Report

By Emma Tekulová, 12.09.2025, Munich

Intro + Problem statement

Before this project, I had only a vague understanding of blockchain. I didn't know how transactions were processed, what the mempool was, or that somebody could profit from ordering transactions strategically. This challange was really great experience where I learned a lot.

To achieve this I chose the begginer sample project where I focused on **sandwich attacks**, a common type of MEV strategy. These attacks involve placing a transaction **before** a victim trade (front-run) and **after** the trade (back-run) to exploit the price impact of the victim's trade.

The steps I followed were:

1. **Obtain blockchain transaction data** – explore tools like Dune and Ethereum-ETL to collect and export block and transaction data.
2. **Learn MEV concepts** – understand the mempool, gas fees, and common MEV strategies such as arbitrage, back-running, and sandwich attacks.
3. **Preprocess and analyze data** – clean, format, and explore blockchain data to identify relevant fields for detecting MEV activity.
4. **Identify potential sandwich attacks** – apply heuristic rules based on transaction ordering, gas fees,....
5. **Investigate individual cases** – verify suspicious addresses using Etherscan and observe patterns among potential attackers.
6. **Draw conclusions and summarize learnings** – reflect on the process, insights gained, and challenges encountered.

This project provided me a practical introduction to blockchain data analysis, allowing me to go from knowing almost nothing about MEV to detecting (maybe) real patterns of sandwich attacks on Ethereum.

1. Obtaining Data

This step was much harder than I expected. I first came across the portal Dune[1]. It was very useful for an initial data exploration because of its user-friendly GUI.

I started with a simple SQL query:

```
SELECT *
FROM ethereum.blocks
LIMIT 10;
```

time	number	gas_limit	gas_used	difficulty	total_difficulty	size	base_fee_per_gas	hash	parent_hash	miner	nonce	date	bl

From this I learned what kind of data is stored in blockchain blocks.
Next, I wanted to check how many blocks there are in total:

```
SELECT COUNT(*) AS total_blocks
FROM ethereum.blocks;
```

Query results Blockchain metrics

total_blocks
23340825

Because the dataset is too large, I decided to limit my analysis to blocks in the range **23300000–23300500**.

However, I could not download the data directly from Dune without paying. I searched for alternatives and eventually found the excellent library Ethereum-ETL[2].

After setting up the environment and installing the library, I only needed to create an Infura account to obtain an API key. With that, I could download data in batches of 100 blocks and save them as CSV files:

```
ethereumetl export_blocks_and_transactions \
  --start-block 23300401 --end-block 23300500 \
  --blocks-output blocks.csv \
  --transactions-output transactions.csv \
  --provider-uri https://mainnet.infura.io/v3/
```

The data came in nicely structured CSV tables, which made further analysis much easier.

2. What MEV is and what to do[3]

This is a short, summary of what I learned.

Mempool and gas tips

- Every transaction submitted to Ethereum first lands in the *mempool*, which is like a waiting room for transactions.
- Miners or validators choose which transactions to include in the next block. A higher *gas price* and *tip* make a transaction more attractive to a validator, so it will likely be processed faster and included sooner.

What MEV (Maximal Extractable Value) means

- MEV is the extra value that block producers or bots can extract by choosing *which* transactions to include and in *what order*.

**Common MEV strategies

- **Arbitrage** - finding and exploiting price differences for the same asset across different markets (for example, buying on one DEX where price is low and selling on another where price is higher).
- **Sandwiching** - inserting transactions *before* and *after* a target swap to profit from the price impact of the target transaction.

**Sandwich attack
A sandwich attack combines a front-run and a back-run around a victim's trade on a DEX:

1. Attacker sees a pending large swap in the mempool that will move the price.
2. **Front-run:** attacker submits a buy transaction with a higher gas tip so it gets mined *before* the victim. This pushes the price up.
3. **Victim's trade** executes at the worse price (higher cost because attacker already moved price).
4. **Back-run:** attacker immediately sells the tokens bought in step 2 at the now-higher price, locking in profit.
   **Key characteristics of sandwich attacks

- Targeted at large swaps
- Require mempool visibility so the attacker can see pending transactions
- Attacker must pay higher gas/tip to get their front-running transaction included before the victim.

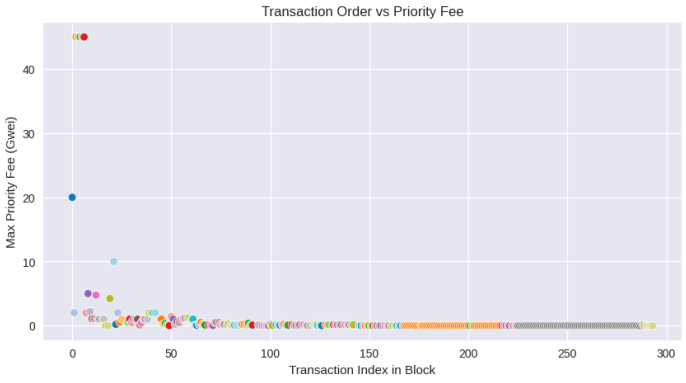
3. Data analysis

I use Jupyter Notebook with Python to preprocess the data. My preprocessing steps include: dropping columns that are mostly `NaN` and converting object or string values to `int` or `float` where possible.
Dataset overview:

- **Total transactions:** ~104,000 across blocks `23300000-23300500`
- **Time period:** 500 consecutive Ethereum blocks

Key fields for MEV analysis:

- `max_priority_fee_per_gas` and `max_fee_per_gas` : Indicators of potential MEV competition
- `transaction_index` : Position within the block, crucial for MEV analysis
- `value` : ETH transfer amount (many MEV transactions have `value = 0`)
- `to_address` : Target contracts
- `from_address` : Sender addresses
- `input` : Function calls and parameters



This plot shows the relationship between transaction priority fees and their order in the block. Transactions with higher priority fees (`max_priority_fee_per_gas` *) are included earlier, confirming that miners/validators prioritize these transactions.* Block number = 23300000

---

4. Sandwich Attacks

1. Filter Transactions

To focus on candidate swap transactions, the following filters are applied:

- **Remove plain ETH transfers**
  Transactions with `input = "0x"` are simple ETH transfers and cannot represent token swaps, so they are excluded.
- **Remove contract creations**
  Transactions where `to_address = None` indicate contract creation calls and are excluded from the analysis
- **Remove transactions with missing or zero priority fees**
  Only transactions with non-null and positive `max_priority_fee_per_gas` are considered, as priority fees are crucial for MEV analysis.

2. Candidate Swaps

- Transactions whose `to_address` corresponds to a known DEX router (Uniswap, SushiSwap, 1inch, etc.) are treated as primary swap candidates.
- This ensures the analysis focuses exclusively on transactions that could be potential sandwich victims.
- Candidate identification is performed manually after.

3. Gas-Price Heuristics

Sandwich attackers rely heavily on **priority fees** to manipulate transaction ordering:

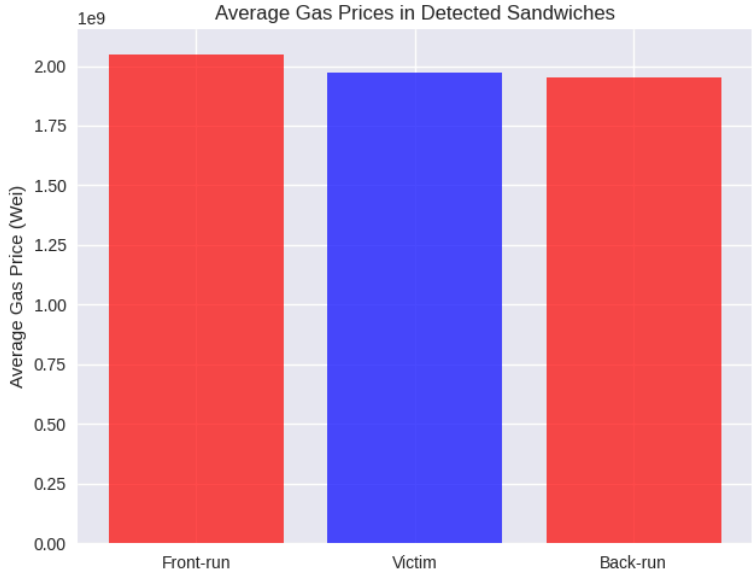
- **Front-run conditions:**
  - The attacker's front-run transaction must have a **higher priority fee** than the victim's transaction.
  - Ideally, the fees is in the **top 10% of gas prices** in the block, increasing the likelihood of early execution.
- **Back-run conditions:**
  - The back-run transaction should either originate from the same attacker (`back.from_address == front.from_address`) or be directed to the attacker (`back.to_address == front.from_address`).
  - Its priority fee should be roughly similar or slightly lower to the victim's to preserve correct execution ordering.

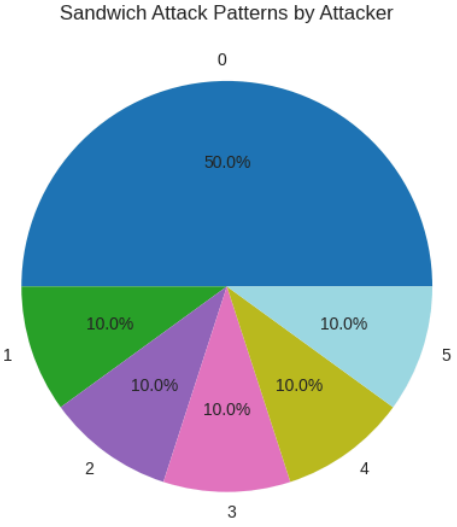
4. Block-Level Analysis

- For each block, the code evaluates **three consecutive transactions** (`front` , `victim` , `back`) to detect potential sandwich patterns.
- Only blocks containing at least three candidate transactions are considered.
- The 90th percentile of `max_priority_fee_per_gas` within a block is used to filter high-priority transactions, likely representing front-run attempts.

---

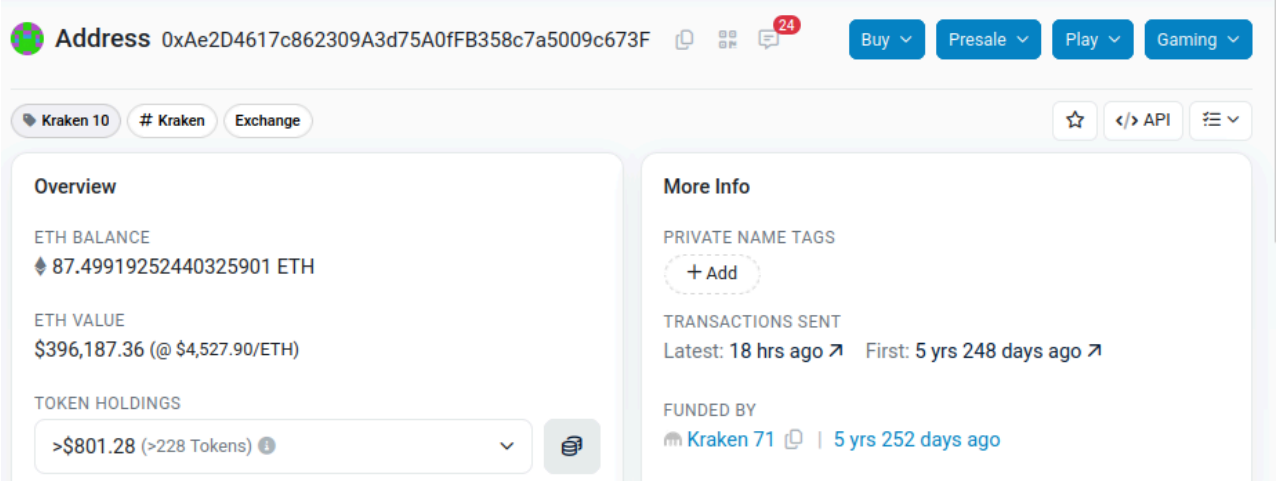
5. Output & Visualization





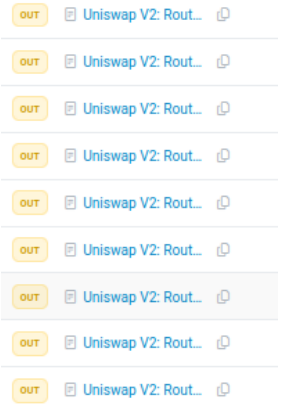
Potential sandwich attacks detected: 10, originating from 6 unique addresses.

Upon further investigation via Etherscan [4], it was found that 4 out of the 6 addresses belong to Kraken, a centralized exchange that often manages user funds through automated wallets. These are likely **not malicious**. Two addresses remain for further analysis. (The judge said can be, so these should also be investigated)



Etherscan shows it is funded by Kraken.

From the two one is also Kraken related.



Only one address draws attention as it does not belong to Kraken and exhibits multiple repeated swap transactions.

For example, address `0x65a8f07bd9a8598e1b5b6c0a88f4779dbc077675` displays a sandwich pattern:

Step	Wallet	Action	Effect
Front-run	`0x65A8F07B...`	Swaps 0.11592 ETH for 58,793 CHILLHOUSE on Uniswap V2	Buys tokens before victim trade
Victim	`0xbabe01c4...`	Swaps 4,445.8 CHILLHOUSE for 0.00885 ETH	Executes trade, moving the price
Back-run	`0x65A8F07B...`	Transfers 58,793 CHILLHOUSE → ETH	Captures profit from slippage

Sandwich Attack #3 – Block 23300130

Attacker: `0x65a8f07bd9a8598e1b5b6c0a88f4779dbc077675`

Step	Gas (wei)	Index	Description
Front-run	3,000,000,000	3	Attacker buys tokens before victim
Victim	2,202,462,127	4	Victim executes trade
Back-run	2,000,000,000	5	Attacker sells tokens after victim

Although the victim's transaction value is small, this example highlights a clear sandwich attack candidate that passes our filtering criteria.

5. conclusion

Most of the transactions I checked ended up being from exchange wallets, mainly **Kraken**, which is a big crypto exchange. These wallets didn't really show any MEV behavior, so they're probably harmless. After digging into the remaining two addresses on Etherscan, one was again linked to Kraken. That left just **one address** that showed signs of a sandwich attack-but even then, the amounts were pretty small.

Overall, this project taught me a lot. I improved my skills in **working with blockchain transaction data** and building basic analysis tools, and I started to get a better sense of how MEV can appear on-chain. It was a valuable hands-on experience to observe the mechanics behind front-running and back-running in practice.

CODE : https://github.com/emmatekulova/EF-Research-Challenge-x-TUM-Blockchain-Club/blob/main/analysis.ipynb

1. https://dune.com/hildobby/ethereum ↩
2. https://github.com/blockchain-etl/ethereum-etl ↩
3. https://info.arkm.com/research/beginners-guide-to-mev ↩
4. https://etherscan.io/ ↩