Proceedings of the 34th European Modeling & Simulation Symposium (EMSS), 41
19th International Multidisciplinary Modeling & Simulation Multiconference

2724-0029 © 2022 The Authors.
doi: 10.46354/i3m.2022.emss.041

Simulation of Front-Running Attacks and Privacy Mitigations in Ethereum Blockchain

Zachary Stucke[1], Theodoros Constantinides[1] and John Cartlidge[1,*]

[1]Department of Computer Science, University of Bristol, Bristol, UK

[*]Corresponding author. Email address: john.cartlidge@bristol.ac.uk

Abstract

Transactions sent to a public blockchain network, such as Ethereum, are initially held in the mempool before they are accepted in a block. While waiting in the mempool, these 'in-flight' transactions are publicly visible and vulnerable to front-running attacks, such that malicious parties use information in the transaction for their own gain and at a direct cost to the transaction owner. In this work, we introduce open-source simulation software for identifying and mitigating these attacks on Ethereum blockchains. Designed for education and research, the software introduces simple smart contracts that elaborate front-running vulnerabilities such as displacement attacks, sandwich attacks, and priority gas auctions. Users can run these attacks in a safe environment, monitor the detailed mechanics of attacks, and mitigate attacks using the MEV-geth protocol for in-flight transaction privacy.

Keywords: Blockchain simulation; decentralised finance; education software; front-running; smart contract

1. Introduction

When trading in financial markets, there are dangers in revealing your hand. Once you have publicly exposed your trading intention, it can be possible for others to profit at your expense by maliciously using your trading intention against you. A classic example of this is front-running. In traditional financial markets, front-running is often associated with intermediaries, such as a broker, using their client's trading information against them. For example, if a client instructs their broker to buy some reasonably large quantity of stock, knowing that the purchase will increase the market price the broker can first buy the stock on their own behalf and then sell the stock back to their client at the new higher price. The broker profits at their client's expense and the client is none the wiser that they have been duped. This practice is illegal, but can be difficult to prove unless performed in a naïve fashion. In traditional markets, so called "dark pool" trading venues are designed to stop front-running activity by enabling traders to post orders that are not displayed. In this way, pre-trade order information remains private and cannot be misused as long as the dark pool operator ensures that information does not leak (e.g., see Cartlidge et al. (2019)). However, incidents where dark pool providers have systematically abused their privileged access to front-run their own clients are depressingly common (e.g., see Cartlidge et al. (2021)). To address this issue, some works have employed multi-party computation to ensure no leakage through cryptographic methods (Cartlidge et al., 2019, 2021).

In decentralised financial markets – i.e., trading of cryptocurrency tokens on a public blockchain, such as Ethereum – security and privacy challenges are greatly exacerbated (e.g., see Massacci and Ngo (2021)). In particular, not only are transactions public and therefore vulnerable to front-running in the traditional sense, but also "in-flight" transactions (TXs) can be observed and attacked before they are confirmed (i.e., "mined") and added to the blockchain. When a TX is sent to the blockchain network, it first waits in the mempool to be confirmed. Each

TX has an associated gas price, which is effectively a "tip" paid to the miner for confirming the TX. Miners will select TXs from the mempool in order of gas price (i.e., selecting the largest tip first). An attacker can observe and exploit a TX waiting in the mempool by sending a new TX with higher gas price to gain priority and be confirmed first, therefore effectively front-running the original TX. This strategy opens up the possibility of a variety of attack strategies such as: *sandwich attacks*, similar to traditional front-running, where the attacker buys (sells) ahead of a large buy (sell) TX and then sells (buys) immediately afterwards; and *displacement attacks*, where the attacker identifies a profitable TX and sends the same TX with higher priority to steal the profit for themselves. Further, if a malicious miner decides to abuse their privileged status in the blockchain network, there are opportunities for additional attack vectors such as re-ordering or replacing TXs, or even performing *time bandit* attacks by mining competing blocks to take advantage of past opportunities (e.g., see Daian et al. (2020)).

Front-running attacks are common and varied in form. Eskandari et al. (2020) summarise attacks witnessed on the top 25 decentralised applications (DApps), including: EtherDelta, a more traditional exchange with a centralised component, being susceptible to spot price manipulation by attackers flooding the market with orders and then canceling them (*taker's griefing*), or by attackers front-running order cancellations (*cancellation grief*); Bancor, a fully decentralised exchange, being vulnerable to sandwich attacks; Fomo3D, a gambling application, succumbing to *suppression attacks*, whereby an attacker can flood the market with high gas transactions so that other users are unable to participate; and reward functions in CryptoKitties NFTs (such as giveBirth()) being susceptible to displacement attacks.

The dollar scale of front-running attacks in public blockchains is worryingly large. The term "maximal extractable value" (MEV; previously defined as "miner-extractable value") describes the deterministic value that can be extracted by agents from placing, reordering, and excluding transactions in a block (Daian et al., 2020). It has been estimated that more than USD \$500 million was extracted from the Ethereum blockchain over 2.5 years from sandwich attacks, liquidations, and arbitrage opportunities (Qin et al., 2022); with pure arbitrage profits alone estimated at USD \$1.6 million per year (Daian et al., 2020); and USD \$0.46 million extracted during the single *bZx* attack (Zhou et al., 2021). To mitigate these attacks, the *Flashbots Auction* (https://docs.flashbots.net/flashbots-auction/overview) network protocol has been introduced to provide a mechanism for agents to send TXs and have them included by miners without ever being visible in the public mempool. Resembling a first-price sealed-bid auction mechanism, Flashbots Auction attempts to eliminate front-running vulnerabilities by effectively creating a dark pool architecture. While the latest version of the software currently relies on trusted intermediaries (and is therefore susceptible to the same forms of information misuse that we see dark pool providers performing in traditional financial markets), the development road map aims for a fully permisionless system with no trusted intermediaries.

Contribution: In this work, we introduce a simple open source simulation toolbox for performing front-running attacks and mitigations in Ethereum blockchain. The simulation software is designed for educational purposes and has been used in teaching at the University of Bristol, UK, to elaborate smart contract vulnerabilities and mitigation strategies for students registered on the MSc in Financial Technology with Data Science. Despite the scale of front-running vulnerabilities in blockchain, there is a lack of educational resources to train the next generation of blockchain application developers in security vulnerabilities (see Section 2). To address this gap, we develop and release open source simulation code of simple smart contracts with known vulnerabilities, code to perform a selection of front-running attacks, and code to mitigate these attacks using the mev-geth protocol of Flashbots Auction (see Section 3). As a use case for the simulation code, we develop a web frontend dashboard UI for teachers and students to easily perform and inspect these attacks and mitigations (see Section 4).

For more details on this work, see the longer BSc project report written by Stucke (2022). Simulation code is available open source at: https://github.com/zakstucke/ethereum-front-running.

2. State of the art: software applications

There are a number of software bots available online that perform sandwich attacks on automated market makers by exploiting the mempool. The earliest example we were able to find is the Bancor exchange exploit on the Ethereum main net (https://github.com/bogatyy/bancor). A plethora of similar bots are also available, including the popular *Subway* bot (https://github.com/libevm/subway). Other publicly available bot strategies include *liquidity sniping* (https://github.com/Supercycled/cake_sniper) and *transaction pool sniping* (https://github.com/a04512/txPoolSniper). Additionally, there are many tools that can inspect and query the mempool and identify arbitrage opportunities, find evidence of front running, and discover tampering by miners, e.g., *mev-inspect-py* (https://github.com/flashbots/mev-inspect-py) and *Helios* (https://github.com/taarushv/helios). Most disturbingly, there are also a number of unethical paid services that give buyers access to proprietary bots (links not given).

However, none of these software tools are designed for educational purposes. There is a research gap for simulation software to aid learning about blockchain vulnerabilities and mitigation strategies.

Algorithm 1: Displacement attack

```
forkedNet = forkNet(net)
for tx in mempool do
    attacker.net, tx.net = forkedNet
    startBal = attacker.balance()
    tx.data["from"] = attacker.address
    tx.send()
    finalBal = attacker.balance()
    if finalBal > startBal then
        attacker.net, tx.net = net
        tx.gas_speed = GAS_FAST
        tx.send()
    end
end
```

Algorithm 2: Sandwich attack

```
for tx in mempool do
    if tx.data["to"] = simplePool.address then
        if simplePool.getFuncName(tx) =
         "swapEthForTokens" then
            tx.data["gasPrice"] = GAS_FAST
            tx.data["from"] = attacker.address
            tx.send()
            sellTx =
              simplePool.createTx("swapTokensForEth",
              attacker, GAS_SLOW)
            sellTx.send()
        end
    end
end
```

3. Simulation Software

To demonstrate front-running attacks, we have developed a simulation toolbox. Our toolbox is based on two vulnerable smart contracts, written in Solidity, which we use as examples to perform the attacks on. We then have additional code that performs the actions of an agent, simulating a normal user who wants to interact with one of the two contracts, and an attacker, who monitors the activity on the blockchain, looks for profitable opportunities and executes the appropriate transactions to capitalise on them. We also implement mitigation code to enable an agent to send their their transactions to a miner that supports the Flashbot's MEV-geth protocol, such that an attacker is unable to view those transactions in-flight. The code is open source and can be extended.

3.1. Displacement Attacks

A displacement attack occurs when an attacker completes a similar TX before the original agent can complete theirs. Algorithm 1 shows pseudocode for a simplified generalised displacement attack. For each TX in the mempool, the attacker replaces the TX "from" address with their own address, simulates the TX on a Ganache fork, and if profitable, displaces the original TX by sending a replacement with a higher gas price.

In order to demonstrate a displacement attack, we have created a simple "holder" contract that can receive funds from anyone using the `receiveFunds()` method, and will hold the funds until one of our accounts (either the attacker or the agent) extracts the funds by calling the `withdraw()` method. The `withdraw()` method of the contract provides the caller with the assets in the contract, if there are no assets in the contract the call will revert.

A real displacement attack was performed on the holder contract deployed on the Goerli testnet. The TXs of the attack are as follows:

- Attacker: `withdraw()`⇒SUCCESS. Gas price: 2.92 Gwei.
- Agent: `withdraw()`⇒REVERTED. Gas price: 1.25 Gwei.

We see that the attacker identified the agent's `withdraw()` TX in the mempool and performed a displacement attack by posting a `withdraw()` TX with higher gas price, therefore gaining priority and successfully withdrawing the funds from the contract. The agent's TX is subsequently reverted as there are no funds remaining in the contract. The agent loses both the funds and also the gas price paid for the transaction. In this simple context it is clear why the displacement attack is profitable (the attacker withdraws all funds from the contract before the agent has a chance to do so). However, notice that, in general, the attacker does not need to understand why the displacement attack will be successful, only that the TX returns a profit in simulation.

3.2. Sandwich Attacks

Algorithm 2 presents pseudocode for a simplified sandwich attack. For each TX in the mempool, if the attacker recognises the TX is interacting with a known liquidity pool and is a swap, it front-runs the TX with its own information with a higher gas price, then back-runs the TX with a reverse swap with a lower gas price.

To demonstrate sandwich attacks, we have implemented a simplified Automated Market Maker (AMM) contract that uses the constant product model to facilitate trades between ETH and a made-up internal token TOK, i.e., the contract is equivalent to a very simple liquidity pool or exchange that allows two cryptocurrency tokens to be swapped. The contract holds liquidity in the form of a balance of the token and ETH that is sent to it. The contract also keeps track of the balances of the agent and the attacker (in terms of the aforementioned token), and allows them to deposit funds into the contract in return for tokens, or to exchange their tokens for ETH.

Again, the Goerli testnet was used to perform a real sandwich attack. Using our code, we were able to identify the in-flight transaction of the agent in real-time and send the attacker's sandwich transactions. The TXs of the attack are as follows:

- Attacker: `swapEthForTokens()`⇒-0.05 ETH. Gas price: 3.22 Gwei.
- Agent: `swapEthForTokens()`⇒-0.05 ETH. Gas price: 1.78 Gwei.
- Attacker: `swapTokensForEth()`⇒+0.066 ETH. Gas price: 1.25 Gwei. Reverse swap profit: 0.016 ETH.
- Agent: `swapTokensForEth()`⇒+0.034 ETH. Gas price: 1.77 Gwei. Reverse swap loss: 0.016 ETH. NB: TX performed separately in different block, for completeness.

We see that the attacker identified that the agent has submitted a TX to swap ETH for TOK with gas price 1.78 Gwei. The attacker places a TX to make the same swap, but with higher gas price 3.22 Gwei in order to gain priority and be processed first. In this way the attacker front-runs the agent. The attacker also sends a TX to swap TOK back to ETH, this time with lower gas price (1.25 Gwei) than the agent's TX, to ensure that it is processed after the agent's TX has moved the swap price. In this way the attacker makes a profit of 0.016 ETH. For completeness, we also show that if the agent later swaps TOK back to ETH, they make a loss of 0.016 ETH, equal in magnitude to the profit taken by the attacker.

Given that Ethereum blocks are mined approximately every 13 seconds and the maximum size of a block is 30 million gas units, in the worst case scenario where every TX uses the minimum gas possible (21,000), there are a maximum of 1428 TXs in an Ethereum block. However, running our code on a local machine, it is currently possible to simulate a maximum of only 28 TXs from the mempool every 13 seconds. Also, if the attacker is to act upon an identified attack, even fewer TXs can be simulated as the period near the end of the block must be reserved for sending TXs. Since the chances of finding a profitable opportunity in the first few transactions of each block are slim, we were not able to use the software to simulate an attack on a real liquidity pool contract. Therefore, to demonstrate that a sandwich attack is possible, we simulated a manual attack on a Uniswap v2 liquidity pool on the Ropstein testnet using a series of four TXs that exchange Wrapped Ether (WETH) and Uniswap tokens (UNI), as follows:

- Attacker: `swap(10 WETH)`⇒5.76 UNI.
- Agent: `swap(35 WETH)`⇒19.26 UNI.
- Attacker: `swap(5.76 UNI)`⇒10.68 WETH. Reverse swap profit: 0.68 WETH.
- Agent: `swap(19.26 UNI)`⇒32.17 WETH. Reverse swap loss: 2.8 WETH. Presented for completeness.

We can see that the attacker profits by 0.68 WETH from the sandwich attack. The attack works on the real liquidity pool in the same way that we demonstrated with the simple simple AMM contract.

3.3. Attack Mitigation using MEV-geth

To implement a mitigation strategy for the previously described attacks, we use web3-flashbots to send the agent's transactions directly to the miners using the MEV-geth protocol, without passing through the mempool. Algorithm 3 shows pseudocode for sending TXs to miners that support the MEV-geth protocol. The transactions are bundled, and the bundle must have a minimum of 42,000 gas. A dummy TX will be included if the target TX does not meet this requirement. The bundle is sent and specifies the target block. Finally, the logic repeatedly attempts to include the TX in ten consecutive blocks until one TX succeeds.

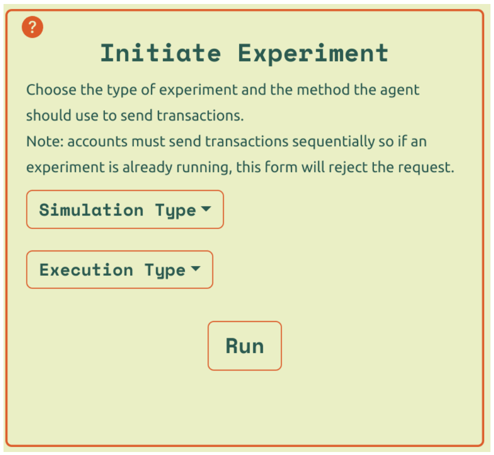
Algorithm 3: MEV-geth transactions

```
bundle = list(txData)
if txData["gas"] < 42000 then
    bundle = list(txData, createDummyTx())
end
blockNum = getCurrentBlock()
for x=blockNum; x+1; x < blockNum + 10 do
    success = sendFlash(bundle, blockNum + x)
    if success = True then
        break
    end
end
```

We chose to implement MEV-geth as this is the most mature protocol available for mitigating vulnerabilities in the mempool. Alternative mitigation strategies have been proposed, but none are suitable for our means. For instance, in their systematization of knowledge, Baum et al. (2021) classify three categories of front-running mitigations: fair ordering (e.g., see Kelkar et al. (2020)), batching of blinded inputs, and private user balances and inputs; the former preventing arbitrary transaction re-ordering, the latter two hiding the contents of the TXs and the user's intent. Fair ordering would require protocol-level changes such as "y-batch-order-fairness", where TX's are ordered based on when y% of nodes receive each TX; this is shown to introduce new TX front-runners that attempt to send a TX and meet the required threshold before the original TX can do so. During batching, users' TXs would be combined and sent through some intermediary smart contract, only releasing user balances once all TXs have been completed. Secret user input stores can also be maintained by DApps off-chain to prevent visibility until the completion of the operation, although this significantly reduces the permissionless nature of the DApp.

protect a specific smart contract.

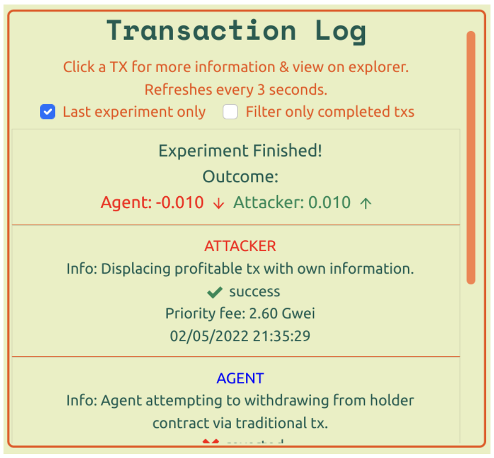
Alternatively, Heimbach and Wattenhofer (2022b) proposed DApp-specific mitigations for transaction reordering, such as an AMM automatically extracting any MEV created by an agent's TX and giving it to the agent themselves, or by redesigning Decentralised Crypto Exchanges (DEXs) to enforce fair ordering through off-chain implementations. They also propose a trusted centralised third party or an algorithmic committee as a fair ordering enforcer and discuss a potential model where a TX is guaranteed to only succeed if the blockchain state has not changed after submission. However, all methods put forward require either significant Ethereum protocol change



Initiate Experiment

Choose the type of experiment and the method the agent should use to send transactions.
Note: accounts must send transactions sequentially so if an experiment is already running, this form will reject the request.

Figure 1. Dashboard configuration.



Transaction Log

Click a TX for more information & view on explorer.
Refreshes every 3 seconds.

☑ Last experiment only ☐ Filter only completed txs

Experiment Finished!
Outcome:
Agent: -0.010 ↓ Attacker: 0.010 ↑

ATTACKER
Info: Displacing profitable tx with own information.
✓ success
Priority fee: 2.60 Gwei
02/05/2022 21:35:29

AGENT
Info: Agent attempting to withdrawing from holder contract via traditional tx.

Figure 2. Dashboard transaction log.

or specific smart contract implementations. Heimbach and Wattenhofer (2022a) also show that some production automated market makers (AMMs) are vulnerable to sandwich attacks and, as mitigation, they propose an optimal slippage tolerance algorithm that they encourage AMMs to adopt. Varun et al. (2022) train an LSTM to detect displacement, insertion, and suppression attacks and deploy the model in a smart contract to monitor and revert malicious transactions as they arrive. However, this approach will only protect a specific smart contract and not the Ethereum blockchain itself.

Eskandari et al. (2020) argue that traditional mitigations such as legal actions, dark pools, and sealed bid auctions do not work in a blockchain setting. They summarise solutions presented in the literature, including building DApps using design patterns, such as commit and reveal schemes or batch ordering, that prevent front-running. They also suggest that some forms of attack can be mitigated by setting up transactions in such a way that they only execute in a particular smart contract state and otherwise fail. However, once again these solutions will only

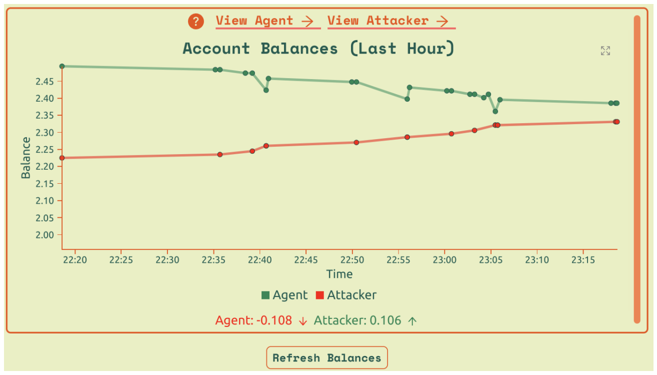


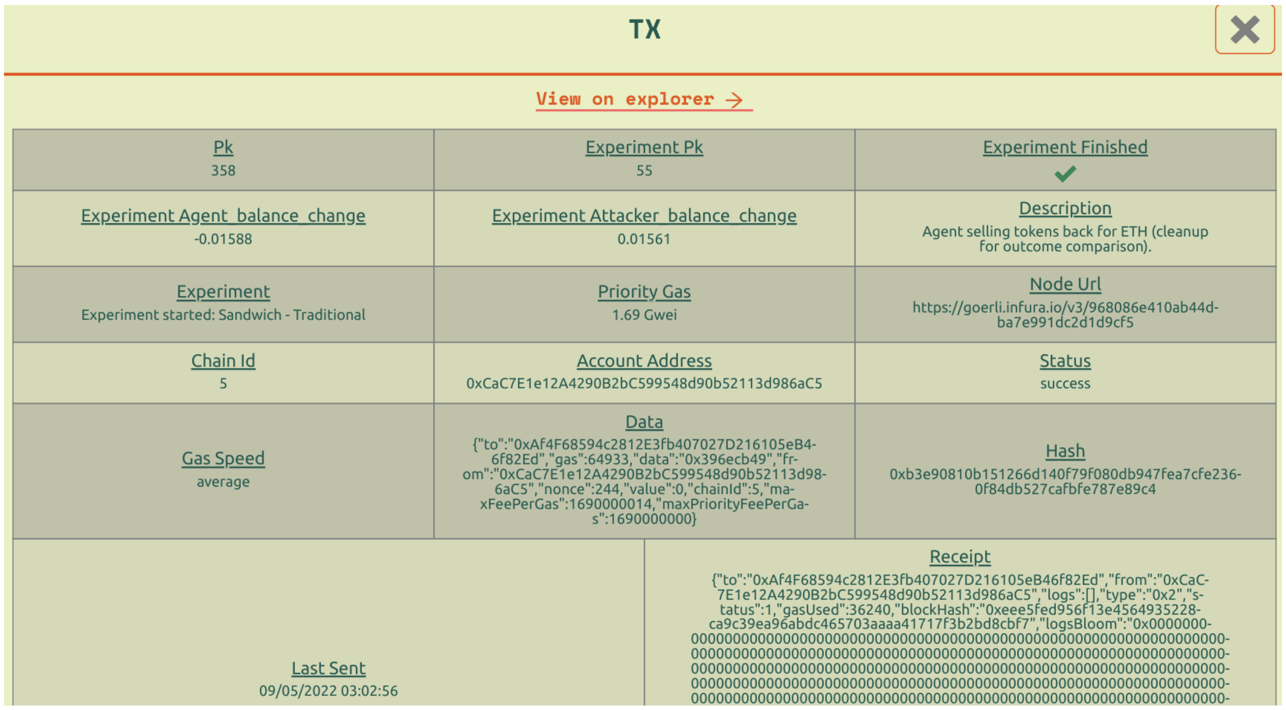
Figure 3. Dashboard balance monitor.

4. Use Case: UI Dashboard

Here, we extend the simulation code presented in Section 3. We present an interactive web interface that acts as a dashboard for educational purposes. The dashboard enables users to easily perform sandwich and displacement attacks on the real Goerli test-net and study the transactions that are produced. The dashboard visualises the interactions between agents and attackers during these experiments, the financial outcome of the attacks on all parties. The dashboard also demonstrates an attack mitigation, such that if an agent sends their transactions to a miner that supports the Flashbot's MEV-geth protocol, the attacker is unable to see those transactions and is thus unable to capitalise on them.

As shown in Figure 1, users are initially presented with a form to configure an attack simulation: displacement or sandwich. Users can also specify the execution type as "traditional", where TXs are sent using the canonical method such that they can be observed pre-completion in the mempool, or using "MEV-geth" protocol, which prevents pre-completion visibility and therefore stops the attacker from interfering. For simplicity, the system enforces a single experiment to be running at any one time. If an experiment is already running, this form will reject the request and inform the user to wait for a few minutes.

As shown in Figure 2, the dashboard provides a transaction log which presents a list of recent TXs. The list is refreshed every three seconds with the most recent TXs in the database; by default, the list is configured to show only the most recent experiment TXs but this can be changed to see up to 30 recent TXs across experiments. Incomplete TXs can also be filtered out of the list view. When showing for a specific experiment, the experiment configuration is included at the base of the list; upon completion, the agent and attacker outcome balances are also shown at the top of the list. In the example presented, we see that the latest simulation has completed, with the attacker gaining 0.010 ETH in profit from the agent. Initially, each TX shows its description, current status, priority fee, and send time. Upon clicking a particular TX, the complete information is provided in a pop-out (see Figure 4), including the TX data and a link to the TX on the explorer. This is only available


TX

View on explorer →

Pk 358	Experiment Pk 55	Experiment Finished ✓
Experiment Agent_balance_change -0.01588	Experiment Attacker_balance_change 0.01561	Description Agent selling tokens back for ETH (cleanup for outcome comparison).
Experiment Experiment started: Sandwich - Traditional	Priority Gas 1.69 Gwei	Node Url https://goerli.infura.io/v3/968086e410ab44d-ba7e991dc2d1d9cf5
Chain Id 5	Account Address 0xCaC7E1e12A4290B2bC599548d90b52113d986aC5	Status success
Gas Speed average	Data {"to":"0xAf4F68594c2812E3fb407027D216105eB4-6f82Ed","gas":64933,"data":"0x396ecb49","fr-om":"0xCaC7E1e12A4290B2bC599548d90b52113d98-6aC5","nonce":244,"value":0,"chainId":5,"ma-xFeePerGas":1690000014,"maxPriorityFeePerGa-s":1690000000}	Hash 0xb3e90810b151266d140f79f080db947fea7cfe236-0f84db527cafbfe787e89c4
Last Sent 09/05/2022 03:02:56	Receipt {"to":"0xAf4F68594c2812E3fb407027D216105eB46f82Ed","from":"0xCaC-7E1e12A4290B2bC599548d90b52113d986aC5","logs":[],"type":"0x2","s-tatus":1,"gasUsed":36240,"blockHash":"0xeee5fed956f13e4564935228-ca9c39ea96abdc465703aaaa41717f3b2bd8cbf7","logsBloom":"0x0000000-0000000000000000000000000000000000000000000000000000000000000000-0000000000000000000000000000000000000000000000000000000000000000-0000000000000000000000000000000000000000000000000000000000000000-0000000000000000000000000000000000000000000000000000000000000000-0000000000000000000000000000000000000000000000000000000000000000-	

Figure 4. Dashboard TX pop-out. Full TX information is shown along with a link to the TX on the blockchain explorer (if applicable).

when the TX is on the Goerli network and the TX has been completed.

As shown in Figure 3, the balance changes of both the attacker and agent wallets are visualised with a line graph. Transactions for each account are polled over the last hour. Balances are then calculated for the block height of each transaction; the balances at the start and end of the period are also shown. These balances can be refreshed using a button below the graph. Over the time period shown, the attacker has gained 0.106 ETH profit at a direct cost to the agent. Finally, the site page also contains an information tab that provides details on the smart contracts used and further information relating to the experiment and execution types.

5. Conclusion

We have introduced a minimal simulation framework for exploring front-running attacks and mitigation strategies in Ethereum blockchains. The holder smart contract used for displacement attacks and the AMM contract used for sandwich attacks are deliberately designed to be simple as the aim of the simulation software is for educational purposes. However, there are some limitations to the software and obvious areas for extension. We consider these below.

5.1. Dangers for the attacker

In the real-world, agents' TXs will be placed at random times during the mining of a block, i.e., the agent might place a transaction one second before the next block is mined and their transaction could be included in that block. Therefore, some TXs will mine before the attacker has noticed there is an opportunity, so the attacker will miss them. More dangerously for the attacker, some will mine after an attacker has initialised their attack. When this occurs, the attacker's TXs will still be included in the block, which might lead to a loss for the attacker. Furthermore, the attacker might itself be front-run by other attackers. The attacker could rectify these issues by operating through both an intermediary smart contract and utilizing MEV-geth. The attacker could create a contract that executes the attacking TXs but reverts upon an unprofitable outcome. Given reversions are not included when using MEV-geth, the attacker would utilise MEV-geth to send the TX to the intermediary contract. The TX would not be included if the attack would fail at the point of inclusion, effectively reducing the risk for the attacker to zero.

5.2. Mitigations are susceptible to malicious miners

Miners can be malicious themselves. It is assumed that TXs will be picked from the mempool by order of gas price, however, miners have the ability to re-order and replace TXs with their own in order to extract MEV themselves. MEV-geth provides no protection from miners, it simply protects agents from other attackers viewing the mempool; the miner who mines the block still has complete access. Whilst still providing significant protection from other agents in the system, the requirement to trust the miner is still a significant limitation in the current mitigations provided by MEV-geth. As of writing, the Flashbots organisation is upgrading its MEV-geth protocol continuously. Flashbots aim to eventually make the contents of a TX unavailable even to the miner; if this is successful, this limitation will be rectified.

5.3. Liquidity pools are less susceptible to attacks

The liquidity pools we consider in this work use the constant product model (CPM). Whilst this is the foundation of most liquidity pools, it's a simplification when it comes to the latest production pools. Uniswap is the largest AMM on the Ethereum network, and whilst its previous iterations used the CPM, its current v3 version utilises concentrated liquidity at specific price intervals. Significant upgrades to our algorithm design would be needed to work against production AMMs like Uniswap v3. Uniswap also implements a 0.3% fee per swap which is given to liquidity providers. This fee limits sandwich attack availability to reasonably large opportunities as the attack must profit at minimum 0.6% plus miner fees. Furthermore, agents can set price slippage tolerances on their swaps currently defaulting at 0.5%; if a sandwich attacker swaps first and the asset price changes by more than the configured tolerance, the agent's TX will revert, saving them from the attack. Whilst these mitigations do provide protections to naive sandwich attackers, as shown in Heimbach and Wattenhofer (2022a), complete protection is only provided based on variable, calculated slippage tolerances which AMMs do not provide by default to agents.

5.4. Simulation speed

Using Ganache to fork the blockchain and simulate transactions, like we do here, is relatively slow. If attempting to run an attacker in a production system, it would need to be simulating thousands of TXs per block, unavailable with the current setup without significant computational power. Whilst in its current form the system could analyse TXs and front-run them, it would not be able to analyse a significant quantity of unmined TXs before the block was mined and the pending TXs were included. Upgrading the method used for simulating TXs would be a desirable improvement as both forking and simulating TXs in the current environment take up significant time. Improvements could include: maintaining the current ganache fork, rebasing it on new blocks for each block simulation round; batching simulated TXs together when simulating multiple in a block along with a method to separate their outcomes or potentially using a local node instead for simulating TXs, discarding Ganache entirely.

5.5. Knowing what information to change

In generalised displacement attacks on the holder contract, the attacker can alter the TX with their own information by simply altering the "from" data parameter with their own address. In this scenario, this is all that is needed to make a profitable TX for the attacker. However, it may often be the case that contract method parameters need altering to the attacker's information. Current functionality could be extended to test multiple different combinations of updated parameters and untouched parameters, as some fields may need to be left untouched to reach a profitable TX. Contract parameter information is encoded using the contract's ABI; if the ABI is not known, it is not possible to decode the parameter inputs and re-encode with the attacker's. The lack of ABI can be mitigated by querying sites like Etherscan or by decompiling the contracts. For sandwich attacks, the same problem is true but to a lesser extent. In the current implementation, we need to know the address and the ABI of the AMM contract to perform the attacks. This is not such a huge issue in this case, as these contracts should be accessible by their users and as such that information is usually public.

5.6. Mempool visibility

We are currently using Infura as our provider. The RPC endpoint *txpool_content* is not available when using Infura nodes. Given the agent and attacker are both operating from the same system in our experiments, we can store the pending transactions locally and use that as an approximation of the mempool. However, if this were a real environment, this approach would not work as the local pool would have no visibility into other agents' TXs. This is a severe limitation of the current implementation. To remedy this, another node provider that implements the RPC endpoint *txpool_content* would be required, or a full local node instance would need to be run to access the endpoint directly.

5.7. Attacks that analyse blockchain state

Currently, we do not look into attack styles that analyse the current state of the blockchain and previously mined TXs to identify opportunities. For example, if an agent were to identify the holder contract from before as insecure, they could just monitor for TXs that send funds to the contract, and instantly realise that a profitable opportunity to withdraw the funds is available without having to monitor the mempool at all. An attacker could build a system to automatically test contracts on the blockchain, find these vulnerabilities and then monitor them until they become profitable. Mempool based attackers would not be able to compete in these scenarios as there are no TXs to analyse.

Funding

This work received financial support from UKRI through Innovate UK and the Economic and Social Research Council (ESRC) Knowledge Transfer Partnership between University of Bristol and Claritum Limited [KTP grant number 11952].

References

Baum, C., Chiang, J. H., David, B., Frederiksen, T. K., and Gentile, L. (2021). Sok: Mitigation of front-running in decentralized finance. Cryptology ePrint Archive, Paper 2021/1628. https://eprint.iacr.org/2021/1628.

Cartlidge, J., Smart, N. P., and Talibi Alaoui, Y. (2019). Mpc joins the dark side. In *ACM Asia Conference on Computer and Communications Security*, Asia CCS '19, page 148–159. https://doi.org/10.1145/3321705.3329809.

Cartlidge, J., Smart, N. P., and Talibi Alaoui, Y. (2021). Multi-party computation mechanism for anonymous equity block trading: A secure implementation of Turquoise Plato Uncross. *Intelligent Systems in Accounting, Finance and Management*, 28(4):239–267. https://doi.org/10.1002/isaf.1502.

Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., and Juels, A. (2020). Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE Symposium on Security and Privacy (SP)*, pages 910–927. https://doi.org/10.1109/SP40000.2020.00040.

Eskandari, S., Moosavi, S., and Clark, J. (2020). Sok: Transparent dishonesty: Front-running attacks on blockchain. In *Financial Cryptography and Data Security*, volume 11599 of *LNCS*, pages 170–189. https://doi.org/10.1007/978-3-030-43725-1_13.

Heimbach, L. and Wattenhofer, R. (2022a). Eliminating sandwich attacks with the help of game theory. In *ACM Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 153–167. https://doi.org/10.1145/3488932.3517390.

Heimbach, L. and Wattenhofer, R. (2022b). Sok: Preventing transaction reordering manipulations in decentralized finance. arXiv. https://arxiv.org/abs/2203.11520.

Kelkar, M., Zhang, F., Goldfeder, S., and Juels, A. (2020). Order-fairness for byzantine consensus. In *Advances in Cryptology – CRYPTO 2020*, volume 12172 of *LNCS*, pages 451–480. https:doi.org//10.1007/978-3-030-56877-1_16.

Massacci, F. and Ngo, C. N. (2021). Distributed financial exchanges: Security challenges and design principles. *IEEE Security & Privacy*, 19(1):54–64. https://doi.org/10.1109/MSEC.2020.2994826.

Qin, K., Zhou, L., and Gervais, A. (2022). Quantifying blockchain extractable value: How dark is the forest? In *IEEE Symposium on Security and Privacy (SP)*. Preprint available: https://arxiv.org/abs/2101.05511.

Stucke, Z. (2022). Generalised front-running attacks in blockchain: Building, formalising and mitigating generalised front-running techniques in blockchain environments. Bachelor's thesis, Department of Computer Science, University of Bristol, UK. https://github.com/zakstucke/ethereum-front-running/blob/main/StuckeBSc.pdf.

Varun, M., Palanisamy, B., and Sural, S. (2022). Mitigating frontrunning attacks in ethereum. In *ACM International Symposium on Blockchain and Secure Critical Infrastructure*, BSCI '22, page 115–124. https://doi.org/10.1145/3494106.3528682.

Zhou, L., Qin, K., Cully, A., Livshits, B., and Gervais, A. (2021). On the just-in-time discovery of profit-generating transactions in DeFi protocols. In *IEEE Symposium on Security and Privacy (SP)*, page 919–936. https://doi.org/10.1109/SP40001.2021.00113.