two-fold effect on frontrunning. First, the attacker has little time to observe pending transactions and issue their own. However, on the other hand, fast confirmation may increase the amount of slippage users experience. On Ethereum, the AMM price change between two blocks is minimal [18]. However, the duration between two blocks is around 12 seconds, giving arbitrageurs more time to insert transactions that re-balance the AMM Pool. Furthermore, the arbitrageurs can affect where in the block their transaction executes.

In contrast, XRP Ledger produces 3 to 4 ledgers in the same time span. Each produced ledger may cause further AMM price disbalance. Furthermore, arbitrageurs cannot influence transaction execution orders. Thus, they must submit the re-balancing transactions in a subsequent ledger.

We postulate that the fast transaction confirmation may cause a higher price fluctuation than on Ethereum. As a result, traders would have to incorporate higher slippage tolerance to account for the fluctuation. The higher slippage tolerance would provide more lucrative frontrunning opportunities.

B. Profitability

Transaction costs on XRP Ledger are low. A standard transaction costs 0.00001 XRP or equivalent of 0.000006 USD. Furthermore, XRP Ledger does not have an incentive system. Thus, the fees are burnt. In contrast, the average transaction cost on Ethereum is $\approx 6.83\$$. These fees go to the miners as part of the incentive system. A 2022 report on Ethereum frontrunning shows that the revenue of Sandwich attacks on Ethereum was $126M\$$ [6]. However, the frontrunners paid $101M\$$ in transaction costs. The frontrunners paid nearly 79% of their revenue to block builders and validators. Thus, we summarize that when the XRP Ledger AMM popularity reaches and surpasses that of Ethereum AMMs, frontrunning on XRP Ledger will be more profitable.

C. Mitigation

The following blockchain properties enable frontrunning:

- The content of submitted transactions is public. Furthermore, due to the necessity of a consensus, there is a delay window between the transaction being submitted and finalized in the ledger. Therefore, adversaries can act based on the content of the yet-to-be-executed transactions.
- In blockchains, transaction execution order is important. Traditionally, this order is on transaction fees, with higher fees granting priority. However, the XRP Ledger takes a distinctive approach. Transactions are processed in a pseudo-random order, making frontrunning probabilistic. Nonetheless, due to the low transaction fees on the XRP Ledger, frontrunners can use multiple accounts to increase their odds of success.

Removing one of these properties would prevent or make frontrunning significantly more difficult. In the remainder of this section, we propose mitigation strategies that address these limitations. However, these changes require significant modifications to the XRP Ledger.

1) Hiding Transaction Content: One solution to prevent frontrunning is to hide transaction content with cryptographic primitives, such as Threshold Encryption. In this approach, the user encrypts their transaction using a global public key. A group of decryptors hold shares of the corresponding private key. Once the consensus finalizes a ledger of encrypted transactions, the decryptors collectively decrypt the transactions. Threshold encryption is a natural fit for the XRP Ledger, as the consensus protocol is quorum-based. Therefore, the validators can act as decryptors.

While threshold encryption solves the frontrunning problem, it increases the time between the submission and finalization of the transaction. Intuitively, the delay comes from an additional communication round between the decryptors to decrypt the transaction. Therefore, a threshold encryption scheme may increase the amount of slippage traders experience.

2) Fair Transaction Order: The second family of frontrunning mitigation strategies prevent frontrunning by ordering transactions in a ledger fairly.

A transaction order is *fair* when no one can include or exclude transactions from a ledger after seeing their content. In addition, no party can insert additional transactions before any transaction whose contents has already been observed [19].

Committee-based fair-order approaches have been well studied [20], [21], [22]. At a high level, they rely on a committee of nodes observing incoming transactions and agreeing on

a fair order through a consensus. For example, given two transactions *tx* and *tx'* sent by users, the receive order fairness ensures that *tx* will be finalized before *tx'* if a majority of consensus nodes received *tx* before *tx'*.

Similarly to threshold encryption, a committee-based approach is a natural fit for the XRP Ledger due to its unique consensus protocol. Unfortunately, the same drawbacks apply. The consensus would have to agree on an additional property: transaction execution order, slowing down the consensus protocol. Furthermore, implementing such an approach would require modifying the existing battle-tested XRP Ledger Consensus Protocol. Therefore, practically, the benefits of such an approach must significantly outweigh the risks.

VI. Related Work

Tumas *et al.*[10] are the first to analyse the possibility of frontrunning on XRP Ledger. They provide two strategies an attacker may use to increase the probability of success. The authors also show the existence of frontrunning on XRP Ledger. We expand on their work by conducting frontrunning analysis on the XRP Ledger AMM.

Frontrunning is a well-known issue on Ethereum. Eskandari *et al.*[23] created the first taxonomy of frontrunning attacks. Daian *et al.* [4] demonstrate evidence of frontrunning from monitoring Ethereum's mempool of pending transactions. Torres *et al.*[3] provided the first lower bound on the frontrunner profits by measuring different frontrunning attacks in Ethereum's transaction history.

Several works propose frontrunning mitigation strategies based on varying techniques. Most recently, Constantinescuẽtal[24] proposed a Layer-2 consensus protocol to determine the transaction order. A group of synchronised clock nodes run a consensus protocol that provides each transaction with a receive timestamp. The Layer-1 consensus protocol then uses the timestamps to order the transactions. A work by Misra *et al.*[25] provides a causal-ordering preserving protocol in a synchronous blockchain. Although these works effectively address the frontrunning problem, they make synchrony assumptions that may not be feasible in existing blockchain systems.

Fair transaction order is a prevalent topic [26], [20], [22], [21]. However, these techniques are designed for private blockchains and may not work on a public blockchain like XRP Ledger.

Other solutions prevent frontrunning with advanced commit-and-reveal scheme [27], tighter slippage protection [18], or new more frontrunning-resistant decentralized exchanges [28], [29], [30], [31], [32], [33]. These solutions slow down the blockchain, introduce higher fees or do not necessarily protect against every type of frontrunning attack.

VII. Conclusion

In this paper, we conduct a pre-emptive analysis of frontrunning on the upcoming XRP Ledger AMM. We show that some of the frontrunning techniques applicable to Ethereum also work well on XRP Ledger AMM. Furthermore, we discovered a new AMM/CLOB Insertion attack that leverages interaction between AMM and the CLOB. Finally, we validate these attacks on the XRP Ledger *AMM-Devnet*.

References

[1] C. A. Makridis *et al.*, "The rise of decentralized cryptocurrency exchanges: Evaluating the role of airdrops and governance tokens," *Journal of Corporate Finance*, vol. 79, p. 102358, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092911992300007X

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, 2014.

[3] C. F. Torres *et al.*, "Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain." USENIX Association.

[4] P. Daian *et al.*, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 910–927, 2020.

[5] L. Zhou *et al.*, "High-frequency trading on decentralized on-chain exchanges," *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 428–445, 2020.

[6] EigenPhi, "Mev outlook 2023: Walking through the dark forest," Feb 2023. [Online]. Available: https://eigenphi.substack.com/p/mev-outlook-2023

[7] D. Schwartz *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, 2014.

[8] "rippled - CanonicalTXSet," Oct. 2022, [Online; accessed 10. Oct. 2022]. [Online]. Available: https://github.com/XRPLF/rippled/blob/fe05b8c4feb815e1aa6afd7ff8c85dbc92bcd651/src/ripple/ app/misc/-CanonicalTXSet.cppL25

[9] J. Piet *et al.*, "Extracting godl [sic] from the salt mines: Ethereum miners extracting value," *arXiv:2203.15930 [cs]*, Mar 2022, arXiv: 2203.15930. [Online]. Available: http://arxiv.org/abs/2203.15930

[10] V. Tumas *et al.*, "A ripple for change: Analysis of frontrunning in the xrp ledger," *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–9, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259835397

[11] A. Malhotra and D. J. Schwartz, "0030 xls 30d: Automated market maker on xrpl," Aug 2023. [Online]. Available: https://github.com/XRPLF/XRPL-Standards/discussions/78

[12] G. Tsipenyuk, "Payment engine system design overview - ripple." [Online]. Available: https://ripple.com/reports/Payment-Engine-System-Design.pdf

[13] K. Qin *et al.*, "Quantifying blockchain extractable value: How dark is the forest?" 2021.

[14] G. A. F. Rebello *et al.*, "Security and performance analysis of quorum-based blockchain consensus protocols," *Electrical Engineering Program, COPPE/UFRJ, Tech. Rep*, 2020.

[15] "UNL – XRP Ledger Foundation," Nov. 2023, [Online; accessed 30. Nov. 2023]. [Online]. Available: https://foundation.xrpl.org/unl

[16] X. L. Foundation, "XRPL.org," 2023, [Online; accessed 30. Nov. 2023]. [Online]. Available: https://xrpl.org

[17] G. Peduzzi *et al.*, "Jack the rippler: Arbitrage on the decentralized exchange of the xrp ledger," *3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2021.

[18] L. Heimbach *et al.*, "Eliminating sandwich attacks with the help of game theory," in *ASIA CCS '22: ACM Asia Conference on Computer and Communications Security, Nagasaki, Japan, 30 May 2022 - 3 June 2022*, Y. Suga, K. Sakurai, X. Ding, and K. Sako, Eds. ACM, 2022, pp. 153–167.

[19] ——, "Sok: Preventing transaction reordering manipulations in decentralized finance," 2022. [Online]. Available: http://dx.doi.org/10.1145/3558535.3559784

[20] M. Kelkar *et al.*, "Order-fairness for byzantine consensus," in *40th Annual International Cryptology Conference, CRYPTO*, 2020.

[21] ——, "Themis: Fast, strong order-fairness in byzantine consensus," *IACR Cryptol. ePrint Arch.*, 2021.

[22] K. Kursawe, "Wendy, the good little fairness widget: Achieving order fairness for blockchains," in *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*. ACM, 2020, pp. 25–36.

[23] S. Eskandari *et al.*, "Sok: Transparent dishonesty: Front-running attacks on blockchain," *Econometrics: Computer Programs & Software eJournal*, 2019.

[24] A. Constantinescu *et al.*, "A fair and resilient decentralized clock network for transaction ordering," 2023.
[25] A. Misra *et al.*, "Towards stronger blockchains: Security against front-running attacks," 2023.
[26] Solana, "Proof of History Explained by a Water Clock," Jun. 2018, [Online; accessed 07. oct. 2022]. [Online]. Available: https://medium.com/solana-labs/proof-of-history-explained-by-a-water-clock-e682183417b8
[27] A. Capponi *et al.*, "The evolution of blockchain: from lit to dark," *arXiv preprint arXiv:2202.05779*, 2022.
[28] C. Baum, B. David, and T. K. Frederiksen, "P2DEX: privacy-preserving decentralized cryptocurrency exchange," in *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, K. Sako and N. O. Tippenhauer, Eds., vol. 12726. Springer, 2021, pp. 163–194.
[29] M. Ciampi *et al.*, "Fairmm: A fast and frontrunning-resistant crypto market-maker," in *Cyber Security, Cryptology, and Machine Learning - 6th International Symposium, CSCML*, 2022.
[30] C. McMenamin *et al.*, "Fairtradex: A decentralised exchange preventing value extraction," *IACR Cryptol. ePrint Arch.*, p. 155, 2022. [Online]. Available: https://eprint.iacr.org/2022/155
[31] L. Zhou *et al.*, "A2MM: mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges," *CoRR*, vol. abs/2106.07371, 2021. [Online]. Available: https://arxiv.org/abs/2106.07371
[32] E. Sariboz *et al.*, "FIRST: frontrunning resilient smart contracts," *CoRR*, vol. abs/2204.00955, 2022.
[33] "CoW Protocol - Batch Auctions," Oct. 2022, [Online; accessed 30. Nov. 2023]. [Online]. Available: https://docs.cow.fi/overview/batch-auctions