A Large Scale Study of the Ethereum Arbitrage Ecosystem

Robert McLaughlin, Christopher Kruegel, Giovanni Vigna
University of California, Santa Barbara
{robert349, chris, vigna}@cs.ucsb.edu

Abstract

The Ethereum blockchain rapidly became the epicenter of a complex financial ecosystem, powered by decentralized exchanges (DEXs). These exchanges form a diverse capital market where anyone can swap one type of token for another. Arbitrage trades are a normal and expected phenomenon in free capital markets, and, indeed, several recent works identify these transactions on decentralized exchanges.

Unfortunately, existing studies leave significant knowledge gaps in our understanding of the system as a whole, which hinders research into the security, stability, and economic impacts of arbitrage. To address this issue, we perform two large-scale measurements over a 28-month period. First, we design a novel arbitrage identification strategy capable of analyzing over 10x more DEX applications than prior work. This uncovers 3.8 million arbitrages, which yield a total of $321 million in profit. Second, we design a novel arbitrage opportunity detection system, which is the first to support modern complex price models at scale. This system identifies 4 billion opportunities and would generate a weekly profit of 395 Ether (approximately $500,000, at the time of writing). We observe two key insights that demonstrate the usefulness of these measurements: (1) an increasing percentage of revenue is paid to the miners, which threatens consensus stability, and (2) arbitrage opportunities occasionally persist for several blocks, which implies that price-oracle manipulation attacks may be less costly than expected.

1 Introduction

Decentralized Finance (DeFi) is an alternative financial infrastructure primarily run on the Ethereum blockchain (the blockchain) [46]. The blockchain is fundamentally a distributed state machine intended to facilitate transaction settlement between non-trusting parties [52]. Financial assets and instruments alike execute on the blockchain as "smart contracts" – small interoperable programs that run on the Ethereum Virtual Machine (EVM) [46, 52].

For example, USD Coin [21] (USDC) is one such ownable asset represented by a smart contract on the blockchain. That smart contract uses an interface compliant with the ERC-20 Token API [50], which is *de facto* standard for financial instruments of this type. In particular, this standard interface includes functions to check an account's balance and transfer value from one user to another.

Other financial instruments are likewise deployed as smart contracts on the blockchain. These include exchanges [4, 16, 20], leverage providers [3], derivatives [17], loans [2], and other applications.

In this work, we focus specifically on a type of transaction, arbitrage, performed with a specific type of exchange, an Automated Market Maker (AMM). Arbitrage is defined as the simultaneous purchase and sale of the same asset in two different markets for advantageously different prices, and is widely considered a benign, yet critical element of modern efficient markets [40, 47]. An AMM, put simply, is a smart contract that enables users to swap one token for another at an automatically determined price. AMMs are the most popular form of exchange in the blockchain ecosystem and they represent a very large Total Value Locked (TVL). For example, the Uniswap v2 [20] AMM has more than $5 billion of value deposited at the time of writing [7].

A recent publication from Zhou et al. made the shocking finding that by using a simple arbitrage-detection algorithm one could feasibly generate a weekly revenue of $76,592 [55]. Moreover, Qin et al. found that, over a study period of 32 months, $277.02 million in value was extracted by arbitrageurs [44]. Making matters yet more dramatic, Wang et al. found that unexploited arbitrage opportunities consistently yield more than 1 ETH (at the time of their publication, about $4,000) [51]. Daian likewise finds that arbitrageurs are not only pervasive but also compete with each other aggressively [33].

However, we need more methods and materials to properly understand the current situation of arbitrage on the blockchain, above and beyond what has been studied by past works, which have a number of limitations. Zhou's method of arbitrage opportunity detection is engineered to be real-time capable, but this is accomplished by limiting the focus to a small subset of decentralized exchanges, which results in missing substantial portions of the arbitrage phenomenon. Daian's work determined that arbitrage bots bid reactively to one another, but since then, Qin notes the emergence of a centralized, private transaction relayer, called FLASHBOTS. This destroys one arbitrageur's ability to react to the actions of another, forcing a new type of fee bidding, which must be revisited. Qin's recent study performs a broad-based identification of profit-generating transactions, but in doing so settles for applying application-specific knowledge of a handful of AMMs in order to detect arbitrage, missing important parts of the exchange ecosystems. Lastly, Wang's study revealing persistent, high-value arbitrage opportunities needs to be properly explained, so that we can assess the security and economic implications of an inefficient market in terms of the ability of an attacker to maliciously influence price oracles.

In this work, we perform two key measurements over a study period of about 28 months, ranging from block number 9,569,113 (February 28, 2020) to block number 15,111,876 (July 10, 2022) using novel methods to address these gaps and expand the community's knowledge of arbitrage.

First, we perform a historical review of the actions real arbitrageurs have taken on the Ethereum blockchain. We use a novel method to perform application-agnostic identification of AMM-based arbitrages, which is able to account for ten times more AMM applications with respect to prior work and identifies 4 million arbitrages performed on the blockchain.

Second, we take the six most arbitraged AMM applications, as identified by activity from the prior measurement, and we run a novel arbitrage opportunity detection system over the study period. This yields 4 billion arbitrage opportunities. We then selectively execute each opportunity on a private fork of the blockchain, as if it were to have run at that point in history, showing that these opportunities would have generated a weekly profit of 395 Ether, a much larger figure than what was previously computed.

In summary, our contributions are the following:

- We present an application-agnostic method of identifying arbitrage transactions on the blockchain, which automatically recognizes arbitrage across ten times more applications than prior systems.
- We design and construct a system capable of identifying and profit-maximizing arbitrage opportunities. To our knowledge, this is the first study that designs such a system that is (i) capable of efficiently exploring large numbers of exchanges, (ii) supports exchanges that do not use the "constant-product" pricing invariant, and (iii) is capable of reasoning about "fee-on-transfer" tokens.
- We present several findings from both systems, including (i) increasing share of arbitrage revenue sent to the block producer as fees, which threatens consensus stability by simplifying "time-bandit" attacks, (ii) arbitrage used as a tool in sandwich-attacks, which requires specific attention when measuring arbitrage, and (iii) a typical arbitrage opportunity duration of 1 to 3 blocks, which significantly reduces the expected cost of price-oracle attacks as modeled in prior work [38].

We make our source code and artifacts available at https://github.com/ucsb-seclab/goldphish.

2 Background

The Ethereum blockchain is a distributed, permissionless, strictly ordered ledger of transactions. Networked participants coordinate with each other in a consensus protocol to emit blocks that bundle transactions together. When a user requests the inclusion of a transaction in the blockchain, the transaction is added to the public *mempool*, waiting to be selected by a block producer. Transactions have an associated incentive that expresses the willingness of the transaction originator to pay a premium to the block producer to get their transaction selected for inclusion. However, each block producer has the exclusive privilege of deciding which transactions are included in a block and in which order the transactions occur. Also, a block producer can include in a block its own transactions or transactions that have not been previously added to the mempool.

Transactions. Ethereum transactions can be of three types: they either (i) send Ether – Ethereum's native currency – from one address to another, (ii) deploy a small piece of software, called a "smart contract," or (iii) execute a smart contract's function with user-supplied input data.

Smart Contracts. Transactions (ii) and (iii) execute on the Ethereum Virtual Machine (EVM), a deterministic and Turing-complete computing environment. Every smart contract has a unique *address* on the blockchain, and its code is immutable. When a transaction invokes a smart contract's function the corresponding EVM bytecode is executed, which may modify the smart contract's state, send or receive Ether, emit event logs, and invoke other contracts. Every instruction on the EVM costs a fixed amount of *gas*. After the EVM exits the transaction, the transaction originator pays a fee equal to the amount of gas times a *gas price*.

Tokens. The EVM code in a smart contract can serve several high-level purposes, such as managing loans, implementing games, or supporting a voting system. A very common form of a smart contract is used to issue and track ownership of an asset. Such contracts are typically called *tokens*, and ERC-20 [50] standardizes their function interface and behavior. These tokens have become extremely popular – there are currently more than $400,000$ token smart contracts deployed on the Ethereum blockchain – and they can have a substantial economic value – at the time of writing, the USDC token [21] has a market capitalization of over $40 billion. In addition, Ether, the native currency of the Ethereum blockchain, is often traded in its "tokenized" form, called *Wrapped Ether* (WETH).

DeFi. Decentralized Finance (DeFi) is an ecosystem of smart contracts that operate advanced, composable financial functions on the blockchain. In this paper, we specifically focus on decentralized exchanges (DEX) – a type of smart contract that allows one to swap a token for another type of token (or a token for Ether). In particular, we will focus on a type of DEX called an Automated Market Maker (AMM).

AMM Structure. AMMs are a new type of exchange that does not maintain buy-and-sell order books – i.e., they do not match buyers and sellers. Instead, an AMM maintains a balance of the tokens it trades, i.e., a *liquidity pool*. When a user wishes to execute a swap, they invoke the AMM's smart contract; the AMM then automatically determines a fair price, accepts tokens from the user as payment, deducts some fees, and then invokes the *transfer* function on the purchased token to send some of the liquidity pool's balance to the user. An AMM's liquidity pool is maintained by *liquidity providers*, who deposit tokens into the AMM in exchange for a proportional share of the fees collected on each swap.

An AMM determines price according to its *swap invariant*. In Section 4, we make use of three swap invariants: constant-product [31], weighted constant-product [39], and Uniswap v3's bounded-liquidity constant-product [27]. Other invariants, such as StableSwap [26], have also been proposed.

Briefly, a constant-product AMM permits any swap of tokens α and β such that

$$R_\alpha R_\beta \leq (R_\alpha + n_{in})(R_\beta - n_{out}),$$

where R_α and R_β are the contract's reserves of tokens α and β before the swap, n_{in} is the amount of token α paid to the AMM, and n_{out} is the amount of token β sent to the user by the AMM.

Creating an AMM. AMMs commonly follow the *factory* pattern – anyone can add support for trading between a given pair of tokens by invoking the factory smart contract. Then, the factory smart contract deploys on the blockchain a new contract that trades the specified pair of tokens. The liquidity pool starts at zero balance, and users who wish to be liquidity providers then begin adding tokens. At the time of writing, we measure that the Uniswap v2 factory has deployed over 100,000 of such token-pair contracts.

Arbitrage. Arbitrage is "the process of earning risk-less profits by taking advantage of differential pricing for the same ... asset or security" [47]. This is a common and expected component of capital markets and develops naturally whenever exchange prices for the same asset deviate significantly.

Arbitrage on AMMs. AMMs offer a unique opportunity for those seeking to make nearly risk-free profits. Because the smart contracts running these AMMs are immutable, one can anticipate an AMM's pricing model and construct a profitable arbitrage transaction when an asset has different prices on different AMMs. Moreover – since transactions are atomic – an arbitrageur can interact with each exchange, one after the other, within a single transaction and ensure that everything executes appropriately. If anything is not as expected, then the transaction is reverted, and no exchanges take place. This helps guard against the risk of unfavorable AMM price changes between the time of transaction construction and its execution in the blockchain. Arbitrageurs typically deploy a utility smart contract on the blockchain that manages trades and routes value to and from exchanges. These utility contracts are simple executors and are controlled by off-chain bots that implement the sophisticated financial analysis logic that detects arbitrage opportunities.

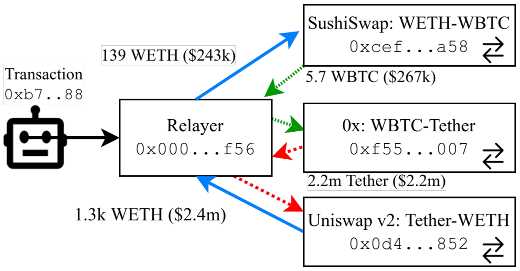

Figure 1: Arbitrage transaction in February 2021, which profits the bot operator $2 million in a single atomic transaction. An arbitrage bot sends a transaction invoking their relayer contract on the blockchain. Then, the relayer sends 139 WETH from itself to SushiSwap's WETH-WBTC liquidity pool as payment for 5.7 WBTC tokens. Next, the relayer uses the 5.7 WBTC tokens to buy 2.2 million Tether tokens via 0x exchange. Last, the relayer sends the 2.2 million Tether tokens to Uniswap's Tether-WETH liquidity pool as payment for 1,352 WETH. Sushi Swap sends the WETH back to the relayer, and the bot profits about 1,373 ETH ($2 million). This was made possible by a counter-party's very large order on the 0x exchange, which swaps WBTC for Tether well above the prevailing market rate.

We include a concrete example of such an arbitrage in Figure 1, which yields the bot owner $2 million.

FLASHBOTS. The FLASHBOTS system was motivated as a response to Daian's [33] work, which shows that high activity among arbitrage bots leads to network congestion, as competing bots try to bypass each other to capture arbitrage opportunities that are published in the mempool. The FLASHBOTS system is a private, centralized, third-party relay between bot operators and block producers. Bot operators are able to use this system to purchase priority placement of their transactions near the top of the block without going through the mempool, preventing other bots from stealing the arbitrage opportunity by offering higher incentives to the block producers.

On Proof-of-Stake. While preparing this publication, the Ethereum network changed the consensus mechanism from Proof-of-Work to Proof-of-Stake, in an event called *The Merge*. While our data was collected in the Proof-of-Work period, we re-run the analyses described in the remainder of this paper over a span of 14 days and find that the situation is fundamentally unchanged. Arbitrages still occur with approximately the same frequency as before. Furthermore, the Ethereum community remains concerned about the impact of excessive arbitrage activity and other profit-generating transactions, even more so now that this activity might create "economies of scale" that centralize capital among the group of already-wealthy staking block producers [14].

3 Executed Arbitrages

In this section, we present a novel analysis approach to identify arbitrages that were executed on the Ethereum blockchain. We run this analysis over a study period of about 28 months – from block 9,569,113 (February 28, 2020) to block 15,111,876 (July 10, 2022), encompassing about 1 billion transactions. We then analyze the results and discuss several findings.

3.1 Identification Algorithm

One method of identifying arbitrage transactions is to parse and analyze application-specific event log structures that are emitted by DEXs when executing a (token) swap. This strategy is employed by Qin [44], Wang [51], and Daian [33] alike. This method is fast and simple, but it lacks generality because these *swap* event logs are not standardized. Consequently, a non-trivial amount of manual effort is required *for each individual DeFi application* to extract a high-level financial interpretation of the activity that occurred within a transaction. This strategy does not scale as the DeFi ecosystem grows, since expanding the scope of the analysis requires both the knowledge of the mere existence of these applications and manual effort to interpret their swap event logs.

To broaden our view of on-chain arbitrage activity, we use an application-agnostic identification strategy that lifts this limitation. Our approach makes the following assumptions:

1. All arbitrages are atomic – i.e., the multiple exchange operations that execute the arbitrage are wholly contained within a single transaction that executes the entire flow.

2. All assets transferred in an arbitrage trade conform to the ERC-20 standard [50], which states that tokens emit *Transfer* event logs whenever value is sent from one address to another. Different from application-specific *Swap* event logs, *Transfer* event logs are standardized, so we use them as the basis for our analysis.

3. A DEX exchange executes a swap by accepting one token and emitting another one.

Given these assumptions, we design a new algorithm to identify arbitrage transactions, leveraging standardized ERC-20 *Transfer* events that are found within a given transaction. Intuitively, we extract meaning on two levels: first, we infer exchange (swap) operations, and then we extract the financial strategy that was performed.

Our algorithm operates in four steps. We begin by inferring a high-level interpretation of the exchange operations that a transaction executes. Next, we construct a directed graph indicating the flow of value. Then, we perform cycle detection on this graph to determine if the value flows in a closed loop, a key indicator of arbitrage. Last, we analyze the cycles to extract information about the arbitrage. The details of these four steps are discussed below.

The algorithm runs once for each transaction on the blockchain and outputs whether that particular transaction was or was not an arbitrage – and if so, what strategy was used.

Step 1: Exchange Inference. We examine all ERC-20 *Transfer* events and, for each kind of token, sum the net value *sent from* and *received by* each address. We infer likely exchanges in the transaction by identifying any address that accepted exactly one kind of token and emitted exactly one other kind of token. We perform this anew every single transaction. This process is similar to that of Wu [53], who also infers exchange operations using ERC-20 *Transfer* events, but for the purpose of identifying price-manipulation attacks.

Step 2: Graph Construction. We build a directed multi-graph G where the vertices $v_i \in V$ are kinds of tokens, and the edges $v_i \rightarrow v_j \in E$ are the inferred exchanges from the prior step. Edges are drawn from the payment token (net value received) toward the emitted token (net value sent.)

Step 3: Cycle Detection. We run Johnson's cycle-detection algorithm [36] to look for loops in the multi-graph G constructed in the prior phase. A loop of edges $v_0 \rightarrow v_1, \ldots, v_j \rightarrow v_0$ indicates an arrangement of exchange operations such that each exchange feeds into the next, and the first kind of token sent as payment is the same as the final one emitted. If no cycle is found, then the transaction is labeled as *not an arbitrage* and the procedure completes.

Step 4: Cycle Analysis. If only a single cycle is found, then we ensure that there exists a token that appears to have been both bought and sold for differing prices – forming an arbitrage. We call this token the *pivot token*.

Occasionally, multiple tokens meet this definition. For example, a "fee-on-transfer" token may charge 2% tax on each transfer. If 100 units of the token are bought on a DEX, then 98 units of the token are received by the trade executor contract. The executor then sells 96.04 units of the token to the next exchange, appearing as if they have taken 1.96 tokens as profit – but, in fact, that was taken as a fee. To account for these cases, we adopt this simple heuristic: when multiple tokens appear to show a profit, we select as the *pivot token* the one with the *highest* profit.

Example. Consider a transaction with the following ERC-20 *Transfer* events: Alice sends 100 token α to address "AMM1," address "AMM2" sends 120 token α to Alice, and "AMM1" sends 200 token β to "AMM2."

Each of the addresses "AMM1" and "AMM2" accepted one type of token, and emitted a different type, so we mark them as potential exchanges. Then, we generate a graph with the directed edges $\alpha \rightarrow \beta$ and $\beta \rightarrow \alpha$ (i.e., the directions of token movements through exchanges). We can clearly see the cycle: α, β, α. Finally, we notice that Alice both sold 100 units of token α, and bought 120 units of token α. We label α as the pivot token, describe the arbitrage as profiting Alice 20 units of α.

Limitations of Identification Algorithm. Our approach to arbitrage detection allows for a more comprehensive characterization of on-chain arbitrage activity compared to previous approaches that leveraged application-specific *swap* event logs. However, this generality comes with some limitations.

First, because this strategy of analysis is blind to the movement of (un-wrapped) Ether, we may experience both false-positive and false-negative detections of some exchanges. For example, Uniswap v1 is written to trade ERC-20 tokens for Ether, but we cannot detect these exchanges. In Appendix A, we evaluate the risk of false-positive detections and determine that this risk is minimal.

Second, because we do not parse application-specific logs, it is possible that some actions are mislabeled. For example, a "wrapping" smart contract may accept deposits of one token and then emit a different token always in a 1:1 ratio, which we may mislabel as an exchange. Our exchange attribution in Section 3.2.3 only finds a small number of unexplained exchanges, so we accept this as a small but possible source of error.

Lastly, we make the assumption that an AMM contract must use an ERC-20 *Transfer* to emit tokens. This assumption is violated, for example, by Balancer v2 [4]. All Balancer v2 AMM contracts keep their balances in a "vault" smart contract, which centralizes token custody for all of their application's liquidity pools. This is contrary to the design of Uniswap v2 and v3, for example, where every trading pair of two tokens has a separate contract, and balances are kept under the custody of that contract. Token transfers from one Balancer v2 exchange to another Balancer v2 exchange are accomplished by internal accounting in the "vault," without performing an ERC-20 *Transfer* to change token custody. However, *Transfer* events are still emitted when Balancer v2 emits tokens externally (see Figure 2). We accept as a source of error that Balancer v2's activity (and potentially the activity of other, similar exchanges) will be under-represented.

3.2 Results

We execute our identification algorithm on every transaction within the study period – about 1 billion. This results in 4,070,938 arbitrage transactions.

Based on these identified arbitrage transactions, we ask ourselves the following questions: What strategies are used to perform arbitrage? How much money is made by arbitrageurs? What are the security implications of arbitrage?

3.2.1 False Positives and False Negatives

We immediately notice a high number of transactions with a surprisingly large number of arbitrage loops – up to 15 contained within just one transaction. We manually inspect a subset of these and identify 193,941 false positives arising from the exchange aggregators CoW Swap [5], Dexible [8], and Tokenlon [19]. Those applications bundle users' exchange orders into one transaction, which creates several spontaneous arbitrage-like patterns of token movement. However, users do not control this bundling behavior, so we dismiss these transactions as false positives, reducing the results to 3,876,997 transactions.

We manually inspect a random sample of 100 arbitrages. 93 samples are completely correct (true positives). We find three false positives due to other exchange aggregators (in addition to the three mentioned above). Additionally, four instances are correctly identified as arbitrages, but have a minor classification issue. Specifically, three of these four arbitrages incorrectly identified the profit-token because of a fee-on-transfer not observed in event logs, and one arbitrage was labeled as single-cycle but in fact contains two (one uses raw Ether). These four cases do not impact our statistics of cycle length nor exchanges used. Moreover, no WETH-profiting arbitrage was found to be incorrect – we attribute this to the fact that WETH does not have fee-on-transfer, and conforms to expected ERC-20 behavior.

False negatives may arise from the various limitations identified above. However, given our large set of automatically identified exchanges in Section 3.2.3, we believe that we have achieved sufficient coverage of the ecosystem in general, and that our aggregate statistics are a fair high-level representation.

3.2.2 Arbitrage Cycle Properties

We find that 3,797,259 arbitrages (or 98%) contain exactly one exchange cycle, which demonstrates that bots typically use a basic strategy of one arbitrage per transaction. To simplify the analysis, hereon we only focus on the transactions that have just one exchange cycle.

Table 1 counts each of the one-cycle arbitrages by cycle length. It is important to note that 91% of these arbitrages use either two or three exchanges, which also validates the simplifying heuristics that Wang [51] and Zhou [55] make when computing profitable arbitrage opportunities.

Additionally, Table 2 counts all arbitrages by their *pivot token* (token taken as profit). Here, we can clearly see that nearly all arbitrages (92%) use Wrapped Ether [23] (WETH) as their chosen token to take a profit. This validates another important heuristic used by Wang [51] and Zhou [55]: when seeking arbitrage opportunities, one may constrain the search space to only cycles that use WETH and still effectively examine the vast majority of opportunities that arbitrageurs are willing to execute.

We use this measurement as justification to re-use these two simplifying heuristics ourselves in Section 4, where we look back in history and compute profitable arbitrage *opportunities*.

3.2.3 Exchange Attribution

As a result of running our arbitrage detection, we have identified $50,081$ addresses as decentralized exchanges (as a reminder, per Section 3.1, exchanges are the edges on multi-graph *G* that participated in the arbitrage cycle).

Exchange Count	Arbitrages (#)	Arbitrages (%)
2	1,797,940	47.3%
3	1,658,735	43.7%
4	280,078	7.4%
5+	60,506	1.6%
Total	3,797,259	100%

Table 1: Count of arbitrage cycle lengths, by number of exchanges used.

Token	Arbitrages (#)	Arbitrages (%)
Wrapped Ether [23]	3,507,128	92.4%
USDC [21]	75,701	2.0%
Tether [18]	48,381	1.3%
Dai [13]	41,389	1.1%
Other	124,660	3.3%
Total	3,797,259	100%

Table 2: Count of arbitrages by token taken as profit.

We manually examined these exchanges and, when possible, attributed them to DEX applications. Attribution was done by checking for smart contract source code or vanity address labels added to Etherscan [9]. When we uncover a new application in this manner, we consult their documentation for an automated method to list all of the application's DEX contracts and then use this data to attach labels to the application's other contract addresses. This manual investigation reveals that a significant number of applications are direct clones of Uniswap v2 with minimal source code modifications, if any.

We take the remaining unexplained exchanges and automatically scan the blockchain for the type of event log that a Uniswap v2 clone's *factory* contract would emit if it were to deploy that exchange, and then verify via dynamic trace that it did indeed deploy that contract. This reveals 180 unique Uniswap v2 factories, each a separately deployed AMM application.

The results of this attribution are displayed in Figure 2, and we also provide an extended accounting in Appendix C. Our result shows that while Uniswap V2 is dominant in the arbitrage ecosystem (44% of exchange-uses), Uniswap v3 is also quite popular among arbitrageurs (14% of exchange-uses). This is significant because, at the time of writing, we are not aware of any study that attempts to detect and analyze arbitrage *opportunities* with Uniswap v3. We accomplish this in Section 4, where we model Uniswap v3 and several other frequently arbitraged DEX applications from this list, and detect profit opportunities.

3.2.4 Sandwich Attacks

A *sandwich attack* is a strategy used to extract profit from a victim attempting to trade on a DEX [29, 44, 49]. Sandwich attacks consist of three transactions, in the following order: (1) the attacker manipulates DEX exchange prices, (2) the victim executes at unfavorable prices, and (3) the attacker de-manipulates DEX exchange prices and collects profit.

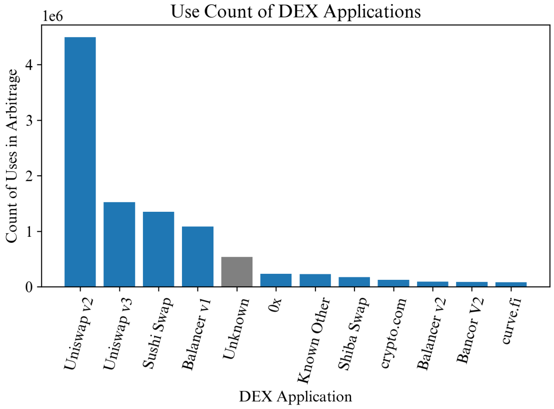

Figure 2: Exchanges used in arbitrage, by frequency.

While examining the arbitrages identified by our system, we noticed a large number of extremely profitable trades performed by a small number of actors. Further investigation revealed that these are not simple arbitrages, but rather Phase (3) – the de-manipulation step – of sandwich attacks. These sandwich-attack arbitrages are paired with a Phase (1) transaction, which appears as an arbitrage that *loses* an amount of money slightly less than that gained in Phase (3). In effect, the appearance of extreme arbitrage profit in Phase (3) is because of the great expense paid in Phase (1) to manipulate the exchanges.

We check for each of the three phases listed above and identify 63,257 sandwich-attack arbitrages. This reduces the apparent sum of arbitrage profit by $5.2 billion – future research must take care to avoid inclusion of these arbitrages, to avoid skewing the data. These transactions are removed from our dataset.

Attack Example. We illustrate with a real-world example where the attacker profits about $59 before fees / $10 after fees.

Manipulation transaction (`0x6fd4..5564`): The attacker spends 1,239,768 DAI to buy 298,613 HOPR on Uniswap v2, and then spends the 298,613 HOPR to buy 38,122 DAI on Uniswap v3. Notice that a naive interpretation would label this as an arbitrage that *loses* about 1.2 million DAI.

Victim transaction (`0x5ca7..72f5`): The victim sells 500,000 HOPR for 63,143 DAI on Uniswap v3, then 63,143 DAI for 63,139 USDC, and finally 63,139 thousand USDC for 54 Ether. Because of the manipulation, the victim's swap executes at a less favorable price than expected.

De-manipulation transaction (`0x8269..7ac4`): This transaction operates on a reverse of the exchange circuit used during the manipulation phase. The attacker sells 38,122 DAI for 300,868 HOPR on Uniswap v3, then sells 300,868 thousand HOPR for 1,239,827 DAI on Uniswap v2. Accounting for the initial losses, this yields the attacker 59 DAI ($59).

Behavior	Count	%
No change	2,108,655	55.5%
No arbitrage	875,515	23.1%
Reverted	506,213	13.3%
Profit changed	238,329	6.3%
Sandwich	63,257	1.7%
New pivot token	5,260	0.1%
Cycle count changed	30	0.0%
Total	3,797,259	100%

Table 3: Observed behavior when executing identified arbitrages at the top of the block.

3.2.5 Back-Running

It is well-known that arbitrageurs use a strategy called *back-running* [44,51,57]. In this strategy, an arbitrageur monitors the queue of pending transactions (the *mempool*) and examines it for other users' transactions that modify DEX prices. Occasionally, one can observe a pending transaction that modifies prices such that profit can be made from arbitrage. The arbitrageurs then seek to construct and include this arbitrage immediately after the observed target transaction. This is accomplished in one of two ways, by either (a) exploiting the fact that block producers tend to include transactions in order of descending transaction fees or (b) by using FLASHBOTS [22] (see Section 2).

Strategy (a) is a commonly noted method of exploiting transactions placed in the mempool [33, 34, 42–44, 49, 57]. Using this approach, one can strategically place a transaction either immediately before, or immediately after a target transaction.

Strategy (b), however, is a more recent emergence, and is still lightly studied [34,42,44]. The FLASHBOTS system is a centralized, third-party service that connects users seeking to avoid the publicly observable mempool with block producers who are willing to include their transactions for a fee. This keeps a transaction's content private (with respect to the Ethereum peer-to-peer network) until its inclusion in a block. As an additional feature, the FLASHBOTS system allows users to specify that a block producer should relay several transactions in a predetermined order. Arbitrageurs use this to back-run transactions.

In order to identify back-running arbitrages, we introduce the following approach. For each identified arbitrage transaction, we use a modified version of GANACHE [11] to fork the blockchain at the block immediately preceding its inclusion. Then, we re-play the transaction as if it were to execute first in its block. Lastly, we observe the resulting ERC-20 token movement logs and use them to run our arbitrage identification algorithm. The results of this experiment are displayed in Table 3.

We define a back-running arbitrage as one which, when re-ordered and executed as the first transaction in its block, fails to re-produce an arbitrage – i.e., either it reverts or is no longer labeled as an arbitrage under our analysis. This yields a total of 1,381,728 back-running transactions. The profit-share of back-running behavior is also plotted over time in Figure 3.

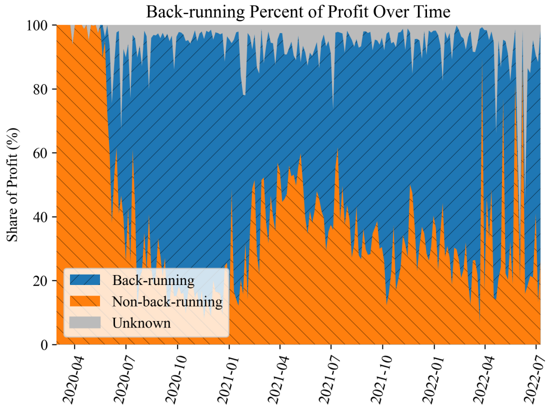

Figure 3: Share of total arbitrage profits taken by back-running vs non-back-running transactions, drawn as a percentage over a window of 6,646 blocks (about one day). We see back-running emerge as a strategy around May 2020.

We augment this experiment with the publicly available FLASHBOTS data feed [10], which identifies the transactions that were relayed through their system. Based on this feed, we find that 468,431 back-running transactions (or 32.4%) leverage FLASHBOTS.

3.2.6 Revenue and Fees

In this section, we analyze the revenues, profits, yields, and fees paid for arbitrage transactions. In order to make a fair apples-to-apples comparison of revenue without dealing with token-to-token conversion rates, we further limit the scope of this analysis to only those arbitrages that took profit in WETH.

We say that the *revenue* of a transaction is the amount of pivot token received minus the amount emitted to fund the arbitrage trade.

The *profit* of a transaction is the revenue *minus* any fees paid to the block producer. This is an important distinction. As Daian [33] observed, arbitrage runners compete with each other in "progressive gas-price auctions," where arbitrageurs bid ever-increasing fees for the privilege of being included before all others seeking the same arbitrage opportunity. In such gas auctions, the revenue stays the same, but the profit kept by the arbitrageur approaches zero as they attempt to out-bid the other bots. Note that transactions relayed through FLASHBOTS may optionally include a direct transfer of Ether to the block producer in addition to the standard transaction fee, so we also collect all these direct transfers of Ether and include them within the fees of the transaction.

Finally, we define the *yield* of an arbitrage transaction to be the percentage of revenue that is taken as profit, where revenue is positive.

The aggregate statistics of profit, revenue, and yield of our identified arbitrages are summarized in Table 4. There are

	Revenue (ETH)			Profit (ETH)			Fee (ETH)			Yield (%)		
Percentile	25th	50th	75th	25th	50th	75th	25th	50th	75th	25th	50th	75th
Back-running	0.023	0.050	0.115	0.004	0.019	0.065	0.008	0.018	0.032	13.11	51.39	75.35
Not back-running	0.011	0.023	0.045	0.001	0.004	0.013	0.007	0.015	0.029	6.2	20.93	40.28
FLASHBOTS	0.011	0.021	0.044	0.001	0.002	0.005	0.003	0.012	0.024	3.29	7.6	19.0
Not FLASHBOTS	0.017	0.036	0.079	0.003	0.012	0.041	0.009	0.018	0.033	20.24	42.39	65.84
All	0.014	0.030	0.067	0.001	0.007	0.027	0.008	0.016	0.031	8.06	28.34	56.57

Table 4: Revenue, Profit, Fee, and Yield of various sets of identified arbitrages, broken down by percentile.

several important findings to highlight here. First, we can independently confirm Piet's [42] observation that, for FLASHBOTS-relayed arbitrages, the vast majority of revenue is paid out in mining fees (only 7.6% yield). In contrast, arbitrages not relayed through FLASHBOTS achieve a much higher yield of 42%. Consequently, non-FLASHBOTS arbitrages achieve median *six-fold* higher profits. Interestingly, we also show that the median fee paid by FLASHBOTS arbitrages and non-FLASHBOTS arbitrages is somewhat lower, with 0.012 vs 0.019 Ether, respectively. So, the difference in yield is best explained *not* by the fact that fees are typically higher, but by the fact that FLASHBOTS transactions tend to have smaller revenue.

The situation is similar for back-running versus non-back-running arbitrages. We see that those who are able to use the more advanced *back-running* strategy are rewarded with median *five-fold* higher profits, and much higher yield (51% vs 21%). Because the median fee paid is in fact higher when the transaction is back-running, the increase in yield is best explained by the higher median revenue.

3.2.7 Impact on MEV

Maximal Extractable Value (MEV) – sometimes called Block Extractable Value, or BEV [44] – commonly refers to the maximum amount of risk-free value that a block producer can extract from a block by selectively including, excluding, and/or re-ordering transactions in a block [29, 33, 34, 42–44, 55]. Prior work has raised the alarm about MEV – as too much of it can set the stage for a so-called "time-bandit attack" [33, 37, 44]. In fact, Qin [44] demonstrates that several thousand blocks in the Ethereum blockchain *already exist* with MEV far exceeding the typical block reward, setting the scene for this attack.

In brief, this attack is performed by a financially motivated block producer who notices that a recently included block contains an abnormally high MEV. Then, the rational choice is no longer to produce blocks honestly, which yields nominal profits, but instead fork the blockchain immediately before the valuable block's inclusion and attempt to *itself* become the producer, which results in a windfall profit.

Piet's [42] study period of about two weeks found that an alarmingly large share of arbitrage profit is being sent to the block producer. Our much larger window enables us to see that this is, in fact, an ongoing and long-running trend. Figure 4 shows that the block producer's share of the revenue from arbitrage is rapidly approaching 100% in recent times. This implies that time-bandit attacks remain a viable and worrisome threat.

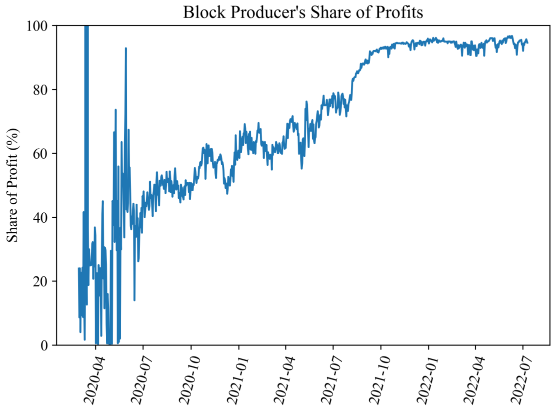

Figure 4: Block producers' share of profits, computed as a median over a 1-day rolling time window. We see that block producers' share is rapidly approaching 100%.

4 Detecting Arbitrage Opportunities

In this section, we perform an experiment to examine which arbitrage *opportunities* existed in the past, and in the process make several important observations about the difficulties arbitrageurs encounter when attempting to perform arbitrage identification in real-time. We begin in Section 4.1 by describing the system for detecting potential arbitrages based on DEX prices and optimizing for maximum profit. Then, in Section 4.2, we execute a selection of these potential arbitrages and automatically diagnose execution failures. Finally, in Section 4.3, we examine the results of executing these (potential) arbitrages.

4.1 Arbitrage Detection System

Consulting Figure 2, we decide to simplify the task of detecting arbitrage opportunities by limiting the supported DEX applications to Uniswap v2 and v3 [20], Shushi Swap [16], ShibaSwap [15], and Balancer v1 and v2 [4]. We do not use the exchange 0x [1], as it is not an Automated Market Maker (AMM), but rather an exchange settlement application with an off-chain order book, for which we do not have records to analyze. We also opted to use Balancer v2 over crypto.com [6], as we know that the former is under-counted (see Section 3.1). Thus, we include in our analysis the six most frequently used AMM-based exchanges. These exchanges total 94,495 smart contracts and include $78,605$ unique ERC-20 tokens. See Appendix E for a breakdown of count by DEX application.

The detection of arbitrage opportunities is challenging. At a high level, it includes two tasks. First, one must use the spot prices of all (supported) tokens of each AMM to compute exchange rates and determine where arbitrage is possible (Section 4.1.2). Second, one must determine *how much* profit can be made and, simultaneously, how much money to spend funding the trade – we contribute a novel approach to solving this problem (Section 4.1.3).

Only Uniswap v2, Sushi Swap, and ShibaSwap use the "constant-product" swap invariant, shown in Section 2. This is both well-studied and known to have fast, analytic solutions to those two problems [28, 29, 45]. Constant-product swap invariants maintain the AMM's token reserves at a 50% / 50% share of value between its two paired tokens.

However, both Balancer v1 and v2 use a weighted version of the constant-product invariant pricing model [39]. Here, administrators may manually set the percentage of the total pool's value that is allocated to each token. Moreover, Balancer AMMs may maintain a liquidity pool of up to 8 assets, any of which may be exchanged for any other. So, for example, a Balancer AMM could have 3 tokens, α, β, and γ, occupying 50%, 25%, and 25% of the pool, respectively. Balancer also provides a generalized analytical solution to arbitrage on weighted constant-product AMMs [39].

Making matters yet more complex, Uniswap v3 uses a different swap invariant entirely [27]. Uniswap v3 allows *liquidity providers* to bound their price tolerance, which removes assets from market-making when the price deviates beyond the user's parameters. In effect, reserves of its two paired tokens increase and decrease dynamically as the price moves into and out of liquidity providers' pre-set tolerance bounds. This adjustment may occur several times within a single atomic swap, as the price moves due to *slippage* (defined below).

Wang's approach [51] relies on the known fast solution for constant-product exchanges, and hence, will not work for all our exchanges – owing to the complexity of Uniswap v3's swap invariant, which frequently adjusts available token reserves, frustrating analytic solutions. Zhou's approach [55] works by gradually increasing the amount of funds sent in order to find the optimal profit. This strategy is not efficient enough to process swaps (exchanges) at the scale we require – in fact, Zhou only considers 25 tokens when looking for arbitrage opportunities, versus our 78,605 tokens.

In what follows, we present an approach that efficiently searches for arbitrage opportunities.

4.1.1 Definitions

First, we define some key terms and notation.

Spot Price. The *spot price* of an AMM is the current rate of exchange from one token to another. We denote the spot price of an exchange e when swapping token α for token β at blockchain state σ as:

$$p^{e}_{\alpha\to\beta} : (\sigma) \to \mathcal{R}^{+}.$$

Slippage. Every exchange on our selected AMMs experiences *slippage* on every exchange operation, which describes the situation where the actual price charged when executing an exchange is worse than the AMM's initial spot price.

Marginal Price. The *marginal price* of an AMM is simply the spot price after spending n units of token α to buy token β on exchange e in blockchain state σ. Because slippage is guaranteed, the marginal price increases with n. We write this as:

$$\phi^{e}_{\alpha\to\beta} : (\sigma, n) \to \mathcal{R}^{+}.$$

Swap Function. The *swap function* of an AMM returns the amount of token β one can purchase by spending n units of token α in blockchain state σ. We write this as:

$$s^{e}_{\alpha\to\beta} : (\sigma, n) \to \mathcal{R}^{+}.$$

Each of our modeled exchanges imposes a constant tax rate on each transaction. For simplicity, we include this tax in the spot price, marginal price, and swap functions, simply by multiplying by the tax rate.

Fee-on-transfer Token Fees. Some ERC-20 tokens deduct a small tax each time a transfer from one address to another occurs. We assume that, for token α, the amount received when transferring from one address to another is a constant multiple of the quantity transferred, which we capture as the parameter $f_\alpha \in (0,1]$.

Exchange Cycle. An *exchange cycle* indicates a closed, directed cycle of exchanges. This can be expressed as an ordered list of k tuples:

$$E = (e_1, \alpha_1, \beta_1), \ldots, (e_k, \alpha_k, \beta_k),$$

where exchange e_i accepts token α_i and emits token β_i. Because this is a cycle, $\beta_i = \alpha_{i+1}$ for $i < k$ and $\alpha_1 = \beta_k$, i.e., the output token of each exchange is the input token the next, and the first and last token are the same.

4.1.2 Cycle Detection

Our first task is to identify candidate exchange cycles, which we later optimize for maximum profit.

We begin by observing that each of our target AMMs emits event logs whenever their internal state updates. These event logs contain enough information to fully infer the internal state of the AMMs, which we refer to as σ.

We manually build five models for the functions p, ϕ, and s: one for the constant-product model used in Uniswap v2, Sushi Swap, and ShibaSwap, one for Balancer v1, one for Uniswap v3, one for Balancer v2's "weighted pool," and one for Balancer v2's "liquidity bootstrapping pool." The structure of these exchanges is more rigorously examined by Xu [54].

For each model, we also build the transition function $q(l,\sigma) \rightarrow \sigma'$, which accepts event log l and state σ and returns new state σ' after applying the information contained in the event log. Logs contain information about various actions that people have taken on the AMM – for example, they could indicate a price update after a swap, the contribution of liquidity, an adjustment of the relative weight of a token (for Balancer v2), or an adjustment of a liquidity provider's parameters (Uniswap v3).

Each time that we examine a block, we apply all logs to the state update function, q, and collect the set of all exchanges whose internal state was updated. Then, we construct all cycles that include two or three exchanges (and at least one exchange that was just updated).

Note that we filter out exchange cycles that are not profitable. We do this by pushing a tiny amount of 10^{-8} WETH forward through the cycle. If this operation already proves unprofitable (and returns under 10^{-8} WETH in output), we discard the cycle. Otherwise, we proceed to profit maximization.

4.1.3 Generalized Profit Maximization

Using our models and transition functions, we can compute the revenue gained by running an arbitrage on a fixed cycle E of length i using n units of the pivot token on state σ:

$$\mathrm{Out}(n,E) = \begin{cases} s^{e}_{\alpha\rightarrow\beta}(\sigma,n)\cdot f_{\beta} & E=(e,\alpha,\beta) \\ \mathrm{Out}\left(s^{e}_{\alpha\rightarrow\beta}(\sigma,n)\cdot f_{\beta}, E'\right) & E=(e,\alpha,\beta),E' \end{cases}$$

$$\mathrm{Revenue}(n,E) = \mathrm{Out}(n,E)\cdot f_{\alpha_1} - n$$

where $\mathrm{Out}(n,E)$ recursively computes the revenue of the arbitrage cycle, including fees-on-transfer.

In order to maximize our revenue, we must find the appropriate amount N that satisfies

$$N = \underset{n}{\mathrm{argmax}}[\mathrm{Revenue}(n,E)].$$

The following insight allows us to significantly simplify the search for this value N. As stated previously, the marginal price of each exchange e_i for $\alpha_i \rightarrow \beta_i$ monotonically increases as n increases – in other words, the more value we send through each exchange, the less favorable the prices will be. This makes sense financially as well, as the very act of performing an arbitrage equalizes prices across exchanges, making that arbitrage cycle no longer profitable [47].

In fact, for some small value ε, there must exist some n such that $\mathrm{Revenue}(n,E) > \mathrm{Revenue}(n+\varepsilon,E)$ – that is, the rate of change in revenue with respect to n eventually turns negative, reducing revenue as n increases. We can compute this rate of change in revenue – we will call it *marginal revenue*, or $\Phi(n,E)$ – using our model for marginal price:

$$\Phi(n,E) = \begin{cases} \gamma\cdot f_{\alpha} & E=(e,\alpha,\beta) \\ \gamma\cdot f_{\alpha}\cdot\Phi\left(s^{e}_{\alpha\rightarrow\beta}(n)\cdot f_{\alpha}, E'\right) & E=(e,\alpha,\beta),E' \end{cases}$$
$$\text{where } \gamma = \phi^{e}_{\alpha\rightarrow\beta}(n)$$

where $\Phi(n,E)$ multiplies together the marginal prices of the cycle E after n tokens are applied. Put simply, we are computing the ratio of the cycle's output per the amount of input n we attempt to send. If the marginal revenue is less than 1, then this is sub-optimal – sending more input into the cycle will cost less than the amount of output we receive. If the marginal revenue is above 1, then it is past the optimal amount – as each additional unit of input sent will not yield more than that amount of output.

Since we know that all ϕs are monotonically increasing as n increases (guaranteed slippage), and that s increases as n increases, we can conclude that $\Phi(n,E)$ is also monotonically increasing with respect to n.

From here, the task of finding N is reduced to solving the equation $\Phi(N,E) = 1$ – where the ratio of revenue to input is exactly equal. In other words, we must find the turning point where purchasing any additional output costs more in input than will be received, which decreases revenue. Since $\Phi(n,E)$ is monotonically increasing, we use a simple binary search to discover N, the input amount which maximizes profit.

We plot an example optimization problem in Appendix G.

4.1.4 Experimental Setup

We have developed a system that loads the blockchain state as needed by the models and performs the transition function $q(l,\sigma)$ to update the model state in response to logs observed in the blockchain history. We set the fee multiple f_α for eleven common fee-on-transfer tokens, and initialize the remainder f_α to 1, which optimistically assumes zero fees (this is automatically corrected later in Section 4.2).

Because our set of exchanges is so large, a naive application of our cycle detection algorithm (Section 4.1.2) would quickly result in a combinatorial explosion of possible exchange cycles to consider. To combat this, we use four simplifying heuristics to keep the problem tractable:

1. We limit exchange cycle length to 3, which Table 1 indicates was used in 91% of observed actual arbitrages.

2. We fix the pivot token to WETH, which Table 2 indicates was used in 92% of arbitrages.

3. We set minimum exchange balance thresholds on all exchanges that hold WETH [23], USDC [21], Tether [18], Wrapped Bitcoin (WBTC) [24], and Uniswap Token [25]. We disregard any exchange that trades one of these

tokens with a balance smaller than this threshold. See Appendix F for specifics.

4. We disregard any potential arbitrages which do not produce at least 0.001 Ether in revenue.

We break the about 5.5-million blocks observed during our study period into 5,500 contiguous segments of approximately one thousand blocks each. Each of these segments of blocks forms a work queue. Workers in the system are assigned segments to analyze, and they do so block by block, starting from the oldest one and scanning forward into the future. For each block, we only analyze exchange cycles that contain an exchange that updated its state in the prior block.

We use GNU Parallel [48] to spawn approximately 500 processes across a set of five machines, each with dual Intel Xeon Gold 6330 CPUs, a minimum of 350 GB of RAM, running Ubuntu 20.02. We load-balance across two full-archive Go Ethereum [12] nodes. The computation takes about two weeks and yields 4,580,282,058 potential arbitrages.

4.1.5 Comparison to Prior Work

We use this experimental setup to compare our method of arbitrage profit optimization vs Zhou's [55] strategy of gradually-increasing linear search. To accomplish this, we replace our strategy with a linear search, set to stride in increments of 10^{-5} Ether (about $0.01). We randomly select 100 contiguous segments of 20 blocks each within our study window, and run both strategies over this selection while measuring both wall-clock time consumed and the number of model-queries performed. We find that the linear search strategy consumes 44 times more wall-clock time, and executes 55 times more model-queries. Moreover, our strategy runs to much greater precision.

4.2 Executing Arbitrages

The prior experiment yields more than four and a half billion potential arbitrages, but we still need to know which of these are truly possible to execute, and if so, how much gas each transaction consumes. This will aid us in gaining a realistic view of which arbitrages were once possible in the blockchain.

First, in Section 4.2.1, we compute a simple gas-price oracle to help us reason about expected fees. Then, in Section 4.2.2, we discuss our system that automatically constructs and executes an arbitrage transaction to verify that it is actually possible to take. In Section 4.2.3 we overview the experimental setup. Finally, the results are compiled in Section 4.3.

4.2.1 Estimating Fees and Gas Prices

Although an arbitrage may have positive revenue, we need to know whether it is *profitable* after we deduct the transaction fees paid to the block producer. To do this, we need to estimate how much gas price an arbitrageur would typically need to bid to get their transaction included in a block. We devise a simple but effective strategy to create a gas-price oracle that estimates how much this might be.

To devise this oracle, we examine the identified arbitrages in Section 3. We limit our view to arbitrages that meet two criteria: (i) they must only use the six labeled exchanges supported by our detection system, and (ii) they must be non-back-running arbitrages. Then, we categorize each into a *bucket*: a 3-tuple that indicates (1) whether this was a FLASHBOTS transaction, (2) the set of DEX applications that it used, and (3) the cycle length. If a transaction included a direct Ether transfer to the block producer, we factor that into the gas price as well. More precisely, we compute how much additional Ether was transferred per gas expended, and then add this value to the gas price.

We perform a linear scan over the arbitrages and construct the gas-price oracle for each block based on a rolling window of the previous 276 blocks (about 1 hour). For each period, we record the 25th, 50th (median), and 75th percentile gas price that arbitrages used within that window.

4.2.2 Automated Execution and Failure Diagnosis

Executing an arbitrage on the blockchain requires a relaying smart contract to invoke the correct exchanges and to manage the flow of tokens. Abstractly, correctly invoking this relayer is all that is required to execute an arbitrage. To this end, we write our own relayer contract and ensure that it is capable of invoking each of our supported exchanges.

Our system executes arbitrages against historical blockchain states. We use GANACHE [11] to fork the blockchain exactly at the block where the arbitrage should be possible (that is, the block where our system detects the opportunity). We then deploy our relayer contract, fund it with a sufficiently large amount of Ether, and finally run the transaction that executes the arbitrage.

Failure Diagnosis. In certain cases, the arbitrage fails to properly execute when running. When this failure occurs, we take a dynamic trace of the execution and attempt to diagnose the issue. We discuss the two main sources of failure below and describe a few more in Appendix H.

Token Reverts. ERC-20 token smart contracts occasionally insert application-specific logic into the functions `transfer(..)` or `balanceOf(..)`. This logic might cause either of these functions to revert the transaction, causing failure. Logic to revert transactions can be intentional or unintentional, based on a whitelist, based on a blacklist, or other reasons. For example, some tokens have an imposed limit on the maximum transfer value. Others are whitelisted to only known exchanges and initial investors, and any transfer to an address outside of the whitelist will fail. In our dynamic trace, if we ever identify that one of these two ERC-20 functions reverts, then we mark the token as faulty and discard the potential arbitrage.

Non-supported Tokens. Balancer v1 requires that tokens return `true` upon calling `transfer(..)`, which should indicate success. However, many tokens do not return any data whatsoever, and, instead, indicate failure by simply reverting the transaction. If we ever see that Balancer v1 attempts to transfer a token that does not return data, then we mark the token as “not supported” and discard the potential arbitrage.

4.2.3 Experimental Setup

It is computationally difficult to execute the entire set of four and a half billion possible arbitrages. Thus, we select a subset to replay. Specifically, we create two sets of arbitrages.

For the first set, we execute all 20,622,390 arbitrages that purport to generate over one Ether in revenue. That is, we want to further analyze all large arbitrage opportunities that we identified.

If our system manages to successfully executes one of these arbitrages, we search backward and forward (from the current block) for other potential arbitrages that involve the exact same exchange cycle E. We then continue executing these arbitrages (in both directions) until the arbitrage disappears. Finally, we store these arbitrages (for a particular cycle) together as a contiguous *campaign*. This is important to do for two reasons – so we can reason about an arbitrage’s properties over time, and so that we can disambiguate arbitrages which, in reality, cannot all be executed simultaneously. For example, attempts to naïvely sum arbitrage profits block-by-block will dramatically over-represent the real situation, as taking a single arbitrage in a contiguous campaign would move prices to an unprofitable state.

For the second set, we divide the study period into 834 segments of 6,646 blocks (about 1 day). We randomly sample 30 of these segments and execute all 126,147,388 potential arbitrages within these segments. The selected segments are listed in Appendix **??**. When an arbitrage is successfully executed, we again perform a linear scan through the segment and group possible arbitrages with the same exchange cycles into contiguous *campaigns*.

We again use GNU Parallel to execute this analysis across the same five machines described in Section 4.1.4. This experiment takes about one week to complete.

4.3 Results

In this section, we discuss the results of the arbitrage opportunity detection, execution, and failure diagnosis system.

4.3.1 Discovered Arbitrage Opportunities

As mentioned, we find about 4.5 billion arbitrage opportunities. This means that there are, on average, 826 new (or updated) potential arbitrage opportunities in each block. Recall that we record arbitrages only when an exchange’s state updates, so this is, in fact, a lower bound on the number of arbitrage opportunities that are available within a block – some remain from previous blocks, unchanged. These arbitrages purport an average revenue of 0.097 ETH and range from the threshold minimum – 0.001 ETH – to 1,039 ETH.

4.3.2 Execution and Diagnosis

In this section, we discuss the results of executing all 20,622,390 potential arbitrages with over 1 Ether in revenue.

The results are shown in Table 5. We see that only 106,139 (0.51%) of these arbitrage opportunities are executed successfully. Moreover, the vast majority of those transactions that execute successfully do not produce profit significantly greater than 1 Ether – we display the distribution in Figure 5. Finally, and most importantly, the results clearly indicate that one *cannot* naïvely assume that an arbitrage is present without verifying that the transaction can actually be concretely executed. In practice, 93% of the arbitrage opportunities cannot be executed because the token they require is either incompatible with the exchange it is listed on, or a token simply reverts.

	Count	%
All transactions	20,622,390	100%
Successfully executed	106,139	0.5%
All failures	20,516,251	99.5%
Token reverts	11,305,290	54.8%
Non-supported token	8,832,238	42.8%
Other	220,760	1.1%
Exchange-balance disorder	102,834	0.5%
Interference	29,102	0.1%
No arb. after fee-on-transfer	26,027	0.1%

Table 5: Results for potential arbitrages over 1 Ether.

Our results indicate that actually executing any perceived opportunities is critical for a large-scale study of arbitrage. However, this has not been discussed in previous work on arbitrage, such as in Zhou’s [55] or Wang’s [51] studies. Although Zhou’s [55] work is likely not significantly impacted because they hand-picked a small number of known good exchanges and tokens, Wang’s [51] study likely over-represents arbitrage opportunities.

We now focus on the 106,139 arbitrages that were executed successfully. The results show that – using the “median” gas-price oracle – the successfully executed arbitrages arrange into 57,392 contiguous campaigns. This is another important realization: When we reason about arbitrages over time, we discover that, in fact, many of these are simply prior arbitrages carrying forward. Critically, this is another source of over-counting that studies must avoid.

We then use the following method to compute the window of opportunity for each of these campaigns. First, we locate the earliest block which maximizes the campaign’s profit after fees. Then, we scan forward and locate the first block where this profit falls below either half of the maximum profit or below 1 Ether, whichever is smaller. The distance between these two blocks is the duration of the window of opportunity for the arbitrage. Our analysis shows that the duration of opportunity is just 1 block at the 50th percentile, 4 blocks at the 75th percentile, and 548 blocks at the 95th percentile.

This result indicates that large arbitrages *do not* persist for long periods of time and that, generally, such opportunities are quite rare. This is a distinct development in the ecosystem

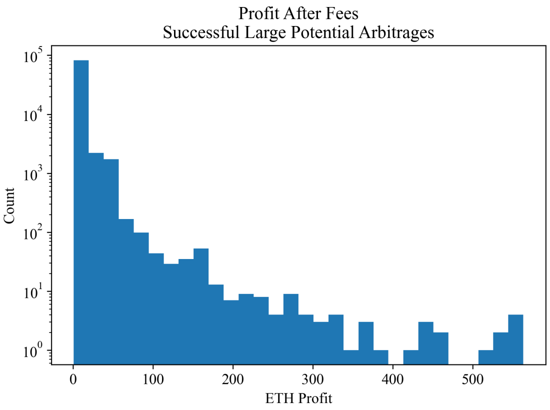

Figure 5: Profit distribution of successfully executed "large" arbitrages giving revenue over 1 ETH. We use the "median" gas-price oracle to compute fees. To show more detail, we have used a log scale on the y-axis.

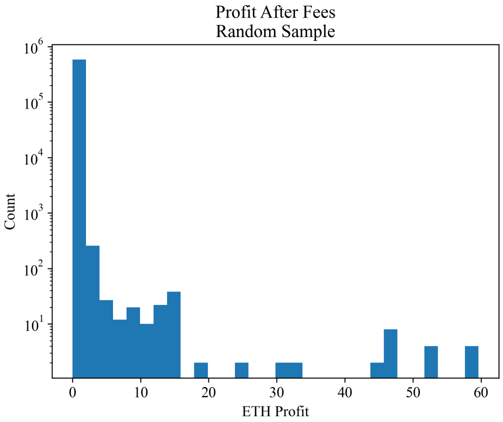

Figure 6: Profit distribution of arbitrage campaigns derived from 30 randomly-sampled segments of time about 1 day long. Profits are computed using the "median" gas-price oracle. Notice that to show more detail we have used a log scale on the y-axis.

since Wang's [51] publication, which found that the revenue of the most profitable arbitrage in Uniswap v2 is persistently higher than 1 Ether.

Finally, we compute the total value of the arbitrage opportunities. To accomplish this, we cannot simply sum the maximum profit value in each campaign, as campaigns that share exchanges may interfere with each other – i.e., taking one arbitrage may move prices such that the other is no longer profitable. Thus, it is important to ensure that we only sum the profit of arbitrages that do not interfere with one another, or else we risk over-counting (again).

We cast this computation as an instance of the maximum weighted independent set optimization problem over conflict graph G, where the vertices are the arbitrage campaigns, the weights on the vertices are their respective profits, and the edges indicate campaigns that overlap in both time and exchanges used. Because this optimization problem is known to be strongly NP-hard [35], we use a simple greedy algorithm to find a maximal set of non-conflicting campaigns. The algorithm produces $2,065$ non-conflicting campaigns, yielding a total of $3,860$ Ether, or $5,715,862 in total value, as measured at the time of arbitrage opportunity.

Comparison with Identified Arbitrages. We build confidence in our modeling by comparing identified arbitrages that were *actually executed* on the blockchain (Section 3) to the set of detected opportunities that we claim were possible to execute (as described in this section). We detect 93% of the arbitrages that were actually taken and for which we should detect a corresponding opportunity. Next, we randomly sample and analyze the identified (real) arbitrages for which we did not have a corresponding opportunity. About one-third were arbitrages that were executed as part of a more complex DeFi transaction. These can be considered false positives of our arbitrage identification algorithm. Hence, it makes sense that our model did not detect the opportunity. Another half was due to an inaccuracy in our model that led us to disregard some exchange cycles involving Balancer v1 (and that were specific to this exchange). The remainder was due to execution failure, where a closed-source token either reverts or has token-exchange interference that our bot is not capable of overcoming. Overall, we believe that the strong alignment between our models and the identified arbitrages provides a good indication of the correctness of our analyses.

4.3.3 Execution and Diagnosis – Random Sample

In this section, we discuss the results of executing all 126,147,388 potential arbitrages within 30 randomly-sampled day-long segments of time. This set of potential arbitrages shows a dramatically higher rate of success than the prior set – only 7,214,976 (5.72%) failed to execute. We break down the results in Table 8 in Appendix I, which shows that the main cause of failure was a token reverting the transaction.

Unlike in the prior section, the arbitrages may make revenue as low as 0.001 Ether. In fact, the median attainable profit, after computing campaigns, is just 0.006 Ether when we use the realistic gas-price oracle – we plot the full distribution of profits in Figure 6. This plot reveals that the overwhelming majority of actually executable arbitrage opportunities yields very little in profit.

Next, we compute the total possible profit over the 30-day sample (full detail of this analysis are in Appendix I). Recall again that taking one arbitrage in a contiguous campaign will cause it to be no longer profitable in the future, so we must again solve the maximal weighted independent set problem as described in Section 4.3.2. In total, the algorithm selects 50,533 non-conflicting arbitrages, which totals $1,692$ Ether in profit using the realistic gas-price oracle.

This amounts to a potential *weekly* profit of 395 Ether, which greatly exceeds Zhou's [55] system's estimated weekly profit of 191 ETH.

In order to compare to Zhou's work, we greedily gather exchanges from the most profitable arbitrages, greatest to least, until we have 100 total exchanges. When we limit our scope to just these 100 exchanges, and again solve the maximal weighted independent set problem as above, we see a weekly profit of just 72 Ether – which demonstrates the necessity of large-scale analysis.

Note that this differs from the profit in Section 3 because many bots are back-running, which this analysis does not support (as it operates on the state at the start of each block).

4.4 Security Implications on Price Oracles

Smart contracts on the blockchain occasionally need access to price data in order to value a token. For example, this assists automated loan providers in valuing the collateral supplied for a loan. AMMs are a convenient source for this sort of price data, but ultimately prove to be insecure.

In the bZx hack, an attacker managed to steal nearly $1 million by manipulating an AMM's price oracle [41]. At a high level, the attack worked as follows. First, the attacker made a large purchase of tokens on an AMM, which forced the price significantly higher. Then, they use these tokens as collateral to take out a loan from an automated provider. But the loan provider used the manipulated AMM's price, which valued the collateral too high. The attacker was able to loan out much more value than expected. The attacker walked away with the loaned Ether, and the loan provider was left with the debt.

A modern defense against this attack is the Time-Weighted Price oracle (TWAP), which functions similarly to a sliding-window price average. This forces an attacker to keep the price manipulated for a long period of time, expending significant capital in the process. Recent work has modeled and discussed costs and methods of attack that overcome the TWAP – the typical multi-block manipulation assumes "no fees, an infinitely liquid reference market, and the no-arbitrage condition, meaning that arbitrageurs are assumed to de-manipulate the price every block" [38]. The attacker must re-manipulate the oracle each time arbitrageurs de-manipulate the price.

Recall that we just measured that arbitrages regularly extend beyond one block in duration, even for large arbitrages. In fact, our "large" arbitrage campaigns from recent blocks – 14.5 million and on, since about April 2022 – have a *mean* duration of 6 blocks. This implies that the true cost of launching a price-oracle manipulation attack is likely significantly cheaper than expected.

5 Related Work

Daian et al. [33] first described the concept of MEV. This work also introduces the "time-bandit" attack, a threat that occurs when too much MEV is available on the blockchain. The authors report that this entices profit-motivated miners into forking the blockchain, possibly causing instability in the consensus system. They also first identify and discuss what they call "Priority Gas Auctions" (PGAs), where bots bid against each other in order to obtain priority execution for MEV extraction. This publication directly motivated the creation of FLASHBOTS [22], an effort to mitigate network congestion arising from PGAs and to create fairer conditions for relaying profit-generating transactions.

Qin et al. [44] quantify MEV – which they call BEV – in the form of arbitrages, "sandwich" attacks, and liquidations. They demonstrate that the setting for a time-bandit attack is already present. Moreover, they argue that the introduction of FLASHBOTS in fact aggravates both network congestion and MEV, by creating a situation of highly increased competition. Piet et al. [42] use a method of detecting arbitrage, back-running, and front-running transactions on the blockchain to analyze the MEV extraction ecosystem, focusing on the impacts of FLASHBOTS and private transaction relaying. Eskandari et al. [34] provides an SoK on these MEV-generating transactions.

A separate but related research thrust focuses on detecting MEV opportunities. Zhou et al. [55] design a system capable of detecting MEV in real-time, both via arbitrage cycle detection – with heavily limited scope – and via solver-aided modeling, which uncovers advanced DeFi attacks. Wang et al. [51] create a system that can find arbitrage opportunities, but it is limited to those that use the "constant-product invariant" pricing model, which limits the analysis to Uniswap v2 and Sushi Swap.

While there are a number of other works that explore various aspects of MEV and attacks against AMMs [29, 30, 32, 56, 57], the works described above are the closest related to our research, and we see them as fundamental first steps towards understanding the arbitrage phenomenon. Our work builds upon these results and introduces a modeling approach that allows for the accounting of more AMMs, resulting in more complete identification of arbitrage transactions and more accurate prediction of arbitrage opportunities.

6 Conclusions

Decentralized finance applications and protocols based on the Ethereum blockchain have a Total Value Locked (TVL) of tens of billions of dollars. However, the mechanisms that govern the market are sometimes opaque and not well understood. This might result in security risks for users and a barrier to adoption.

In this paper, we presented a novel analysis of arbitrage. Our analysis, which has a larger scale than previous work, both confirms prior observations and uncovers new insights about the arbitrage phenomenon. In particular, we observe that the way in which fees are distributed to the block producers might incentivize attacks that threaten consensus stability. In addition, an analysis of the duration of certain arbitrage opportunities indicates that price-oracle attacks might be less expensive than what was previously assumed. We also measure the potential arbitrage opportunities, showing that the extractable value is higher than previously thought.

References

[1] 0x: Powering the decentralized exchange of tokens on ethereum. https://www.0x.org/.

[2] Aave - open source liquidity protocol. https://aave.com/.

[3] Alpha homora - yield farming on leverage. https://homora.alphaventuredao.io/.

[4] Balancer amm defi protocol. https://balancer.fi/.

[5] CoW protocol overview. https://docs.cow.fi/.

[6] Crypto.com: The best place to buy bitcoin, ethereum, and 250+ altcoins. https://crypto.com.

[7] Decentralized finance - rankings, analysis and news. https://dappradar.com/defi/protocol/ethereum.

[8] Dexible - the professional dex aggregator. https://dexible.io/.

[9] Ethereum (ETH) Blockchain Explorer. https://etherscan.io/.

[10] Flashbots Blocks API. https://blocks.flashbots.net/.

[11] Ganache: A tool for creating a local blockchain for fast ethereum development. https://github.com/trufflesuite/ganache.

[12] Go ethereum. https://geth.ethereum.org/.

[13] The maker ecosystem. https://docs.makerdao.com/.

[14] MEV in Ethereum Proof-of-Stake (PoS). https://ethereum.org/en/developers/docs/mev/#mev-in-ethereum-proof-of-stake.

[15] Shibaswap. https://shibaswap.com/.

[16] Sushi swap. https://www.sushi.com/.

[17] Synthetics | the derivatives liquidity protocol. https://synthetix.io/.

[18] Tether. https://tether.to/en/.

[19] Tokenlon protocol. https://www.tokenlon.im/.

[20] Uniswap protocol. https://uniswap.org/.

[21] Usd coin. https://www.coinbase.com/usdc/.

[22] Welcome to Flashbots. https://docs.flashbots.net/.

[23] wETH | ERC20 tradable version of eth. https://weth.io/.

[24] Wrapped bitcoin (WBTC) an erc20 token backed 1:1 with bitcoin. https://wbtc.network/.

[25] Introducing UNI. https://uniswap.org/blog/uni, Sept 2020.

[26] Curve documentation. https://curve.readthedocs.io/_/downloads/en/latest/pdf/, 2022.

[27] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. https://uniswap.org/whitepaper-v3.pdf, 2021.

[28] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. An analysis of Uniswap markets, 2019.

[29] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch Lafuente. Maximizing Extractable Value from Automated Market Makers. In *Financial Cryptography and Data Security*, May 2022.

[30] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A theory of automated market makers in defi. In Ferruccio Damiani and Ornela Dardha, editors, *Coordination Models and Languages*, pages 168–187, Cham, 2021. Springer International Publishing.

[31] Yiling Chen and David M. Pennock. A utility framework for bounded-loss market makers. *CoRR*, abs/1206.5252, 2012.

[32] Michele Ciampi, Muhammad Ishaq, Malik Magdon-Ismail, Rafail Ostrovsky, and Vassilis Zikas. Fairmm: A fast and frontrunning-resistant crypto market-maker. In Shlomi Dolev, Jonathan Katz, and Amnon Meisels, editors, *Cyber Security, Cryptology, and Machine Learning*, pages 428–446, Cham, 2022. Springer International Publishing.

[33] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927, 2020.

[34] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano

Sala, editors, *Financial Cryptography and Data Security*, pages 170–189, Cham, 2020. Springer International Publishing.

[35] M. R. Garey and D. S. Johnson. “strong” np-completeness results: Motivation, examples, and implications. *J. ACM*, 25(3):499–508, jul 1978.

[36] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[37] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gaži, Sarah Meiklejohn, and Edgar Weippl. SoK: Algorithmic Incentive Manipulation Attacks on Permissionless PoW Cryptocurrencies. In *Financial Cryptography Workshop on Trusted Smart Contracts*, 2021.

[38] Torgin Mackinga, Tejaswi Nadahalli, and Roger Wattenhofer. Twap oracle attacks: Easier done than said? pages 1–8, 05 2022.

[39] Fernando Martinelli and Nikolai Mushegian. A non-custodial portfolio manager, liquidity provider, and price sensor. https://balancer.fi/whitepaper.pdf, 2019.

[40] Ronald W. Melicher. *Introduction to finance : markets, investments, and financial management*. Wiley, Hoboken, New Jersey, fifteenth edition. edition, 2014 - 2014.

[41] palkeo. The bzx attacks explained. https://www.palkeo.com/en/projets/ethereum/bzx.html, 02 2020.

[42] Julien Piet, Jaiden Fairoze, and Nicholas Weaver. Extracting godl [sic] from the salt mines: Ethereum miners extracting value, 2022.

[43] Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An Empirical Study of DeFi Liquidations: Incentives, Risks, and Instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference*, IMC ’21, pages 336–350, New York, NY, USA, 2021. Association for Computing Machinery. event-place: Virtual Event.

[44] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 198–214, 2022.

[45] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the defi ecosystem with flash loans for fun and profit. In Nikita Borisov and Claudia Diaz, editors, *Financial Cryptography and Data Security*, pages 3–32, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[46] Fabian Schär. Decentralized finance: On blockchain- and smart contract-based financial markets. *Review*, 103(2):153–174, Feb 2021.

[47] William F. Sharpe. *Investments*. Prentice-Hall,, Englewood Cliffs, N.J. :, 3rd ed. edition, 1990.

[48] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, Feb 2011.

[49] Christof Ferreira Torres, Ramiro Camino, and Radu State. Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1343–1359. USENIX Association, August 2021.

[50] Fabian Vogelsteller and Vitalik Buterin. EIP-20: Token Standard. https://eips.ethereum.org/EIPS/eip-20, Nov 2015.

[51] Ye Wang, Yan Chen, Haotian Wu, Liyi Zhou, Shuiguang Deng, and Roger Wattenhofer. Cyclic arbitrage in decentralized exchanges. In *Companion Proceedings of the Web Conference 2022*, WWW ’22, page 12–19, New York, NY, USA, 2022. Association for Computing Machinery.

[52] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151:1–32, 2014.

[53] Siwei Wu, Dabao Wang, Jianting He, Yajin Zhou, Lei Wu, Xingliang Yuan, Qinming He, and Kui Ren. Defiranger: Detecting price manipulation attacks on defi applications. *CoRR*, abs/2104.15068, 2021.

[54] Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols, 2021.

[55] Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-in-time discovery of profit-generating transactions in defi protocols. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 919–936, 2021.

[56] Liyi Zhou, Kaihua Qin, and Arthur Gervais. A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges. https://arxiv.org/abs/2106.07371, 06 2021.

[57] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 428–445, 2021.

Appendix

A Evaluating Risk of False-Positive from Unwrapped Ether

Only 11% of our set of identified transactions contains a transfer of Ether that is not either (a) a direct payment to the miner or (b) a wrapping or un-wrapping operation executed by Wrapped Ether [23]. We randomly sampled 10 of these transactions for manual analysis, and the transfers contained within all of them are benign transfers that do not impact the correctness of the analysis. So we are confident that this limitation does not compromise our results.

B Example Transactions

One cycle, two exchanges.

```
0x603761e4c2acdd3cef3a9e2e29f55009b18cdffd5696935bf59649910bb89943
0xd9f9aad09530cc54bbff46e1847bea281f54862cb10a595ff2163b6fa44fcffd
```

One cycle, three exchanges.

```
0x5739e2ab65ae48bbf60f89777a205d0056ddc0400f80034d05bb7526977cfad1
```

Two arbitrage cycles.

```
0x6d9e998d9ef0b1c3920c72f64e29a066b9e8ac29fdb19cea66dbb283f09d5cc0
```

C DEX Attribution from Labeling

Application	Use in Arbitrages (#)	Use in Arbitrages (%)
Uniswap v2	4,491,762	44.9%
Uniswap v3	1,524,600	15.2%
Sushi Swap	1,350,260	13.5%
Balancer v1	1,085,147	10.8%
Unknown	535,942	5.4%
0x	233,133	2.3%
Shiba Swap	172,880	1.7%
crypto.com	127,129	1.3%
Balancer v2	93,461	0.9%
Bancor V2	85,761	0.9%
curve.fi	81,409	0.8%
PowerPool	37,037	0.4%
SakeSwap	36,565	0.4%
indexed.finance	30,679	0.3%
Cream	24,749	0.2%
Orion V2	19,005	0.2%
Kyberswap	16,658	0.2%
SwipeSwap	12,415	0.1%
Bitberry	11,399	0.1%
WSwap	10,103	0.1%
Equalizer	9,611	0.1%
Convergence	7,636	0.1%
dodo	6,450	0.1%
defi plaza	4,976	0.0%
1inch	4,349	0.0%

Table 6: Count of times each DEX application was identified as an exchange in labeled arbitrage.

D Sandwich Attack with Arbitrage: Example Transaction Hashes

Manipulation transaction.

```
0x6fd4fc11d21877c4ee087790262da0fefaafd1f0f9e7ae8f35c1d53519525564
```

Victim transaction.

```
0x5ca70135b394e5a709b83032145216b2348ac9302423030ed1a622ea482572f5
```

De-manipulation transaction.

```
0x826903513f9077f0fe6ef19d3dd57af4db8956f381f57d72451e93f7eb8f7ac4
```

E AMM Exchanges in Selected DEXs

DEX Application	Count of Exchanges
Uniswap v2	79,884
Uniswap v3	7,272
Sushi Swap	2,984
ShibaSwap	475
Balancer v1	3,279
Balancer v2	601
All	94,495

F Thresholds on Exchange Balances

Token	Minimum Threshold
WETH	0.01
USDC	10
Tether	10
UNI	0.25
WBTC	0.0001

Table 7: Minimum holding thresholds imposed.

G Example of Optimization Procedure

Consider that we have two Uniswap v2 constant-product AMMs, AMM1 and AMM2. Both of these AMMs trade token α and β. AMM1 has 1 million units of α and 2 million units of β, whereas AMM2 has 1.5 million units of α and 2 million units of β. In this set-up, the spot exchange rate on AMM1 from α to β is 1:2, and for AMM2 it is 1.5:2, setting up the possibility of an arbitrage.

In Figure 7 we plot both the revenue, in units of α, and the marginal revenue, in units of α output per α input. Notice that the marginal revenue falls below 1 exactly at the point of optimal profit (about 111,400) of token α input. Our optimization procedure takes advantage of this fact to search for this marginal revenue crossing-point. Because the exchanges we model guarantee price slippage, all revenue curves must be concave-down, guaranteeing a unique solution like that shown here.

H Other Sources of Failure

Fee-on-transfer Required. Initially, we optimistically assume that a token's fee-on-transfer tax is zero. Occasionally, this assumption turns out to be incorrect. We precisely encode the expected amount of each token that is moved between exchanges at each step of the arbitrage. If our relayer smart contract encounters an unexpected balance, the execution reverts. We detect this by carefully tracking all calls to ERC-20 token's `transfer(..)` function, and we keep an accounting of the balance that we expect each address to possess. Each time a contract calls a token's `balanceOf(address)` function, we

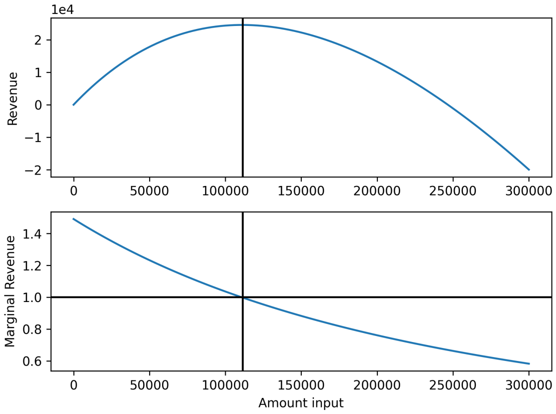

Figure 7: Simple arbitrage optimization problem example.

ensure that the balance returned for the specified address is exactly as expected by our accounting. If our accounting is incorrect, we automatically infer the fee multiple f_α and re-run the arbitrage opportunity routine with the updated fee-on-transfer.

Token Interferes with Exchange. Some ERC-20 fee-on-transfer tokens boost the profits of the token administrators by selling a portion of that fee on every call to `transfer(..)`. Occasionally, this interferes with the prices of an exchange in our cycle, which causes unexpected price behavior and ultimately transaction failure. If we ever observe such interference, we discard the potential arbitrage.

Exchange Balance Disorder. Some ERC-20 tokens use complex application-specific logic to determine an address' balance. Occasionally, this logic may actually *reduce* an address' balance without informing the owner. The AMM exchanges in our study keep an internal cache of the last known balance, and if the token's logic causes the cached value to become out of sync with the actual balance, the pricing behavior becomes unexpected. We detect and discard any affected potential arbitrages.

Other. We group all other failures into this category, including undiagnosed failures.

I Random Sample Analysis

When analyzing the arbitrages in the 30-day random sample, a significant number of them will turn unprofitable after accounting for gas fees paid to the block producer. We use the three gas-price oracles described in Section 4.2.1 to give an optimistic, realistic, and pessimistic gas price estimate. After application, profitable arbitrages amount to only 1.6, 1.3, and 1.1 million, respectively – a reduction of about 99% across the board.

Next, we compute the duration of these arbitrages. We find that all three gas price oracles yield the same aggregate duration statistics – the campaigns last for a duration of 1, 2, and 6 blocks at the 25th, 50th, and 75th percentile, respectively.

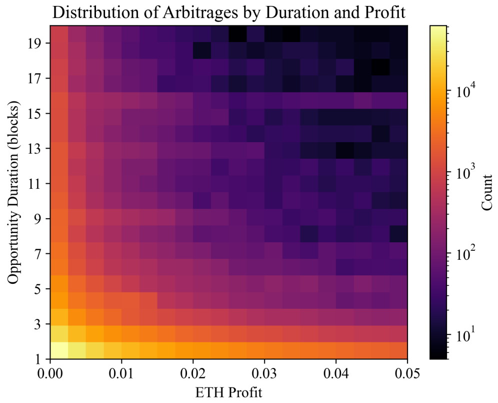

Figure 8: 2-dimensional histogram of profit and duration of arbitrage opportunities

Lastly, we compute the maximum total profit from these arbitrages. We perform this measurement from two directions. First, we maximize for extractable value on a per-block basis. Then, we use the method from Section 4.3.2 to sum extractable value across time.

To measure the total possible arbitrage profit on a per-block basis, we perform a linear scan through the study period and maintain a rolling record of the active arbitrages. At each block, we solve the maximal weighted independent set problem to find a conflict-free set of active arbitrages. We sum the profits from this set of conflict-free arbitrages to arrive at the maximum profit in each block. This shows that the total possible profit in a single block ranges from 0 to 60 Ether, with an average of 0.290 Ether and a median of 0.081 Ether.

	Count	%
All transactions	126,147,388	100%
Successfully executed	118,932,412	94.3%
All failures	7,214,976	5.7%
Token reverts	6,354,589	5.0%
Non-supported token	402,390	0.3%
Other	241,344	0.2%
No arb. after fee-on-transfer	135,497	0.1%
Interference	54,004	0.0%
Exchange-balance disorder	27,152	0.0%

Table 8: Results for a 30-day random sample.