Estimating Sandwich Attack Profits And User Losses

Overview

Zeromev uses Flashbots mev-inspect-py for MEV detection. While it works well, there are some issues with sandwich profit calculations which this document aims to address.

Profit Calculation Problem

- The simple way of calculating sandwich profit is backrun_out - frontrun_in
- Unfortunately, this calculation produces misleading results when the attacker takes a position while sandwiching (ie: is overweight on the frontrun or the backrun swap)
- This happens frequently, especially on the backrun
- While position taking can also be MEV, it is not strictly part of the sandwich, and must be estimated separately

Detecting Position Taking

- If the attacker's frontrun output does not equal their backrun input, then they will be left with a position:
    - If backrun_in > frontrun_out, profitability will be overestimated by the naive profit calculation, and the attacker is left with excess tokens from the backrun
    - If backrun_in < frontrun_out, profitability will be underestimated by the naive profit calculation, and the attacker is left with excess tokens from the frontrun

Method Overview

- In order to differentiate profits from position taking, we first recalculate the sandwich with the frontrun_out and backrun_in limited to each other
- Any remaining amounts can then be attributed to frontrunning and backrunning
- To do this exhaustively, we would need to recreate chain state at the time of the attack and replay the sandwich transactions through the EVM with modified parameters
- This is beyond the scope of what mev-inspect-py and zeromev currently offer
- Instead, we extract AMM pool parameters from the sandwich transactions and use these to replay the attack

- This technique also allows us to estimate user losses from the frontrun

Pool Extraction Method

- By taking the inputs/outputs of any two consecutive swaps on the same AMM pair, we can calculate with mathematical certainty what the Uniswap v2 pool state (x, y and k values) were just before the swaps
- Even where the transactions are not consecutive and/or where the sandwich uses AMMs other than Uniswap v2, we still get good estimates because most AMMs behave similarly and because we adjust for errors in the calculations (see Error Minimization)

Backrun Heavy Sandwich

- If backrun_in > frontrun_out, we set the backrun_in = frontrun_out and recalculate the swaps using extracted pool values
- sandwich profit = calculated_backrun_out - calculated_frontrun_in
- Remaining backrun = calculated_backrun_out - original_backrun_out

Frontrun Heavy Sandwich

- If backrun_in < frontrun_out, we set frontrun_out = backrun_in
- We then reverse swap the frontrun_in from this new frontrun_out and recalculate the swaps forwards again using extracted pool values
- sandwich profit = calculated_backrun_out - calculated_frontrun_in
- Remaining frontrun = calculated_frontrun_in - original_frontrun_in

User Losses

- We limit our analysis to the frontrun when calculating user losses, as it directly impacts the victim transactions that come after it
- Extract pool values from just before the frontrun
- Zero the frontrun_in amount and recalculate to find what each victim transaction would have received if they had not been frontrun
- User loss = calculated_victim_out - original_victim_out

Error Minimization

We can minimize any potential error from differing protocol mechanics and fee structures in the Pool Extraction Method as follows:

- First recalculate the swaps using unmodified in and out values (calculated original)
- Then modify the swap in and out values as required and calculate again (calculated new)
- Final calculated values = calculated new * (original value / calculated original)

Exchange Rates

Zeromev calculates USD exchange rates for sandwich profit calculations and arbs on a tick by tick basis directly from swap data with the following policies:

- Uniswap v2 and v3 are exclusively used for rates data
- The USD rate for any token is taken directly from the latest swap with a base currency
- Base currencies are USD stablecoins (Tether USD,USD Coin,Dai,Binance USD,Gemini Dollar) and WETH (calculated indirectly from the latest WETH:USD rate)
- Initial rates are only set when converting tokens to base currency (this is more trustworthy than swapping base currency to tokens)
- After this initial swap, rates can be set in both directions with rate changes >50% in a single tick filtered out to avoid bad results
- Profits/losses are calculated in USD rates immediately after the last transaction in an attack

Handling Outliers

These outlier cases are handled differently when calculating:

Pool Imbalance Attacks

- Pool imbalance attacks switch which AMM pool is larger and which is smaller during the sandwich and back again
- This often messes up user loss calculations which require an additional rate conversion, because the rate cannot be trusted while pools are being manipulated in this way
- When detected, user losses are taken to be the negative of the profit calculation instead, as profits do not require the additional rate conversion and are usually more trustworthy in this situation

Low Liquidity / Malicious Tokens

- Low liquidity and malicious tokens can lead to unexpected and inaccurate profit and loss calculations using the methods above
- Happily these cases are often wrapped in more liquid and well behaved swaps to allow the attacker to withdraw profits safely (eg: WETH > malicious token > sandwiched victim > malicious token > WETH)
- As such, when a wrapped sandwich is detected, we take wrapped_token_output - wrapped_token_input as profit and the negative of this as the user loss
- The naive calculation usually works fine here. This is because wrapped sandwiches tend to be well balanced as the attacker is trying to get their money out and not take a position in what is usually a base currency

Split Transactions

- mev-inspect-py expects frontruns / backruns to be single swaps
- In some cases, frontrun and backrun amounts are split over several swaps
- By coalescing swap amounts for the same pair within the same transaction, errors arising from this are avoided

Undetected Reverts

- Mev-inspect-py misses some transaction reverts which can lead to sandwiches being included which never occurred
- Zeromev removes these pending a fix for this in mev-inspect-py