SharkTeam

A Vulnerability Perspective Analysis of Move Language Security —— Sandwich Attack

Feb 16, 2023

SharkTeam, a leading blockchain security service team, offers smart contract audit services for developers. To satisfy the demands of different clients, thesmart contract audit services provide both manual auditing and automated auditing.

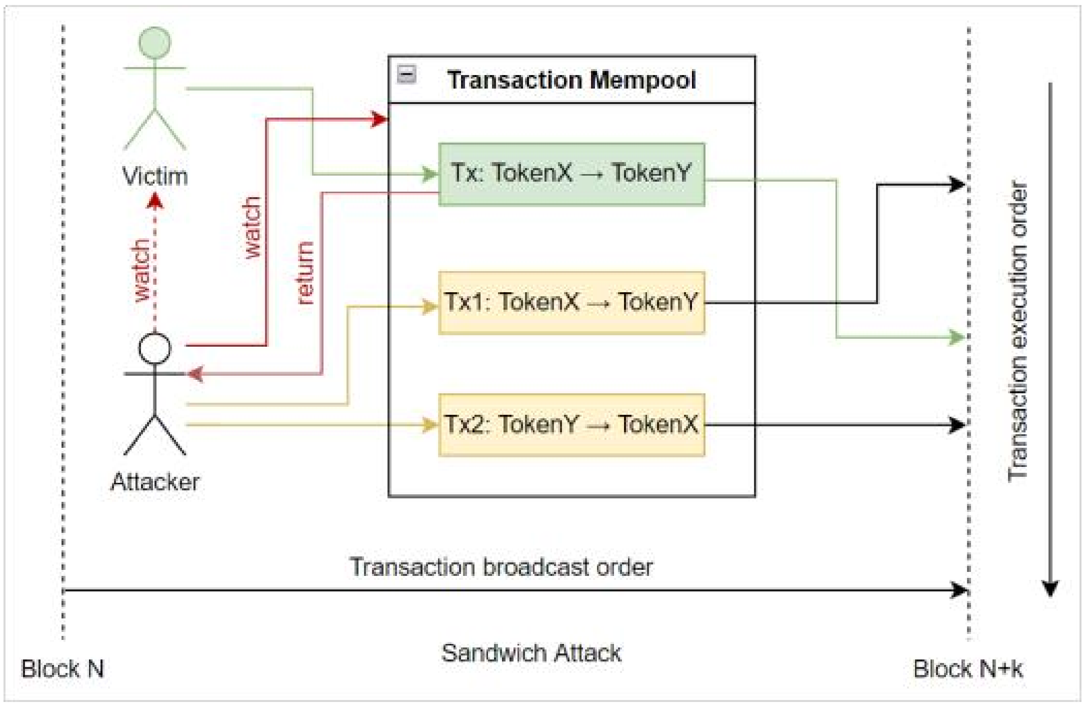
We implement almost 200 auditing contents that cover four aspects: high-level language layer, virtual machine layer, blockchain layer, and business logiclayer, ensuring that smart contracts are completely guaranteed and Safe.

In the previous series of “Top 10 Smart Contract Security Threats”, SharkTeam summarized and analyzed the top 10 most harmful vulnerabilities in the field of smart contracts based on historical smart contract security incidents.

These vulnerabilities usually appeared in Solidity smart contracts before, so will the same harm exist for the emerging Move smart contracts?

The Sandwich Attack is a common attack in the DeFi and is a front-end attack that is not related to smart contracts at the blockchain level. To achieve a sandwich attack, an attacker finds a pending victim transaction in the transaction memory pool and then attempts to sandwich the victim transaction by sending a front-run and a back-run transaction, forming a “sandwich” transaction structure. Essentially, a sandwich attack is an act of pre-emption. The transparency of transactions in the blockchain transaction memory pool and the latency of executing transactions under the consensus mechanism (especially in the case of network congestion) make it easier for preemptive transactions to occur, which provides the conditions for sandwich attacks. An attacker launching a sandwich attack aims to prey on the victim’s revenue and generally exploits the difference in price to achieve his goal.

1. Attack process



As shown in the diagram above, the main flow of a sandwich attack is as follows.

(1) The attacker obtains the transaction Tx submitted by the attacker by observing the pending transactions in the transaction memory pool, gaining insight into the victim's transaction intent before Tx is executed and deeming it profitable. The attacker then submits the predicate transaction Tx1 at a higher Gas Price than Tx, causing an unexpected price slippage by using 10 TokenX to exchange 100 TokenY (1TokenX = 10TokenY, 1TokenY = 0.1TokenX), inflating the price of TokenY.

(2) The victim, unaware of anything, exchanges 5 TokenX for 40 TokenY through unexpected price slippage, when originally 5 TokenX could have been exchanged for 50 TokenY (1 TokenX = 8 TokenY, 1 TokenY = 0.125 TokenX). At this point, the price of TokenY is further increased.

(3) The attacker submits Tx2 and converts the TokenY redeemed in step (1) into TokenX. Since the price of TokenY is twice too high (1TokenX=7TokenY, 1TokenY=0.143TokenX), 100 TokenY can be redeemed for 14.286 TokenX, compared to the 10 TokenX paid at the time of redemption. This is a profit of 4.286 TokenX compared to the 10 TokenX paid for the exchange.

The attacker has caused an unexpected price slippage through the sandwich attack, increasing the actual price slippage of the victim's transaction and thus robbing the victim of his earnings.

Using the above attack process as an example, let's say the victim's desired transaction price is TokenX = 11TokenY.

1. If no sandwich attack occurs, the transaction price is the same as the previous transaction price, 1TokenX=10TokenY, at which point the price slippage.

• TokenX: 10–11 = -1

• TokenY: 1/10–1/11 = 0.0091

2. After the sandwich attack, the actual transaction price is 1TokenX = 8TokenY

and the price slippage is

• TokenX: 8–11 = -3

• TokenY: 1/8–1/11 = 0.0341

The unexpected slippage caused by a sandwich attack (front-running trade) is

• TokenX: (-3) — (-1) = 8–10 = -2

• TokenY: 0.0341–0.0091 = 1/8–1/10 = 0.025

That is, because of the sandwich attack, the price of TokenX unexpectedly falls by 2 and the price of TokenY unexpectedly rises by 0. 025.

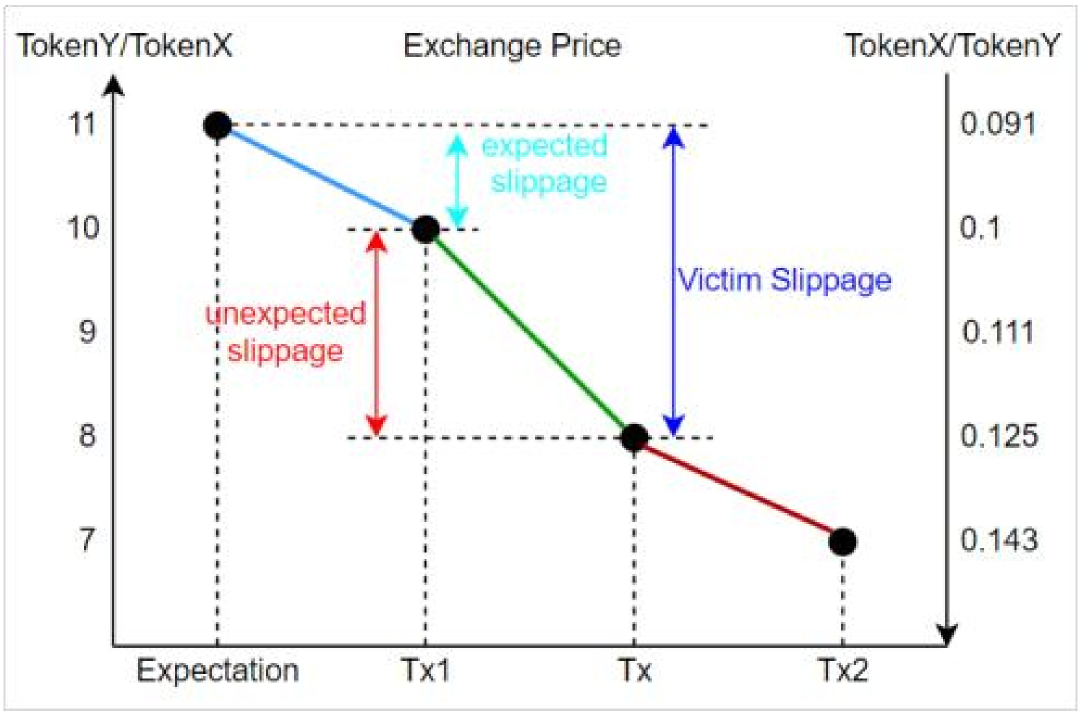


Figure: Sandwich exchange prices

A sandwich attack is one in which the victim's actual trade is made to create an unexpected slippage by front-loading the trade, increasing the price difference when the trade is executed, and then profiting from it.

The attacker initiates two transactions and needs to pay a Gas fee as a cost, especially if the Gas Price of Tx1 is higher. If the value of the profitable 2.5 Tokens is higher than the Gas Fee for both transactions, then the attacker will make a pure profit from this sandwich attack; otherwise, a loss will be incurred. Thus, a sandwich attack does not necessarily result in a net gain, but can also result in a loss, which is related to the cost of the sandwich attack (i.e. the Gas fee) and the price of the Token.

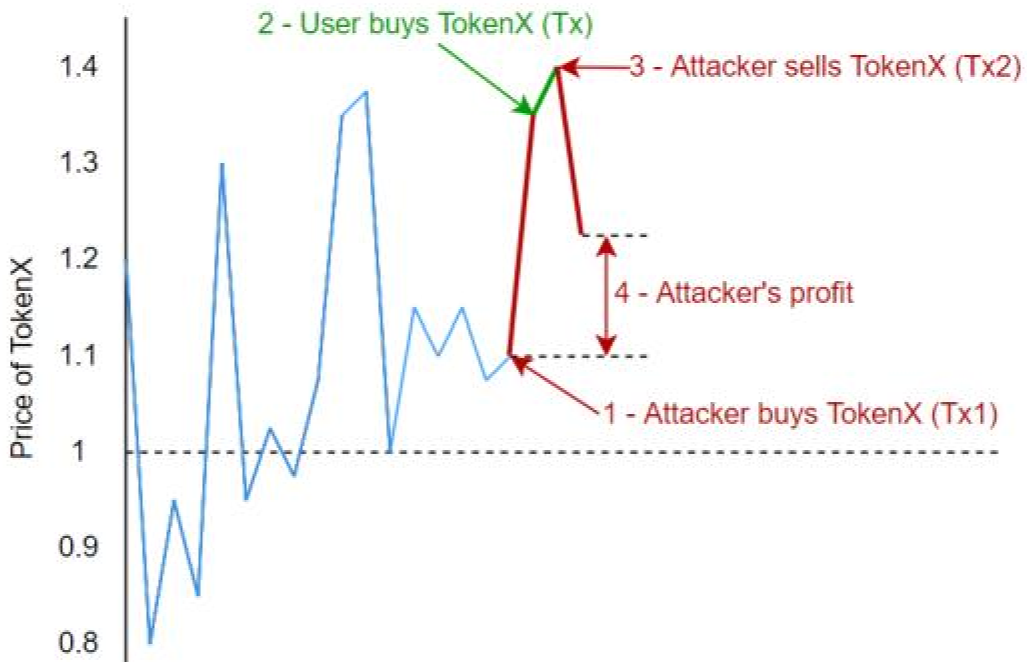


Figure 3 TokenX price fluctuations

As in the TokenX price fluctuation graph above, in the DEX ecology of a decentralized exchange with a constant product market maker X * Y = k, every time TokenX is purchased, its price goes up. Thus, the attacker buys TokenX first, the price of TokenX rises, and then others (the victims) buy TokenX, and the price of TokenX rises again. The attacker then sells the bought TokenX immediately, at which point the TokenX price rises twice and the attacker buys low and sells high, thus making the attacker an additional profit. In this way arbitrage of TokenX is achieved.

Arbitrage can, to a certain extent, maintain the balance of the market price ecology. However, sandwich attacks can come at the expense of normal traders (victims), ultimately robbing them of their profits.

2. Attack

Through the analysis of the attack process and price slippage, we find that for the sandwich attack to successfully prey on the victim, it must generate unexpected slippage through the front-running trade before the execution of the sandwiched trade, thereby increasing the price slippage when the sandwiched trade is executed, and thus widening the price difference between the front-running and back-running era coins. The attacker uses this price difference to prey on the victim’s gains.

The more the price slippage increases within the allowed trading price range, the greater the token trading spread between the front and back trades, and the greater the gain from the sandwich attack. If the price of the sandwiched transaction exceeds the allowed range at execution, the transaction will not be executed and the cost paid by the attacker for the front and back trades will become a loss and the attacker will not gain anything.

For AMM-related transactions, there are two ways to increase the price slippage and then widen the price differential, i.e. preemptive exchange of tokens versus preemptive reduction of liquidity.

(1) Preemptive exchange of tokens

The attacker preemptively exchanges TokenX for TokenY in the preemptive transaction. As in the attack flow above, the price of TokenY is inflated after the execution of the preemptive transaction. When the pinned transaction executes TokenX for TokenY, an unexpected price slippage occurs because the price of TokenY is inflated by the preemptive transaction. This, combined with the original price slippage, causes the actual slippage to be increased and the price of TokenY to be inflated again after the transaction is completed. The attacker sells the TokenY exchanged for the front transaction in the back transaction at the twice inflated price (into TokenX) and can thus profit from it.

(2) Preemptively reducing liquidity

For the constant product model AMM, the less liquidity in the trading pool, the greater the price volatility and the greater the price slippage. Therefore, the liquidity provider can increase the price slippage by reducing liquidity and thus increase the price spread for profit making purposes. The process by which this is achieved is as follows.

Front-running trade: preemptively removes liquidity TokenX/TokenY, increasing the price slippage when the victim trades by reducing liquidity.
pinned trade: executes the victim’s trade to exchange TokenX for TokenY
post-trade: adds liquidity to bring the liquidity in the TokenX/TokenY trading pool back to the pre-attack level.
Assume that the liquidity of the pool is 1000 * 1000 = 1e6; the attacker holds 200 * 200 liquidity; the victim submits a transaction to exchange 100 TokenX for TokenY.
Pre-trade: Attacker removes all liquidity. The liquidity of the pool after the transaction execution is 800 * 800 = 64e4; the attacker holds 200 TokenX and 200 TokenY.
Pinned transaction: The victim exchanges 100 TokenX for TokenY. The number of TokenY that can be exchanged is 800–640,000 / (800 + 100) = 88.889. If there is no front-running transaction, the number of TokenY that can be exchanged by the victim is 1000–1e6 / (1000 + 100) = 90.909. The liquidity of the pool of transactions after the execution of the pinned transaction is 900 * 711.111 = 64e4.
post-trade: adds liquidity to bring k back to 1e6. After execution, the pool liquidity is 1100 * 909.091 = 1e6 or 1097.561 * 911.111 = 1e6, leaving the attacker with 2.02 TokenY or 2.44 TokenX. The remaining tokens are effectively the The remaining tokens are actually the proceeds of the sandwich attack. If this gain is less than the cost, the attack will incur a loss.
With a sandwich attack in this manner, the attacker needs to hold the liquidity in the pool of transactions and then remove the liquidity before the sandwiched transaction is executed, while forfeiting the victim’s fee commission on the transaction.
In addition, sandwich attacks can also be business related, such as some auction NFT scenarios, where there is a price differential, which could be subject to a sandwich attack. The attacker achieves profit by adjusting the

order of transactions, inserting front and back transactions between the sell and buy transactions, to buy NFTs from the auctioneer at a low price and then sell them to bidders at a high price, robbing the auctioneer of its proceeds.

The conditions for this scenario to occur are relatively high and are related to the business of the scenario, such as the auction having a large price difference and the outcome of the auction sale must be related to the order of the transactions. Therefore, sandwich attacks in this scenario are easy to prevent and only require strict business and implementation design so that the result is not dependent on the order of the transaction.

3. Conditions for a successful attack

Sandwich attacks are not related to smart contracts and cannot be written to predict returns. Moreover, sandwich attacks require a higher transaction cost to initiate than the victim’s transaction, and therefore, sandwich attacks are risky in that not every attack will result in a net profit and may incur a loss. An attack without a net profit is a failure, and for an attack to be successful, two conditions need to be satisfied simultaneously.

(1) From a results perspective

The profit from a sandwich attack needs to be higher than the Gas fee (the cost of launching the attack) and if the benefit is lower than the cost, the attacker will not only have no net profit, but will also incur a financial loss.

There are two factors that affect the net profit of the final gain: Gas fee and price slippage. gasPrice determines the order of transactions, and the order of transactions affects the price slippage at the time of the transaction. gas fee is calculated from the Gas consumed by the transaction as well as GasPrice, and is the largest cost of the attack. price slippage directly affects the gain of the attack. the effect of Gas fee and price slippage on the net gain of the attack has Uncertainty, rather this impact is unpredictable. This makes it difficult to assess the success rate of a sandwich attack and the net profitability of the attack before it is launched. This uncertainty also reduces the frequency of sandwich attacks compared to attacks launched using contractual vulnerabilities, especially for some small transactions.

(2) From a transaction perspective

Firstly, it is necessary to ensure that the order of execution of the transactions is the preceding transaction, the victim being sandwiched transaction, and the latter transaction; secondly, it is necessary to ensure that all three transactions can be executed successfully. For example, if the victim is caught in a transaction that requires slippage, the sandwich attack will fail and the attacker will bear the loss if the victim is caught in a transaction that cannot be executed because of excessive slippage after the execution of the front transaction.

4. Preventive measures

By analyzing the success conditions of sandwich attacks, we can take the following three preventive measures against sandwich attacks:

(1) Limiting Gas Fees

Sandwich attacks happen because transactions with higher gas costs are prioritized over other transactions. Setting a certain limit on the gas fee can reduce the impact of the gas fee on the transaction sequence and increase the difficulty of the sandwich attack.

(2) Avoid low liquidity trading pools

Lack of good fluidity is good for a sandwich attack. Because the lower the liquidity of the trading pool, the greater the price fluctuation and the greater the price slippage. And the higher the slippage, the more profit an attacker can make from a sandwich attack. Therefore, low-liquidity trading pools are more vulnerable to sandwich attacks. Therefore, it is recommended to choose a high-liquidity trading pool for trading as much as possible, which can reduce the possibility of sandwich attacks.

(3) Select small transaction

Sandwich attacks are more interested in large transactions, because the larger the transaction value, the greater the profit margin. One way to avoid a sandwich attack is to split our transaction into several smaller transactions, since these small transactions are less interesting for an attacker, since small transactions may incur losses.

5. Risk Analysis of Potential Sandwich Attacks in the Move Ecology

The sandwich attack is a front-end attack method for price manipulation at the blockchain level. It has nothing to do with smart contracts, but only with the execution order of blockchain transactions. Therefore, to determine whether there is a sandwich attack on the public chain of the Move ecosystem, it is necessary to analyze the consensus mechanism of the public chain and the order of transaction execution in the transaction memory pool.

Blockchain verification nodes (miners) collect transaction information broadcast in the blockchain network, and these transactions will be stored in the transaction memory pool for verification and execution. Different blockchains adopt different consensus protocols, and different consensus protocols also have different transaction verification and execution schemes. For example, the PoW protocol adopted by Bitcoin and the PoS protocol adopted by Ethereum both execute transactions sequentially. The transactions in the memory pool are sorted according to the high and low GasPrice. implement. This also creates conditions for speculators to front-run trade, and it is also a prerequisite for sandwich attacks.

Move ecological public chain, such as Sui, in order to improve the performance of the public chain, has chosen an improved Byzantine agreement, using a combination of parallel and sequential transaction execution schemes. Unrelated transactions without shared state can execute in parallel, while related transactions that share state can only be executed sequentially. From the above analysis, it can be known that the three sandwich transactions in the sandwich attack process share the state of the AMM transaction pool, so they cannot be executed in parallel and can only be executed sequentially. The condition of whether the public chain constitutes a sandwich attack depends on whether there is a possibility of preemptive transaction in the sequence of execution:

(1) If the final execution order of the transaction is related to the time of submission, and the transaction submitted first will be executed first, the possibility of sandwich attack will be eliminated from the beginning, and the preemptive transaction of the victim transaction will not be realized;

(2) If the final execution order of transactions is related to GasPrice, which is the same as the execution order of transactions in the Ethereum PoS mechanism, and transactions with high GasPrice are executed first, then such a public chain cannot avoid sandwich attacks;

(3) If the final execution order of transactions is calculated and arranged by GasPrice according to a certain algorithm, then such a public chain may have conditions for a sandwich attack, which depends on the arrangement algorithm, but the difficulty of a sandwich attack will be more difficult than directly sorting by GasPrice Somewhat higher, but the possibility of no sandwich attack cannot be ruled out.

The public chains of the Move ecosystem include Starcoin, Aptos, Sui, 0L, Pontem, etc. These public chains have chosen different consensus protocols and made improvements. For example, Starcoin has chosen an enhanced PoW protocol and improved it; Sui adopts a scheme in which irrelevant transactions are executed in parallel and related transactions are executed sequentially. Whether there may be a sandwich attack in these public chains needs to analyze its transaction execution method. Whether there will be a sandwich attack depends on the activity of the public chain, especially the lock-up, transaction volume and transactions of AMM-based decentralized transactions. quantity. Generally, sandwich attackers will not choose an AMM with low lockup, small transaction volume, and low transaction volume, because the net profit is low, and the loss is even greater than the gain.

At present, the public chain of the Move ecosystem is in the development stage. From the perspective of project coverage, project maturity, lock-up and other indicators, the Move project is much less likely to suffer from a sandwich attack than the Solidity project. The security of the Move language itself is much higher than that of Solidity, and there may be fewer loopholes in the Move smart contract. In addition, the Move ecological public chain is diversified, each public chain has unique characteristics, and each public chain needs to be analyzed separately. Therefore, sandwich attacks are a security issue worth considering in the Move ecosystem.

About Us

Our vision is to improve security globally. We believe that by building this security barrier, we can significantly improve lives around the world.SharkTeam composes of members with many years of cyber security experiences and blockchain, team members are based in Suzhou, Beijing, Nanjing and Silicon Valley, proficient in the underlying theories of blockchain and smart contracts, and we provide comprehensive services including threat modeling, smart contract auditing, emergency response, etc. SharkTeam has established strategic and long-term cooperations with key players in many areas of the blockchain ecosystem, such as Huobi Global, OKX, polygon, Polkadot, imToken, ChainIDE, etc

Twitter：https://twitter.com/sharkteamorg

Discord：https://discord.gg/jGH9xXCjDZ

Telegram：https://t.me/sharkteamorg

web：https://www.sharkteam.org/

SharkTeam

In Math, We Trust!