Demystifying Just-in-Time (JIT) Liquidity Attacks on Uniswap V3

Xihan Xiong, Zhipeng Wang, William Knottenbelt, and Michael Huth
Imperial College London

***Abstract*—Uniswap is currently the most liquid Decentralized Exchange (DEX) on Ethereum. In May 2021, it upgraded to the third protocol edition named Uniswap V3. The key feature update is concentrated liquidity, which allows Liquidity Providers (LPs) to provide liquidity in custom price ranges. However, this design introduces a new type of Miner Extractable Value (MEV) source called Just-in-Time (JIT) liquidity attack, where the adversary mints and burns a position right before and after a sizable swap.**

In this paper, we first formally define the JIT liquidity attack and then conduct empirical measurements on Ethereum. We detect that the JIT liquidity attack is indeed a whales' game dominated by few bots, where the most active bot 0xa57...6CF siphons 92% of the attack profit. We observe that the attack presents extremely high barriers to entry, since it requires the adversary to add liquidity that is on average 269 times higher than the swap volume. In addition, we detect that the attack demonstrates poor profitability, with an average Return On Investment (ROI) ratio of only 0.007%. Furthermore, we find the attack detrimental to existing LPs in the pool, whose liquidity shares are diluted by an average of 85%. However, it is beneficial to liquidity takers, who obtain execution prices 0.139% better than before. We further dissect top MEV bots' behaviors and evaluate their strategies via local simulation. We find that the top first bot 0xa57...6CF issued 27% non-optimal attacks, thus failing to capture at least 7,766 ETH (16.1M USD) of the attack profit.

***Index Terms*—Decentralized Exchange, Blockchain, Decentralized Finance, Miner Extractable Value**

I. Introduction

Smart contracts enable building an ecosystem of financial products and services on top of permissionless blockchains, commonly referred to as Decentralized Finance (DeFi). DeFi is becoming increasingly popular, with the Total Value Locked (TVL) hitting an all-time-high of 179B USD on December 1st, 2021[1]. In addition to the fundamental functions drawn from traditional finance, DeFi also brings new innovative designs such as Automated Market Maker (AMM) DEX.

In contrast to centralized exchanges that rely on custodial infrastructures, DEXs allow users to trade cryptocurrencies directly in a non-custodial environment. AMM DEXs replace the traditional order matching system with smart-contract-enabled algorithmic models that determine the prices at which buyers and sellers can trade assets in liquidity pools. Constant Function Market Maker is a board class of AMMs that is widely adopted by most DEXs (e.g., Uniswap, SushiSwap, Curve). Uniswap is the most liquid DEX on Ethereum, with TVL of 3.85B USD and daily trading volume of 1.86B USD.

Adams *et al.* launched Uniswap V1 [1] in November 2018. Uniswap V2 [2] went alive in May 2020, and further upgraded to V3 [3] in May 2021. The Uniswap V1 contract supports only the ETH–ERC-20 liquidity pool, while V2 allows the swap between any ERC-20–ERC-20 token pairs. Besides, Uniswap V2 also introduces Wrapped Ether (WETH) into its core contract. However, Uniswap V2 requires the LPs to provide liquidity in the entire price range (e.g., from 0 to $+\infty$), which results in capital inefficiency because only a small fraction of assets are available at a given price [3]. To improve capital efficiency, Uniswap V3 introduces a new design called *concentrated liquidity*, which allows LPs to concentrate their liquidity in custom price ranges smaller than $[0, +\infty]$ to supply more liquidity at targeted prices. Uniswap V3 also introduces the concept of *active liquidity*. If the price of assets trading in the liquidity pool moves outside LP's specified price range $[p_l, p_u]$, the LP's position becomes inactive and stops earning fees. Therefore, a rational LP is incentivized to concentrate its liquidity in a profitable price range yet keep its position active.

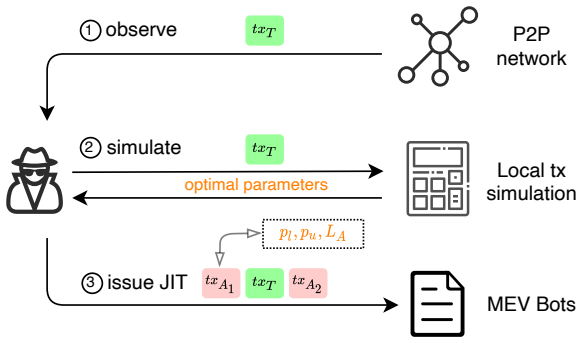


Fig. 1: Overview of JIT liquidity attack.

Nevertheless, Uniswap V3's concentrated liquidity design introduces a new type of MEV source, known as *JIT liquidity attack*, where the adversary mints and burns a position immediately before and after a sizable swap transaction (cf. Figure 1). More specifically, the adversary monitors the mempool via its spy node. Once observing a sizable pending swap, the adversary simulates the JIT liquidity attack locally with chosen parameters and launches the attack if profitable.

While the common MEV sources (e.g., arbitrage, liquidation and sandwich attack) have been extensively researched in recent academic work, the mechanism of JIT liquidity attacks has been studied to a very limited extent. This study aims to provide a scientific formalization of JIT liquidity attack, understand the attack impact by empirical measurement, and evaluate the adversarial strategies via local simulation. The

[1]https://defillama.com/, last accessed on April 14th, 2023.

main contributions of this paper are summarized as follows:

Attack Formalization. We formalize the JIT liquidity attack and adversarial utility function. We find that the price range $[p_l, p_u]$ and the liquidity amount L_A are parameters to optimize.

Empirical Measurement. We conduct empirical measurements of JIT liquidity attacks on Ethereum. During the course of 20 months, We identify 36,671 attacks with a total profit of 7,498 ETH. We recognize the JIT liquidity attack as a whales' game dominated by few bots, where the most active bot 0xa57...6CF siphons 92% of the total profit. Besides, we observe that the attack presents extremely high barriers to entry, since it requires the adversary to add liquidity that is on average 269 times higher than the swap volume. In addition, we discover that JIT liquidity attack demonstrates poor profitability performance, with an average ROI of only 0.007%. Furthermore, we detect JIT liquidity attack detrimental to existing LPs in the pool, whose liquidity shares are diluted by an average of 85%. However, it is beneficial to liquidity takers, who obtain execution prices 0.139% better than before.

Comparision Study. We compare JIT liquidity attacks with sandwich attacks. We find that the sandwich attack presents lower entry barriers and higher profitability performance. The initial capital needed for a sandwich attack is only 6 times the swap volume. While the top first JIT bot 0xa57...6CF only achieves an average ROI of 0.013%, the top first sandwich bot 0x000...B40 manages to achieve an average ROI of 1.642%.

Strategy Analysis. We first analyze how top MEV bots choose attack parameters. We find that the top first bot 0xa57...6CF always provides all its token balance to add liquidity in each JIT liquidity attack. We then evaluate the top bots' strategies via local simulation. Interestingly, our simulation result shows that 0xa57...6CF issued 27% non-optimal attacks, thus failing to capture at least 7,766 ETH (16.1M USD) of the attack profit.

II. Related Work

Heimbach *et al.* [4] empirically investigate Unsiwap V3's resilience to unexpected price shocks and find that the price on Uniswap is inaccurate during UST–USDT stablecoin price shocks. Loesche *et al.* [5] study the impermanent loss in Uniswap V3 by analyzing 17 liquidity pools. The result shows that providing liquidity in V3 pools makes around 50% of the LPs unprofitable compared to simply holding the assets. Heimbach *et al.* [6] examine the risks and returns of Uniswap V3 LPs and find that high returns can only be obtained by accepting high risks. Hashemseresht *et al.* [7] analyze the effect of concentrated liquidity on the return of the liquidity providers. Aigner *et al.* [8] discusses the impermanent loss and risk profile of Uniswap V2 and V3 LPs. Wan *et al.* [9] investigate the historical JITs on Uniswap V3 and discuss the microeconomic considerations. The primary distinctions between this study and [9] are that: *(i)* we formally defined the adversarial utility function and the parameters to optimize; *(ii)* We provide a more thorough empirical analysis; and *(iii)* we study the top MEV bots' behaviors and evaluate their strategies.

III. Uniswap Liquidity Math

A. Uniswap V2: Zero to Infinity

For a simple X–Y liquidity pool, the V2 contracts require the token reserves to satisfy $x * y = k$, where k is a constant.

Liquidity Taker. Let p_0 be the marginal price of X (i.e., $p_0 = \frac{y_0}{x_0}$). Given the spot price p_0, for a liquidity taker who is willing to trade Δ_x amount of X for Δ_y amount of Y, the pool reserve should change to $x_1 = x_0 + \Delta_x$ and $y_1 = y_0 - \Delta_y$ such that $(x_0 + \Delta_x) * (y_0 - \Delta_y) = k$ holds when the swap ends.

Liquidity Provider. An LP who provides liquidity to the X–Y token pool must provide liquidity across the entire price range of $[0, +\infty]$. Such liquidity provision design results in capital inefficiency because most of the liquidity remains unused [3].

B. Uniswap V3: Concentrated Liquidity

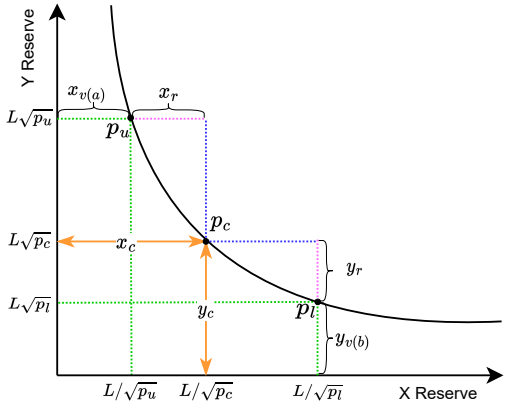


Fig. 2: Uniswap V3 price curve. Given the spot price p_c, an LP who wants to mint a position in the price range $[p_l, p_u]$ only needs to provide the real reserve of x_r and y_r.

1) Liquidity and Price: To improve capital efficiency, Uniswap V3 introduces a novel AMM design called *concentrated liquidity*, which allows LPs to add liquidity in a customized price range of $[p_l, p_u]$. In Uniswap V3, a liquidity pool can be thought of as having *virtual liquidity* L and *virtual reserves* x and y such that $x * y = k = L^2$. The pool contract tracks `Liquidity` (L) and the marginal token price `sqrtPriceX96` ($\sqrt{P}$). As such, the relationship between price and liquidity can be expressed as follows:

$$\begin{aligned} L &= \sqrt{xy}, \quad \sqrt{P} = \sqrt{\frac{y}{x}} \\ x &= \frac{L}{\sqrt{P}}, \quad y = L\sqrt{P} \end{aligned} \tag{1}$$

2) Positions and Ticks: When an LP adds liquidity in $[p_l, p_u]$, a new *Position* is created and an Non-fungible Token (NFT) is minted. To guarantee customized liquidity provision, Uniswap V3 used discrete *ticks* to delineate the possible prices in a range. Ticks represent prices at which the contract's virtual liquidity may change. The price at tick i is 1.0001^i.

$$p(i) = 1.0001^i, \quad \sqrt{p(i)} = 1.0001^{\frac{i}{2}} \tag{2}$$

Every pool in Uniswap V3 is initialized with a given `tickSpacing`. Uniswap V3 supports fee tiers of $\{0.01\%, 0.05\%, 0.3\%, 1\%\}$, which corresponds to the `tickSpacing` of $\{1, 10, 60, 200\}$. Tick i is initialized when liquidity is added to a range where tick i serves as the lower/upper bound and tick i is not referenced by any existing position. Note that only ticks that are divisible by `tickSpacing` can be initialized.

3) Liquidity Provision: Given spot price p_c, consider an LP who aims to provide liquidity in the price range $[p_l, p_u]$ ($p_l < p_c < p_u$) (cf. Figure 2). Recall that Uniswap v3 always ensures that the virtual reserves satisfy $x * y = L^2$. As such, the LP must provide x_r amount of X token and y_r amount of Y token, such that when the provided token amounts are added to the virtual reserves of X and Y in $[p_l, p_u]$, the resulting curve will behave according to the Constant Product Market Maker pricing curve. From Equation 1, the virtual reserves of X token at price p_u is $x_{v(u)} = \frac{L}{\sqrt{p_u}}$, and the virtual reserves of Y token at price p_l is $y_{v(l)} = L\sqrt{p_l}$. Hence, the x_r and y_r reserve provided by the LP must satisfy Equation 3 as follows:

$$(x_r + \frac{L}{\sqrt{p_u}})(y_r + L\sqrt{p_l}) = L^2 \tag{3}$$

From p_c, we know that the reserves are $x_c = \frac{L}{\sqrt{p_c}}$ and $y_c = L\sqrt{p_c}$. Hence, we can derive x_r and y_r by Equation 4:

$$x_r = \frac{L}{\sqrt{p_c}} - \frac{L}{\sqrt{p_u}}, \quad y_r = L\sqrt{p_c} - L\sqrt{p_l} \tag{4}$$

Note that this position only needs to hold enough amounts of token X and Y to support trading in the price range $[p_l, p_u]$. Particularly, when the marginal price P (i.e., the price of X in terms of Y) moves to the upper bound p_u, the reserve of X is fully depleted and the LP's position only holds Y token. Similarly, when the marginal price P moves to the lower bound p_l, the LP's position only holds X token. In other words, a position is active only if the pool's current price is within the specified price range. While Figure 2 describes the scenario where the current price is within $[p_l, p_u]$, it can be generalized to a broader format to resolve x_r and y_r (cf. Equation 5).

$$x_r = \begin{cases} \frac{L}{\sqrt{p_l}} - \frac{L}{\sqrt{p_u}} & p_c < p_l \\ \frac{L}{\sqrt{p_c}} - \frac{L}{\sqrt{p_u}} & p_l \le p_c \le p_u \\ 0 & p_u < p_c \end{cases}$$
$$y_r = \begin{cases} 0 & p_c < p_l \\ L\sqrt{p_c} - L\sqrt{p_l} & p_l \le p_c \le p_u \\ L\sqrt{p_u} - L\sqrt{p_l} & p_u < p_c \end{cases} \tag{5}$$

4) Transaction Fees: The LPs of a given liquidity pool earn fees when traders transact one asset for another. For instance, a trader who aims to swap Δx amount of USDC for ETH in the USDC–WETH pool with a fee tier of 0.05% pays $0.05\%\Delta x$ as the trading fee. The trading fees will be distributed among active LPs based on the liquidity they provide. For Uniswap V3, an LP receives fees only if the corresponding position is active. In case the price moves outside the specified range, the LP's position becomes inactive and thus stops earning fees. When the price reenters the range, the position becomes active and receives fees again. It is worth noting that while fees earned in Uniswap V2 are auto-compounded in the pool, they are stored separately in Uniswap V3 and held as tokens in Uniswap V3 [3]. Furthermore, there might be multiple V3 pools with different fee tiers for the same X–Y asset pairs.

IV. Models

A. System Model

We consider a blockchain peer-to-peer (P2P) network and assume the existence of the following participants in the system:

Trader T: who initiates a swap tx_T on Uniswap V3;
Adversary A: who observes the pending transaction tx_T through its own spy node (e.g., a custom Ethereum client) and launches a JIT liquidity attack over the targeted transaction;
LPs L: who served liquidity to the liquidity pool implied by tx_T, supporting the swaps in the pool and earning fees;
Miners and Validators $\mathcal{M}$: a set of miners (in a Proof-of-Work blockchain) or validators (in a Proof-of-Stake blockchain) who have the capacity to manipulate the order of transactions.

B. Threat Model

We consider an economically rational adversary $\boldsymbol{A}$ who is well-connected to the P2P network and is able to monitor unconfirmed transactions in the mempool through its spy node. We also assume that $\boldsymbol{A}$ holds a sufficient balance of assets required by the corresponding JIT liquidity attack to issue transactions. In addition, $\boldsymbol{A}$ can participate in auctions for priority transaction inclusion, in an attempt to extract the MEV contained in the JIT liquidity attack. More specifically, $\boldsymbol{A}$ can submit transactions to miners/validators either *(i)* publicly through the P2P network or *(ii)* privately through a centralized Front-running as a Service (e.g., Flashbots). To compete with other MEV searchers, $\boldsymbol{A}$ can bribe the miners/validators by offering high gas prices or through direct Coinbase transfer.

C. Attack Model

1) Pool Mechanism: We consider the following pool actions.

`Mint`: An LP add liquidity ΔL to Pool(X,Y,ϕ) by specifiying a price range $[p_l, p_u]$ and the amount (ΔX_d, ΔY_d) desired to be added (cf. Equation 6). A unique NFT is minted accordingly to represent this position. The periphery contract computes ΔL automatically (cf. Equation 7). It first calculates ΔL according to the amount (ΔX_d, ΔY_d), and then computes the actual amounts (ΔX_a, ΔY_a) added to the pool based on ΔL.

$$(L, \sqrt{P}) \xrightarrow[\Delta X \in R+,\ \Delta Y \in R+]{\texttt{Mint}(p_l, p_u, \Delta X_d, \Delta Y_d)} (L + \Delta L, \sqrt{P}) \tag{6}$$

$$L = \begin{cases} \Delta X_d(\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}) & p_c < p_l \\ \min\{\Delta X_d(\frac{1}{\sqrt{p_c}} - \frac{1}{\sqrt{p_u}}), \Delta Y_d \frac{1}{\sqrt{p_c} - \sqrt{p_l}}\} & p_l \le p_c \le p_u \\ \Delta Y_d \frac{1}{\sqrt{p_u} - \sqrt{p_l}} & p_u < p_c \end{cases} \tag{7}$$

Burn: An LP burns its position by removing liquidity ΔL from Pool(X,Y,ϕ). The LP will collect the remaining X, Y reserves, and the swap fees (held separately in tokens) up to a maximum amount of fees owed to its position to the recipient.

$$(L, \sqrt{P}) \xrightarrow[\text{NFTID} \in R+]{\texttt{Burn}(\text{NFTID})} (L - \Delta L, \sqrt{P}) \tag{8}$$

Swap: A liquidity taker can swap one token for another. `ZeroForOne` is a boolean variable and is true when swapping X for Y. `AmountSpecified` configures the swap as exact input when it is positive, and exact output when it is negative. While a `Swap` changes $\sqrt{P}$, it does not necessarily change L.

$$(L, \sqrt{P}) \xrightarrow[\text{ZeroForOne} \in \{0,1\},\ \text{AmountSpecified} \in R]{\texttt{Swap}(\text{ZeroForOne},\ \text{AmountSpecified})} (L', \sqrt{P'}) \tag{9}$$

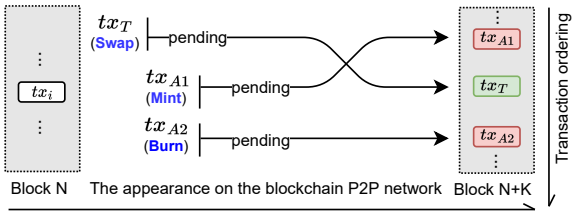


Fig. 3: JIT liquidity attack mechanism.

2) Attack Mechanism: ❶ The adversary $\boldsymbol{A}$ observes a sizable pending (zero-confirmation) swap tx_T in the mempool via its spy node; ❷ $\boldsymbol{A}$ predicts the price range $[p_l, p_u]$ into which the tx_T will be in range for, and issues tx_{A1} right before tx_T to mint a new position by adding liquidity in $[p_l, p_u]$. ❸ $\boldsymbol{A}$ issues tx_{A2} immediately after tx_T to burn its position by removing liquidity in $[p_l, p_u]$ and collect the earned fees .

3) Utility Formalization: By launching a JIT liquidity attack, a rational adversary $\boldsymbol{A}$ aims to maximize its financial gain.

Revenue. The adversarial revenue comes from two sources: *(i)* the *swap fee* taken unfairly from other LPs; and *(ii)* the *portfolio value change* caused by price impact of the trade.

$$\text{Revenue} = \text{Fee} + \Delta\text{Value} \tag{10}$$

The liquidity taker is responsible for paying fees for swap execution. The adversary earns swap fees proportional to how much liquidity it contributes compared to the total liquidity during the swap interval (cf. Equation 11). If the swap is executed crossing ticks, $\boldsymbol{A}$ earns swap fees only for the swap range covered by the price range in which $\boldsymbol{A}$ supplies liquidity.

$$\text{Fee}(L_A) = \begin{cases} \Delta x_{\text{in}} \cdot \phi \cdot L_A / L_T \cdot P_{(ETH/X)} & \text{ZeroForOne} \\ \Delta y_{\text{in}} \cdot \phi \cdot L_A / L_T \cdot P_{(ETH/Y)} & \text{!ZeroForOne} \end{cases} \tag{11}$$

$\boldsymbol{A}$ may also benefit from the change of its portfolio value. The marginal price of asset X (i.e., `sqrtPriceX96`) changes because of the swap execution, thus affecting the adversarial portfolio value. Equation 12 calculates the adversarial portfolio value change (in ETH) as the difference between the value after $(p_1 \cdot x_1 + y_1)$ and before $(p_0 \cdot x_0 + y_0)$ swap, where P_0 and P_1 denote X's marginal prices before and after respectively.

$$\Delta\text{Value}(p_1, x_1, y_1) = [(p_1 \cdot x_1 + y_1) - (p_0 \cdot x_0 + y_0)] \cdot P_{(ETH/Y)} \tag{12}$$

Let $V_0 = p_0 \cdot x_0 + y_0$ and $L_T = L_A + L_0$, Equation 13 represents ΔValue as the function of L_A, p_l and p_u (cf. Appendix A).

$$\Delta\text{Value}(L_A, p_l, p_u) = \begin{cases} [L_A(\sqrt{p_u} - \sqrt{p_l}) - V_0] \cdot P_{(ETH/Y)} \\ \qquad \text{if } p_u < p_1, \text{ which implies !ZeroForOne;} \\ [L_A(\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}}) \frac{(L_A + L_O)\sqrt{p_0}}{(L_A + L_O) + \sqrt{p_0}\Delta x_{\text{in}}} - V_0] \cdot P_{(ETH/Y)} \\ \qquad \text{if } p_1 < p_l, \text{ which implies ZeroForOne;} \\ [L_A(\sqrt{p_u} - \sqrt{p_l} - \frac{1}{\sqrt{p_u}}(\frac{(L_A + L_O)\sqrt{p_0}}{(L_A + L_O) + \sqrt{p_0}\Delta x_{\text{in}}} - \sqrt{p_u})^2) - V_0] \cdot P_{(ETH/Y)} \\ \qquad \text{if } p_l \le p_1 \le p_u \text{ and ZeroForOne;} \\ [L_A(\sqrt{p_u} - \sqrt{p_l} - \frac{1}{\sqrt{p_u}}((\sqrt{p_0} + \frac{\Delta y_{\text{in}}}{L_A + L_O}) - \sqrt{p_u})^2) - V_0] \cdot P_{(ETH/Y)} \\ \qquad \text{if } p_l \le p_1 \le p_u \text{ and !ZeroForOne;} \end{cases} \tag{13}$$

Cost. The adversarial cost comes from two sources: *(i)* the gas cost of the mint (tx_{A1}) and burn (tx_{A2}) transaction; and *(ii)* the amount of ETH transferred to miners/validators for priority inclusion. $\boldsymbol{A}$ may participate in auctions for the priority transaction ordering in the block. As such, in addition to the gas cost, $\boldsymbol{A}$ may also bribe the miners/validators via direct Coinbase transfer through smart contracts.

$$\text{Cost} = \text{gas}(tx_{A1}) + \text{gas}(tx_{A2}) + \text{transfer} \tag{14}$$

Profit. $\boldsymbol{A}$ aims to maximize its profit by choosing optimal values (cf. Equation 15) for the following parameters: *(i)* the upper and lower bound for the price range $[p_l, p_u]$ in which $\boldsymbol{A}$ adds liquidity; and *(ii)* the amount liquidity L_A to add. Without loss of generality, we assume that the cost is invariant to the parameter values. Hence, maximizing profit is equivalent to maximizing revenue (cf. Equation 15).

$$(L_A, p_l, p_u) = \underset{(L_A, p_l, p_u)}{\text{argmax}}\ (\text{Fee} + \Delta\text{Value}) \tag{15}$$

V. Empirical Measurement

This section presents empirical measurement on JIT liquidity attacks based on real-world data of Uniswap V3. We provide the overall statistics of JIT liquidity attacks, analyze the adversarial revenue and cost, and dissect the adversarial strategies.

A. Data Collection

We conduct our empirical measurement for JIT liquidity attacks on Ethereum from blocks 12545219 (June 1st, 2021) to 16530247 (Jan 31st, 2023), during the course of 20 months. We apply the following heuristic to identify JIT liquidity attacks.

Heuristic 1 (JIT Liquidity Attack Heuristics). $\boldsymbol{A}$ mints (tx_{A1}) and burns (tx_{A2}) an LP position in the same block. The target transaction (tx_T) is a swap in the same pool. The issuer of tx_{A1} and tx_{A2} is different from the issuer of tx_T. The transaction indices of tx_{A1}, tx_T and tx_{A2} are consecutive.

B. Overall Statistics

By applying Heuristic 1, we have successfully identified 36,671 JIT liquidity attacks on Ethereum. We calculate the

distance between the tick before and after the swap as $d = |i_1 - i_0|/\texttt{tickSpacing}$. We discover that 97.84% the price change is less than or equal to one `tickSpacing` (cf. Table III).

Insight 1. The JIT liquidity attacks present extremely high barriers to entry. Participating in JIT liquidity attacks requires holding a significant amount of initial funds.

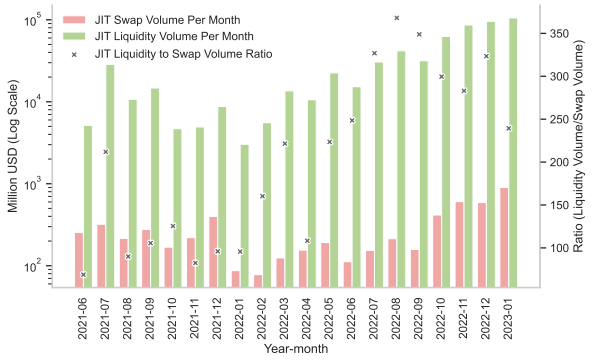


Fig. 4: The average JIT liquidity to swap volume ratio is 269×.

Our data show that the total swap volume impacted by JIT liquidity attacks is 5.64B USD, and the average swap volume is 153.8K USD. Hence, we can infer that ***A*** only targets sizable swaps in the mempool to maximize its utility. Interestingly, we discover that the total liquidity provided by adversaries amounts to 602.3B USD, and the average liquidity volume exceeds 16.4M USD. This result indicates that to initiate a JIT liquidity attack on tx_T, an adversary needs to provide liquidity that is on average 269 times higher than the swap volume of tx_T (cf. Figure 4). Hence, we conclude that the JIT liquidity attack presents extremely high barriers to entry.

We find that the USDC–WETH–0.03% (0x88e...640) is the most popular Uniswap V3 pool for JIT liquidity attacks. Our data show that the USDC–WETH-0.03% pool attracts 47% of the JIT liquidity attacks (17,368 / 36,671, cf. Figure 12b), and 60% of the revenue (6,123 / 10,125 ETH, cf. Figure 12e). In addition, Figure 12a and 12d shows that the USDC–WETH–0.03% takes up most of the JIT liquidity volume (72%) and swap volume (61%). Besides, while comparing different fee tiers, we observe that the 0.05% fee tier captures most of the JIT liquidity volume (51%) and swap volume (33%).

C. Profitability Analysis

While the accumulative attack profit is growing, the profit-making potential of various JIT attackers varies. In the following, we provide a detailed analysis of adversarial profitability.

1) Revenue: We detect that the total revenue amounts to 10,125 ETH, and the average revenue is 0.276 ETH.

Insight 2. JIT liquidity attacks dilute existing LPs's liquidity shares in the pool by an average of 85%, thus significantly damaging their financial interest.

JIT adversaries have taken a tremendous amount of swap fees from existing LPs in the liquidity pool. Our data show that the total revenue of 10,125 ETH earned by JIT adversaries consists of 7,111 ETH of portfolio change revenue and 3,014 ETH of swap fee revenue taken from other LPs. For each attack, an average of 85% swap fees are exploited by adversaries.

Besides, we find that the portfolio value change is not always beneficial to JIT adversaries. From November 2022, the change of portfolio value is the main source for JIT revenue (with up to 83% contribution). However, before November 2022, the swap fee and portfolio revenue dominate the revenue generation alternatively. In fact, we detect that the portfolio value change presents a positive contribution only in 17,743 out of 36,671 JIT liquidity attacks, which implies that the adversary must carefully consider the impact of portfolio value change.

Insight 3. Instead of the gas fee, the direct transfer incurs most of the cost. Over 99% of the detected JIT liquidity attacks do not use the gas fee to reward miners/validators.

The main source of JIT cost is the direct Coinbase transfer used to bribe the miners/validators. The empirical result shows that the average cost of a JIT liquidity attack is 0.072 ETH, and the total cost amounts to 2,027 ETH, which consists of 539 ETH of gas cost and 2,088 ETH of direct transfer to miners/validators. We condcut a detailed breakdown of JIT cost. The direct Coinbase transfer is increasingly serving as the main source of JIT cost, with a cost proportion of 81% in January 2023. This result indicates that the MEV game is becoming increasingly competitive. In fact, we detect that on average, the JIT adversaries are willing to sacrifice $43.7\% \pm 25.5\%$ of their revenue to bribe the miners/validators.

Interestingly, in most of the attacks, adversaries do not use gas fees to bribe miners/validators. Before EIP-1559, the adversary could set the gas price of a transaction to zero, and bribe the miners only via direct transfer. We detect 2,464 (99.91%) such JIT attacks. After EIP-1559, the adversary pays both the base fee and priority fee as the gas cost. The gas price cannot be set as zero because the adversary must pay the base fee. However, the adversary can set the priority fee to zero and only use direct transfer as the reward to miners. We detect 34,048 (99.54%) attacks with zero priority fee.

2) Profit: We calculate adversarial profit using Equation 15.

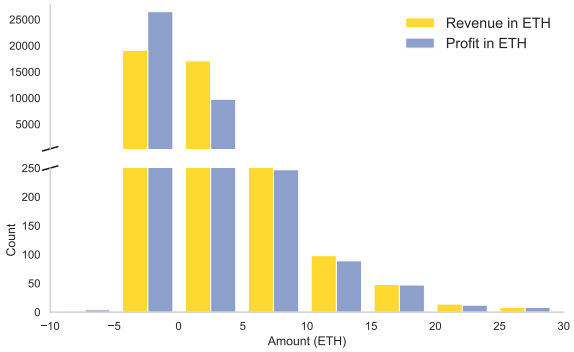


Fig. 5: Histogram of JIT revenue and profit.

Insight 4. JIT liquidity attacks demonstrate poor profitability performance because they have extremely low ROI.

The adversaries obtain a total profit of 7,498 ETH, and an average profit of 0.204 ETH. Surprisingly, only 20,068 (54.7%) JIT liquidity attacks are profitable. Figure 5 shows the distribution of adversarial revenue and profit, which is right-skewed with a tail in the positive region. For unprofitable attacks, the adversary's loss concentrates in $[-0.1, 0]$ ETH. The adversary earns more when the attack is profitable.

Besides, we observe that JIT liquidity attacks demonstrate extremely low ROI. ROI is a metric to evaluate the performance of an investment. It is calculated as the profit over the invested capital. The average ROI for JIT liquidity attacks is only 0.007%, indicating poor profitability performance.

3) Top MEV Bots: We proceed to analyze the strategy and behavior of top MEV bots who have the highest profitability (i.e., 0xa57...6CF, 0x57C...c94 and 0xCD9...5C4).

Insight 5. The JIT liquidity attack is indeed a whales' game that is inaccessible to retail MEV searchers.

Interestingly, the 36,671 detected JIT liquidity attacks were related only to 18 MEV bots. By tracing the coin flow of the top first MEV-bot 0xa57...6CF, we detect that it uses 308 Externally-Owned Accounts to launch attacks but the earned funds are all transferred to the same receiver 0x561...Bf9 (cf. Figure 13). Surprisingly, 0xa57...6CF has issued 27,983 (76%) attacks, consumed 435 ETH (81%) as the gas fee, transferred 1,628 (78%) ETH to bribe miners/validators, and siphons 6,900 ETH (92%) as attack profit (cf. Table I). It is evident that 0xa57...6CF has dominated the entire JIT game, leaving few opportunities for retail MEV searchers. Does the top first bot 0xa57...6CF achieve higher profitability? Yes, as 0xa57...6CF earns an average profit of 0.247 ETH, which is higher than the average profit. However, we can see that the average ROI achieved by 0xa57...6CF is still very low (only 0.013%). This again highlights that JIT liquidity attacks are low-return investments with poor profitability performance.

D. Impact on Liquidity Takers

How do JIT liquidity attacks affect liquidity takers (i.e., the issuer of tx_T)? In order to answer the question, for every JIT liquidity attack, we simulate the pool state for the same swap tx_T as if the attack (i.e., tx_{A1} and tx_{A2}) had never happened.

Insight 6. Liquidity takers benefit from JIT liquidity attacks. They get better swap prices than they would have otherwise.

We compare the liquidity taker's price slippage with/without JIT liquidity attack. Interestingly, we find that liquidity takers obtain better execution prices with JIT liquidity attacks. The average price improvement is 0.139%. This is reasonable, since after the adversary mints its position, the pool's liquidity increases and the price slippage decreases. We further compare the price improvement across pools with different fee tiers (cf. Table IV). We detect that liquidity takers in 1% fee tier pools obtain the highest average price improvement of 0.699%.

E. Comparison Study

This section compares JIT liquidity attacks with sandwich attacks by analyzing their attack mechanisms and statistics.

1) Attack Mechanism: A JIT liquidity attacker $\boldsymbol{A}$ attempts to mint and burn a concentrated position immediately before and after the target swap transaction. In contrast, upon observing a target pending swap tx_T (e.g, swap X for Y), a sandwich attacker $\boldsymbol{A'}$ attempts to front-run tx_T by purchasing asset Y and back-run tx_T by purchasing asset X. As such, the liquidity taker bears higher slippage than anticipated.

2) Attack Statistics: We proceed to compare the JIT and sandwich attack statistics on Uniswap V3. We conduct our empirical measurement on Ethereum from blocks 12545219 (June 1st, 2021) to 16530247 (Jan 31st, 2023). We apply the heuristics mentioned in [10] to identify sandwich attacks.

Insight 7. Compared with JIT liquidity attacks, sandwich attacks present lower entry barriers and higher profitability.

Overall Statistics. we identify in total 208,149 sandwich attacks, which is 5.7× the number of JIT liquidity attacks. We observe that the sandwich attacks present lower entry barriers. In contrast to JIT liquidity attacks which require the adversarial to hold a significant amount of initial funds that is on average 269 times higher than the swap volume, a sandwich attack requires an initial capital of 8.37 ETH, which is on average 6 times higher than the swap volume, indicating that the sandwich game is more accessible to retail MEV searchers.

Profitability. Our data show that sandwich attackers earn a total and an average profit of 12,242 ETH and 0.059 ETH respectively. The average ROI for sandwich attacks is 1.629%, implying a better profitability performance than JIT liquidity attacks. In addition, we observe that on average, sandwich attackers are willing to sacrifice $11.5\% \pm 25.9\%$ of their attack revenues to bribe the miners/validators, which is much lower than that of JIT liquidity attacks.

Top MEV Bots Statistics. We detect 143 bots participating in the sandwich game. The top sandwich MEV bots by profits are 0x000...B40, 0x000...e7D and 0x000...94e, who siphon 38%, 18% and 16% of the total profit respectively. This result indicates that the sandwich game is less monopolized than the JIT game. Surprisingly, while the top first JIT bot 0xa57...6CF only achieves an average ROI of 0.013%, the top first sandwich bot 0x000...B40 manages to achieve an average ROI of 1.642%.

Impact on Liquidity Takers. While JIT liquidity attacks improve the liquidity takers' execution price by an average of 0.139%, our simulation result show that sandwich liquidity takers' execution prices are worsened by an average of −5.93%.

VI. JIT Strategy Analysis

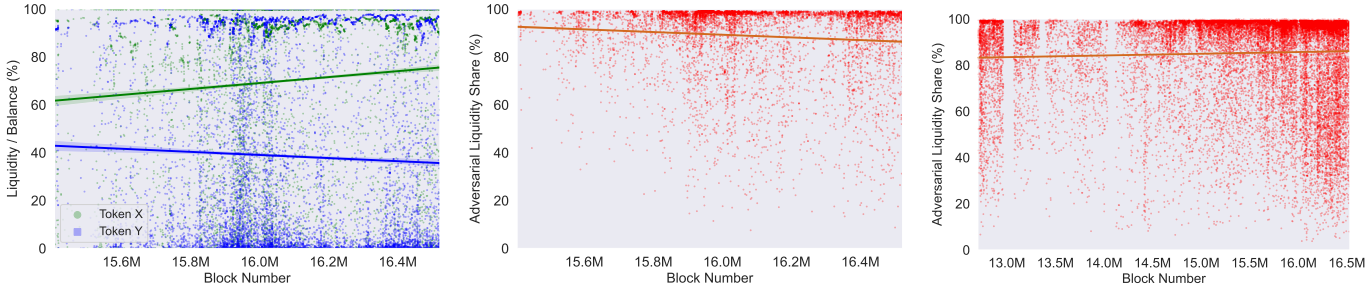
A. Attack Stratgy

As shown in Figure 1, we observe that the existing JIT attackers all adopt the following strategies in practice:

① The adversary $\boldsymbol{A}$ observes a sizable pending and potentially profitable swap transaction tx_T in the network via its spy node;

	num. Attacks	Revenue	Avg. Rev.	Gas Fee	Avg. Gas	Transfer	Avg. Trans.	Profit	Avg. Pro.	Liq. Share	Bribe Ratio	ROI
All	36,671	10,125	0.276	539	0.015	2,088	0.057	7498	0.204	85.42%	23.94%	0.007%
0xa57	27,983 (76%)	8,963 (88%)	0.320	435 (81%)	0.016	1628 (78%)	0.058	6900 (92%)	0.247	85.27%	23.39%	0.013%
0x57C	7,808 (21%)	981 (10%)	0.126	94 (17%)	0.012	403 (19%)	0.052	484 (6%)	0.062	88.85%	26.30%	0.005%
0xCD9	454 (1%)	84 (1%)	0.186	1 (0%)	0.002	30 (1%)	0.066	53 (1%)	0.117	45.16%	21.78%	0.033%
Others	426 (1%)	95 (1%)	0.225	10 (2%)	0.022	26 (1%)	0.062	60 (1%)	0.141	75.32%	18.81%	-0.325%

TABLE I: Overview of top MEV bots statistics. The revenue, gas fee, Coinbase transfer, and profit are measured in ETH.



(a) The liquidity/balance ratio for 0x57C...c94. (b) Adversarial liquidity share for 0x57C...c94. (c) Adversarial liquidity share for 0xa57...6CF.

Fig. 6: Statistics of top MEV bots behaviors.

② ***A*** simulates tx_T locally to predict the pool price after the execution of tx_T. In this step, ***A*** searches the optimal values for price range$[p_l, p_u]$ and the amount liquidity L_A to add.
③ ***A*** issues the add transaction tx_{A_1} and the burn transaction tx_{A_2} by calling the MEV bot contract. The input data of tx_{A_1} includes the optimal parameters generated in Step ②.
Note that Step ② is performed off-chain and the adversary searches the optimal values privately. However, we can leverage the on-chain public data to decode tx_{A1}'s input and execution process, which can help analyze how the adversary chooses the liquidity amount L_A and the price range $[p_l, p_u]$.

1) Liquidity Amount L_A Choice: We observe two methods to choose the liquidity amount L_A: *(i)* For all of the 7,932 (22%) JIT attacks launched by the bots 0x57C...c94 and 0x596...87E, L_A is explicitly specified in the add transaction input. *(ii)* For add transactions issued by other JIT bots, e.g., 0xa57...6CF, L_A is not specified in the transaction input data. Instead, the bots input ΔX_d and ΔY_d that they desire to add to the pool.

We take 0xa57...6CF and 0x57C...c94 as examples to compare their adding token amounts with the corresponding pool liquidity and their balances. We observe that, although the bot 0x57C...c94 does not always choose to supply all its token balance to add liquidity to the pool (cf. Figure 6a), the average liquidity share occupied by 0x57C...c94 is larger than that occupied by 0xa57...6CF (cf. Figures 6b and 6c).

2) Price Range $[p_l, p_u]$ Choice:

3) Price Range $[p_l, p_u]$ Choice: To analyze the price range choice of JIT attackers, we compare the tick ranges $[i_l, i_u]$ set in their mint transactions with the tick i_0 before the swap transaction and the tick i_1 after the swap. We observe that 18,766 (51%) swap transactions do not trigger the tick update in the liquidity pool (i.e., $i_0 = i_1$), and 17,112 (47%) swap transactions only trigger the tick update less than one tick space (i.e., $|i_1 - i_0| \leq 1\times$ `tickSpacing`, cf. Table III and II). This can well explain why almost all (more than 99.99%) JIT attacks only cover one single tick space when setting the tick ranges $[i_l, i_u]$, i.e., $|i_u - i_l| = 1\times$ `tickSpacing`.

Algorithm 1: Attack Simulation for 0xa57...6CF

Data: Original JIT transactions tx_{A_1}, tx_T and tx_{A_2}
Result: optimal lower tick $i_l^{optimal}$, maximum profit P_{max}

i_l^{actual}, `tickSpacing` $\leftarrow$ decode tx_{A_1} input data ;
$i_l^{optimal} \leftarrow i_l^{actual}$;
$P_{max} \leftarrow 0$;
while $-4 \leq \delta \leq 4$ **do**
  $\hat{tx_{A_1}} \leftarrow$ modify tx_{A_1} by setting the lower tick provided in the input data as $i_l^{actual} + \delta \cdot$ `tickSpacing`;
  $P \leftarrow$ execute $\hat{tx_{A_1}}$, tx_T and tx_{A_2} sequentially;
  if $P > P_{max}$ **then**
    $P_{max} \leftarrow P$;
    $i_l^{optimal} \leftarrow i_l^{actual} + \delta \cdot$ `tickSpacing`;
  $\delta \leftarrow \delta + 1$;

4) Simulating the Attacks of 0xa57...6CF: Our analysis in Section VI-A indicates that bot 0xa57...6CF adopts the following strategy: *(i)* providing the maximum amount of available token balance to add liquidity in each JIT, and *(ii)* setting the lower tick i_l and the upper tick i_u with the constraints of $|i_u - i_l| = 1\times$ `tickSpacing`. Through analyzing the input data of 0xa57...6CF's mint transactions, we find that the adversary only explicitly specifies the distance between the lower tick and the current tick in the transaction data. Because the current tick is obtained by calling the Uniswap V3 pool smart contract, upon receiving the transaction data, the bot contract can execute the JIT liquidity attack with the corresponding parameters chosen by the adversary.

As shown in Algorithm 1, we simulate the JIT liquidity attacks issued by 0xa57...6CF with modified parameter values, in an attempt to test whether the modified attacks yield higher profits. For each JIT, we first decode the input data of the

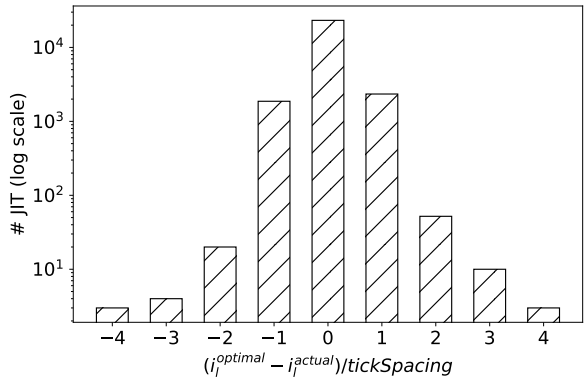


Fig. 7: Distribution of optimal tick ranges after simulating JITs of 0xa57...6CF. $i_l^{optimal}$ is the optimal lower tick obtained via simulation, and i_l^{actual} is the actual lower tick. 0xa57...6CF always ensures $|i_u - i_l| = 1\times$ `tickSpacing`.

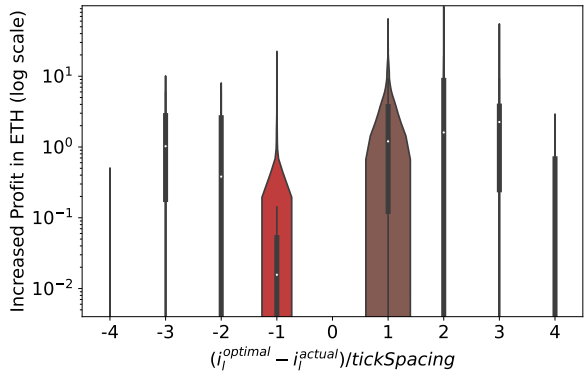


Fig. 8: Distribution of the increased profit after simulating JITs.

mint transaction to extract to actual lower tick i_l^{actual} chosen by the adversary. We then modify the add transaction data by changing the lower tick to $i_l^{actual} + \delta \cdot$ `tickSpacing`, where $-4 \leq \delta \leq 4$. We finally output the optimal parameter $i_l^{optimal}$ which yields the highest attack profit P_{max}.

Insight 8. The most active bot 0xa57...6CF does not always choose the optimal parameters to launch attacks, thus failing to capture at least 7,766 ETH attack profit.

After simulating the JIT liquidity attacks issued by 0xa57...6CF, we observe that 73% of them indeed achieve the maximum attack profit, i.e., $i_l^{actual} = i_l^{optimal}$ (cf. Figure 7). For the remaining 27% JITs, we can choose better lower tick values in the range of $[i_l^{actual} - 4 \cdot$ `tickSpacing`$, i_l^{actual} + 4 \cdot$ `tickSpacing`$]$ to achieve higher attack profits. While the original profit for these non-optimal JITs is 6,900 ETH, we detect that the adversary could have achieved a total profit 14,667 ETH if the adversary had chosen the optimized lower tick values. As a result, 0xa57...6CF fails to capture at least 7,766 ETH (16.1M USD) attack profit (cf. Figure 8).

VII. Discussion

In this study, we provide an overview of JIT liquidity attacks on Uniswap V3. Based on the findings of our empirical measurement, we suggest the following potential future research avenues. First, it is worth investigating the existence of analytic solutions for maximizing adversary utility. While it is difficult to solve Equation 15, the adversarial strategies could be evaluated more effectively if an analytical solution could be found. Second, the protocol should consider how to defend JIT liquidity attacks for existing LPs. A potential solution could be to prevent minting and burning in the same block. However, this may negatively impact the traders who have rebalancing needs (e.g., 0x9ec...832, 0x2b0...dd3, 0x8a6...e2d). The protocol must thus devise an elegant solution while taking the welfare economics of various market participants into account.

VIII. Conclusion

This paper provides an empirical analysis of JIT liquidity attacks, a new type of MEV source introduced by the concentrated liquidity design of Uniswap V3. We find that the attack presents extremely high entry barriers, requiring the adversary to hold a sufficient amount of initial capital that is on average 269 times higher than swap volume. Besides, the JIT game is inaccessible to retail MEV searchers, because few bots manage to dominate the entire game. In addition, we find the JIT game being a low-return investment with an average ROI ratio of only 0.007%. Moreover, we show the attack detrimental to existing LPs, whose liquidity shares are diluted by an average of 85%. Interestingly, we observe JIT liquidity attacks beneficial to liquidity takers, who obtain better execution prices. We dissect the JIT strategies of top MEV bots and evaluate their strategies via local simulation. We find that 27% of the attacks issued by 0xa57...6CF is non-optimal, which fails to capture at least 7,766 ETH of the attack profit. We recommend that the protocol develop appropriate features to protect the existing LPs from JIT liquidity attacks.

References

[1] "Uniswap whitepaper," https://hackmd.io/@HaydenAdams/HJ9jLsfTz.
[2] "Uniswap v2 core," https://docs.uniswap.org/whitepaper.pdf.
[3] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, "Uniswap v3 core," *Tech. rep., Uniswap, Tech. Rep.*, 2021.
[4] L. Heimbach, E. Schertenleib, and R. Wattenhofer, "Exploring price accuracy on uniswap v3 in times of distress," in *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, 2022.
[5] S. Loesch, N. Hindman, M. B. Richardson, and N. Welch, "Impermanent loss in uniswap v3," *arXiv preprint arXiv:2111.09192*, 2021.
[6] L. Heimbach, E. Schertenleib, and R. Wattenhofer, "Risks and returns of uniswap v3 liquidity providers," *arXiv preprint arXiv:2205.08904*, 2022.
[7] S. Hashemseresht and M. Pourpouneh, "Concentrated liquidity analysis in uniswap v3," in *Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, 2022, pp. 63–70.
[8] A. A. Aigner and G. Dhaliwal, "Uniswap: Impermanent loss and risk profile of a liquidity provider," *arXiv preprint arXiv:2106.14404*, 2021.
[9] X. Wan and A. Adams, "Just-in-time liquidity on the uniswap protocol," *Available at SSRN 4382303*, 2022.
[10] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 428–445.

APPENDIX A
DERIVATION OF EQUATION 13

Equation 12 can be written as $\Delta\text{Value}(p_1, x_1, y_1) = [(p_1 \cdot x_1 + y_1) - V_0] \cdot P_{(ETH/Y)}$. Therefore, we only need to focus on $p_1 \cdot x_1 + y_1$. To represents ΔValue as the function of L_A, p_l and p_u, we discuss the following scenarios.

Scenario 1: $\mathbf{p_u < p_1}$. In this scenario, tx_T swaps Y for X (i.e., !ZeroForOne). According to Equation 5, when $p_u < p_1$, $\boldsymbol{A}$'s portfolio after swap consists of asset Y only. Plugging $x_1 = 0$ and $y_1 = L_A(\sqrt{p_u} - \sqrt{p_l})$ into Equation 12, we have:.

$$\Delta\text{Value} = [L_A(\sqrt{p_u} - \sqrt{p_l}) - V_0] \cdot P_{(ETH/Y)} \tag{16}$$

Scenario 2: $\mathbf{p_1 < p_l}$. In this scenario, tx_T swaps X for Y (i.e., ZeroForOne). According to Equation 5, when $p_1 < p_l$, $\boldsymbol{A}$'s portfolio after swap consists of asset X only. Plugging $x_1 = L_A(\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}})$ and $y_1 = 0$ into Equation 12, we have:

$$\Delta\text{Value} = [L_A(\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}})p_1 - V_0] \cdot P_{(ETH/Y)} \tag{17}$$

When swapping X for Y, Equation 18 holds.

$$\Delta x_{\text{in}} = \Delta\frac{1}{\sqrt{p}}L_T = (\frac{1}{\sqrt{p_1}} - \frac{1}{\sqrt{p_0}})L_T \tag{18}$$

Hence, p_1 can be derived by Equation 19:

$$\sqrt{p_1} = \frac{L_T\sqrt{p_0}}{L_T + \sqrt{p_0}\Delta x_{\text{in}}} = \frac{(L_A + L_O)\sqrt{p_0}}{(L_A + L_O) + \sqrt{p_0}\Delta x_{\text{in}}}, \text{ if ZeroForOne} \tag{19}$$

Plugging Equation 19 into Equation 17 we have:

$$\Delta\text{Value} = [L_A(\frac{1}{\sqrt{p_l}} - \frac{1}{\sqrt{p_u}})\frac{(L_A + L_O)\sqrt{p_0}}{(L_A + L_O) + \sqrt{p_0}\Delta x_{\text{in}}} - V_0] \cdot P_{(ETH/Y)} \tag{20}$$

Scenario 3: $\mathbf{p_l \le p_1 \le p_u}$. In this scenario, $\boldsymbol{A}$'s portfolio after swap consists of both asset X and Y. Plugging $x_1 = L_A(\frac{1}{\sqrt{p_1}} - \frac{1}{\sqrt{p_u}})$ and $y_1 = L_A(\sqrt{p_1} - \sqrt{p_l})$ into Equation 12:

$$\Delta\text{Value} = [L_A \cdot (\sqrt{p_u} - \sqrt{p_l} - \frac{1}{\sqrt{p_u}}(\sqrt{p_1} - \sqrt{p_u})^2) - V_0] \cdot P_{(ETH/Y)} \tag{21}$$

Since the swap direction (i.e., the value of ZeroForOne) is unknown, we should further discuss Scenario 3(a) and 3(b).

Scenario 3(a): $\mathbf{p_l \le p_1 \le p_u}$ and ZeroForOne. When swapping X for Y, we can derive p_1 using Equation 19.

$$\Delta\text{Value} = [L_A(\sqrt{p_u} - \sqrt{p_l} - \frac{1}{\sqrt{p_u}}(\frac{(L_A + L_O)\sqrt{p_0}}{(L_A + L_O) + \sqrt{p_0}\Delta x_{\text{in}}} - \sqrt{p_u})^2) - V_0] \cdot P_{(ETH/Y)} \tag{22}$$

Scenario 3(b): $\mathbf{p_l \le p_1 \le p_u}$ and !ZeroForOne. When swapping Y for X, Equation 23 holds.

$$\Delta y_{\text{in}} = \Delta\sqrt{p}L_T = (\sqrt{p_1} - \sqrt{p_0})L_T \tag{23}$$

Hence, p_1 can be derived by Equation 24:

$$\sqrt{p_1} = \sqrt{p_0} + \frac{\Delta y_{\text{in}}}{L_T} = \sqrt{p_0} + \frac{\Delta y_{\text{in}}}{L_A + L_O}, \text{ if !ZeroForOne} \tag{24}$$

Plugging Equation 24 into Equation 21 we have:

$$\Delta\text{Value} = [L_A(\sqrt{p_u} - \sqrt{p_l} - \frac{1}{\sqrt{p_u}}((\sqrt{p_0} + \frac{\Delta y_{\text{in}}}{L_A + L_O}) - \sqrt{p_u})^2) - V_0] \cdot P_{(ETH/Y)} \tag{25}$$

APPENDIX B
SUPPLYMENTARY INFORMATION

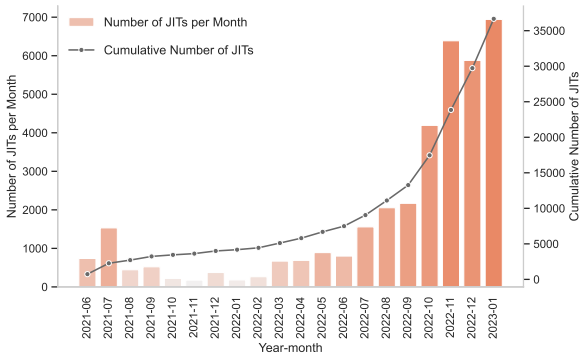


Fig. 9: Number of detected JIT liquidity attacks on Ethereum.

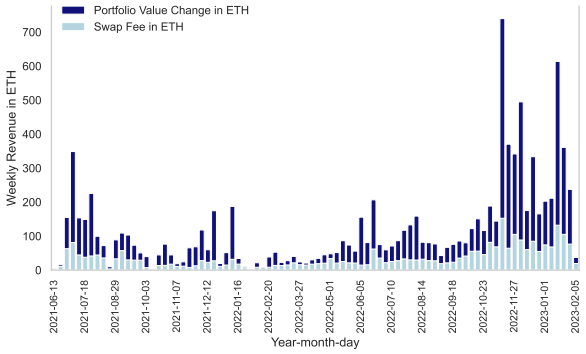


Fig. 10: Weekly portfolio change and swap fee revenue.

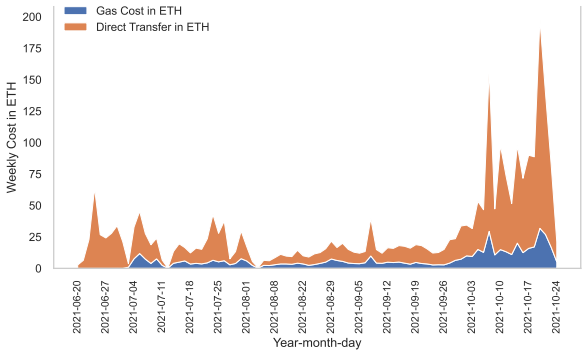
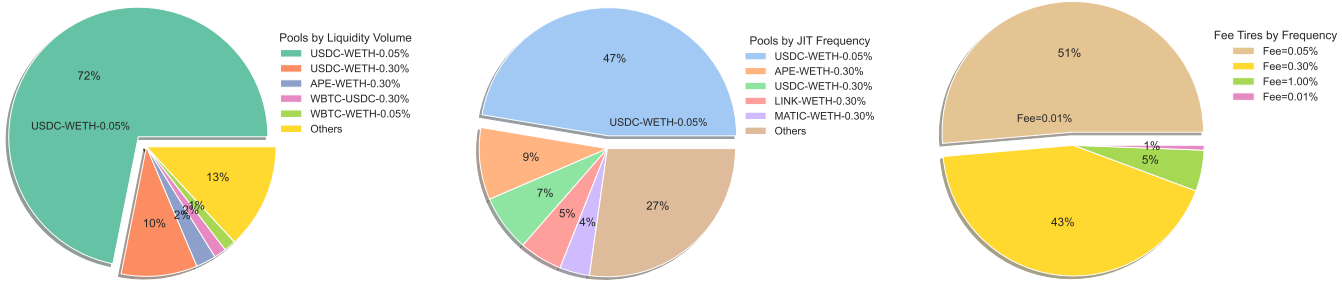


Fig. 11: Gas cost and direct transfer to miners/validators.



(a) Pools by liquidity volume.

(b) Pools by attack frequency.

(c) Fee tiers by attack frequency.

(d) Pools by swap volume.

(e) Pools by attack revenue.

(f) Fee tiers by attack revenue.

Fig. 12: Statistics of Uniswap V3 pools involve in JIT liquidity attacks.

Categories			Number of JIT liquidity attacks
$i_u - i_l = 1 \times \texttt{tickSpacing}$	$i_0 < i_1$	$i_1 < i_l$	2
		$i_0 < i_l, i_l \le i_1 \le i_u$	2,727
		$i_l \le i_0 \le i_u, i_l \le i_1 \le i_u$	5,441
		$i_l \le i_0 \le i_u, i_1 > i_u$	83
	$i_0 = i_1$	$i_l \le i_0 = i_1 \le i_u$	18,766
	$i_0 > i_1$	$i_1 < i_l, i_l \le i_0 \le i_u$	27
		$i_l \le i_1 \le i_u, i_l \le i_0 \le i_u$	5,980
		$i_l \le i_1 \le i_u, i_0 > i_u$	3,614
		$i_1 > i_u$	6
$i_u - i_l > 1 \times \texttt{tickSpacing}$			5

TABLE II: Distribution of JIT adversaries' choices of tick parameters i_l and i_u.

$\lvert i_1 - i_0 \rvert / \texttt{tickSpacing}$	≤1	(1, 2]	(2, 3]	(3, 4]	>4
Number of Attacks	35878	1210	218	78	77
Percentage of Attacks	97.84%	1.65%	0.30%	0.11%	0.11%

TABLE III: Summarization of the distance between p_0 and p_1.

	Quantiles (%)				Statistics (%)			
Fee	10%	25%	50%	75%	min	max	mean	std
1%	0.135	0.232	0.466	0.911	0.014	12.611	0.699	0.737
0.3%	0.022	0.046	0.110	0.252	0.119	22.752	0.194	0.341
0.05%	0.008	0.014	0.029	0.052	0.000	0.734	0.040	0.039
0.01%	0.000	0.000	0.000	0.001	0.000	0.662	0.009	0.066

TABLE IV: Price improvement statistics for different fee tiers.

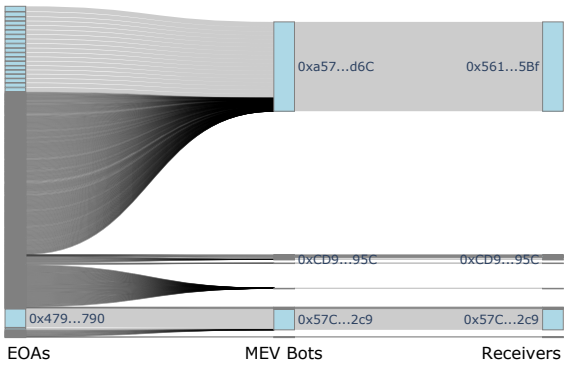


Fig. 13: The coin flow of JIT MEV bots. The attack revenue of 0xa57...6CF always flows to 0x56...Bf9.