A Ripple for Change: Analysis of Frontrunning in the XRP Ledger

Vytautas Tumas
University of Luxembourg
Luxembourg, Luxembourg
vytautas.tumas@uni.lu

Beltran Borja Fiz Pontiveros
University of Luxembourg
Luxembourg, Luxembourg
beltran.fiz@uni.lu

Christof Ferreira Torres
ETH Zurich
Zurich, Switzerland
christof.torres@inf.ethz.ch

Radu State
University of Luxembourg
Luxembourg, Luxembourg
radu.state@uni.lu

2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) | 979-8-3503-1019-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICBC56567.2023.10174999

***Abstract*—Blockchains are disrupting traditional finance by reducing the number of intermediaries and providing transparency. Blockchains, however, come with their own set of prominent issues. One such challenge is frontrunning. Attackers try to influence the transaction order so that their transaction executes before their victims' transaction. While frontrunning is a well-studied topic on Ethereum, it is unknown whether other blockchains are also susceptible to such attacks.**

One proposed defence strategy against frontrunning attacks is to randomize the transaction execution order. XRP Ledger is the highest-value blockchain to use such a strategy. Furthermore, it runs a Decentralized Exchange, which provides ample frontrunning opportunities. Therefore, in the context of XRP Ledger, we examine whether randomized transaction order provides sufficient protection against frontrunning.

Our results show that the mechanism embedded in the XRP Ledger protocol is insufficient to prevent these attacks. We showcase two strategies to perform frontrunning attacks. The first, "naive" strategy, uses randomly generated accounts, whereas the second uses carefully selected accounts to improve the attack's success. Based on our analysis of the XRP Ledgers' historical data, we estimate that attackers could generate up to approx. 1.4M USD profit over two months, provided they succeeded to frontrun every opportunity.

***Index Terms*—Frontrunning, XRP Ledger, Blockchain, Security**

I. Introduction

Blockchain technology is disrupting traditional finance with the rise of decentralized finance (DeFi). A core part of the DeFi ecosystem are so-called Decentralized Exchanges (or "DEXes"). A DEX is a peer-to-peer marketplace where users can exchange assets in a non-custodial way. In contrast to centralized exchanges, DEXes offer transparency and remove intermediaries such as banks, brokers, payment processors, or other institutions, thereby reducing costs. Despite their benefits, DEXes have a severe drawback: pending blockchain transactions are public. Adversaries can *frontrun* pending orders by observing them and placing their competing orders.

For example, on Ethereum [1], the transaction execution order is based on the transaction fee, from high to low. Therefore, adversaries can influence where their transactions occur in a block by changing the transaction fee. As a result, many users of Ethereum DEXes become victims of these attacks. Frontrunning is a well-studied issue on Ethereum[2], [3], [4]. However, to our knowledge, these attacks are yet to be examined on other blockchains.

XRP Ledger [5], 7th by market capitalization, is one of the oldest cryptocurrencies. Its unique consensus protocol allows the blockchain to process up to 1,500 transactions per second, with a settlement time of around 3 to 5 seconds.

XRP Ledger users were victims to frontrunning in the past [6]. Similarly to other blockchains, the pending transactions on XRP Ledger are public. Adversaries could generate low transaction IDs to ensure their adversarial orders are applied first. As a response, the XRP Ledger developers introduced a new transaction ordering strategy to make the transaction order difficult to predict [7]:

> *"The order transactions execute within a ledger is designed to be unpredictable, to discourage frontrunning."*

The new transaction ordering strategy, similar to suggestions in previous works [8], creates a pseudo-random shuffle of the to-be-executed transactions. As a result, an attacker using a single account has a 50% probability of a successful attack.

A naive attacker may use multiple randomly generated accounts to increase their likelihood of success. However, we show that the winning probability does not increase as expected when using multiple accounts.

Therefore, we propose a more advanced strategy that leverages carefully crafted attacker accounts to increase the effectiveness of frontrunning attacks. Using historical mainnet data, we simulate how often a frontrunning attack would have been successful on the XRP Ledger and measure the profitability of mounting such an attack. Furthermore, we inspect the mainnet data for existing attacks.

Contributions. We summarize the contributions presented in this paper as follows:

- We present an advanced attack strategy that leverages selected accounts, which weakens the resistance of the XRP Ledger to frontrunning attacks.
- We evaluate two frontrunning attack strategies on the XRP Ledger. We show that with *five* accounts, an attacker would have a success rate of 80% and 96% when using random accounts and carefully crafted ones, respectively.
- We analyze historical mainnet data and estimate an upper bound of approx. 1.4M USD profit attackers could have

made when frontrunning every opportunity on the XRP Ledger over a two-month period.

We organize the remainder of this paper as follows. In Section II, we provide background information on frontrunning attacks and the XRP Ledger. In Section III, we describe the attack strategies and evaluate them in Section IV. In Section V, we discuss our results, and in Section VI, we highlight related work. Finally, we conclude our paper in Section VII.

II. Background

In traditional markets, frontrunning is the use of insider information on a future deal that is about to happen in order to place a competing order right before to reap benefits at its expense. It is a well-studied issue in centralized exchanges [9]. Recently, frontrunning started to take place on decentralized exchanges (DEXes). Given the public nature of the transactional data, these attacks are common and pose a serious concern across multiple DEXes [2], [3], [4].

A frontrunning attack is successful only when the attacker's transactions execute before the victim's transaction. For example, on Ethereum, miners process transactions based on their fee, starting from the highest. Therefore, an attacker can ensure their transaction executes before the victim by paying a greater fee than the victim.

In contrast, XRP Ledger uses a pseudo-random shuffling algorithm to determine the execution order. Therefore an attacker cannot guarantee that their transaction will occur before the victim's transaction. However, as shown later in Section III, an attacker can employ other mechanisms to increase their chance of victory. In the remainder of this section we provide the background on XRP Ledger and discuss its transaction ordering mechanism.

A. XRP Ledger

The XRP Ledger is a blockchain running on a distributed peer-to-peer network. XRP is the native currency of the ledger. At its core, XRP Ledger is a public, distributed database that maintains records of accounts, balances, offers, and traded assets. Users submit cryptographically signed instructions, called transactions, which trigger the state of the database to change. The users of XRP Ledger identify themselves via cryptographic identities derived from a pair of public/private keys. Cryptographic signatures matching these identities authorize the execution of transactions. Every server in the network processes transactions based on known, deterministic rules. The goal is for every server in the network to have an identical ledger state.

The XRP Ledger uses the XRPL Consensus Protocol [5], which is a variant of the federated byzantine agreement (FBA) protocol [10]. The principle behind FBA is to have multiple trusted parties, called validators on XRP Ledger, voting to agree on which transactions to include in the ledger. Each participant selects their trusted validators and only works with them to advance the ledger history.

1) XRPL's Decentralized Exchange: A decentralized exchange (DEX) is a peer-to-peer marketplace where users can exchange assets in a non-custodial way. In contrast to centralized exchanges, DEXes offer transparency and remove intermediaries such as banks, brokers, payment processors, or other institutions, thereby reducing costs.

XRP Ledger users can represent any asset, other than XRP, as a *token*. The users trade their tokens on the XRPLs native order book decentralized exchange. In order book-based DEXes, traders determine the price of their trade orders and place them in real-time. Trade orders, called *offers*, represent a limit order to buy a specific amount of one token (or XRP) in exchange for another. The network will execute the user's trade if there are matching offers. Offers are consumed by starting with the best exchange rate first. An offer becomes an object in the ledger for the remaining amount if it is not filled. Later, other offers can match and consume it. Due to this, offers can execute at a better rate than their requested exchange rate or at exactly their stated exchange rate later on.

a) Accounts: An account in XRPL corresponds to a holder of the XRP native currency and a sender of transactions. A unique *address* and a *sequence* number identify each account. The former is a 20-byte address in Base58 format derived from the account's public key. The latter is a 32-bit unsigned integer that ensures that transactions execute only once and in the correct order.

To create an account, an existing account on the ledger has to send a *Payment* transaction to a valid account address that does not have an account. This is called *funding* an account. The account address is derived from its public key. Thus, the user can generate as many keys as they desire offline until they find an account address with desirable properties.

b) Transactions: A Transaction is the only way to update the state of the XRP Ledger. A unique *hash* identifies each transaction. In addition, a transaction has a *sequence* number and the *address* of the sending account. A transaction is valid only if its *sequence* number is exactly *1* greater than the previous transaction from the same account. The sequence number ensures that transactions submitted by the same account execute in the order of submission.

2) Transaction Order: Consensus requires that validators execute accepted transactions in the same order to reach the same result. XRP Ledger uses a shuffle algorithm to create a deterministic, pseudo-randomly ordered list of transactions [11]. The algorithm uses a random salt derived from the accepted transactions to ensure that all participants compute the same transaction order. In the remainder of this section, we describe the shuffling algorithm and its properties.

Each node stores the accepted transactions in a Merkle Tree. A Merkle Tree is a tree data structure in which the label of each leaf is a cryptographic hash of the data block, and the label of the inner nodes is the hash of its children's labels. The hash of the root node is the source of randomness for the transaction shuffling. The XRP Ledger uses *SHA512-Half* as the hashing function. The work by Kaminsky et. al. [12] shows

that *SHA512-Half* hashes can be used as a decent source of entropy for pseudo-randomness.

The shuffling algorithm works as follows. First, for each transaction, it calculates an *account key* by combining the account ID for that transaction and the salt using the *XOR* operation. The algorithm sorts the transactions by a pairwise comparison using the following rules:

- **Rule 1:** If *account keys* are not equal, return ascending order. Otherwise, follow *Rule 2*.
- **Rule 2:** If transaction sequence numbers are not equal, order them in ascending order. Otherwise, follow *Rule 3*.
- **Rule 3:** Order by transaction hash.

The shuffling algorithm produces a pseudo-randomly shuffled list of transactions, where transactions of a single account occur one after another in their submission order.

XRP Ledger nodes process transactions one at a time. If a transaction fails, it still appears in the ledger, but with a *tec*-class result code, at the end of the canonical list. Once all transactions are processed, failed transactions are retried. However, they remain at the end of the canonical list.

3) Order Analysis: The address of an account is 160 bits long, whereas the hash of the Merkle Tree root is 256 bits. During the shuffling, the address of an account occupies 256 bits, but the first 96 most significant bits are all zeros. Therefore, the first 96 bits of each *account key* are equal, and only the remaining 160 bits effectively determine the transaction order. In the remainder of our work, we assume the starting position is the 160th bit.

Further, we assume that the account addresses and the salts come from a uniform distribution (i.e. each value has an equal probability of occurring). Thus, each bit can have a value of 0 or 1, with an equal likelihood.

Let A and V denote the attacker's and victim's account addresses and S the salt. The attacker wins when

$$A \oplus S < V \oplus S \tag{1}$$

Let A_i denote the value of the *i-th* most significant bit, $A_i \in [0, 1]$. We can say that Rule 1 is true, when

$$A_i \oplus S_i < V_i \oplus S_i \tag{2}$$

We can further deduce that Rule 2 is true if and only if $A_i \neq V_i$, and $A_i \oplus S_i = 0$, and $V_i \oplus S_i = 1$. We know that, $A_i \oplus S_i = 0 \iff A_i = S_i$, and $V_i \oplus S_i = 1 \iff V_i \neq S_i$.

With the information above, we can express the probability of A winning as:

$$\begin{aligned} P(A_{win}) &= P(A_i = S_i \wedge V_i \neq S_i) \wedge P(A_i \neq V_i) \\ &= \frac{1}{4} + \left(\frac{1}{4} \cdot \frac{1}{2^1}\right) + \left(\frac{1}{4} \cdot \frac{1}{2^2}\right) + ... + \left(\frac{1}{4} \cdot \frac{1}{2^n}\right) \\ &= \frac{1}{2} \sum_{n=1}^{160} \frac{1}{2}^n = \frac{1}{2} - \frac{1}{2}^{160} \approx 0.5 \end{aligned} \tag{3}$$

Let's assume the attacker uses two accounts, A and B, to increase their odds of winning. There will be pairs of salt and victim address, for which A and B will both win, and pairs for which both will lose. The probability of winning with either A or B is

$$P(A_{win}) \vee P(B_{win}) = P(A_{win}) + P(A_{lose} \wedge P(B_{win})) \tag{4}$$

Therefore we need to estimate the probability of B winning when A loses. We illustrate this scenario in Figure 1. The event of B winning and A losing can occur only when $A_i \neq B_i$ and A loses, and B, V reach a draw at the ith position. Therefore B can only win with a probability of $\frac{1}{4}$ from the $i+1^{th}$ round. This occurs as a result of B drawing i times and winning from the $i+1$ turn onwards, meaning x

$$P(A_{lose} \wedge B_{win}) = \frac{1}{2} \sum_{n=1}^{160-i} \frac{1}{2^i} \frac{1}{2^n} = \frac{1}{2^{i+1}} \left(1 - \frac{1}{2^{160-i}}\right) \tag{5}$$

For small values of i, the probability of losing with A and winning with B is $\approx \frac{1}{2}^{i+1}$. Thus, the probability of winning with either A or B, is:

$$\begin{aligned} P(A_{win}) \vee P(B_{win}) &= \frac{1}{2} - \frac{1}{2^{160}} + \frac{1}{2^{i+1}} \left(1 - \frac{1}{2^{160-i}}\right) \\ &\approx \frac{1}{2} + \frac{1}{2^{i+1}} \end{aligned} \tag{6}$$

We can say that the probability of winning with two accounts, with different MSB at position i, for low values of i is $\frac{1}{2} + \frac{1}{2}^{i+1}$. For example, let's assume the first three MSBs of A and B are 000 and 001, respectively. The first difference in the MSB happens at position 3, $i = 3$. The combined probability of victory with these two accounts would be $\frac{1}{2} + \frac{1}{2}^4 = 0.5625$.

The XRP Ledger shuffling mechanism ensures that an attacker using a single account has only 50% chance of winning. Furthermore, as we have shown, XRP Ledger provides some defence against frontrunning attacks even when using multiple randomly generated accounts.

In the following section, we will discuss our proposed frontrunning model and showcase a strategy which bypasses the XRP Ledger resilience to frontrunning.

III. Methodology

In this section, we describe the attack model and our frontrunning strategy.

A. Attacker Model

We begin with an example of an opportunity. Imagine Alice is selling 2 *FOO* for 1 *XRP* and Bob is selling 5 *FOO* for 5 *XRP*, at an exchange rate of 0.5 *XRP/FOO* and 1 *XRP/FOO*, respectively. A victim submits an offer to buy 7 *FOO* for 7 *XRP* at an exchange rate of 1 *XRP/FOO*. This is an opportunity for the attacker to buy Alice's offer before the victim and sell it back to the victim at a higher price. Therefore, the attacker submits two offers: (i) a buy offer to take Alice's offer and (ii) an order to sell 2 *FOO* for 2 *XRP*. The attacker wins if the

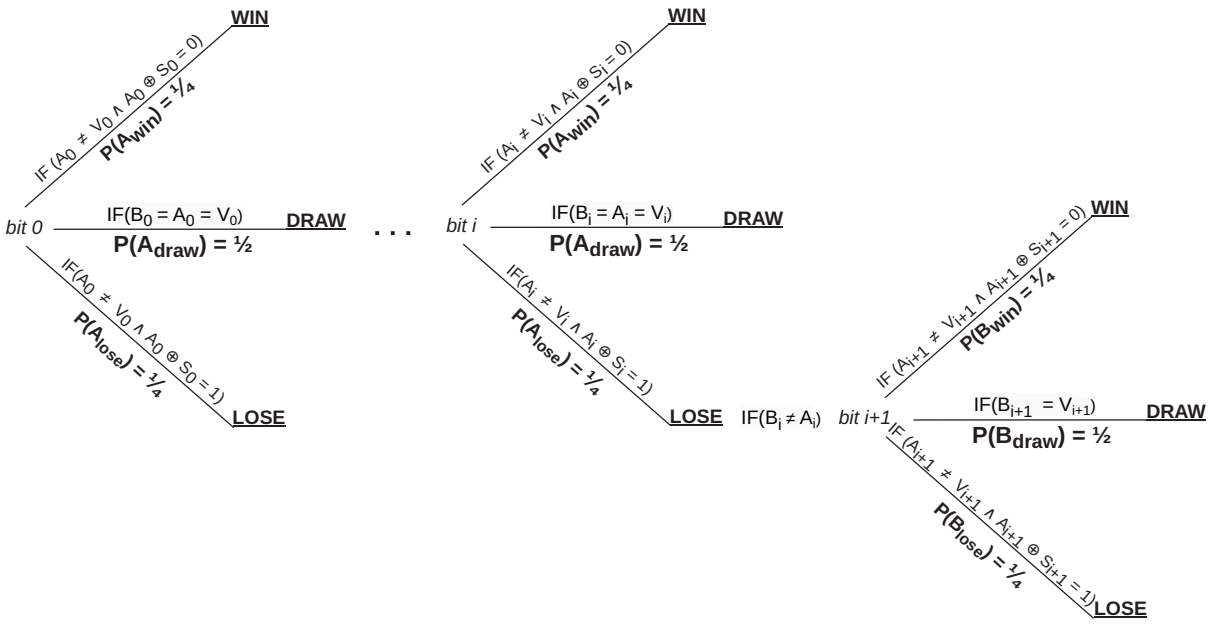

Fig. 1: Illustrative example of XOR ordering as a sequential series of bit comparisons.

attacker's buy offer occurs before the victim's. The victim will take the attacker's and Bob's offers, resulting in the attacker making a profit of 1 *XRP* minus the transaction fee.

Our attacker is required to monitor opportunities (pending offer transactions) in real-time. We define an opportunity as follows. Assume an available sell offer O_{sell} for a token T at a price P, and an incoming buy offer O_{buy} at a price P'. We define the case when $P' > P$ as an opportunity to make a profit $\delta P = P' - P$, also called the spread.

During some preliminary analysis, we found 65,291 such opportunities. For example, in the ledger 74,037,904, the transaction **2F6CAF5C87F...** created a buy order at a price 13.3 thousand times higher than the market average.

B. Frontrunning Strategy

1) Identifying Opportunities: The attacker requires a stream of pending offers and existing sell orders on the ledger to detect an opportunity. They can get this information via the XRP Ledger WebSocket API [7]. The API allows anyone to subscribe to a stream of pending transactions. The same API also provides book order information. The arbitrage system *JACK* [13] successfully used the WebSocket API to detect arbitrage opportunities in real-time. Therefore, we use the same API to detect frontrunning opportunities.

As discussed in Section II, candidate transactions propagate through the network via broadcasting. We can reduce the delay between transaction submission and detection by running a *rippled* server connected to many peers. Such a server will quickly learn about new transactions submitted anywhere in the network.

2) Frontrunning Attack: The attacker controls N accounts $A = \{a_1, a_2, a_N\}$, with which it submits the frontrunning transactions. The attacker monitors incoming proposed offer transactions for an offer O_{V-buy}, which fully consumes the cheapest existing O_{sell} offer for some token T. The attacker ignores the opportunity if the spread between the victim's and the seller's offers is below a profit threshold. Otherwise, the attacker prepares two transactions for each of the N accounts: a buy offer O_{A-buy} to consume O_{sell}, and a sell offer O_{A-sell} with the price equal to O_{V-buy}. The attacker cancels the O_{A-buy} offer if the account does not win. The O_{A-sell} transaction will fail automatically if the accounts do not hold token T. However, the attacker will incur a loss of $2 \times$ transaction fees. The attack will be successful when the attacker's buy offer occurs *before* the victim's buy offer. The attacker makes a profit of $P' - P - 2 \times$ transaction fees.

3) Improving the Odds: A single account has a 50% probability of winning. An attacker can use multiple accounts to increase their odds. However, as we discussed in Section II-A2, randomly generated accounts do not provide sufficient winning guarantees. In the following, we outline a strategy to improve the winning odds.

The attacker's account A appears before the victim's account V when $A \oplus S < V \oplus S$.

The outcome of the XOR between the most significant bits of the salt and the account address will determine if the result ends up in the lower half of the range of possible accounts or the upper half.

Therefore in the event that $V_0 \oplus S_0 = 1$ where they represent the first MSB of the victim and the salt respectively, we want our attacker to have $A_0 \oplus S_0 = 0$, such that we win the ordering 50% of the time. With the same reasoning, in the event that $V_0 \oplus S_0 = 0$, we need to focus on the second MSB. Therefore if $V_1 \oplus S_1 = 1$, then we need to pick an attacker such that: $A_0 \oplus S_0 = 1 \wedge A_1 \oplus S_1 = 0$.

We continue this technique for the first n most significant

bits, as described in Algorithm 1.

Algorithm 1 Selecting attackers from pool

```
procedure SELECTATTACKERS(A_victim, N, P)
    V ← V ≪ (256 − N)
    A = []
    for i in range(N) do
        Index = V ⊕ 2^(N+1−i)
        A.append(P[Index])
    return P
```

A limitation of this approach is that the attacker has to maintain a pool of accounts. Each account with a unique combination of the first n bits, for a total of 2^n accounts.

The pool has to be initialised ahead of conducting the frontrunning attacks. Therefore, our proposed strategy can result in a high initial capital requirement, given that each account requires a reserve of XRP (see Section III-B4). An attacker may leverage partial order fulfilment if they do not have enough capital allowing the attacker to "bootstrap" the initial funds. Although, they might not be able to take full advantage of all opportunities.

4) The Price of Victory: There are three costs associated with each frontrunning attack.

a) Fees.: The transaction fee is a small amount of XRP paid when submitting a transaction. It is non-recoverable and is the only source of loss for the attacker. Transaction fees scale exponentially, as outlined in Section II-A1b. Therefore, depending on the potential profit from a transaction, an attacker may pay larger fees than expected to ensure their transactions appear in the next ledger.

b) Reserve.: The attack has associated reserve requirements. The *Base Reserve* is the minimum XRP required for each account in the ledger. These funds are frozen but are released when the attacker deletes the account. During the writing, the base reserve was 10 XRP for an account [7]. The *Owner Reserve* freezes 2 XRP for each object an account owns within the ledger. The relevant objects for the frontrunning attacks are offers and trustlines. Each attacker's account has to reserve $10 + 2 * offers + 2 * trustlines$.

c) Liquidity.: Liquidity represents the funds that are necessary to consume an offer. It forms a significant portion of the attacker's capital. Every attacker's account has to contain enough funds to place the buy orders. XRP Ledger DEX supports partial order fulfilment. Therefore, the attacker may perform frontrunning with fewer funds. Though, this will reduce the profit, making some attacks unprofitable.

IV. Evaluation

In this section, we evaluate our frontrunning strategy on the XRP Ledger testnet and mainnet.

A. Frontrunning on the Testnet

The goal of performing the frontrunning attack on the testnet is to show that the attack is possible without causing harm. The *testnet* is the testing ground for the XRP Ledger. Users can experiment with the XRP Ledger without risking real money.

To not interfere with other users of the testnet, we issued our token: *TOK*. We summarise the transactions submitted for each experiment in Table I.

a) Setup: Our experimental setup is as follows. First, we create an artificial opportunity for the attacker to make 2 XRP profit. Then, we attack with six randomly generated accounts. After the attack, we clear the offers and restore *TOK* balances. We experiment with two sets of accounts, performing 3,000 attacks with each, and capture the XRP balance of each account every 50 iterations.

Owner	Type	Offer	Exchange Rate
Seller	Sell	1 TOK : 1 XRP	1
Seller	Sell	2 TOK : 3 XRP	1.5
Seller	Sell	3 TOK : 6 XRP	2
Victim	Buy	6 TOK : 12 XRP	2
Attacker	Buy	1 TOK : 1XRP	1
Attacker	Buy	2 TOK : 3 XRP	1.5
Attacker	Sell	3 TOK : 6 XRP	2

TABLE I: Testnet transaction summary.

b) Results: The evaluation on the testnet shows positive results (see Fig. 2). The frontrunning attack is feasible and profitable. In both experiments, all accounts made a profit but exhibited different profit growths.

From the first set of accounts (see Fig. 2 top), accounts 2 and 4 were the most successful, with a profit of 250 and 350 XRP, respectively. In contrast, the combined earnings of the other accounts were around 33 XRP.

From the second set of accounts (see Fig. 2 bottom), account 6 was the most prosperous, with a final balance of 115 XRP. The least profitable account, 3, made only 6 XRP. The others profited between 62 and 73 XRP. These results demonstrate that randomly selected accounts do not win a proportional number of times.

It is important to note that, during the evaluation, our transactions overwhelmingly filled each ledger, and we attacked the same victim with a few randomly generated accounts. This may have affected the transaction order. Therefore, we evaluate frontrunning on the mainnet to verify the attack further and ensure its success was not the effect of the testnet.

B. Frontrunning on the Mainnet

1) Opportunities in the Wild: We scanned the XRP Ledger for opportunities defined in Section III. Between June 30th and August 30th 2022, we found 65,291 profitable transactions and evidence of frontrunning.

a) Total Profit: The potential profit in two months was 4,402,433.5815 (4.4M) XRP or an estimated 1.4M USD. The most profitable token was *SOLO*. Opportunities involving this token were worth 3.4M XRP or 1.0M USD. An order of magnitude more valuable than opportunities with the 10th most profitable *USD* token. These opportunities had a profit potential of 9.5K USD.

A significant portion of the profit came from a single order to buy *SOLO* at 13.3 thousand times higher price

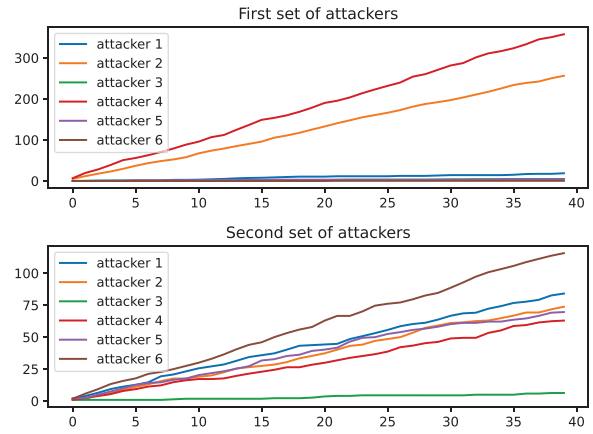

Fig. 2: Testnet cumulative profit growth using two different sets of attackers.

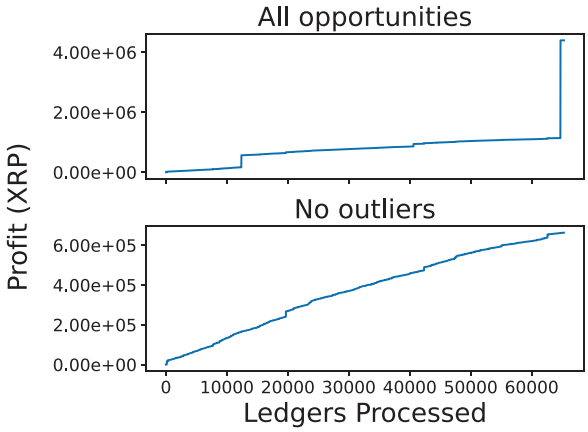

Fig. 3: Cumulative profit growth on the mainnet.

than the market average (see Fig. 3) in ledger *74,037,904*. The second most profitable transaction was *xSPECTAR* in ledger *72,890,193*, 1.9 thousand times higher than the market average. It is hard to say whether these transactions were attempts at market manipulation or human errors. For example, we found one transaction in which a user bought *BLA* token at an exchange rate of 0.45 when the market average exchange rate was 0.045. Without the outlying transactions, the profit growth is linear (see Fig. 3 bottom).

b) Profit and Investment: The profitability of each opportunity O is the difference between the revenue, required liquidity for one account, and transaction fees. Liquidity is the amount of XRP needed for one attacker's account to frontrun one opportunity.

We illustrate the ratio between profitability and liquidity in Fig. 4. Both axes are on a logarithmic scale due to the high variability between the data points. We divide the figure into four quadrants by the average profit and liquidity. Based on profitability and capital requirements, it is clear that opportunities are not equal.

The 4th quadrant contains the least desirable opportunities. These require exceedingly high liquidity and provide small

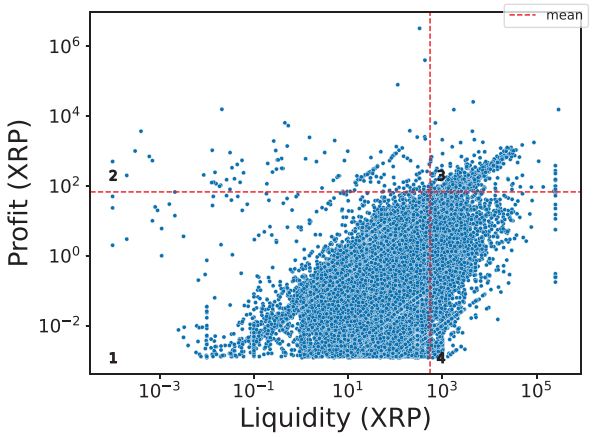

Fig. 4: Illustrative example of profit and liquidity requirements for opportunities.

returns. The worst offender required a liquidity of $2,000$XRP and returned only 0.002XRP profit. We conclude that it is impractical to frontrun these opportunities.

Members of the 3rd quadrant require careful consideration. Some of them require liquidity greater than $87,000$ XRP. These liquidity requirements make the majority of these opportunities infeasible to be frontrun. However, an attacker may use partial order fulfilment to reap partial profits.

The opportunities in the 1st quadrant are of low reward, but they also need equally little liquidity. These opportunities might be worth attacking. However, increasing the number of accounts will further diminish their profitability. For the same reason, these opportunities should be fully consumed and not rely on partial order fulfilment.

The 2nd quadrant provides the best return on liquidity. In this quadrant, we find all the profit outliers. The attacker should use the highest number of accounts to maximise the chances of winning.

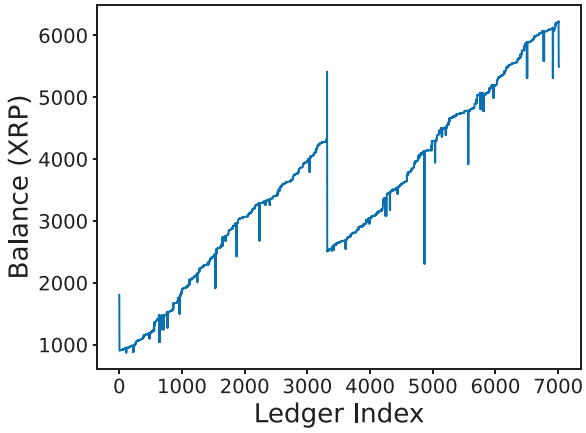

Fig. 5: Frontrunner balance growth.

2) Evidence of Frontrunning: During our analysed period, we also found an attacker performing frontrunning attacks. For example, in ledger 73,698,802, the buy offer of account

r3Vh9ZmQ... was frontrun by account **robp8v3o...**. The attacker was active for three weeks. On September 10th, they deleted their accounts. During this time, the attacker performed 4,075 attacks. Out of which, 2,435 were successful, and 1,640 were not. In Fig. 5, we illustrate the balance growth of the attacker's account. We estimate that the attacker made a total profit of around 8,000 XRP or approx. 3,200$. The account started with an initial balance of around 1,000 XRP. Halfway, they sent their profits to another account, as indicated by the sharp balance drop.

3) Optimizing the Attack: We have started the process of responsible disclosure of our findings to the XRP Ledger developers. Therefore, we evaluate our attack strategy with a simulator to not interfere with their investigation.

The period of two months, during which we found the opportunities and discovered the frontrunner, provide the ideal data to evaluate our attack strategy. In the remainder of this section we discuss our results.

a) Simulator: The simulator scans ledgers for offers O_{buy} that consumed offers O_{sell}, such that the asking price of O_{sell} was lower than O_{buy}. The simulator injects frontrunning transactions using up to M accounts and orders the transactions following the rules discussed in Section II-A2.

We validated the correctness of our implementation of the shuffling algorithm using 100,000 salts sampled from ledgers between the 10th and 14th of August 2022. For every sample, we were able to recreate the exact transaction order.

As outlined in Section II-A2, the ordering mechanism requires a random salt, which is the hash of the root of a Merkle Tree. In addition to the sampled seeds, we generated 150,000 random 256-bit seeds. We used these values to simulate the transaction ordering. The simulator then captures the number of times in which at least one attacker's transaction occurred before the victim. In addition, the simulator calculates the potential profit, assuming the win is guaranteed.

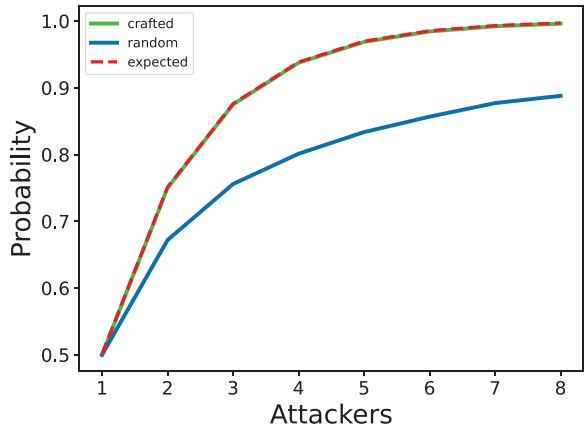

Fig. 6: Winning probabilities per frontrunning strategy.

b) Data: We created two pools of accounts using different generation strategies. The first pool contains randomly generated accounts created with the XRP Ledger Python library [14]. We paid no attention to the bit sequence of these accounts. Furthermore, during the attack, the attacker picks accounts from this pool in an arbitrary order.

The second pool contains accounts that cover every combination of the first eight most significant bits. The attacker picks from this pool in a way that maximises the odds of winning. Both pools contained 256 entries. We simulate both strategies on a set of 2,000 victims randomly picked from the opportunities we found. For each victim, we select up to 8 frontrunning accounts.

c) Results: We illustrate the winning probabilities of these strategies in Fig. 6. The red dashed line indicates theoretical probability derived via $\frac{2^M-1}{2^M}$. The blue line indicates the winning probability with the naive strategy, and the green uses the advanced strategy.

Both approaches achieve a winning probability of 50% when attacking with a single account. However, we already see a divergence between the strategies of approx. 8% when using two accounts. The gap increases up to 11% as we increase their number. The approaches slowly converge when raising the count beyond 8 (not shown). However, even with 30 accounts, the naive approach does not reach a 96% success probability.

Randomly selected accounts perform poorly due to overlaps within the first 8 MSB patterns. An arbitrarily added account will provide some benefit. However, in most cases, another existing account would have complemented the victim's ID. Therefore, unless selected carefully, a new account adds little benefit.

V. Discussion

a) Order Fairness: Achieving transaction order fairness in blockchains is a prevalent topic. Several works have demonstrated that order fairness based on transaction submission time is impossible [15], [16]. Instead, works rely on weaker fairness definitions. For example, [15] achieve batch order fairness, which states that if most honest participants receive transaction t_a before t_b, then t_a must be in an older or the same block as t_b. However, this does not prevent in-block reshuffling.

On the other hand, authors of [16] propose applying transactions linearly, such that if all honest participants receive t_a before t_b, then t_a must be applied before t_b. However, delays in message propagation still allow attackers to send their transactions before all nodes receive t_a and thus execute the malicious transactions before t_a.

In contrast, XRP Ledger offers account-level fairness. Assume an account submits two transactions in order t_a and t_b. The transaction sequence number will ensure that t_a will occur before t_b. Therefore, transactions submitted by the same account execute in their submission order. However, XRP Ledger does not guarantee the transaction execution ordering for differing accounts.

b) Strengths: The XRP Ledger shuffling mechanism guarantees that the order in which transactions execute is unpredictable. An attacker has a 50% probability of a successful attack with a single account. However, even when using multiple randomly generated accounts, the odds of winning do not grow as expected. We demonstrate that due to overlapping

most significant bits of the account addresses, there will be cases when accounts share their wins or loses. Therefore, the shuffling algorithm offers some security against naive frontrunning, although it does not prevent it.

c) Limitations: We demonstrate that an attacker can generate accounts with alternating bit sequences to maximise their odds of winning. Furthermore, the new ordering mechanism introduced by the XRP Ledger displays some properties that might deter its use in traditional shuffle requirements. For example, when shuffling two ”neighbour accounts” (i.e. accounts only differing in the last bit), they will end up together in the result but in reverse order. While this type of preserved locality might be an issue in some shuffling scenarios, the XRP Ledger uses it to shuffle 256-bit hashes that, by design, are near impossible to reverse. Therefore, it is unrealistic to take advantage of this property. For example, if an attacker could generate an account that is one bit off from the victim in a realistic time frame, they could also create the keys to the victim's account and steal their funds directly.

Frontrunning on the XRP Ledger was an issue in the past, which was addressed by executing transactions in a pseudo-random ordering. However, combined with low transaction fees, this ordering does not provide sufficient protection against frontrunning. As we demonstrate in our work, an attacker can use multiple accounts to increase their odds of winning without significantly reducing the profit. We have disclosed our findings to the XRP Ledger developers at Ripple. We are working together with the team to improve the robustness of XRP Ledger's blockchain to frontrunning attacks. We leave the evaluation of any future mitigation strategy to future work.

VI. Related Work

Frontrunning is thoroughly studied on the Ethereum blockchain [17], [2], [4]. Eskandari et al. [17] were the first to provide a taxonomy of frontrunning attacks. Daian et al. [2] were the first to provide evidence of frontrunning practices by monitoring Ethereum's pool of pending transactions. Torres et al. [4] were the first to provide a lower bound on the actual profit made by attackers by measuring the prevalence of different frontrunning attacks across Ethereum's transaction history. Our work is the first to study frontrunning on a blockchain other than Ethereum. In particular, we demonstrate the practicality of frontrunning attacks on the XRP Ledger and provide the first lower bound regarding the maximum profit attackers could make on the XRP Ledger by applying our attacks.

Besides analyzing the prevalence of frontrunning, several countermeasures mitigate the effects of frontrunning on Ethereum. One solution is to prevent frontrunning at the application layer either via an advanced commit-and-reveal scheme [18], splitting large trades into smaller trades with tighter slippage protection [19], or designing new decentralized exchanges that are more frontrunning-resistant [20], [21], [22], [23], [24], [25]. However, these solutions often introduce higher transaction costs and do not protect against every type of frontrunning attack. Another solution is to prevent frontrunning at the consensus layer by either fair transaction ordering [26], [15], [16], [27] or making transactions private [28], [29], [30]. However, these techniques do not apply to large dynamic blockchains and thus are not widely adopted.

Projects based on private transaction pools such as Flashbots [31] and Eden [32] are an alternative to modifying the application layer or the consensus layer. These pools establish private agreements with miners, allowing users to bypass the public pending transaction pool and submit their transactions privately. However, Weintraub et al. [33], and others [8], [18] discovered that private pools are used excessively to perform frontrunning attacks on Ethereum, and the miners receive the most of the profit distribution within Flashbots.

Despite the thorough analysis of = the liveness and safety properties of the XRP Ledger's consensus protocol [34], [35], [36], and its topology [37], no works analyze whether XRP Ledger transaction ordering strategy is resistant to frontrunning attacks. Peduzzi et al. [13] demonstrated that observing XRP Ledger transactions in real-time to perform arbitrage on the decentralized exchange is possible. However, they did not study whether frontrunning attacks would be feasible. Our work shows that frontrunning is possible and that the countermeasures employed by the XRP Ledger to prevent frontrunning are not effective enough. We demonstrate that due to the low transaction fees on the XRP Ledger, sharp-witted attackers will be able to perform frontrunning attacks and make a significant profit.

VII. Conclusion

Frontrunning is a well-studied and understood issue on Ethereum. However, little effort has been made to analyze whether other blockchains are potentially vulnerable to these attacks. In this paper, we conduct the first frontrunning analysis on the XRP Ledger. We show that the XRP Ledger is vulnerable to frontrunning despite efforts to make frontrunning more difficult by introducing a pseudo-random transaction order. We demonstrate that these attacks are profitable. Furthermore, we showcase two attack strategies. In the first strategy, accounts are chosen randomly. In the second, accounts are selected carefully to maximize the success probability of frontrunning the victim. We first evaluate the feasibility of our strategies on the XRP Ledger testnet by creating and successfully frontrunning our own generated opportunities. We then show the profitability of the two attack strategies on the historic mainnet data. We discover, over two months, frontrunning opportunities worth 1.4M USD, including evidence of a user already performing frontrunning attacks in the wild.

Acknowledgment

The authors would like to thank Arno Michel Denis Geimer for his input. This work was supported by Ripple UBRI.

References

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

[2] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "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.

[3] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 428–445, 2020.

[4] C. F. Torres, R. Camino, and R. State, "Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain." USENIX Association.

[5] D. Schwartz, N. Youngs, A. Britto *et al.*, "The ripple protocol consensus algorithm," *Ripple Labs Inc White Paper*, 2014.

[6] D. Hide, "Exploiting ripple transaction ordering for fun and profit," 2015. [Online]. Available: http://availableimagination.com/exploiting-ripple-transaction-ordering-for-fun-and-profit/

[7] X. L. Foundation, "XRPL.org," 2022, [Online; accessed 14. Sep. 2022]. [Online]. Available: https://xrpl.org

[8] J. Piet, J. Fairoze, and N. Weaver, "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

[9] J. W. Markham, "'front-running' - insider trading under the commodity exchange act," 1988.

[10] G. A. F. Rebello, G. F. Camilo, L. Guimaraes, L. A. C. de Souza, and O. Duarte, "Security and performance analysis of quorum-based blockchain consensus protocols," *Electrical Engineering Program, COPPE/UFRJ, Tech. Rep*, 2020.

[11] "Transaction set canonical ordering," 2022. [Online]. Available: https://github.com/XRPLF/rippled/blob/e32bc674aa2a035ea0f05fe43d2f301b203f1827/src/ripple/app/misc/CanonicalTXSet.cpp

[12] A. Kaminsky, "Testing the randomness of cryptographic function mappings," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 78, 2019.

[13] G. Peduzzi, J. G. James, and J. Xu, "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.

[14] "xrpl-py," Oct. 2022, [Online; accessed 7. Oct. 2022]. [Online]. Available: https://pypi.org/project/xrpl-py

[15] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, "Order-fairness for byzantine consensus," in *40th Annual International Cryptology Conference, CRYPTO*, 2020.

[16] 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.

[17] S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: Front-running attacks on blockchain," *Econometrics: Computer Programs & Software eJournal*, 2019.

[18] A. Capponi, R. Jia, and Y. Wang, "The evolution of blockchain: from lit to dark," *arXiv preprint arXiv:2202.05779*, 2022.

[19] L. Heimbach and R. Wattenhofer, "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.

[20] 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.

[21] M. Ciampi, M. Ishaq, M. Magdon-Ismail, R. Ostrovsky, and V. Zikas, "Fairmm: A fast and frontrunning-resistant crypto market-maker," in *Cyber Security, Cryptology, and Machine Learning - 6th International Symposium, CSCML*, 2022.

[22] C. McMenamin, V. Daza, and M. Fitzi, "Fairtradex: A decentralised exchange preventing value extraction," *IACR Cryptol. ePrint Arch.*, p. 155, 2022. [Online]. Available: https://eprint.iacr.org/2022/155

[23] L. Zhou, K. Qin, and A. Gervais, "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

[24] E. Sariboz, G. Panwar, R. Vishwanathan, and S. Misra, "FIRST: frontrunning resilient smart contracts," *CoRR*, vol. abs/2204.00955, 2022.

[25] C. Protocol, "CoW Protocol - Batch Auctions," Oct. 2022, [Online; accessed 05. oct. 2022]. [Online]. Available: https://docs.cow.fi/overview/batch-auctions

[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] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, "Themis: Fast, strong order-fairness in byzantine consensus," *IACR Cryptol. ePrint Arch.*, 2021.

[28] E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, "CALYPSO: private data management for decentralized ledgers," *Proc. VLDB Endow.*, 2020.

[29] D. Yakira, A. Asayag, G. Cohen, I. Grayevsky, M. Leshkowitz, O. Rottenstreich, and R. Tamari, "Helix: A fair blockchain consensus protocol resistant to ordering manipulation," *IEEE Trans. Netw. Serv. Manag.*, 2021.

[30] T. P. Protocol, "The Penumbra Protocol - ZSwap," Oct. 2022, [Online; accessed 07. oct. 2022]. [Online]. Available: https://protocol.penumbra.zone/main/zswap.html

[31] "Overview | Flashbots Docs," Nov 2021, [Online; accessed 23. Nov. 2021]. [Online]. Available: https://docs.flashbots.net/flashbots-auction/overview

[32] C. Piatt, J. Quesnelle, and C. Sheridan, "Eden network," 2021. [Online]. Available: https://edennetwork.io/EDEN_Network___Whitepaper___2021_07.pdf

[33] B. Weintraub, C. F. Torres, C. Nita-Rotaru, and R. State, "A flash(bot) in the pan: measuring maximal extractable value in private pools," *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022.

[34] L. Mauri, S. Cimato, and E. Damiani, "A formal approach for the analysis of the XRP ledger consensus protocol," in *Proceedings of the 6th International Conference on Information Systems Security and Privacy, ICISSP*. SCITEPRESS, 2020, pp. 52–63.

[35] B. Chase and E. MacBrough, "Analysis of the XRP ledger consensus protocol," *CoRR*, vol. abs/1802.07242, 2018. [Online]. Available: http://arxiv.org/abs/1802.07242

[36] I. Amores-Sesar, C. Cachin, and J. Micic, "Security analysis of ripple consensus," in *24th International Conference on Principles of Distributed Systems, OPODIS*, ser. LIPIcs, Q. Bramas, R. Oshman, and P. Romano, Eds., 2020.

[37] V. Tumas, S. Rivera, D. Magoni, and R. State, "Topology Analysis of the XRP Network," *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23)*, 2023.