Liquidity Management Attacks on Lending Markets

Alireza Arjmand
University of Alberta, Canada

Majid Khabbazian
University of Alberta, Canada

Abstract

Decentralized Finance (DeFi) continues to open up promising opportunities for a broad spectrum of users, with lending pools emerging as a cornerstone of its applications. While prominent platforms like Compound and Aave maintain a large share of the funds in lending pools, numerous other smaller pools also exist. Many of these smaller entities draw heavily from the design principles of their larger counterparts due to the complex nature of lending pool design.

This paper asserts that the design approaches that serve larger pools effectively may not necessarily be the most beneficial for smaller lending pools. We identify and elaborate on two liquidity management attacks, which can allow well-funded attackers to exploit specific circumstances within lending pools for personal gain. Although large lending pools, due to their vast and diverse liquidity and high user engagement, are generally less vulnerable to these attacks, smaller lending protocols may need to employ specialized defensive strategies, particularly during periods of low liquidity. We also show that beyond the six leading lending protocols, there exists a market value exceeding \$1.75 billion. This considerable sum is dispersed among over 200 liquidity pools, posing a potentially attractive target for bad actors.

Furthermore, we evaluate existing designs of lending pools and suggest a novel architecture that distinctly separates the liquidity and logic layers. This unique setup gives smaller pools the adaptability they need to link with larger, well-established pools. Despite encountering certain constraints, these emerging pools can leverage the considerable liquidity from larger pools until they generate sufficient funds to form their own standalone liquidity pools. This design cultivates a setting where multiple lending pools can integrate their liquidity components, thus encouraging a more diverse and robust liquidity environment.

2012 ACM Subject Classification Security and privacy → Distributed systems security

Keywords and phrases Lending Pools, DeFi, Interest Rate, Liquidity Management Attack

Digital Object Identifier 10.4230/LIPIcs.AFT.2023.23

1 Introduction

Decentralized Finance (DeFi) protocols offer a solid foundation for financial investors seeking to earn returns on their assets in a decentralized manner. Lending and borrowing, one of the oldest financial applications, is transformed by blockchains, enabling the creation of liquidity pools that consolidate lender funds and facilitate borrowing. This arrangement presents an appealing opportunity for both parties; lenders earn interest from the moment they contribute their funds to the liquidity pools, while borrowers are assured of paying a fair interest rate for their borrowed amount.

This paper primarily focuses on over-collateralized lending pools [4], where, after liquidity providers contribute their funds to the pool, borrowers can access these funds by offering collateral in other assets. The collateral amount must exceed the borrowed sum to allow the lending protocol to guarantee a return of funds to the liquidity providers. Should the collateral amount drop below a certain threshold, the collateral can be converted into the borrowed asset, incentivizing third parties to repay the liquidity providers in a process known as liquidation [24].

Advances in Financial Technologies - AFT 2023.
Editors: Alireza Arjmand and Majid Khabbazian; Article No. 23; pp. 23:1–23:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Despite experiencing a decline in 2022 [26], the lending markets continue to expand, amassing a Total Value Locked (TVL) in excess of $13.2b across a multitude of blockchains [10]. While dominant lending markets such as Compound [7, 18] and Aave [1, 2] maintain the bulk of this value, new lending protocols inspired by these major lending pools are constantly emerging, contributing novel capabilities to the application layer for users. To gain traction, these newer lending protocols need to incentivize users to entrust their funds to their platforms. This often requires competition with larger lending pools through attractive incentives such as higher interest rates and novel application layer opportunities.

A key part of any lending pool is its interest rate formula, which determines how much borrowers have to pay back based on what they borrow. The importance of this formula lies in its potential to encourage certain behaviors: (i) It should incentivize borrowing by decreasing interest rates when ample liquidity is available; (ii) It should attract external liquidity providers to participate in the protocol by elevating interest rates when a significant portion of the liquidity is borrowed; (iii) It should stimulate the retention of some liquidity in the pool, enabling providers to withdraw at any time. To encourage these behaviors, several recognized formulas/models are frequently used by lending protocols [15].

In the widely adopted model, lending pools implement high interest rates on borrowed funds when usage approaches 100%. Consequently, if this level of usage persists for an extended duration, borrowers will be subject to significantly increased fees compared to the norm. To address this issue, these lending protocols depend on diligent users who actively monitor the situation. These users are incentivized to inject funds into the pool when interest rates are high. However, if these users lack sufficient funds to effectively reduce the usage or if there is a delay in their actions, borrowers in the lending pools may suffer substantial losses due to the elevated interest rates.

In this paper, we focus on small lending pools that adopt similar models. We postulate that a malicious liquidity provider, owning a significant share of a liquidity pool's reserves, can manipulate other actors to align with certain conditions for their benefit, potentially causing harm to others. We demonstrate that the relative lack of substantial liquidity funds and centralized liquidity providers in these smaller pools can expose them to various threats. In particular, we make the contributions:

- **Liquidity Management Attacks:** To highlight the vulnerability of small lending pools, we present two different liquidity management attacks on these pools. Furthermore, we delve into a general strategy that could be implemented by an attacker with sufficient funds, highlighting the incentives for users and a long-term approach that could prove profitable for the attacker but detrimental to the ecosystem. We also evaluate potential mitigation as well as risks involved in launching the proposed attacks.
- **Liquidity Aggregator:** We present a model in which lending pools separate their liquidity layer from their logic layer. By this means, smaller lending pools can integrate their applications with larger lending pools, thereby enhancing their liquidity safeguards. In this model, lending pools can coexist in dependent or standalone modes, allowing the community to avoid scattering liquidity across numerous platforms.
- **Lending Protocol Data Extraction:** We gathered data from the six biggest lending pools. Even though they hold most of the TVL, it's important to note that there is still a considerable amount of value in the remaining lending pools. This could potentially make them targets for malicious users.

The rest of this paper is organized as follows, Section 2 provides necessary background information. Section 3 introduces the mathematical model that forms the basis for our discussions throughout the paper. Section 4 outlines the logic behind two types of liquidity management attacks we investigate and illustrates how malicious actors can manipulate economic principles to meet their goals. Section 5 presents a design proposal to bolster the security of emerging lending pools, especially those with limited overall liquidity. In Section 6, we analyze the total value locked in on-chain lending pools, focusing on the six largest protocols from various perspectives. Section 7 surveys related work in this field. Finally, in Section 8, we wrap up our discussions and suggest potential avenues for future research.

2 Background

In this section, we present the fundamental concepts necessary to comprehend the subsequent content of the paper.

2.1 Blockchains

Blockchains comprise numerous underlying nodes that disseminate transactions throughout the system using a Peer-to-Peer (P2P) network [20, 6]. Each transaction typically aims to uniquely alter the global state. Transactions are appended to the blockchain within blocks in each round, following a consensus algorithm that determines the transactions' inclusion and sequence.

2.2 Decentralized Finance (DeFi)

Ethereum [32] employs a Turing-complete language named Solidity, enabling users to deploy *smart contracts*. These contracts broaden user capabilities by facilitating the creation of decentralized applications, giving rise to DeFi applications [31]. At present, Ethereum employs the Proof of Stake (PoS) consensus algorithm, which designates a block builder each round to select the transactions' order, which is then subjected to voting by other block builders. Once a block is produced in each round, all users can sequentially execute each transaction within the Ethereum Virtual Environment (EVM) to ascertain the current global state. One distinctive feature of the EVM is that its operations are deterministic and atomic, altering the state only upon success. Therefore, given any pre-state and specific inputs, each node would produce identical outputs. These attributes, coupled with Ethereum's high throughput, have led to novel, transparent DeFi applications not traditionally found in Centralized Finance (CeFi) [23]. Furthermore, Ethereum's allowance for smart contract composability has resulted in the establishment of complex ecosystems.

DeFi has continued to thrive over the past year, attracting numerous users and boasting more than $41.5b in TVL. The absence of third parties and the transparency offered by DeFi applications make them an attractive prospect for many. Popular applications of DeFi include lending pools [4], Decentralized Exchanges [33], Yield aggregators [8], and stablecoins [19].

2.3 Attacks on DeFi

While code transparency is beneficial, it can also simplify the task of spotting faulty code. If such vulnerabilities are detected by attackers, they could lead to massive security breaches. In some of the most significant hacks, such as [22, 5], attackers exploited application layer bugs to siphon user funds. The classification of attack strategies has been thoroughly documented in the literature [35, 3, 14, 11], which is essential in assisting the community in identifying and avoiding patterns that could lead to undesirable consequences. Concurrently, there exist open-source libraries [21] that strive to provide secure building blocks for contracts. This enables protocol developers to ensure the safety of their code's foundational elements.

2.4 High frequency trading

Decentralized markets have given rise to on-chain high-frequency trading [9, 34]. This environment, while presenting many opportunities, also attracts malicious users aiming to seize on-chain opportunities by tampering with transaction ordering. Tactics such as front-running and sandwich attacks are used to drain funds or steal opportunities away from unsuspecting users. To mitigate this, private relayers such as Flashbots [12] have emerged. These entities promise users certain assurances about their transaction inclusion, thereby safeguarding them from generalized front-runners.

3 System model

In this section, we aim to formalize the actions of users who can impact a lending protocol. To simplify the analysis, we focus on a specific subset of actions in lending pools and disregard other activities such as liquidations and absorptions. We assume the presence of numerous users in the system. A user u in our system model is a tuple $u = (S, B, C)$, where S is the amount of fund the user has supplied to the protocol, B is the amount of funds borrowed by the user, and C is the total collateral the user provided to the protocol. For simplicity, in our model, we convert the values of S, B, and C to a common base value (e.g. USD).

The balance of a user $u_i = (S_i, B_i, C_i)$ is defined as $S_i - B_i$. If a user's balance is greater than zero, the user is considered a *liquidity provider*; otherwise, if its balance is less than zero, the user is identified as a *borrower*. A borrower must have adequate collateral in the system for the borrowed balance. Since liquidations are not factored into our model, the following condition should be true for each user u_i:

$$S_i + EC_i > B_i,$$

where EC_i is the effective collateral for each user, that is

$$EC_i = \Sigma_j c_{ij} \times f_j \times rate_{USD/j}$$

Here, f_j represents the collateral factor for each asset. We denote the total amount of each variable in the entire protocol using the "total" subscript, such as S_{total}.

In our system, the borrowers in the system are subject to an interest R calculated using the *kinked interest rate model* as follows:

$$R = \begin{cases} R_0 + R_{low} \times U & if\ U \leq kink \\ R_0 + R_{low} + R_{high} \times (U - kink) & if\ U > kink \end{cases} \tag{1}$$

Table 1 Terminology used in system model.

Character	Meaning
L	Supplied liquidity
B	Borrowed amount
R	Interest rate
U	Utilization
α	Attacker liquidity percentage
EC	Effective collateral
$kink$	Optimal utilization

In this formulation, U denotes the protocol's utilization, calculated as $\frac{B_{total}}{S_{total}}$, where $kink$ represents the optimal utilization rate, often referred to as the 'kink rate'. The terms R_0, R_{low}, and R_{high} signify the base interest rate, the lower slope for utilization, and the sharp increase in interest rates when utilization surpasses the kink rate, respectively. Borrowers are assumed to accrue interest with each passing block, adhering to this interest rate model:

$$Fee_i = R_U \times B_i \times t \tag{2}$$

We also assume that the protocol reserve doesn't accumulate any yields and all borrower fees are shared among the liquidity providers. To model the reserve, we can consider the reserve amount as one of the liquidity providers.

Collusion model: In the context of lending protocols, it is conceivable that a group of users may collude to achieve a common objective. Thus, we consider an adversary A who can compromise multiple accounts with cumulative supply of up to fraction α, such as:

$$\alpha \geq \frac{\Sigma_e S_e}{S_{total}} \tag{3}$$

Where α is the maximum fraction of overall funds that an attacker can control.

4 Attacks on lending markets

In this section, we examine the overarching structure of lending pools and present two forms of attacks that enable an adversary to impose specific conditions on the liquidity pool by employing economic strategies to secure a desired outcome. These outcomes could be:

- **More income:** An attacker can augment the fees extracted from other participants within the pool over a specific time frame.
- **Denial of Service:** An attacker can obstruct access to the rest of the participants, effectively preventing them from either borrowing or withdrawing their liquidity from the pool.

While these attacks pose potential complications for other users, they necessitate a substantial amount of liquidity from the attacker to fulfill the preconditions of launching the attack. Consequently, the attacker's risk level escalates in correlation with the growth of this prerequisite amount. The Compound and Aave protocol models are currently the most influential among the lending pools, widely implemented by smaller lending pools and occasionally forked from the main projects. Given the vast liquidity diversity and substantial user base of the top protocols with the highest TVL, an adversary would face a formidable task executing these attacks. However, the situation is different for smaller pools. Here, an attacker could instigate these attacks with a lower risk and initial capital, thereby realizing a profit. Thus, we demonstrate that smaller pools cannot merely replicate the strategies of larger entities. They must devise additional defence mechanisms against such attacks while their liquidity pool is relatively small, thereby safeguarding their liquidity providers and borrowers.

In the remainder of this section, we commence by elucidating the potential attacks and demonstrating how an attacker with sufficient liquidity can enforce other actors to comply with specific conditions. We then proceed with an analysis of the attacker's risk before deliberating on some design decisions that new lending pools should avoid.

4.1 Utilization kink attack

While borrowers secure funds by depositing an overcollateralized quantity of tokens in the protocol, they pay ongoing fees determined by the length of their loan. These fees fluctuate based on the degree of liquidity utilization, with adjustments made following each transaction processed by the protocol. Generally, it is anticipated that the borrowing rate maintains proportionality with the borrowed amount and the R_{low} delineated in the interest rate formula. However, when the utilization quantity exceeds a predetermined threshold or "kink", all borrowers become liable to pay supplemental fees to the liquidity providers. The objective of this kink value is to motivate all participants to act, thereby releasing liquidity within the protocol: (1) as a liquidity provider, the increased fees offer an incentive to contribute more liquidity from out of the protocol, and (2) as a borrower, the prospect of evading excessive fees incentivizes the repayment of the borrowed amount. Both actions lead to a decrease in total utilization and consequently a reduction in fees. By comparing the fees at maximum lending protocol utilization and at the kink value, we notice that in some protocols the fees can unexpectedly jump to more than ten times. This indicates that if an attacker were to elevate these values by either borrowing the rest of the remaining liquidity, or pulling out his own liquidity out of the protocol, they could compel borrowers to bear extensive fees. In such scenarios, smaller pools face two significant threats compared to their larger counterparts:

- **Lesser liquidity required:** Attackers need a smaller volume of liquidity to drive up fees, consequently exposing themselves to lower risks.
- **Smaller group of active users:** In such circumstances, the lending pool requires either active external liquidity providers or borrowers to regulate utilization. A smaller lending pool implies a lower number of participants monitoring such activities in the system, hence increasing the likelihood of such attacks.

4.1.1 Simplified attack

In order to exemplify this attack, we explore a hypothetical scenario involving a single liquidity provider, Alice, and a borrower, Bob. This analysis demonstrates how Alice can increase the

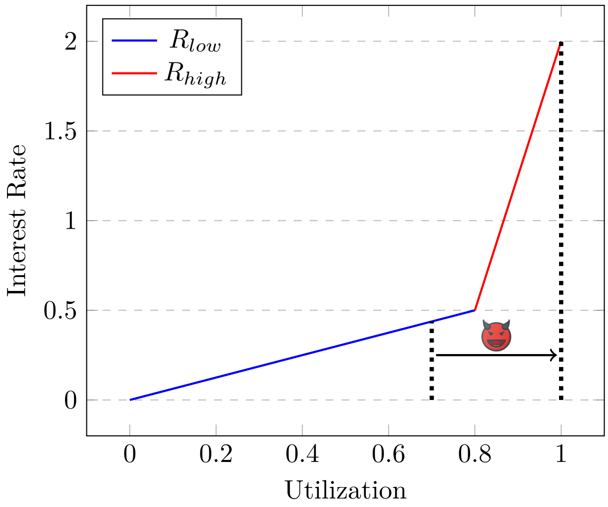

Figure 1 The kinked rate model can be exploited by an attacker through either increasing the utilization of the protocol by borrowing more or withdrawing funds.

utilization potentially to secure additional fees from Bob. Subsequently, real-world protocol figures are utilized to replace the formulas and estimate the possible damage an attacker can cause borrowers to pay.

Scenario Setup: Consider a lending platform characterized by parameters R_{low}, R_{high}, and $kink$, which are used to compute the interest rate. Initially, Alice contributes S initial funds to the protocol. Subsequently, Bob borrows an amount B, setting the protocol's utilization at the $kink$ amount by offering C in collateral value with collateral factor f.

Attack Execution: Alice currently receives fees from Bob proportionate to $kink * R_{low}$. Nonetheless, Alice can elevate the utilization by opting for one of the following strategies to increase the protocol's utilization:

- She may withdraw $(1 - kink) * S$ liquidity from the protocol.
- She might borrow the remaining amount of $(1 - kink) * S$ and pay those fees to herself, since she is the sole liquidity provider. In this case, Alice needs more funds comparing to the previous method to borrow and execute the attack.

Any of these actions would surge the protocol utilization to 100%, thereby significantly escalating Bob's fee. We can calculate the Bob's new fee, which is proportionate to $kink * R_{low} + (1 - kink) * R_{high}$. We can see that Bob needs to pay $1 + \frac{(1-kink)*R_{high}}{kink*R_{low}}$ times more fees.

Aftermath: Although Bob retains the option to stop this attack at any point by repaying his borrowed positions, he remains accountable for fees corresponding to the duration he borrowed the funds from the protocol. Nevertheless, Bob's response may be hindered for various reasons:

- He may not have enough liquidity to repay the borrowed sum, especially if these funds have been invested and locked elsewhere.
- He may be offline or negligent in monitoring the protocol's fees.

Furthermore, many protocols accumulate fees for borrowers in a manner that escalates their borrowing position over time. This means that by exploiting these circumstances, Alice not only forces Bob to endure higher fees but could also cause the liquidation of his position if the accumulated fees surpass Bob's initial estimations. Bob's position can even get liquidated if the following formula becomes true:

$$EC_{Bob} < B + fee \tag{4}$$

While Bob may have provided ample collateral to cover the protocol's standard fees, Alice could potentially elevate Bob's fees, leading to the liquidation of his position and opening up another potential profit source.

Numerical example: As a straightforward example, consider a lending pool emulating the interest rate parameters of Compound V2's cETH contract. As of this writing, this contract has an R_{high}/R_{low} ratio of 217.78 and a kink value of 0.8. Consequently, for utilization rates exceeding 80 percent, we observe a significant increase in the fees taken from borrowers. Yet, Compound V2 is a well-known contract, frequently monitored by numerous users. In contrast, for newly generated contracts which are copying these values, the utilization kink attack can present a genuine threat. An attacker could amplify fees by escalating utilization from 80 to 100 percent, by $((1 - 0.8)/0.8) \times 217.78 = 54.445$ times. Thus, if Alice successfully executes this attack against Bob for merely a single day, the profits generated would approximate those accrued from nearly two months of honest investment.

4.1.2 Utilization kink attack in general setting

While the prior example was a basic version of the attack with just two actors in the system, it served to illustrate that such attacks are indeed possible. However, in real-world situations, the number of actors, including both honest users and adversaries, is typically greater than one. In this section, we aim to shape a scenario involving multiple actors, where adversaries might work together to conduct the explained attack on a specific lending pool.

Collusion among liquidity providers: In order to examine the attack in a broader context, we need to account for realistic interactions among actors. In this section, we concentrate on a specific scenario where attackers could potentially enhance the utilization rate by withdrawing their available liquidity. To simplify this without compromising the mathematical validity of our analysis, we assume that a fraction, represented as α, of all liquidity provided to the pool is controlled by colluding adversaries. In this system, where $1 - \alpha$ represents honest participants, the adversaries decrease their shares by withdrawing their funds. Interestingly, under certain conditions met by the interest rate formula, attackers could increase their fees even after reducing their shares. One approach for adversaries to collude atomically, would be through a smart contract. The progression of steps is outlined below:

1. Any adversary could deploy an attack smart contract, equipped with three key functionalities: (1) obtaining permission from users to manage their liquidity tokens, (2) withdrawing funds from each adversary's account to increase the utilization while reducing their respective shares, and (3) returning funds to the liquidity pool if the liquidity kink attack ceases to be profitable.
2. Each adversary could then grant a certain amount of liquidity provider tokens to the deployed contract using the pool's functions, permitting the contract to manage liquidity on behalf of each adversary.
3. Once all permissions are received, a specific threshold of signatures from adversaries could initiate the event of pulling liquidity from the protocol to boost utilization.
4. At this point, adversaries can monitor on-chain events to assess the profitability of the lending pool.
5. Should a new honest liquidity provider join the lending pool, or borrowers repay their borrowed amounts to an extent that it no longer remains profitable for attackers to withhold their funds, they can refund all the liquidity and revert to the initial state.

This strategy enables adversaries to minimize liquidity management risks and, in the worst-case scenario, return to the starting state. By providing adequate permissions, adversaries can utilize the attack contract to impose higher fees when feasible.

Scenario Setup: In this particular situation, we presume that attackers are already in possession of α percent of the total liquidity pool, denoted as L. The borrowed amount is represented by B. The kinked model, which we discussed earlier, guides the calculation of the interest rate. Moreover, we operate under the assumption that the attackers have already initiated the attack contract and have authorized it to either deposit or withdraw funds as necessary. We assume that prior to the attack, the utilization U is less than the kink value. We also assume that attackers possess sufficient liquidity to elevate the protocol's utilization above the kink value. If they lack this amount, the attack would be ineffective and they would merely diminish their own shares. Finally, we operate under the assumption that all fees derived from borrowers are directed to the liquidity providers, with none retained by the protocol itself. This simplifying assumption aids in streamlining the model, though in real-world applications, a portion of the fees is typically allocated to a community wallet managed by a DAO or an admin. Should the attackers choose to retain all their funds within the liquidity pool, behaving honestly, the fees they would receive would equate to the following amount:

$$fee_{honest} \propto (R_0 + \frac{B}{L} * R_{low}) * \alpha \tag{5}$$

Attack Execution: For attackers to boost the utilization, they initially need to calculate the exact amount of funds, termed as x, to withdraw from the protocol to yield higher fees. We assume that when attackers extract this x amount from the protocol's reserves, it drives the utilization beyond the kink value. As a consequence, the fees that would then accrue to the attackers can be computed as follows:

$$fee_{attack} \propto (R_0 + R_{low} \times kink + ((\frac{B}{L-x}) - kink) \times R_{high})(\alpha - \frac{x}{L}) \tag{6}$$

In the preceding equation, the attackers' shares drop from α to $\alpha - x/L$. Simultaneously, the total amount of funds in the protocol diminishes by x, though the borrowed amount remains unchanged.

Our objective is to pinpoint the ideal amount that adversaries should extract from the protocol to maximize fee_{attack}. We attain this by identifying the global maximum obtained from the function's derivative. The solution to this is realized when the condition $dfee_{attack}/dx = 0$ is fulfilled, the optimal amount can be determined by solving the following equation:

$$\frac{B \times R_{high} \times (a - \frac{x}{L})}{(L-x)^2} = \frac{R_{high} \times (\frac{B}{L-x} - U) + R_{low} \times U + R_0}{L} \tag{7}$$

This, naturally, would be the ideal value according to the condition if it lies within the range $x < L - B$, and $x > kink * L - B$.

Risks: Even though attackers stand to profit while the utilization remains high, they are simultaneously accepting certain risks. We explore these primary risks in this section.

- **Borrower Attrition:** By initiating the utilization kink attack, attackers risk compromising their long-term income. Specifically, they may incentivize borrowers to withdraw their money, potentially redirecting it to other protocols. Consequently, a lending pool subject to such attacks may fail to instill trust in new borrowers. Nonetheless, an attacker could easily shift their funds to other protocols, given there are multiple that offer such services.
- **Monitoring Challenges:** The preceding section demonstrated that certain conditions need to be met for a profitable scenario. Given these conditions may change as new actors join and leave the system, attackers can respond quickly when the situation ceases to be profitable. Failure to do so could result in a loss of potential fees that could have been earned through honest investing.
- **Security Considerations:** Participating in a protocol implies that users, both honest and dishonest, trust the protocol to be secure. However, there's always a risk that a protocol may contain a bug leading to a loss of all funds. When an attacker moves between protocols to execute liquidity management attacks, they are inherently trusting these protocols not to be compromised. If a breach does occur, they might lose all their funds.

Mitigation recommendations: The potential threat of liquidity kink attacks can be partially mitigated at the protocol's design phase, offering some level of protection to borrowers. One potential remedy involves demanding a commitment of liquidity from providers. The majority of honest liquidity providers aim to keep their resources in the market for an extended duration. In defense of borrowers, the protocol could stipulate a minimum time commitment from these providers, thereby inhibiting attackers from removing their funds and artificially increasing the protocol's utilization. An alternative could be the establishment of "fee tiers", whereby the protocol rewards providers who have pledged their resources over a longer time frame with higher fees. However, this strategy only stops attackers from withdrawing their funds, while the possibility of borrowing the remaining amount to amplify utilization still exists.

4.2 DoS attack on liquidators

When liquidity providers contribute funds to a protocol, it is generally assumed that sufficient funds will be available for regular withdrawals when needed. The portion of funds supplied to the protocol but not borrowed is typically eligible for withdrawal. However, it is crucial to acknowledge that this mechanism does not guarantee withdrawals, as it is incentivized by imposing fees on borrowers when the total protocol utilization exceeds the specified threshold (kink). Additionally, the fee mechanism is often time-based, considering the duration between borrow and repayment transactions to calculate the final fee. Consequently, if liquidity is borrowed and repaid within the same block, the borrower only needs to cover the gas fee and is not subject to additional fees from the protocol.

An adversary could exploit (1) the absence of guaranteed withdrawals and (2) borrow fees based on time, to launch a DoS attack. This attack could impact liquidity providers who are trying to withdraw their funds from many lending protocols, as well as borrowers attempting to secure a loan after providing sufficient collateral.

4.2.1 Simplified Attack

Here, we discuss a simple attack scenario, Suppose Alice is a liquidity provider in a lending protocol, supplying \$300,000 out of a \$1 million pool. The utilization level is currently at 70%, meaning \$300,000 of the pool remains available for both borrowers and liquidity providers to utilize. Alice urgently needs to withdraw the entire \$300,000 from the protocol. Bob, observing this, aims to prevent Alice's withdrawal opportunity. He already has sufficient collateral provided to the protocol and initiates two transactions: (1) a transaction with a higher gas fee than Alice's to front-run her transaction and borrow the entire \$300,000, resulting in 100% utilization, and (2) a transaction with a lower gas fee than Alice's to back-run her transaction and push the borrowed amount back into the protocol. By sandwiching Alice in this manner, Bob effectively denies her the withdrawal by causing her transaction to fail since there are no available free funds in the pool.

It is worth noting that in the above example, any other withdrawal requests from third parties would also fail since Bob has drained the protocol of funds. Furthermore, during this process, Bob would only pay the gas fees for the two transactions, which is a relatively small amount compared to the disruptive impact inflicted upon Alice within the system.

In addition to targeting specific users, an attacker can also attempt a generalized DoS attack against the entire network. In this scenario, the attacker aims to include one transaction at the beginning of a block and another transaction at the end of the same block. If successful, this strategy can effectively prevent anyone within the system from withdrawing funds from the protocol.

4.2.2 DoS attacks in general setting

In order for an adversary to launch DoS attacks on real-world systems, they require access to an amount of funds denoted as x. They can cause any withdrawal to fail if its size surpasses this threshold:

$$Withdrawal > L - B - x \tag{8}$$

Assuming that liquidity pools typically maintain utilization up to their optimal utilization, an attacker could disrupt any withdrawal provided they have access to $L * (1 - kink)$ funds. If the attacker's funds are already in the protocol as liquidity, they could withdraw their funds. Alternatively, if their funds are outside of the protocol, they could borrow the necessary amount temporarily for just one block. Given they can perform both these actions within a single block, they neither forfeit any income nor incur any fees. This is because the duration of the liquidity withdrawal or borrowing within the same block is effectively zero.

Risks: To execute a Denial of Service attack on users submitting transactions to a public mempool, an attacker can attempt to accomplish this objective by sending one transaction with a higher gas price and another transaction with a lower gas price. However, there is a risk involved as these transactions may not be included in the desired block. To mitigate this risk, an attacker can minimize the issue by bribing block builders within the blockchain network, requesting them to include all the target transactions in their subsequent block. By doing so, the attacker's risk exposure would be reduced. Alternatively, the attacker can opt to send transactions to a private relayer, such as flashbots, which ensures the "next-block-or-never" attribute. This approach allows the attacker to bundle the user's transactions into a meticulously constructed bundle and transmit it to the private relayer. In cases where an attacker is unable to successfully execute sandwich attacks on their target, their transactions remain valid and can be processed on the network. Hence, they might incur borrowing fees over several blocks, which could be a considerable amount given that the utilization is boosted to 100 percent, and the borrowed sum is substantial.

Mitigation recommendations: To effectively mitigate such attacks, implementing protocol-level measures is crucial. It is important to acknowledge that the DoS attack described does not incur a protocol-level fee, making it relatively inexpensive for an attacker to execute. One effective mitigation strategy is to introduce a percentage-based fee within the borrowing process. This means that when a user borrows a certain amount, they would be required to pay a fee calculated as follows:

$$Fee_i = R_U \times B_i \times t + B_i \times proportionalFee \tag{9}$$

By implementing this approach, the cost for an attacker to execute a DoS attack would increase proportionally with the size of the borrowed amount. As the attacker needs to deplete the remaining funds in the pool, the associated cost becomes significant, acting as a deterrent for such attacks. Furthermore, users can proactively protect themselves against these attacks by opting to send their transactions through a private relayer. This approach helps safeguard users from becoming targets of DoS attacks orchestrated by the attacker. However, it is important to note that these solutions may not be effective against the generalized DoS attacks previously discussed.

4.3 Economical games by adversary

In the present analysis, an attempt is made to envision the potential tactics of an adversary within the domain of lending pools to gain profits over an extended period. There are several incentives that may prompt adversaries to initiate such maneuvers, which are discussed in the ensuing sections:

- **Profit Realization:** The most straightforward objective for an adversary could be to accumulate profits. In the event an adversary consistently executes a kink utilization attack, they could potentially accrue multiple rounds of rewards. However, repeated instances of such attacks may compel borrowers to discontinue using the protocol.
- **Control over Access:** By leveraging a DoS attack, adversaries could exercise control over the liquidity providers' access to their funds. In theory, adversaries may be able to immobilize users' funds. However, in practice, it is more possible to cause delays in withdrawals from the protocol resulting in weak censorship [29]. Such delays can prove critical, particularly during periods of financial instability [30].
- **Attrition of Protocol Users:** A possible adversary objective could be to deter users from engaging with a specific protocol. If the adversary's liquidity is sizable in comparison to the entire pool, by performing such attacks, they could result in actors blacklisting the protocol. This is feasible through two mechanisms, for liquidity providers, they may join the protocol when they observe a spike in utilization but as the attacker re-infuses funds, utilization and consequently fees drop. Borrowers, on the other hand, may be subjected to substantially higher fees frequently, making the protocol a less attractive option.

An attacker can meticulously plan and execute such attacks over an extended duration following several steps:

1. Firstly, the attacker must amass significant funds, either through their own capital or via colluding with other adversaries.
2. Subsequently, they must identify vulnerable protocols with a small liquidity pool, relative to their initial funds.
3. Initial investment in the protocol may be conventional, followed by an inflow of investment which reduces the overall fees paid by borrowers. This leads to a situation where other liquidity providers exit the protocol in pursuit of higher returns elsewhere, or more borrowers enter the pool. The attacker must wait until their share is significantly higher than the remaining liquidity to borrow in the protocol, a stage that may occur over an extended period, such as a week. During this time, adversaries earn interest at a standard rate.
4. Once utilization has risen and remaining liquidity is considerably lower than the adversaries' shares, attacks can be launched to achieve their objectives. This stage should ideally be of a short duration since the execution of a utilization kink attack incentivizes other actors to balance utilization. Attackers can respond by further reducing their position upon other actors' actions, thereby continuing to accrue interest. If a large liquidity provider enters the system, attackers can reinfuse all withdrawn funds back into the protocol to sustain fee earnings. However, honest liquidity providers might have no incentive to aid a pool under attack if they anticipate temporary high utilization, making it unadvisable for them to move large volumes of liquidity to help the pool.
5. Continued attacks may lead to general actors in the network blacklisting the attacked protocol, in such situations attackers can easily migrate to a new vulnerable protocol.

In this economic game, attackers stand to profit over the long term. Two primary issues arise:

- **Low-Risk, High-Reward Game for Attackers:** Attackers stand to gain exponentially from 5 to 50 times more fees during the attack period without facing any substantial risks unless the protocol experiences a major hack. This allows them to perpetuate such activities over a long duration.

- **No Financial Incentives for Honest Players:** Existing pools incentivize players by raising interest rates; however, if attackers respond swiftly to honest actors joining the pool, there would be no financial incentive for honest players to rescue minor protocols.

Hence, protocols need to address these attacks at the design level to foster growth and safeguard their users against malicious activities.

While it is feasible for an attacker to simultaneously execute the mentioned attacks by elevating the utilization to its maximum, the objectives for conducting each attack differ. Here, we discuss some of these variations:

- **Utilization Kink Attack:** To execute this attack, malicious liquidity providers need to initially supply liquidity to a specific pool and wait until a part of their liquidity is borrowed. Only then can they employ the remainder of their funds to increase the utilization. In such attacks, all borrowers within the pool are targeted, and the attacker's profit accumulates over time.
- **DoS Attack:** In order to carry out a DoS attack, attackers can retain their funds outside the protocols, monitor multiple systems, and potentially target specific actors if their funding is sufficient. A DoS attack is intended to transpire swiftly within a specific block and is not a continuous action. This approach aims to avoid associated fees.

5 Liquidity aggregation

In previous discussions, we explored the issue of liquidity attacks. We proposed some tactical solutions, like extending liquidity commitments and setting base fees, to deal with such issues. But in this segment, our aim is to get to the core of the problem and offer a comprehensive solution. Our solution could safeguard new lending pools from potential attacks while facilitating their rapid growth.

Often, smaller lending pools try to emulate the larger ones such as Compound and Aave. This leads many protocols to design their logic layer centered around their liquidity pool. In this setup, the logic and liquidity components become inseparable parts of a single, large project. Consequently, each pool has to grow independently. Our proposition is to separate the liquidity and logic layers in the design of such protocols. This separation could let several protocols combine their liquidity layers, possibly strengthening the weaker pools. We recommend the following three-step launch for every new liquidity pool:

1. Design the pool such that the logic and liquidity layers are separate. The logic layer should only interact with the liquidity layer when necessary. This arrangement could allow the liquidity layer to be shared among many protocols.
2. Initially, smaller liquidity pools can connect themselves to larger pools such as Compound. This connection means that they only run out of liquidity when Compound does, protecting them from most liquidity management attacks. This method enforces some limitations on the smaller pool, as it has to conform to the larger pool's constraints.
3. Once the connected pool has sufficient funds, it can operate independently and set its own rules.

By following these steps (as shown in Figure 2), an ecosystem of lending pools can reap mutual benefits. These benefits include:

- **Attack Resilience:** Smaller pools protect their users from attacks. It becomes more difficult for an attacker to raise borrowers' fees. Also, liquidity providers have the freedom to withdraw their funds at any time since the larger underlying pool provides more liquidity.

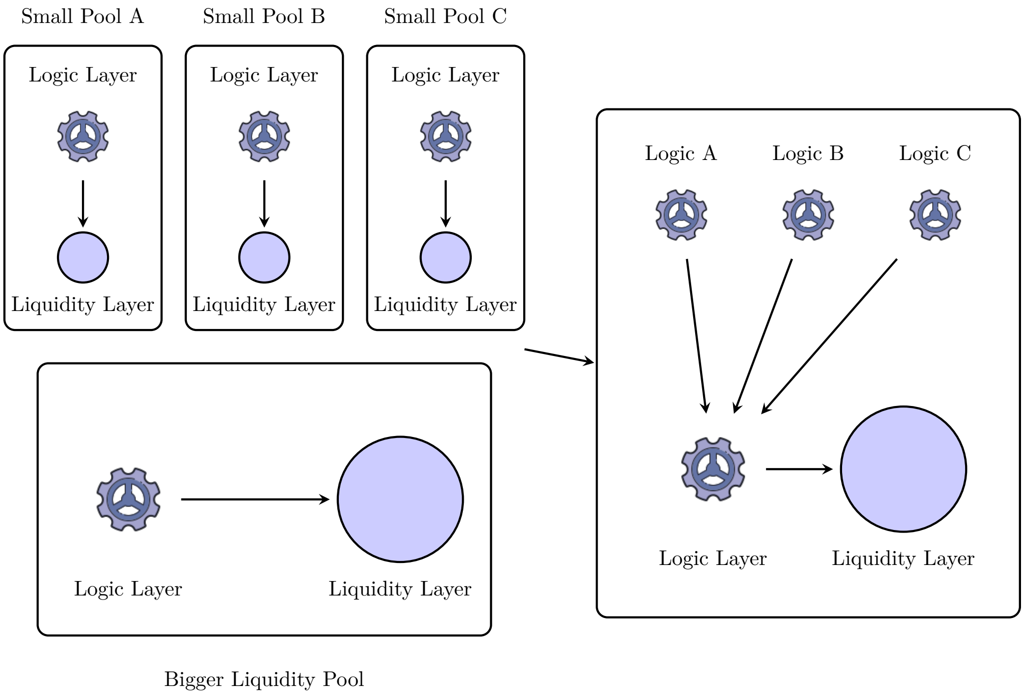

Figure 2 Liquidity aggregation process, how smaller pools can piggyback off larger pools.

- **Larger Shared Pool:** The larger pools also benefit from this arrangement. They now have a larger pool of liquidity providers. Many protocols can use their liquidity for security, while merging their pools to enhance the overall security of the ecosystem.

In the following parts of this section, we aim to explain the complexity in the process of implementing such systems.

5.1 Designing Logic and Liquidity Layers

The goal of this section is to propose a design that separates the logic and liquidity layers of a lending pool. However, we still need these layers to merge together and form a complete lending system. This design expands upon the traditional lending pools' design of one-to-one logic and liquidity layers. It also potentially allows for the integration of multiple logic layers without the need to change the implementation of the liquidity layer.

The logic layer of the lending protocol is deployed via a smart contract, which should be the point of interaction for all users of the protocol. This means the logic layer must handle all bookkeeping and monitor each participant's activity, and it is not designed to hold any funds. When users interact with the protocol via the logic layer, it facilitates the transfer of funds between users and the liquidity pool after conducting necessary checks. On the other side, the liquidity layer, which holds all funds, should only respond to the logic layer contract.

A design layer should have the capability to (1) interface with another logic layer, thereby piggybacking on the infrastructure of another protocol, or (2) function as a standalone liquidity layer, in which it independently manages all of its funds.

5.1.1 Piggybacking Liquidity Pool

When a design layer is in piggybacking mode, it is connected to another design layer. This allows us to establish a system like $D_1, D_2, \ldots, D_N, LL_N$, where D_is are design layers and LL_N is the liquidity layer that only responds to D_N. Here, $D_1, D_2, \ldots, D_{N-1}$ are all in piggybacking mode, and D_N operates in standalone mode. While users can interact with any of the D_i to use their services, their liquidity will be forwarded through $D_i + 1, D_N$ and must comply with all their logic. In this setup, each of D_i has its own users, but all that D_{i+1} sees from the previous logic layer is the entry of D_i, which is using the system just like other users. The simplest version of the use case that interests us is where $N = 2$. Here, D_1 is a small lending pool, and D_2 is one of the largest existing lending pools, such as Compound. In this setting, while users interact with the D_1, their funds are getting accumulated in D_2's pool LL_2. The significant benefit here is that if D_1 runs out of funds, it is backed up by the bigger lending pool's funds and can support its users. We delve deeper into how each basic functionality changes when the design layer is piggybacking off other design layer when a user interacts with D_1:

- **Supply:** Whenever a user supplies amount X to the D_1, then supply of the system changes as:

$$\begin{aligned} S_{D1,user} &\mathrel{+}= X \\ \forall_{1<i\leq N} S_{Di,Di-1} &\mathrel{+}= X \\ L &\mathrel{+}= X \end{aligned} \tag{10}$$

  This means that each logic layer supplies funds to the next one, and the final pool supplies it to the pool.

- **Collateral:** when users supply collateral to the protocol, the state changes are similar to the supply:

$$\begin{aligned} C_{D1,user} &\mathrel{+}= X \\ \forall_{1<i\leq N} C_{Di,Di-1} &\mathrel{+}= X \\ C &\mathrel{+}= X \end{aligned} \tag{11}$$

- **Borrow and liquidation:** For a borrow of amount X to happen, the borrow process is happening in every single layer. Therefore, the collateral that user has provided, should follow the equation below:

$$X > max_i(\Sigma_c(C_{user,c,i} \times f_{c,i})) \tag{12}$$

  This implies that the collateral tokens submitted should exceed the borrowing amount in each logic layer. If the aforementioned condition is not met, the funds could potentially face liquidation in one of the layers. For protocols to ensure that the equation above is never broken, they need to limit their collateral factors, so that $f_{c,i} < f_{c,i+1}$. in such cases the collateral equation gets reduced to a limit against the effective collateral of the user at layer 1:

$$X > \Sigma_c(C_{user,c,1} \times f_{c,1}) = EC_{user,1} \tag{13}$$

  The state changes for borrow are:

$$\begin{aligned} B_{D1,user} &\mathrel{+}= X \\ \forall_{1<i\leq N} B_{Di,Di-1} &\mathrel{+}= X \\ B &\mathrel{+}= X \end{aligned} \tag{14}$$

When a user seeks to borrow from the protocol and a layer runs out of liquidity, the protocol can borrow from the layer beneath it. This mechanism increases the confidence in liquidity availability.

- **Interest Rate Calculation:** Should there be no borrow at layer i, the total liquidity supplied to this layer, denoted as $S_{total,i}$, earns interest at the rate of the succeeding layer, or $i+1$. This follows the formula:

  $$R_{i+1} \times S_{total,i} \tag{15}$$

  Now, if any borrowing occurs from the protocol at layer i, the interest rate from the underlying protocol is given by:

  $$R_{i+1} \times (S_{total,i} - B_{total_i}) + R_i \times B_{total_i} \tag{16}$$

  Which depends on the interest rate of D_i. In order to incentivize more liquidity providers to join the protocol with an increase in borrowing, it is necessary that the condition $R_i \geq R_{i+1}$ be met. This requirement ensures that the previously mentioned formula progressively increases with the growth in borrowing positions. It indicates that the interest rate for layer i should surpass that of layer $i+1$. The proposed interest rate for level i extends from the kinked interest rate algorithm, following the subsequent equation:

  $$\forall_{1\leq i<N}, R_i = \begin{cases} R_{i+1} + R_{low,i} \times U_i & if\ U \leq kink \\ R_{i+1} + R_{low,i} \times kink + R_{high,i} \times (U_i - kink) & if\ U > kink \end{cases} \tag{17}$$

  The interest rate at each level is influenced by U_i. A significant difference in this model is that U_i can exceed the value of one. This is because each layer can lean on the next one for support, and therefore the borrowed amount within a specific protocol can go beyond the supplied amount. However, this also leads to a rise in the interest rate. To stop the growth of the interest rate at max utilization, protocol designers that are using this model could replace the U_i value with $Min(1, U_i)$.

In this setup, the outermost design layers can make use of the liquidity from all underlying protocols. However, this comes at the cost of stricter restrictions on their protocol variables. This implies that for an attacker to carry out a DoS attack on layer i, they now need to have enough funds to exhaust all layers from $i+1$ to N. On the other hand, if a lending protocol wants to connect to another protocol's logic layer, they don't need to set a steep $R_{high,i}$ fee beyond their optimal utilization. Instead, they can rely on the liquidity from the underlying layer. As such, this system is more resistant to utilization kink attacks due to a smaller $R_{high,i}/R_{low,i}$ ratio, compared to standalone pools.

5.1.2 Standalone liquidity pool

Once a protocol has matured and expanded its TVL by piggybacking off another lending pool, it may be time for the protocol owners to consider transitioning into standalone mode. This transition involves the protocol creating its own liquidity pool and transferring its assets into this new pool. It's crucial to note here that when a protocol detaches from the next one, it also severs connections with all its preceding protocols and transfers them as well. In essence, if in the chain $D_1, D_2, ..., D_i, D_{i+1}, ..., D_N, LL_N$, layer i decides to detach, it would result in two separate chains: $D_1, D_2, ..., D_i, LL_i$, and $D_i, D_{i+1}, ..., D_N, LL_N$.

Protocols should only transition to standalone mode when they have accumulated enough liquidity to fend off liquidity management attacks independently. Furthermore, during this

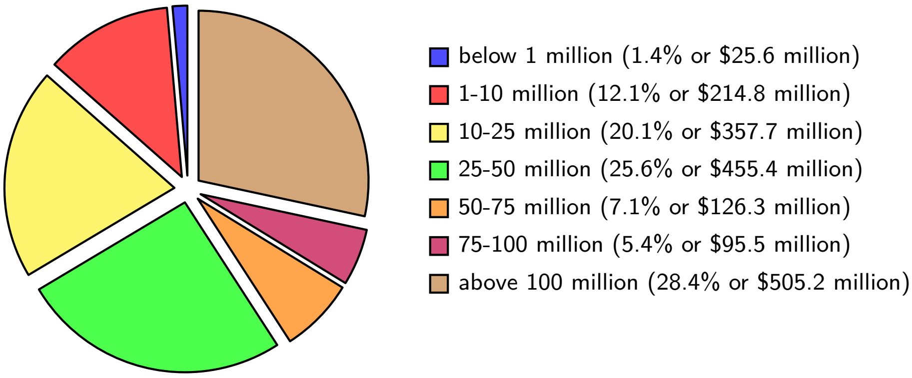

Figure 3 asset distribution beyond the top 6 protocols, totaling $1.75b.

transition, it would be advantageous for the ecosystem if the funds weren't withdrawn all at once. As these lending pools possess large liquidity pools, withdrawing all the funds abruptly could potentially trigger a spike in the underlying pools' utilization. We recommend that, at this stage, lending pools transition to a new pool by gradually vesting all the liquidity over a certain time period. For instance, a protocol could gradually withdraw all funds over the course of a day, after duly notifying the community.

6 Analyzing on-chain lending protocols

In this section, we dive into the lending pools deployed across multiple blockchain networks. Our data collection efforts aim to understand their design, TVL, and potential susceptibilities to liquidity management attacks. Our study includes two types of pools. Initially, we analyze the six most prominent lending pools in the space, and then we shift our focus to scrutinize the rest of the lending pools. Although the larger lending pools are typically secure from liquidity management attacks due to their significant liquidity base, analyzing them remains crucial as they significantly influence numerous emerging lending protocols.

According to reports [10], lending pools on the chain hold over $13.2b in TVL. Of this amount, 86.6% resides within the top six lending pools. We examine each of these influential pools, recognizing their role as templates and foundations for subsequent projects, which may adapt and develop their logic.

We also analyze smaller pools to determine their potential vulnerability to liquidity management attacks. These pools hold over $1.75b across 240 protocols on various chains, posing a tempting target for potential attackers. As shown in Figure 4 our investigation reveals that 32.5% of all 240 smaller lending pools are officially forks of Compound, while over 10% have branched off from Aave. Among the remaining 132 pools, many draw inspiration from the design choices of more established protocols, including aspects such as interest rate determination, supply, borrowing, and liquidation mechanisms. Figure 3 illustrates the distribution of funds across these protocols. When comparing the liquidity distribution of smaller pools with the daily trading volume of Aave, which has consistently exceeded $30 million since the start of 2023, it becomes plausible that such amount of funds is not out of reach for users in the network. Given this amount of funds, attackers could potentially execute the mentioned attacks on these pools.

Our analysis comprises a selection of noteworthy protocols, including Aave, Compound, JustLend [17], Venus [28], Morpho [13], and Radiant [25]. You can find the detailed informa-

Table 2 Data describing the six largest lending pools.

Protocol	TVL Amount	Number of Markets	Interest rate model	Liquidity Management attacks
Aave	\$5.46b	13	Aave Model	Vulnerable
JustLend	\$3.78b	1	Aave Model	Vulnerable
Compound	\$1.92b	4	Compound Model	Vulnerable
Venus	\$804.55m	1	Compound Model	Vulnerable
Morpho	\$341.38m	3	P2P/Compound Model	Possible
Radiant	\$260.09m	3	Aave Model	Vulnerable

tion in Table 2. In the subsequent part of this section, we will delve into each aspect and investigate whether any of the protocols employ innovative approaches:

- **TVL:** We examine the amount of TVL each market holds and the degree of liquidity concentration which is shown by the number of markets. It's common for protocols to be deployed on multiple chains for user accessibility. Additionally, protocols often release new versions over time. While users typically prefer the latest versions, older versions can coexist and continue to serve users. For example, despite the launch of Compound V3 in August 2022, a substantial sum, exceeding \$1.32 billion, is still locked in Compound V2.
- **Supply and Borrow Mechanism:** Most lending pools utilize a similar supply and borrow mechanism, consistent with the one we outlined in our model. However, some protocols incorporate different logic, like P2P lending, and impose additional restrictions. Morpho, for instance, uses a P2P system to pair borrowers with lenders, transferring the borrower to the backup protocol, Compound, if the lender needs to withdraw their funding at any point. This mechanism makes Morpho somewhat resistant to liquidity management attacks, as borrowers borrowing from honest liquidity providers remain secure.
- **Interest Rate Model:** The interest rate model we presented in this paper generalizes those used in the mentioned protocols. Typically, smaller pools widely adopt two main models, those being Compound and Aave, due to their proven efficacy and popularity. The Compound model aligns with the model we utilized in this paper, while Aave's model, though similar, employs different variables:

$$R = \begin{cases} R'_0 + R'_{low} \times \frac{U}{kink} & if\ U \leq kink \\ R'_0 + R'_{low} + R'_{high} \times \frac{U-kink}{1-kink} & if\ U > kink \end{cases} \tag{18}$$

  Even though the formulas bear strong resemblances, they are provided to allow readers to reason with numerical examples. Aave also offers users a choice between stable and variable rates. In this paper, we presumed that protocols only offer variable rates for simplicity. Although stable rates do not alter the assumptions and results of our analysis, we direct the reader to the Aave white paper for more information on stable rates [2].
- **Attack Vulnerability:** We assess whether the pool is generally susceptible to liquidity management attacks. In each case, we assume the attacker possesses ample funds and is pursuing a specific objective. This section highlights the importance of design choices for new protocols adopting each of these larger protocols' designs during their initial public usage, a phase when they may have limited overall liquidity and thus be vulnerable to potential exploitation by an attacker.

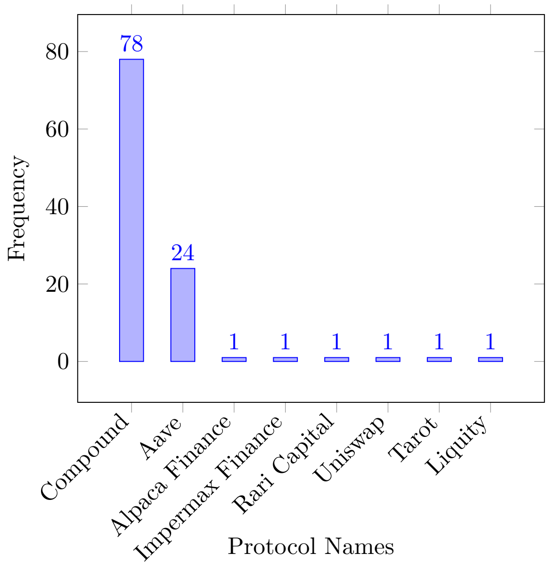

Figure 4 Frequency of protocols forked by newer projects.

7 Related work

Gudgeon et al.[15] use the term *Protocols for Loanable Funds (PLF)* to denote markets for loanable funds. Their work classifies various interest rate models utilized by leading lending protocols, including the "kinked rates" model, which is widely used by the protocols examined in our study. Bartoletti et al.[4] conceptualize the overall structure of lending pools as a state machine, analyzing different state transitions and potential threats. They introduced concepts such as over-utilization and under-utilization attacks, where attackers drive the utilization to its maximum or minimum. Sun et al.[27] explore various liquidity risks, using Aave as a case study to emphasize the significance of the issue. Hafner et al.[16] assess the degree of centralization among liquidity providers in a pool, identifying scenarios where low initial centralization could lead to liquidity shortages following substantial withdrawals. Our work extends these studies by defining liquidity management attacks and examining the motivations of a potential attacker.

8 Conclusion and future work

In this paper, we have introduced and formalized two liquidity management attacks, where an attacker with sufficient resources can exploit specific conditions within lending pools. We have demonstrated that such attacks are not only feasible but also incentivized, given the considerable amount of liquidity dispersed across numerous small liquidity pools. We further explored possible mitigation strategies and risks at the application layer that could aid upcoming lending protocols.

We additionally analyzed a specific design, wherein the design and application layer are structured as separate systems that can interact with each other. This structure enhances the flexibility of options available to liquidity pools and allows for the combination of multiple design layers that can utilize the same liquidity pool. While we scrutinized the overarching design of such systems, there remain considerable complexities to be addressed in their implementation. It is our hope that new lending pools will adopt this design and potentially establish a standard set of defensive mechanisms against liquidity management attacks.

References

1 Aave protocol website, 2023. URL: `https://aave.com/`.

2 Aave protocol whitepaper v1.0, 2020. URL: `https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf`.

3 Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts. Cryptology ePrint Archive, Paper 2016/1007, 2016. `https://eprint.iacr.org/2016/1007`. URL: `https://eprint.iacr.org/2016/1007`.

4 Massimo Bartoletti, James Hsin yu Chiang, and Alberto Lluch-Lafuente. Sok: Lending pools in decentralized finance, 2020. `arXiv:2012.13230`.

5 Bnb bridge - rekt, 2022. URL: `https://rekt.news/bnb-bridge-rekt/`.

6 Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121, 2015. `doi:10.1109/SP.2015.14`.

7 Compound protocol website, 2023. URL: `https://compound.finance/`.

8 Simon Cousaert, Jiahua Xu, and Toshiko Matsui. SoK: Yield aggregators in DeFi. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, may 2022. URL: `https://doi.org/10.1109%2Ficbc54727.2022.9805523`, `doi:10.1109/icbc54727.2022.9805523`.

9 Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges, 2019. `arXiv:1904.05234`.

10 Defillama, 2023. URL: `https://defillama.com/`.

11 Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain, 2019. `arXiv:1902.05164`.

12 Flashbots documentation, 2023. URL: `https://docs.flashbots.net/`.

13 Mathis Gontier Delaunay, Quentin Garchery, Paul Frambot, Merlin Égalité, Julien Thomas, and Katia Babbar. Morpho V1 Yellow Paper. working paper or preprint, May 2023. URL: `https://hal.science/hal-04087388`.

14 Lewis Gudgeon, Daniel Perez, Dominik Harz, Benjamin Livshits, and Arthur Gervais. The decentralized financial crisis, 2020. `arXiv:2002.08099`.

15 Lewis Gudgeon, Sam M. Werner, Daniel Perez, and William J. Knottenbelt. Defi protocols for loanable funds: Interest rates, liquidity and market efficiency, 2020. `arXiv:2006.13922`.

16 Matthias Hafner, Romain de Luze, Nicolas Greber, Juan Beccuti, Benedetto Biondi, Gidon Katten, Michelangelo Riccobene, and Alberto Arrigoni. Defi lending platform liquidity risk: The example of folks finance: Published in the journal of the british blockchain association, Apr 2023. URL: `https://jbba.scholasticahq.com/article/74150-defi-lending-platform-liquidity-risk-the-example-of-folks-finance`.

17 Justlend dao money market protocol v1.0, Dec 2020. URL: `https://portal.justlend.org/docs/justlend_whitepaper_en.pdf`.

18 Robert Leshner and Geoffrey Hayes, Feb 2019. URL: `https://compound.finance/documents/Compound.Whitepaper.pdf`.

19 Amani Moin, Kevin Sekniqi, and Emin Gun Sirer. Sok: A classification framework for stablecoin designs. In Joseph Bonneau and Nadia Heninger, editors, *Financial Cryptography and Data Security*, pages 174–197, Cham, 2020. Springer International Publishing.

20 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, May 2009. URL: `http://www.bitcoin.org/bitcoin.pdf`.

21 OpenZeppelin. Openzeppelin/openzeppelin-contracts: Openzeppelin contracts is a library for secure smart contract development. URL: `https://github.com/OpenZeppelin/openzeppelin-contracts`.

22 Poly network - rekt, 2021. URL: `https://rekt.news/polynetwork-rekt/`.

23 Kaihua Qin, Liyi Zhou, Yaroslav Afonin, Ludovico Lazzaretti, and Arthur Gervais. Cefi vs. defi – comparing centralized to decentralized finance, 2021. `arXiv:2106.08157`.

24 Kaihua Qin, Liyi Zhou, Pablo Gamito, Philipp Jovanovic, and Arthur Gervais. An empirical study of DeFi liquidations. In *Proceedings of the 21st ACM Internet Measurement Conference*. ACM, nov 2021. URL: `https://doi.org/10.1145%2F3487552.3487811`, `doi:10.1145/3487552.3487811`.

25 Radiant documentation, 2023. URL: `https://docs.radiant.capital/radiant/`.

26 Huobi Research. Global crypto industry overview and trends[2022–2023 annual report](first part), Dec 2022. URL: `https://medium.com/huobi-research/global-crypto-industry-overview-and-trends-2022-2023-annual-report-first-part-e15372f29c`.

27 Xiaotong Sun, Charalampos Stasinakis, and Georgios Sermpinis. Liquidity risks in lending protocols: Evidence from aave protocol, 2023. `arXiv:2206.11973`.

28 Venus protocol documentation, 2023. URL: `https://docs.venus.io/docs/getstarted`.

29 Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczentewicz, and Arthur Gervais. Blockchain censorship, 2023. `arXiv:2305.18545`.

30 Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market, 2023. `arXiv:2305.16468`.

31 Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi), 2022. `arXiv:2101.08778`.

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

33 Jiahua Xu, Krzysztof Paruch, Simon Cousaert, and Yebo Feng. SoK: Decentralized exchanges (DEX) with automated market maker (AMM) protocols. *ACM Computing Surveys*, 55(11):1–50, feb 2023. URL: `https://doi.org/10.1145%2F3570639`, `doi:10.1145/3570639`.

34 Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges, 2020. `arXiv:2009.14021`.

35 Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. Sok: Decentralized finance (defi) attacks, 2023. `arXiv:2208.13035`.