Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit

Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais
Imperial College London, United Kingdom
{kaihua.qin,liyi.zhou,b.livshits,a.gervais}@imperial.ac.uk

Abstract. Credit allows a lender to loan out surplus capital to a borrower. In the traditional economy, credit bears the risk that the borrower may default on its debt, the lender hence requires upfront collateral from the borrower, plus interest fee payments. Due to the atomicity of blockchain transactions, lenders can offer flash loans, i.e., loans that are only valid within one transaction and must be repaid by the end of that transaction. This concept has lead to a number of interesting attack possibilities, some of which were exploited in February 2020.

This paper is the first to explore the implication of transaction atomicity and flash loans for the nascent decentralized finance (DeFi) ecosystem. We show quantitatively how transaction atomicity increases the arbitrage revenue. We moreover analyze two existing attacks with ROIs beyond 500k%. We formulate finding the attack parameters as an optimization problem over the state of the underlying Ethereum blockchain and the state of the DeFi ecosystem. We show how malicious adversaries can efficiently maximize an attack profit and hence damage the DeFi ecosystem further. Specifically, we present how two previously executed attacks can be "boosted" to result in a profit of 829.5k USD and 1.1M USD, respectively, which is a boost of 2.37× and 1.73×, respectively.

1 Introduction

A central component of our economy is credit: to foster economic growth, market participants can borrow and lend assets to each other. If credit creates new and sustainable value, it can be perceived as a positive force. Abuse of credit, however, necessarily entails negative future consequences. Excessive debt can lead to a debt default — i.e., a borrower is no longer capable to repay the loan plus interest payment. This leads us to the following intriguing question: What if it were possible to offer credit without bearing the risk that the borrower does not pay back the debt? Such a concept appears impractical in the traditional financial world. No matter how small the borrowed amount, and how short the loan term, the risk of the borrower defaulting remains. If one were absolutely certain that a debt would be repaid, one could offer loans of massive volume – or lend to individuals independently of demographics and geographic location, effectively providing capital to rich and poor alike.

Given the peculiarities of blockchain-based smart contracts, flash loans emerged. Blockchain-based smart contracts allow to programmatically enforce the atomic

arXiv:2003.03810v4 [cs.CR] 20 Mar 2021

execution of a transaction. A flash loan is a loan that is only valid within one atomic blockchain transaction. Flash loans fail if the borrower does not repay its debt before the end of the transaction borrowing the loan. That is because a blockchain transaction can be reverted during its execution if the condition of repayment is not satisfied. Flash loans yield three novel properties, absent in traditional finance:

– No debt default risk: A lender offering a flash loan bears no risk that the borrower defaults on its debt[1]. Because a transaction and its instructions must be executed atomically, a flash loan is not granted if the transaction fails due to a debt default.
– No need for collateral: Because the lender is guaranteed to be paid back, the lender can issue credit without upfront collateral from the borrower: a flash loan is non-collateralized.

– Loan size: Flash loans can be taken from public smart contract-governed liquidity pools. Any borrower can borrow the entire pool at any point in time. As of September 2020, the largest flash loan pool Aave [13] offers in excess of 1B USD [1].

To the best of our knowledge, this is the first paper that investigates flash loans. This paper makes the following contributions:

– Flash loan usage analysis. We provide a comprehensive overview of how and where the technique of flash loans can and is utilized. At the time of writing, flash loan pool sizes have reached more than 1B USD.

– Post mortem of existing attacks. We meticulously dissect two events where talented traders realized a profit of each about 350k USD and 600k USD with two independent flash loans: a pump and arbitrage from the 15th of February 2020 and an oracle manipulation from the 18th of February 2020.

– Attack parameter optimization framework. Given the interplay of six DeFi systems, covering exchanges, credit/lending, and margin trading, we provide a framework to quantify the parameters that yield the maximum revenue an adversary can achieve, given a specific trading attack strategy. We show that an adversary can maximize the attack profit efficiently (in less than 13ms) due to the atomic transaction property.

– Quantifying opportunity loss. We show how the presented flash loan attackers have forgone the opportunity to realize a profit exceeding 829.5k USD and 1.1M USD, respectively. We realize this by finding the optimal adversarial parameters the trader should have employed, using a parametrized optimizer. We experimentally validate the opportunity loss on a locally deployed blockchain mirroring the attacks' respective blockchain state.

– Impact of transaction atomicity on arbitrage. We show quantitatively how atomicity reduces the risk of revenue from arbitrage. Specifically, by analyzing 6.4M transactions, we find that the expected arbitrage reward decreases by 123.49±1375.32 USD and 1.77±10.59 USD for the DAI/ETH and

1 Besides the risk of smart contract vulnerabilities.

MKR/ETH markets respectively when the number of intermediary transactions reaches 5, 000.

Paper organization: The remainder of the paper is organized as follows. Section 2 elaborates on the DeFi background. Section 3 dissects two known flash loan attacks. Section 4 proposes a framework to optimize the attack revenues and Section 5 evaluates the framework on the two analyzed attacks. Section 6 analyses the implications of the atomic transaction property. Section 7 provides a discussion. We conclude the paper in Section 8.

2 Background

Decentralized ledgers, such as Bitcoin [44], enable the performance of transactions among a peer-to-peer network. At its core, a blockchain is a chain of blocks [17,44], extended by miners crafting new blocks that contain transactions. Smart contracts [49] allow the execution of complicated transactions, which forms the foundation of decentralized finance, a conglomerate of financial cryptocurrency-related protocols. These protocols for instance allow to lend and borrow assets [39,4], exchange [24,11], margin trade [24,3], short and long [3], and allow to create derivative assets [4]. At the time of writing, the DeFi space accounts for over 8B USD in smart contract locked capital among different providers. The majority of the DeFi platforms operate on the Ethereum blockchain, governed by the Ethereum Virtual Machine (EVM), where the trading rules are governed by the underlying smart contracts. A decentralized exchange is typically referred to as DEX. We refer to the on-chain DeFi actors as traders and distinguish among the two types of traders:

Liquidity Provider: a trader with surplus capital may choose to offer this capital to other traders, e.g., as collateral within a DEX or lending platform.

Liquidity Taker: a trader which is servicing liquidity provider with fees in exchange for accessing the available capital.

2.1 DeFi Platforms

We briefly summarize relevant DeFi platforms for this work.
Automated market maker (AMM) DEX: While many exchanges follow the limit order book design [40,35,34], an alternative exchange design is to collect funds within a liquidity pool, e.g., two pools for an AMM asset pair X/Y [11,34]. The state (or depth) of an AMM market X/Y is defined as (x, y), where x represents the amount of asset X and y the amount of asset Y in the liquidity pool. Liquidity providers can deposit/withdraw in both assets X and Y to in/decrease liquidity. The simplest AMM mechanism is a constant product market maker, which for an arbitrary asset pair X/Y , keeps the product $x \times y$ constant during trades. When trading on an AMM exchange, there can be a difference between the expected price and the executed price, termed slippage [10]. Insufficient liquidity and other front-running trades can cause slippage on an

AMM [52]. We assume that a constant product AMM ETH/WBTC market is supplied with 10 ETH and 10 WBTC (i.e., the exchange rate is 1 ETH/WBTC). A trader can purchase 5 WBTC with 10 ETH (cf. $10\times10 = (10+ 10)\times(10-5)$) at an effective price of 2 ETH/WBTC. Hence, the slippage is $\frac{2-1}{1} = 100\%$.

Margin trading: Trading on margin allows a trader to take under-collateralized loans from the trading platform and trade with these borrowed assets to amplify the profit (i.e., leverage). On-chain margin trading platforms remain in control of the loaned asset (or the exchanged asset) and hence is able to liquidate when the value of the trader's collateral drops too low.

Credit and lending: With over 3B USD total locked value, credit represents one of the most significant recent use-cases for blockchain based DeFi systems. Due to the lack of legal enforcement when borrowers default, they are required to provide between 125% [24] to 150% [39] collateral of an asset x to borrow 100% of another asset y (i.e., over-collateralization).

2.2 Reverting EVM State Transitions

The Ethereum blockchain is in essence a replicated state machine. To achieve a state transition, one applies as input transactions that modify the EVM state following rules encoded within deployed smart contracts. A smart contract can be programmed with the logic of reverting a transaction if a particular condition is not met during execution. The EVM state is only altered if a transaction executes successfully, otherwise, the EVM state is reverted to the previous, non-modified state.

Flash Loans. Flash loans are possible because the EVM allows the reversion of state changes. A flash loan is only valid within a single transaction and relies on the atomicity of blockchain (and, specifically, EVM) transactions within a single block. Flash loans entail two important new financial properties: First, a borrower does not need to provide upfront collateral to request a loan of any size, up to the flash loan liquidity pool amount. Any borrower, willing to pay the required transaction fees (which typically amounts to a few USD) is an eligible borrower. Second, risk-free lending: If a borrower cannot pay back the loan, the flash loan transaction fails. Ignoring smart contract and blockchain vulnerabilities, the lender is hence not exposed to the risks of a debt default.

2.3 Flash Loan Usage in the Wild

To our knowledge, the Marble Protocol introduced the concept of flash loans [8]. Aave [13] is one of the first DeFi platforms to widely advertise flash loan capabilities (although others, such as dYdX also allow the non-documented possibility to borrow flash loans) since January 2020. At the time of writing, Aave charges a constant 0.09% interest fee for flash loans and amassed a total liquidity beyond 1B USD [1]. In comparison, the total volume of U.S. corporation debt reached 10.5T USD in August, 2020 [12].

9200000 9600000 10000000 10400000 10800000
Block
$1

$10
$100
$1k
$10k
$100k
$1M
$10M
$100M
Accumulative Flash Loan Amount (USD)
Token (price)
DAI ($1)
ETH ($350)
USDC ($1)
BAT ($0.2)
WBTC ($10,000)
ZRX ($0.3)
MKR ($500)
LINK ($10)
USDT ($1)
REP ($15)
KNC ($1.5)
LEND ($0.5)
sUSD ($1)

Fig. 1: Accumulative flash loan amounts of 13 cryptocurrencies on Aave. Note that the y-axis is a logarithmic scale.

By gathering all blockchain event logs from Aave with a full archive Ethereum node, we find 5, 616 flash loans issued from the Aave smart contract (cf. 0x398eC7346DcD622eDc5ae82352F02bE94C62d119) between the 8th of January, 2020 and the 20th of September, 2020. In Figure 1, we show the accumulative flash loan amounts of 13 different loan currencies. Among them, DAI is the most popular with the accumulative amount of 447.2M USD. We inspect and classify the Aave flash loan transactions depending on which platforms the flash loans interact with (cf. Figure 11 in Appendix A). We notice that most flash loans interact with lending/exchange DeFi systems and that the flash loan's transaction costs (i.e., gas) appear significant (at times beyond 4M gas, compared to 21k gas for regular Ether transfer). The dominating use cases are arbitrage and liquidation. Further details are presented in Appendix A.

Flash Loan Arbitrage example: The value of an asset is typically determined by the demand and supply of the market, across different exchanges. Due to a lack of instantaneous synchronization among exchanges, the same asset can be traded at slightly different prices on different exchanges. Arbitrage is the process of exploiting price differences among exchanges for a financial gain [46]. In Figure 2, we present, as an example, the execution details of a flash loan based arbitrage transaction on the 31st of July, 2020. The arbitrageur borrowed a flash loan of 2.048M USDC, performed two exchanges, and realized a profit

1. Flash Loan M USDC
4. Repay M USDC
2. Exchange M USDC
for M DAI
at price USDC/DAI
3. Exchange M DAI
for M USDC
at price USDC/DAI
dYdX arbitrageur
Curve
Y
Curve
Profit: k USDC sUSD

Fig. 2: High-level executions of a flash loan based arbitrage transaction 0xf7498a2

546c3d70f49d83a2a5476fd9dcb6518100b2a731294d0d7b9f79f754a: (1) flash loan; (2) exchange USDC for DAI in Curve Y pool; (3) exchange DAI for USDC in Curve sUSDC pool; (4) repay. Note that Curve provides several on-chain cryptocurrency markets, also known as pools.

of 16.182k USDC (16.182k USD). This example highlights how given atomic transactions, a trader can perform arbitrage on different on-chain markets, without the risk that the prices in the DEX would intermediately change. Flash loans moreover remove the currency volatility risk for arbitrageurs. In Section 6, we quantify the implications of transaction atomicity on arbitrage risks.

Besides arbitrage, we noticed another two use cases for flash loans: (i) wash trading (fraudulent inflation of trading volume), (ii) loan collateral swapping (instant swapping from one collateral to another), and also a variation of flash loan, (iii) flash minting (the momentarily token in- and decrease of an asset). We elaborate further on these in Appendix B and provide real-world examples.

2.4 Related work

There is a growing body of work focusing on various forms of manipulation and financially-driven attacks in cryptocurrency markets.

Crypto Manipulation: Front-running in cryptocurrencies has been extensively studied [25,22,18,32,5,52]. Remarkably, Daian et al. [22] introduce the concept of miner extractable value (MEV) and analyze comprehensively the exploitability of ordering blockchain transactions. Our work focuses on flash loans, which qualify as a potential MEV that miners could exploit. Gandal et al. [26] demonstrate that the unprecedented spike in the USD-BTC exchange rate in late 2013 was possibly caused by price manipulation. Recent papers focus on the phenomenon of pump-and-dump for manipulating crypto coin prices [51,33,28].

Smart Contract Vulnerabilities: Several exploits have taken advantage of smart contract vulnerabilities (e.g., the DAO exploit [9]). The most commonly known smart contract vulnerabilities are re-entrancy, unhandled exceptions, locked ether, transaction order dependency and integer overflow [37]. Many tools and techniques, based on fuzzing [36,31,50], static analysis [48,19,47], symbolic execution [37,43,45], and formal verification [16,14,27,29,30], emerged to detect and prevent these vulnerabilities. In this work, we focus on DeFi economic security, which might not result from a single contract vulnerability and could involve multiple DeFi platforms.

3 Flash Loan Post-Mortem

Flash loans enable anyone to have instantaneous access to massive capital. This section outlines how that can have negative effects, as we explain two attacks facilitated by flash loans yielding an ROI beyond 500k%. We evaluate the proposed DeFi attack optimization framework (cf. Section 4) on these two analyzed attacks (cf. Section 5).

3.1 Pump Attack and Arbitrage (PA&A)

On the 15th of February, 2020, a flash loan transaction (cf. 0xb5c8bd9430b6cc87a 0e2fe110ece6bf527fa4f170a4bc8cd032f768fc5219838 at an ETH price of 264.71 USD/ETH), followed by 74 transactions, yielded a profit of 1, 193.69 ETH (350k

1) Flash Loan Provider (dYdX)
borrow 10,000.00 ETH
2) Lending (Compound)
collateralize 5,500.00 ETH and borrow 112.00 WBTC
3.1) Margin Trade Provider (bZx)
5x short 1,300.00 ETH against WBTC
3.2) Exchange (WBTC Uniswap)
convert 5,637.62 ETH to 51.35 WBTC
4) Exchange (WBTC Uniswap)
convert 112.00 WBTC to 6,871.41 ETH
5) Flash Loan Provider (DyDx)
repay 10,000.00 ETH
3,200.00 ETH 10,071.41 ETH

71.41 ETH
Flash loan
transaction
(block 9484688)
Block
9484917 - 9496602
a) Exchange (Kyber)
convert 4,377.72 ETH to 112.00 WBTC
b) Lending (Compound)
repay 112.00 WBTC and redeem 5,500.00 ETH
ETH Flow
WBTC Flow
Uniswap Pool Size
2817.77 ETH
77.09 WBTC
8455.40 ETH
25.74 WBTC
Uniswap Pool Size
8455.40 ETH
25.74 WBTC
1583.98 ETH
137.74 WBTC

Fig. 3: The pump attack and arbitrage. The attack consists of two parts, a flash loan and several loan redemption transactions.

USD) given a transaction fee of 132.36 USD (cumulative 50, 237, 867 gas, 0.5 ETH). We show in Section 5.1 that the adversarial parameters were not optimal, and that the adversary could have earned a profit exceeding 829.5k USD.

Attack intuition: The core of PA&A is that the adversary pumps the price of ETH/WBTC on a constant product AMM DEX (Uniswap) with the leveraged funds of ETH in a margin trade. The adversary then purchases ETH at a “cheaper” price on the distorted DEX market (Uniswap) with the borrowed WBTC from a lending platform (Compound). As shown in Figure 3, this attack mainly consists of two parts. For simplicity, we omit the conversion between ETH and WETH (the 1:1 convertible ERC20 version of ETH).

Flash Loan (single transaction): The first part of the attack (cf. Figure 3) consists of 5 steps within a single transaction. In step 1 , the adversary borrows a flash loan of 10, 000.00 ETH from a flash loan provider (dYdX). In step 2 , the adversarial trader collateralizes 5, 500.00 ETH into a lending platform (Compound) to borrow 112.00 WBTC. Note that the adversarial trader does not return the 112.00 WBTC within the flash loan. This means the adversarial trader takes the risk of a forced liquidation against the 5, 500.00 ETH collateral if the price fluctuates. In steps 3 , the trader provides 1, 300 ETH to open a short position for ETH against WBTC (on bZx) with a 5× leverage. Upon receiving this request, bZx transacts 5, 637.62 ETH on an exchange (Uniswap) for only 51.35 WBTC (at 109.79 ETH/WBTC). Note that at the start of block 9484688, Uniswap has a total supply of 2, 817.77 ETH and 77.09 WBTC (at 36.55 ETH/WBTC). The slippage of this transaction is significant with $\frac{109.79-36.55}{36.55} = 200.38\%$. In step 4 , the trader converts 112.00 WBTC borrowed from lending platform (Compound) to 6, 871.41 ETH on the DEX (Uniswap) (at 61.35 ETH/WBTC). We remark that the equity of the adversarial margin account is negative after the margin trading because of the significant price movement. The pump attack could have been avoided if bZx checked the negative equity and reverted the transaction. At the time of the attack, this logic existed in the bZx contracts but was not invoked properly. In step 5 , the trader pays back the flash loan plus an interest of 10^{-7} ETH. After the flash loan transaction (i.e., the first part of PA&A), the trader gains 71.41 ETH, and has a debt of 112 WBTC over-collateralized by 5, 500 ETH (49.10 ETH/WBTC). If the ETH/WBTC market price is below this loan exchange rate, the adversary can redeem the loan's collateral as follows.

Loan redemption: The second part of the trade consists of two recurring steps, (step a - b), between Ethereum block 9484917 and 9496602. Those transactions aim to redeem ETH by repaying the WBTC borrowed earlier (on Compound).

To avoid slippage when purchasing WBTC, the trader executes the second part in small amounts over a period of two days on the DEX (Kyber, Uniswap). In total, the adversarial trader exchanged 4, 377.72 ETH for 112 WBTC (at 39.08 ETH/WBTC) to redeem 5, 500.00 ETH.

Identifying the victim: We investigate who of the participating entities is losing money. Note that in step 3 of Figure 3, the short position (on bZx) borrows 5, 637.62 − 1, 300 = 4, 337.62 ETH from the lending provider (bZx), with 1, 300 ETH collateral. Step 3 requires to purchase WBTC at a price of 109.79 ETH/WBTC, with both, the adversary's collateral and the pool funds of the liquidity provider. 109.79 ETH/WBTC does not correspond to the market price of 36.55 ETH/WBTC prior to the attack, hence the liquidity provider overpays by nearly 3× of the WBTC price.

How much are the victims losing: We now quantify the losses of the liquidity providers. The loan provider lose 4, 337.62 (ETH from loan providers) - 51.35 (WBTC left in short position) × 39.08 (market exchange rate ETH/WBTC) = 2, 330.86 ETH. The adversary gains 5, 500.00 (ETH loan collateral in Compound) - 4, 377.72 (ETH spent to purchase WBTC) + 71.41 (part 1) = 1, 193.69 ETH.

More money is left on the table: Due to the attack, Uniswap's price of ETH was reduced from 36.55 to 11.50 ETH/WBTC. This creates an arbitrage opportunity, where a trader can sell ETH against WBTC on Uniswap to synchronize the price. 1, 233.79 ETH would yield 60.65 WBTC, instead of 33.76 WBTC, realizing an arbitrage profit of 26.89 WBTC (286, 035.04 USD).

3.2 Oracle Manipulation Attack

We proceed to detail a second flash loan attack, which yields a profit of 2, 381.41 ETH (c. 634.9k USD) within a single transaction (cf. 0x762881b07feb63c436de e38edd4ff1f7a74c33091e534af56c9f7d49b5ecac15, on the 18th of February, 2020, at an ETH price of 282.91 USD/ETH) given a transaction fee of 118.79 USD. Before diving into the details, we cover additional background knowledge. We again show how the chosen attack parameters were sub-optimal and optimal parameters would yield a profit of 1.1M USD instead (cf. Section 5.2).

Price oracle: One of the goals of the DeFi ecosystem is to not rely on trusted third parties. This premise holds both for asset custody as well as additional information, such as asset pricing. One common method to determine an asset price is hence to rely on the pricing information of an on-chain DEX (e.g., Uniswap). DEX prices, however, can be manipulated with flash loans.

Attack intuition: The core of this attack is an oracle manipulation using a flash loan, which lowers the price of sUSD/ETH. In a second step, the adversary benefits from this decreased sUSD/ETH price by borrowing ETH with sUSD as collateral.

1) Flash Loan Provider (bZx)
borrow 7,500.00 ETH
2) Exchange (sUSD Uniswap)
convert 540.00 ETH to 92,419.70 sUSD
3) Exchange (Kyber)
convert 360.00 ETH to 63,584.09 sUSD
4) Exchange (Synthetix)
deposit 3,517.86 ETH for 943,837.59 sUSD
5) Lending (bZx)
collateralize 1,099,841.39 sUSD and borrow 6,799.27 ETH
6) Flash Loan Provider (bZx)
repay 7,500.00 ETH
3,082.14 ETH
1,099,841.39 sUSD

9,881.41 ETH
2,381.41 ETH ETH Flow sUSD Flow
Uniswap Pool Size
879.76 ETH 243,441.12 sUSD
1,419.76 ETH 151,021.42 sUSD
Kyber Reserve Pool Size
0.91 ETH 107,901.90 sUSD
360.91 ETH 44,317.80 sUSD

Fig. 4: The oracle manipulation attack.

Adversarial oracle manipulation: We identify a total of 6 steps within this transaction (cf. Figure 4). In step 1 , the adversary borrows a flash loan of 7, 500.00 ETH (on bZx). In the next three steps (2 , 3 , 4), the adversary converts a total of 4, 417.86 ETH to 1, 099, 841.39 sUSD (at an average of 248.95 sUSD/ETH). The exchange rates in step 2 and 3 are 171.15 and 176.62 sUSD/ETH respectively. These two steps decrease the sUSD/ETH price to 106.05 sUSD/ETH on Uniswap and 108.44 sUSD/ETH on Kyber Reserve, which are collectively used as a price oracle of the lending platform (bZx). Note that Uniswap is a constant product AMM, while Kyber Reserve is an AMM following a different formula (cf. Appendix C). The trade on the third market (Synthetix) in step 4 is yet unaffected by the previous trades. The adversarial trader then collateralizes all the purchased sUSD (1, 099, 841.39) to borrow 6, 799.27 ETH (at exchange rate
collateral factor = max(106.05, 108.44) × 1.5 = 162.66 sUSD/ETH on bZx). Now the adversary possesses 6, 799.27+3, 082.14 ETH and in the last step repays the flash loan amounting to 7, 500.00 ETH. The adversary, therefore, generates a revenue of 2, 381.41 ETH while only paying 0.42 ETH (118.79 USD) transaction fees.

Identifying the victim: The adversary distorted the price oracle (Uniswap and Kyber) from 268.30 sUSD/ETH to 108.44 sUSD/ETH, while other DeFi platforms remain unaffected at 268.30 sUSD/ETH. Similar to the pump attack and arbitrage, the lenders on bZx are the victims losing assets as a result of the distorted price oracle. The lender lost 6, 799.27 ETH - 1, 099, 841 sUSD, which is estimated to be 2, 699.97 ETH (at 268.30 sUSD/ETH). The adversary gains 6, 799.27 (ETH from borrowing) - 3, 517.86 (ETH to purchase sUSD) - 360 (ETH to purchase sUSD) - 540 (ETH to purchase sUSD) = 2, 381.41 ETH.

4 Optimizing DeFi Attacks

The atomicity of blockchain transactions guarantees the continuity of the action executions. When the initial state is deterministically known, this trait allows an adversary to predict the intermediate results precisely after each action execution and then to optimize the attacking outcome by adjusting action parameters. In light of the complexity of optimizing DeFi attacks manually, we propose a constrained optimization framework that is capable of optimizing the action parameters. We show, given a blockchain state and an attack vector composed of a series of DeFi actions, how an adversary can efficiently discover the optimal action parameters that maximize the resulting expected revenue.

4.1 System and Threat Model

The system considered is limited to one decentralized ledger which supports pseudo-Turing complete smart contracts (e.g., similar to the Ethereum Virtual Machine; state transitions can be reversed given certain conditions). We assume the presence of one computationally bounded and economically rational adversary A. A attempts to exploit the availability of flash loans for financial gain. While A is not required to provide its own collateral to perform the presented attacks, the adversary must be financially capable to pay transaction fees. The adversary may amass more capital which possibly could increase its impact and ROI.

4.2 Parametrized Optimization Framework

We start by modeling different components that may engage in a DeFi attack. To facilitate optimal parameter solving, we quantitatively formalize every endpoint provided by DeFi platforms as a state transition function $S' = T(S; p)$ with the constraints $C(S; p)$, where S is the given state, p are the parameters chosen by the adversary and S' is the output state. The state can represent, for example, the adversarial balance or any internal status of the DeFi platform, while the constraints are set by the execution requirements of the EVM (e.g., the Ether balance of an entity should never be a negative number) or the rules defined by the respective DeFi platform (e.g., a flash loan must be repaid before the transaction termination plus loan fees). When quantifying profits, we ignore the loan interest/fee payments and transaction fees, which are negligible in the present DeFi attacks. The constraints are enforced on the input parameters and output states to ensure that the optimizer yields valid parameters.

We define the balance state function $B(E; X; S)$ to denote the balance of currency X held by entity E at a given state S and require Equation 1 to hold.

$$\forall (E, X, S), B(E; X; S) \geq 0 \quad (1)$$

The mathematical DeFi models applied in this work are detailed in Appendix C.

Our parametrized optimizer is designed to solve the optimal parameters that maximizes the revenue given an on-chain state, DeFi models and attack vector. An attack vector specifies the execution order of different endpoints across various DeFi platforms, depending on which we formalize a unidirectional chain of transition functions (cf. Equation 2).

$$S_i = T_i(S_{i-1}; p_i) \quad (2)$$

By nesting transition functions, we can obtain the cumulative state transition functions $ACC_i(S_0; p_{1:i})$ that satisfies Equation 3, where $p_{1:i} = (p_1, ..., p_i)$.

$$\begin{aligned} S_i &= T_i(S_{i-1}; p_i) = T_i(T_{i-1}(S_{i-2}; p_{i-1}); p_i) \\ &= T_i(T_{i-1}(...T_1(S_0, p_1)...; p_{i-1}); p_i) = ACC_i(S_0; p_{1:i}) \end{aligned} \quad (3)$$

Therefore the constraints generated in each step can be expressed as Equation 4.

$$C_i(S_i; p_i) \iff C_i(ACC_i(S_0; p_{1:i}); p_i) \quad (4)$$

We assume an attack vector composed of N transition functions. The objective function can be calculated from the initial state S_0 and the final state S_N (e.g., the increase of the adversarial balance).

$$O(S_0; S_N) \iff O(S_0; ACC(S_0; p_{1:N})) \quad (5)$$

Given the initial state S_0, we formulate an attack vector into a constrained optimization problem with respect to all the parameters $p_{1:N}$ (cf. Equation 6).

$$\begin{aligned} &\text{maximize } O(S_0; ACC(S_0; p_{1:N})) \\ &\text{s.t. } C_i(ACC_i(S_0; p_{1:i}); p_i) \quad \forall i \in [1, N] \end{aligned} \quad (6)$$

5 Evaluation

In the following, we evaluate our parametrized optimization framework on the existing attacks described in Section 3. We adopt the Sequential Least Squares Programming (SLSQP) algorithm from SciPy[2]

to solve the constructed optimization problems. Our framework is evaluated on a Ubuntu 18.04.2 machine
with 16 CPU cores and 32 GB RAM.

5.1 Optimizing the Pump Attack and Arbitrage

We first optimize the pump attack and arbitrage. Figure 5 summarizes the notations and the on-chain state when the attack was executed (i.e., S0). We use these blockchain records as the initial state in our evaluation. X and Y denote ETH and WBTC respectively. In the PA&A attack vector, we intend to tune the following two parameters, (i) p1: the amount of X collateralized to borrow Y (cf. step 2 and 3 in Figure 3) and (ii) p2: the amount of X collateralized to short Y (cf. step 4 in Figure 3). Following the methodology specified in Section 4.2, we derive the optimization problem and the corresponding constraints, which are presented in Figure 6. We detail the deriving procedure in Appendix D. We remark that there are five linear constraints and only one nonlinear constraint, which implies that the optimization can be solved efficiently.

We repeated our experiment for 1, 000 times, the optimizer spent 6.1ms on average converging to the optimum. The optimizer provides a maximum revenue of 2, 778.94 ETH when setting the parameters (p1; p2) to (2, 470.08; 1, 456.23), while in the original attack the parameters (5, 500; 1, 300) only yield 1, 171.70 ETH. Due to the ignorance of trading fees and precision differences, there is a minor discrepancy between the original attack revenue calculated with our model and the real revenue which is 1, 193.69 ETH (cf. Section 3). This is

2
https://www.scipy.org/. We use the minimize function in the optimize package.

Description	Variable	Value
Maximum Amount of ETH to flash loan	vX	10, 000
Collateral Factor	cf	0.75
Collateralized Borrowing Exchange Rate	er	36.48
Maximum Amount of WBTC to Borrow	zY	155.70
Uniswap Reserved ETH	uX(S0)	2, 817.77
Uniswap Reserved WBTC	uY(S0)	77.08
Over Collateral Ratio	ocr	1.153
Leverage	`	5
Maximum Amount of ETH to leverage	wX	4, 858.74
Market Price of WBTC	pm	39.08

Fig. 5: Initial on-chain states of the
PA&A.

Objective
uX(S0) + p2×`
ocr − uX(S4) − p2 −
p1×cf×pm
er function
Constraints
p1 ≥ 0, p2 ≥ 0
vX − p0 − p1 ≥ 0
zY −
p1×cf
er ≥ 0
wX + p2 −
p2×`
ocr ≥ 0
B0 + uX(S0) + p2×`
ocr − uX(S4) − p1 − p2 ≥ 0

Fig. 6: Generated PA&A constraints.
uX(S4) is nonlinear with respect to p1
and p2.

a 829.5k USD gain over the attack that took place, using the price of ETH at that time. We experimentally validate the optimal PA&A parameters by forking the

Ethereum blockchain with Ganache [6] at block 9484687 (one block prior to the original attack transaction). We then implement the pump attack and arbitrage in solidity v0.6.3. The revenue of the attack is divided into two parts: part one from the flash loan transaction, and part two which is a followup operation in later blocks (cf. Section 3) to repay the loan. For simplicity, we chose to only validate the first part, abiding by the following methodology: (i) We apply the parameter output of the parametrized optimizer, i.e., $(p_1; p_2) = (2,470.08; 1,456.23)$ to the adversarial validation smart contract. (ii) Note that our model is an approximation of the real blockchain transition functions. Hence, due to the inaccuracy of our model, we cannot directly use the precise model output, but instead use the model output as a guide for a manual, trial, and error search. We find 1, 344 is the maximum value of p_2 that allows the successful adversarial trade. (iii) Given the new p_2 constraint, our optimizer outputs the new optimal parameters (2, 404; 1, 344). (iv) Our optimal adversarial trade yields a profit of 1, 958.01 ETH on part one (as opposed to 71.41 ETH) and consumes a total of 3.3M gas.

5.2 Optimizing the Oracle Manipulation Attack

In the oracle manipulation attack, we denote X as ETH and Y as sUSD, while the initial state variables are presented in Figure 7. We assume that A owns zero balance of X or Y. There are three parameters to optimize in this attack, (i) p_1: the amount of X used to swap for Y in step 2); (ii) p_2: the amount of X used to swap for Y in step 3); (iii) p_3: the amount of X used to exchange for Y in step 4). We summarize the produced optimization problem and its constraints in Figure 8, of which five constraints are linear and the other two are nonlinear. We present the details in Appendix E.

We execute our optimizer 1, 000 times, resulting in an average convergence time of 12.9ms. The optimizer discovers that setting $(p_1; p_2; p_3)$ to (898.58;546.80; 3, 517.86) results in 6, 323.93 ETH in profit for the adversary. This results in a gain of 1.1M USD instead of 634.9k USD. We fork the Ethereum blockchain with Ganache at block 9504626 (one block prior to the original adversarial transac-

Description	Variable	Value
Maximum ETH to flash loan	v_X	7, 500
Uniswap Reserved ETH	$u_X(S_0)$	879.757
Uniswap Reserved sUSD	$u_Y(S_0)$	243, 441.12
Liquidity Rate	lr	0.00252
Min. sUSD Price of Kyber Reserve	$minP$	0.0037
Max. sUSD Price of Kyber Reserve	$maxP$	0.0148
Inventory of ETH in Kyber Reserve	$k_X(S_0)$	0.90658
Market Price of sUSD	p_m	0.00372719
Max. sUSD to Buy	$maxY$	943, 837.59
Collateral Factor	cf	0.667
Max. ETH to Borrow	z_Y	11, 086.29

Fig. 7: Initial on-chain states of the oracle manipulation attack.

Objective function	$B(A; Y; S_4) \times cf \times P_Y(M; S_2) - p_1 - p_2 - p_3$
Constraints	$p_1 \geq 0,\ p_2 \geq 0,\ p_3 \geq 0$
	$v_X - p_1 - p_2 - p_3 \geq 0$
	$maxP - minP \times e^{lr\times(k_X(S_0)+p_2)} \geq 0$
	$maxY - \frac{p_3}{p_m} \geq 0$
	$z_Y - B(A; Y; S_4) \times cf \times P_Y(M; S_2) \geq 0$

Fig. 8: Constraints generated for the oracle manipulation attack.

B(A; Y; S4), PY(M; S2) are nonlinear
components with respect to p1, p2,
p3.
tion) and again implement the attack in solidity v0.6.3. We validate that executing the adversarial smart contract with parameters (p1; p2; p3) = (898.58; 546.8; 3, 517.86) renders a profit of 6, 262.28 ETH, while the original attack parameters yield 2, 381.41 ETH. The attack consumes 11.3M gas (which fits within the current block gas limit of 12.5M gas, but wouldn't have fit in the block gas limit of February 2020). By analyzing the adversarial validation contract, we find that 460 is the maximum value of p2 which reduces the gas consumption below 10M gas. Similar to Section 5.1, we add the new constraint to the optimizer, which then gives the optimal parameters (714.3; 460; 3, 517.86). The augmented validation contract renders a profit of 4, 167.01 ETH and consumes 9.6M gas.

6 Implications of Transaction Atomicity

In an atomic blockchain transaction, actions can be executed collectively in sequence, or fail collectively. Technically, operating DeFi actions in an atomic transaction is equivalent to acquiring a lock on all involved financial markets to ensure no other market agent can modify market states intermediately, and releasing the lock after executing all actions in their sequence.

To quantify objectively the impact of transaction atomicity (specifically, how the transaction atomicity impacts arbitrage profit), we proceed with the following methodology. We consider the arbitrages that involve two trades TA and TB to empirically compare the atomic and non-atomic arbitrages (cf. Figure 9). We define the atomic and non-atomic arbitrage profit as follows.

Atomic arbitrage profit (aarb): is defined as the gain of two atomically executed arbitrage trades TA and TB on exchange A and B.

Non-atomic arbitrage profit (naarb): is defined as the arbitrage gain, if TA executes first, and TB's execution follows after i intermediary transactions.

Conceptually, a non-atomic arbitrage requires the arbitrageur to lock assets for a short time (order of seconds/minutes). Those assets are exposed to price volatility. The arbitrageur can at times realize a gain, if the asset increases in value, but equally has the risk of losing value. A trader engaging in atomic arbitrage is not exposed to this volatility risk, which we denote as holding value.

Asset Price
Atomic arbitrage
profit ( ) Time / Number of transactions executed
purchasing price
Non-atomic arbitrage
profit ( )
selling price (non-atomic)
Average price
Holding
value
selling price (atomic)

Fig. 9: On the impact of transaction atomicity on arbitrage. The arbitrageur submits the first trade TA, which aims to purchase an asset at a "cheaper" prices (•) and sell the asset on another exchange at a "higher" price (•). In a non-atomic environment, TB is not immediately executed after TA. The holding value is the in-/decrease in price when holding the asset between TA and TB.

0
50
100
150 DAI/ETH
0 1000 2000 3000 4000 5000
0

1
2 MKR/ETH
Number of intermediary transactions between TA and TB
Profit difference (USD)

Fig. 10: Simulated impact of intermediary transactions on arbitrage revenue. The average reward decreases by 123.49 ± 1375.32 USD and 1.77 ± 10.59 USD for the DAI/ETH and MKR/ETH markets respectively, at 350 USD/ETH, for 5, 000 intermediary transactions. Note that we present the 95% bootstrap confidence interval of mean [23] for readability.

Holding value (hv): is defined as the change in the averaged price of the given asset pair on the two exchanges, which represents the asset value change during the non-atomic execution period.

We introduce holding value to neutralize the price volatility and can hence objectively quantify the financial advantage of atomic arbitrage. Given these variables, we define the profit difference in Equation 7.

$$\text{profit difference} = aarb - (naarb - hv) \quad (7)$$

We simulate atomic and non-atomic based on 6, 398, 992 transactions we collect from the Ethereum mainnet (from block 10276783 onwards). We insert 0 - 5, 000 blockchain transactions following the trade transaction TA. Note that 0 intermediary transaction is equivalent to the atomic arbitrage. The insertion order follows the original execution order of these transactions, some of which may be irrelevant to the arbitrage. We present the simulated profit difference in Figure 10. We observe that the average profit difference reaches 123.49 ± 1375.32 USD and 1.77 ± 10.59 USD for the DAI/ETH and MKR/ETH markets respectively when the number of intermediary transactions increases to 5, 000.

7 Discussion

The current generation of DeFi had developed organically, without much scrutiny when it comes to financial security; it, therefore, presents an interesting security challenge to confront. DeFi, on the one hand, welcomes innovation and the

advent of new protocols, such as MakerDAO, Compound, and Uniswap. On the other hand, despite a great deal of effort spent on trying to secure smart contacts [38,31,21,50,48], and to avoid various forms of market manipulation, etc. [41,42,15], there has been little-to-no effort to secure entire protocols. As such, DeFi protocols join the ecosystem, which leads to both exploits against protocols themselves as well as multi-step attacks that utilize several protocols such as the two attacks in Section 3. In a certain poignant way, this highlights the fact the DeFi, lacking a central authority that would enforce a strong security posture, is ultimately vulnerable to a multitude of attacks by design. Flash loans are merely a mechanism that accelerates these attacks. It does so by requiring no collateral (except for the minor gas costs), which is impossible in the traditional fiance due to regulations. In a certain way, flash loans democratize the attack, opening this strategy to the masses. As we anticipate in the earlier version of this paper, following the two analyzed attacks, economic attacks facilitated by flash loans become increasingly frequent, which have incurred a total loss of over 100M USD [7].

Determining what is malicious: An interesting question remains whether we can qualify the use of flash loans, as clearly malicious (or clearly benign). We believe this is a difficult question to answer and prefer to withhold the value judgment. The two attacks in Section 3 are clearly malicious: the PA&A involves manipulating the WBTC/ETH price on Uniswap; the oracle manipulation attack involves price oracle by manipulatively lowering the price of ETH against sUSD on Kyber. However, the arbitrage mechanism, in general, is not malicious — it is merely a consequence of the decentralized nature of the DeFi ecosystem, where many exchanges and DEXs are allowed to exist without much coordination with each other. As such, arbitrage will continue to exist as a phenomenon, with good and bad consequences. Despite the lack of absolute distinction between flash loan attacks and legitimate applications of flash loans, we attempt to summarize two characteristics that appear to apply to malicious flash loan attacks: (i) the attacker benefits from a distorted state created artificially in the flash loan transaction (e.g., the pumped market in the PA&A and the manipulated oracle price); (ii) the attacker's profit causes the loss of other market participants (e.g., the liquidity providers in the two analyzed attacks in Section 3). We extend our discussion in Appendix F.

8 Conclusion

This paper presents an exploration of the impact of transaction atomicity and the flash loan mechanism on the Ethereum network. While proposed as a clever mechanism within DeFi, flash loans are starting to be used as financial attack vectors to effectively pull money in the form of cryptocurrency out of DeFi. In this paper, we analyze existing flash loan-based attacks in detail and then proceed to propose optimizations that significantly improve the ROI of these attacks. Specifically, we are able to show how two previously executed attacks can be "boosted" to result in a revenue of 829.5k USD and 1.1M USD, respectively, which is a boost of 2.37× and 1.73×, respectively.

Acknowledgments

We thank the anonymous reviewers and Johannes Krupp for providing valuable comments and helpful feedback that significantly strengthened the paper. We are moreover grateful to the Lucerne University of Applied Sciences and Arts for generously supporting Kaihua Qin's Ph.D.

References

1. Aavewatch - live protocol stats! https://aavewatch.now.sh/
2. Bti market surveillance report - september 2019 - bti. https://www.bti.live/bti-september-2019-wash-trade-report/, (Accessed on 02/24/2020)
3. bzx - a protocol for tokenized margin trading and lending. https://bzx.network/
4. Compound. https://compound.finance/
5. Consensys/0x-review: Security review of 0x smart contracts. https://github.com/ConsenSys/0x-review
6. Ganache — truffle suite. https://www.trufflesuite.com/ganache
7. Home — prevent flash loan attacks. https://preventflashloanattacks.com/
8. marbleprotocol/flash-lending: Flash lending smart contracts. https://github.com/marbleprotocol/flash-lending
9. Report of investigation pursuant to section 21(a) of the securities exchange act of 1934: The dao. https://www.sec.gov/litigation/investreport/34-81207.pdf
10. Slippage definition & example. https://www.investopedia.com/terms/s/slippage.asp
11. Uniswap. https://uniswap.org/
12. U.s. corporate debt soars to record $10.5 trillion - marketwatch. https://www.marketwatch.com/story/u-s-corporate-debt-soars-to-record-10-5-trillion-11598921886#:˜:text=U.S.%20corporations%20now%20owe%20a,new%20BofA%20Global%20Research%20report.
13. Aave: Aave Protocol. https://github.com/aave/aave-protocol (2020)
14. Amani, S., B´egel, M., Bortin, M., Staples, M.: Towards verifying ethereum smart contract bytecode in isabelle/hol. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 66–77 (2018)
15. Bentov, I., Ji, Y., Zhang, F., Li, Y., Zhao, X., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware. Conference on Computer and Communications Security (2019)
16. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., et al.:

Formal verification of smart contracts: Short paper. In: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. pp. 91–96 (2016)
17. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J.A., Felten, E.W.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: Security and Privacy (SP), 2015 IEEE Symposium on. pp. 104–121. IEEE (2015)
18. Breidenbach, L., Daian, P., Tram`er, F., Juels, A.: Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1335–1352 (2018)
19. Brent, L., Jurisevic, A., Kong, M., Liu, E., Gauthier, F., Gramoli, V., Holz, R., Scholz, B.: Vandal: A scalable security analysis framework for smart contracts. arXiv preprint arXiv:1809.03981 (2018)

20. CoinMarketCap: Bitcoin market capitalization (2019)
21. Crytic: Echidna: Ethereum fuzz testing framework, https://github.com/crytic/echidna
22. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. IEEE Security and Privacy 2020 (2020)
23. DiCiccio, T.J., Efron, B.: Bootstrap confidence intervals. Statistical science pp. 189–212 (1996)
24. dYdX: dYdX. https://dydx.exchange/ (2020)
25. Eskandari, S., Moosavi, S., Clark, J.: Sok: Transparent dishonesty: front-running attacks on blockchain. In: International Conference on Financial Cryptography and Data Security. pp. 170–189. Springer (2019)
26. Gandal, N., Hamrick, J., Moore, T., Oberman, T.: Price manipulation in the Bitcoin ecosystem. Journal of Monetary Economics 95(4), 86–96 (2018). https://doi.org/10.1016/j.jmoneco.2017.12.004, https://linkinghub.elsevier.com/retrieve/pii/S0304393217301666
27. Grishchenko, I., Maffei, M., Schneidewind, C.: A semantic framework for the security analysis of ethereum smart contracts. In: International Conference on Principles of Security and Trust. pp. 243–269. Springer (2018)
28. Hamrick, J., Rouhi, F., Mukherjee, A., Feder, A., Gandal, N., Moore, T., Vasek, M.: The economics of cryptocurrency pump and dump schemes (2018)
29. Hildenbrandt, E., Saxena, M., Zhu, X., Rodrigues, N., Daian, P., Guth, D., Rosu, G.: Kevm: A complete semantics of the ethereum virtual machine. Tech. rep. (2017)
30. Hirai, Y.: Defining the ethereum virtual machine for interactive theorem provers. In: International Conference on Financial Cryptography and Data Security. pp. 520–535. Springer (2017)
31. Jiang, B., Liu, Y., Chan, W.: Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 259–269. ACM (2018)
32. Kalodner, H.A., Carlsten, M., Ellenbogen, P., Bonneau, J., Narayanan, A.: An empirical study of namecoin and lessons for decentralized namespace design. In: WEIS. Citeseer (2015)
33. Kamps, J., Kleinberg, B.: To the moon: defining and detecting cryptocurrency pump-and-dumps. Crime Science 7 (12 2018). https://doi.org/10.1186/s40163-018-0093-5
34. Kyber: Kyber. https://kyber.network/ (2020)
35. Labs, A.: Idex: A real-time and high-throughput ethereum smart contract exchange. Tech. rep. (January 2019)

36. Liu, C., Liu, H., Cao, Z., Chen, Z., Chen, B., Roscoe, B.: Reguard: finding reentrancy bugs in smart contracts. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). pp. 65–68. IEEE (2018)
37. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making Smart Contracts Smarter. Proceedings of the ACM SIGSAC Conference on Computer and Communications Security pp. 254–269 (2016). https://doi.org/10.1145/2976749.2978309, http://dl.acm.org/citation.cfm?doid=2976749.2978309
38. Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A.: Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 254–269 (2016)
39. Maker: Makerdao. https://makerdao.com/en/ (2019)

40. MakerDao: Intro to the oasisdex protocol (September 2019), accessed 12 November, 2019, https://github.com/makerdao/developerguides/blob/master/Oasis/intro-to-oasis/intro-to-oasis-maker-otc.md
41. Mavroudis, V.: Market Manipulation as a Security Problem. arXiv preprint arXiv:1903.12458 (2019)
42. Mavroudis, V., Melton, H.: Libra: Fair Order-Matching for Electronic Financial Exchanges. arXiv preprint arXiv:1910.00321 (2019)
43. Mueller, B.: Mythril-reversing and bug hunting framework for the ethereum blockchain (2017)
44. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
45. Nikolić, I., Kolluri, A., Sergey, I., Saxena, P., Hobor, A.: Finding the greedy, prodigal, and suicidal contracts at scale. In: Proceedings of the 34th Annual Computer Security Applications Conference. pp. 653–663 (2018)
46. Shleifer, A., Vishny, R.W.: The limits of arbitrage. The Journal of finance 52(1), 35–55 (1997)
47. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., Alexandrov, Y.: Smartcheck: Static analysis of ethereum smart contracts. In: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain. pp. 9–16 (2018)
48. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F., Vechev, M.: Securify: Practical security analysis of smart contracts. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 67–82. ACM (2018)
49. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum Project Yellow Paper (2014)
50. Wüstholz, V., Christakis, M.: Harvey: A greybox fuzzer for smart contracts. arXiv:1905.06944 (2019)
51. Xu, J., Livshits, B.: The anatomy of a cryptocurrency pump-and-dump scheme. In: Proceedings of the Usenix Security Symposium (Aug 2019)
52. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-frequency trading on decentralized on-chain exchanges. arXiv preprint arXiv:2009.14021 (2020)

A Classifying Flash Loan Use Cases

In Figure 11, we present the DeFi platforms that use a total of 5, 615 Aave flash loan transactions[3] between the 8th of January, 2020 and the 20th of September, 2020. We find that more than 30% of the flash loans are interacting with Kyber, MakerDAO, and Uniswap. Compound and MakerDAO accumulate 433.81M USD flash loans which occupy 90% of the total flash loan amount.

On average, a flash transaction uses 1.43M gas, while the most complex one consumes 6.3M gas.

B Flash Loan Use Cases

B.1 Wash Trading
The trading volume of an asset is a metric indicating its popularity. The most popular assets therefore are supposed to be traded the most — e.g., Bitcoin to

3 We collect in total 5, 616 flash loans with one transaction performing two flash loans.

DeFi Platforms	Transactions	Amount (USD)	Mean gas
Kyber, MakerDAO, Uniswap	1826	6.91M	1.64M±465.69k
Kyber, MakerDAO, OasisDEX, Uniswap	817	6.75M	1.38M±324.09k
Compound, MakerDAO	320	433.81M	1.49M±333.16k
0x, Kyber, MakerDAO, Uniswap	231	888.17k	1.76M±595.93k
Compound	228	5.98M	1.22M±501.97k
0x, Compound, Curve, MakerDAO	168	115.82k	1.31M±603.77k
0x, Kyber, MakerDAO, OasisDEX, Uniswap	153	2.12M	1.80M±432.11k
Compound, Curve	143	1.75M	2.06M±281.84k
MakerDAO	122	8.86M	934.39k±230.73k
0x, Compound, Curve	103	103.00k	1.27M±249.15k
Compound, MakerDAO, Uniswap	93	120.18k	1.31M±314.83k
Kyber, Uniswap	92	80.54k	985.68k±711.43k
0x, MakerDAO	87	1.70M	1.18M±120.70k
Bancor, Compound, Kyber, MakerDAO, Uniswap	77	8.45k	2.14M±705.27k
0x, Uniswap	68	32.97k	694.76k±129.58k
MakerDAO, Uniswap	68	40.83k	1.01M±254.51k
0x, OasisDEX	57	23.79k	716.40k±132.51k
Kyber, MakerDAO	53	437.65k	2.06M±641.44k
0x, Kyber, MakerDAO	42	639.36k	1.78M±352.44k
Compound, Kyber, MakerDAO, Uniswap	37	185.30k	2.72M±740.48k
0x, Kyber, Uniswap	30	23.81k	1.30M±285.27k
Bancor, Compound, Kyber, MakerDAO, OasisDEX, Uniswap	30	13.46k	2.05M±666.87k
Compound, Uniswap	29	45.58k	1.14M±293.59k
MakerDAO, OasisDEX	27	114.31k	823.62k±139.90k
Uniswap	25	56.34k	672.12k±404.84k
0x, Compound, MakerDAO	22	88.57k	1.81M±274.23k
Kyber	21	41.73k	803.54k±207.92k
Compound, Curve, MakerDAO	20	3.10M	1.93M±665.87k
Compound, Kyber, Uniswap	13	18.04k	1.82M±430.46k
0x, Kyber, OasisDEX, Uniswap	13	11.99k	1.42M±291.46k
0x, OasisDEX, Uniswap	12	15.68k	789.94k±193.06k
Compound, Kyber, MakerDAO, OasisDEX, Uniswap	11	63.12k	3.20M±893.03k
0x	9	8.48k	590.03k±111.78k
Kyber, OasisDEX, Uniswap	8	42.55k	858.12k±255.44k
0x, Compound, Curve, Kyber, MakerDAO, Uniswap	7	6.98k	1.87M±301.64k
Kyber, MakerDAO, OasisDEX	6	130.31k	1.84M±512.57k
0x, Compound, MakerDAO, Uniswap	5	2.64k	2.02M±149.59k
Bancor, Compound, Kyber, Uniswap	5	564.52	3.83M±1.50M
Others	537	6.87M	670.22k± 568.05k
Total	5, 615	481.20M	1.43M± 605.97k

Fig. 11: Classifying the usage of flash loans in the wild, based on an analysis of transactions between the 8th of January, 2020 and the 20th of September, 2020 on Aave [13]. Others include the platform combinations that appear less than five times and the ones of which the owner platforms are unknown to us. The total amount is calculated at the price – DAI ($1); ETH ($350); USDC ($1); BAT ($0.2); WBTC ($10, 000); ZRX ($0.3); MKR ($500); LINK ($10); USDT ($1); REP ($15), KNC ($1.5), LEND ($0.5), sUSD ($1).

date enjoys the highest trading volume (reported up to 50T USD per day) of all cryptocurrencies.

Malicious exchanges or traders can mislead other traders by artificially inflating the trading volume of an asset. In September 2019, 73 out of the top 100 exchanges on Coinmarketcap [20] were wash trading over 90% of their volumes [2]. In centralized exchanges, operators can easily and freely create fake trades in the backend, while decentralized exchanges settle trades on-chain. Wash trading on DEX thus requires wash traders to hold and use real assets. Flash loans can remove this “obstacle” and wash trading costs are then reduced to the flash loan interest, trading fees, and (blockchain) transaction fees, e.g., gas. A wash trading endeavor to increase the 24-hour volume by 50% on the ETH/DAI market of Uniswap would for instance cost about 1, 298 USD (cf. Figure 12). We visualize in Figure 12 the required cost to create fake volumes in two Uniswap markets. At the time of writing, the transaction fee amounts to 0.01 USD, the flash loan interests range from a constant 1 Wei (on dYdX) to 0.09% (on Aave), and exchange fees are about 0.3% (on Uniswap).

0% 20% 40% 60% 80% 100%
Growth Rate of Volumes
$0
$500
$1000
$1500
$2000
$2500
$3000
Cost (USD)
ETH/DAI - Interest: 0.09%
ETH/DAI - Interest: 1Wei
ETH/WBTC - Interest: 0.09%
ETH/WBTC - Interest: 1Wei

Fig. 12: Wash trading cost on two Uniswap markets with flash loans costing 0.09% (Aave) and a constant of 1 Wei (dYdX) respectively. The 24-hour volumes of ETH/DAI and ETH/WBTC market were 963, 786 USD and 67, 690 USD respectively (1st of March, 2020).

Wash trading example: On March 2nd, 2020, a flash loan of 0.01 ETH borrowed from dYdX performed two back-and-forth trades (first converted 0.01 ETH to 122.1898 LOOM and then converted 122.1898 LOOM back to 0.0099 ETH) on Uniswap ETH/LOOM market (cf. 0xf65b384ebe2b7bf1e7bd06adf0daac0413defe ed42fd2cc72a75385a200e1544). The 24-hour trading volume of the ETH/LOOM market increased by 25.8% (from 17.71 USD to 22.28 USD) as a result of the two trades.

B.2 Collateral Swapping

We classify DeFi platforms that rely on users providing cryptocurrencies [24,13,39] as follows: (i) a DeFi system where a new asset is minted and backed-up with user-provided collateral (e.g., MakerDAO’s DAI or SAI [39]) and (ii) a DeFi system where long-term loans are offered and assets are aggregated within liquidity pools (e.g., margin trading [3] or long term loans [13]). Once a collateral position is opened, DeFi platforms store the collateral assets in a vault until the new/borrowed asset are destroyed/returned. Because cryptocurrency prices fluctuate, this asset lock-in bears a currency risk. With flash loans, it is possible


```
c o n t r a c t F l a shM in t ab l eC o in i s ERC20 { [ . . . ]
funct ion f l a s hM i n t ( u i n t 2 5 6 amount ) {
// m i n t c o i n s a n d t r a n s f e r t h e m
mint (msg . s end e r , amount ) ;
// b o r r o w e r u s e s t h e l o a n
Borrower (msg . s e n d e r ) . e x e c u t e ( amount ) ;
// r e v e r t s if n o t h a v e e n o u g h to b u r n
```

```
burn (msg . s end e r , amount ) ;
}}
```

Fig. 13: Flash mint example.

to replace the collateral asset with another asset, even if a user does not possess sufficient funds to destroy/return the new/borrowed asset. A user can close an existing collateral position with borrowed funds, and then immediately open a new collateral position using a different asset.

Collateral swapping example: On February 20th, 2020, a flash loan borrowed 20.00 DAI (from Aave) to perform a collateral swap (on MakerDAO), cf. 0x5d5bbfe0b666631916adb8a56821b204d97e75e2a852945ac7396a82e207e0ca. Before this transaction, the transaction sender used 0.18 WETH as collateral for instantiating 20.00 DAI (on MakerDAO). The transaction sender first withdraws all WETH using the 20.00 DAI flash loan, then converts 0.18 WETH for 178.08 BAT (using Uniswap). Finally the user creates 20.03 DAI using BAT as collateral, and pays back 20.02 DAI (with a fee to Aave). This transaction converts the collateral from WETH to BAT and the user gained 0.01 DAI, with an estimated gas fee of 0.86 USD.

B.3 Flash Minting

Cryptocurrency assets are commonly known as either inflationary (further units of an asset can be mined) or deflationary (the total number of units of an asset are finite). Flash minting is an idea to allow an instantaneous minting of an arbitrary amount of an asset — the newly-mined units exist only during one transaction. It is yet unclear where this idea might be applicable to, the minted assets could momentarily increase liquidity.

Flash minting example: A flash mint function (cf. Figure 13) can be integrated into an ERC20 token, to mint an arbitrary number of coins within a transaction only. Before the transaction terminates, the minted coins will be burned. If the available amount of coins to be burned by the end of the transaction is less than those that were minted, the transaction is reverted (i.e., not executed). An example ERC20 flash minting code could take the following form (cf. 0x09b4c8200f0cb51e6d44a1974a1bc07336b9f47f):

C DeFi Models

In the following, we detail the quantitative DeFi models applied in this work. Note that we do not include all the states involved in the DeFi attacks but only those relevant to the constrained optimization.

Flash loan: We assume a flash loan platform F with z_X amount of asset X, which the adversary A can borrow. The required interest to borrow b of X is represented by $interest(b)$.

State: In a flash loan, the state is represented by the balance of A, i.e., $B(A; X; S)$.

Transitions: We define the transition functions of Loan in Equation 8 and Repay in Equation 9, where the parameter b_X denotes the loaned amount.

$$B(A; X; S') = B(A; X; S) + b_X \quad \text{s.t. } z_X - b_X \geq 0 \tag{8}$$

$$B(A; X; S') = B(A; X; S) - b_X - interest(b_X) \quad \text{s.t. } B(A; X; S) - b_X - interest(b_X) \geq 0 \tag{9}$$

Fixed price trading: We define the endpoint SellXforY that allows the adversary A to trade q_X amount of X for Y at a fixed price p_m. max_Y is the maximum amount of Y available for trading.

State: We consider the following state variables:

- Balance of asset X held by A: $B(A; X; S)$
- Balance of asset Y held by A: $B(A; Y; S)$

Transitions: Transition functions of SellXforY are defined in Equation 10.

$$
\begin{aligned}
B(A; X; S') &= B(A; X; S) - qX \\
B(A; Y; S') &= B(A; Y; S) + \frac{qX}{pm} \\
\text{s.t. } & B(A; X; S) - qX \geq 0 \\
& maxY - \frac{qX}{pm} \geq 0
\end{aligned}
\tag{10}
$$

Constant product automated market maker: The constant product AMM is with a market share of 77% among the AMM DEX, the most common AMM model in the current DeFi ecosystem [11]. We denote by M an AMM instance with trading pair X/Y and exchange fee rate f.

State: We consider the following states variables that can be modified in an AMM state transition.

– Amount of X in AMM liquidity pool: $uX(S)$, which equals to $B(M; X; S)$
– Amount of Y in AMM liquidity pool: $uY(S)$, which equals to $B(M; Y; S)$
– Balance of X held by A: $B(A; X; S)$
– Balance of Y held by A: $B(A; Y; S)$

Transitions: Among the endpoints of M, we focus on SwapXforY and SwapYforX, which are the relevant endpoints for the DeFi attacks discussed within this work. pX is a parameter that represents the amount of X the adversary intends to trade.

A inputs pX amount of X in AMM liquidity pool and receives oY amount of Y as output. The constant product rule [11] requires that Equation 11 holds.

$$uX(S) \times uY(S) = (uX(S) + (1 - f)pX) \times (uY(S) - oY) \tag{11}$$

We define the transition functions and constraints of SwapXforY in Equation 12 (analogously for SwapYforX).

$$
\begin{aligned}
B(A; X; S') &= B(A; X; S) - pX \\
B(A; Y; S') &= B(A; Y; S) + oY \\
uX(S') &= uX(S) + pX \\
uY(S') &= uY(S) - oY \\
\text{where } oY &= \frac{pX \times (1 - f) \times uY(S)}{uX(S) + pX \times (1 - f)} \\
\text{s.t. } & B(M; X; S) - pX \geq 0
\end{aligned}
\tag{12}
$$

Because an AMM DEX M transparently exposes all price transitions onchain, it can be used as a price oracle by the other DeFi platforms. The price of Y with respect to X given by M at state S is defined in Equation 13.

$$pY(M; S) = \frac{uX(S)}{uY(S)} \tag{13}$$

Automated price reserve: The automated price reserve is another type of AMM that automatically calculates the exchange price depending on the assets held in inventory. We denote a reserve holding the asset pair X/Y with R. A minimum price $minP$ and a maximum price $maxP$ is set when initiating R. R relies on a liquidity ratio parameter lr to calculate the asset price. We assume that R holds $k_X(S)$ amount of X at state S. We define the price of Y in Equation 14.

$$P_Y(R; S) = minP \times e^{lr \times k_X(S)} \quad (14)$$

The endpoint ConvertXtoY provided by R allows the adversary A to exchange X for Y.

State: We consider the following state variables:

– The inventory of X in the reserve: $k_X(S)$, which equals to $B(R; X; S)$
– Balance of X held by A: $B(A; X; S)$
– Balance of Y held by A: $B(A; Y; S)$

Transitions: We denote as h_X the amount of X that A inputs in the exchange to trade against Y. The exchange output amount of Y is calculated by the following formulation.

$$j_Y = \frac{e^{-lr \times h_X} - 1}{lr \times P_Y(R; S)}$$

We define the transition functions within Equation 15.

$$\begin{aligned}
k_X(S') &= k_X(S) + h_X \\
B(A; X; S') &= B(A; X; S) - h_X \\
B(A; Y; S') &= B(A; Y; S) + j_Y \\
\text{where } j_Y &= \frac{e^{-lr \times h_X} - 1}{lr \times P_Y(R; S)} \\
\text{s.t. } & B(A; X; S) - h_X \geq 0 \\
& P_Y(R; S') - minP \geq 0 \\
& maxP - P_Y(R; S') \geq 0
\end{aligned} \quad (15)$$

Collateralized lending & borrowing: We consider a collateralized lending platform L, which provides the CollateralizedBorrow endpoint that requires the user to collateralize an asset X with a collateral factor cf (s.t. $0 < cf < 1$) and borrows another asset Y at an exchange rate er. The collateral factor determines the upper limit that a user can borrow. For example, if the collateral factor is 0.75, a user is allowed to borrow up to 75% of the value of the collateral. The exchange rate is for example determined by an outsourced price oracle. z_Y denotes the maximum amount of Y available for borrowing.

State: We hence consider the following state variables and ignore the balance changes of L for simplicity.

– Balance of asset X held by A: $B(A; X; S)$
– Balance of asset Y held by A: $B(A; Y; S)$

Transitions: The parameter c_X represents the amount of asset X that A aims to collateralize. Although A is allowed to borrow less than his collateral would allow for, we assume that A makes use the entirety of his collateral. Equation 16 shows the transition functions of CollateralizedBorrow.

$$
\begin{aligned}
&B(A; X; S') = B(A; X; S) - c_X \\
&B(A; Y; S') = B(A; Y; S) + b_Y \\
&\text{where } b_Y = \frac{c_X \times cf}{er} \\
&\text{s.t. } B(A; X; S') - c_X \geq 0;\ z_Y - b_Y \geq 0
\end{aligned}
\tag{16}
$$

A can retrieve its collateral by repaying the borrowed asset through the endpoint CollateralizedRepay. We show the transition functions in Equation 17 and for simplicity ignore the loan interest fee.

$$
\begin{aligned}
&B(A; X; S') = B(A; X; S) + c_X \\
&B(A; Y; S') = B(A; Y; S) - b_Y \\
&\text{s.t. } B(A; Y; S) - b_Y \geq 0
\end{aligned}
\tag{17}
$$

Margin trading: A margin trading platform T allows the adversary A to short-/long an asset Y by collateralizing asset X at a leverage ℓ, where $\ell \geq 1$.

We focus on the MarginShort endpoint which is relevant to the discussed DeFi attack in this work. We assume A shorts Y with respect to X on F. The parameter d_X denotes the amount of X that A collateralizes upfront to open the margin. w_X represents the amount of X held by F that is available for the short margin. A is required to over-collateralize at a rate of ocr in a margin trade. In our model, when a short margin (short Y with respect to X) is opened, F performs a trade on external X/Y markets (e.g., Uniswap) to convert the leveraged X to Y. The traded Y is locked until the margin is closed or liquidated.

State: In a short margin trading, we consider the following state variables:

– Balance of X held by A: $B(A; X; S)$
– The locked amount of Y: $L(A; Y; S)$

Transitions: We assume F transacts from an external market at a price of emp. The transition functions and constraints are specified in Equation 18.

$$
\begin{aligned}
&B(A; X; S') = B(A; X; S) - c_X \\
&L(A; Y; S') = L(A; Y; S) + l_Y \\
&\text{where } l_Y = \frac{d_X \times \ell}{ocr} \times emp \\
&\text{s.t. } B(A; X; S) - c_X \geq 0;\ w_X + d_X - \frac{d_X \times \ell}{ocr} \geq 0
\end{aligned}
\tag{18}
$$

D Optimizing the Pump Attack and Arbitrage

In the following, we detail the procedure of deriving the pump attack and arbitrage optimization problem. Figure 5 summarizes the on-chain state when the attack was executed (i.e., S_0). X and Y denote ETH and WBTC respectively. For simplicity, we ignore the trading fees in the constant product AMM (i.e., $f = 0$

for M). The endpoints executed in the pump attack and arbitrage are listed in the execution order as follows.

1. Loan (dYdX)
2. CollateralizedBorrow (Compound)
3. MarginShort(bZx) & SwapXforY (Uniswap)
4. SwapYforX (Uniswap)
5. Repay (dYdX)
6. SellXforY & CollateralizedRepay(Compound)

In the pump attack and arbitrage vector, we intend to tune the following two parameters, (i) p1: the amount of X collateralized to borrow Y in the endpoint 2) and (ii) p2: the amount of X collateralized to short Y in the endpoint 3). Following
the procedure of Section 4.2, we proceed with detailing the construction of the constraint system.

0): We assume the initial balance of X owned by A is B0 (cf. Equation 19), and we refer the reader to Figure 5 for the remaining initial state values.

$B(A; X; S0) = B0$ (19)

1) Loan: A gets a flash loan of X amounts p1 + p2 in total

$B(A; X; S1) = B0 + p1 + p2$

with the constraints

$p1 \geq 0, p2 \geq 0, vX - p1 - p2 \geq 0$

2) CollateralizedBorrow: A collateralizes p1 amount of X to borrow Y from the lending platform L

$B(A; X; S2) = B(A; X; S1) - p1 = B0 + p2$

$B(A; Y; S2) = p1 \times cf$

er

with the constraint $zY -$

$p1 \times cf$

er

≥ 0

3) MarginShort & SwapXforY: A opens a short margin with p2 amount of X at a leverage of ` on the margin trading platform T; T swaps the leveraged X for Y at the constant product AMM M

$B(A; X; S3) = B(A; X; S2) - p2 = B0$

$uX(S3) = uX(S0) + p2 \times$ `

ocr

$uY(S3) = uX(S0) \times uY(S0)$

$uX(S3)$

$L(A; Y; S3) = uY(S0) - uY(S3)$

with the constraint $wX + p2 -$

$p2 \times$ `

ocr

≥ 0

4) SwapYforX: A dumps all the borrowed Y at M

$B(A; Y; S4) = 0$

$uY(S4) = uY(S3) + B(A; Y; S2)$

$uX(S4) = uX(S3) \times uY(S3)$

$uY(S4)$

$B(A; X; S4) = B0 + uX(S3) - uX(S4)$

5) Repay: A repays the flash loan

$B(A; X; S5) = B(A; X; S4) - p1 - p2$

with the constraint $B(A; X; S4) - p1 - p2 \geq 0$

6) SellXforY & CollateralizedRepay: A buys Y from the market with the market price pm and retrieves the collateral from L

$B(A; X; S6) = B(A; X; S5) + p1 - B(A; Y; S2) \times pm$

A

4) SellXforY: A sells p3 amount of X for Y at the price of pm

$B(A; X; S4) = B(A; X; S3) - p3 = 0$
$B(A; Y; S4) = B(A; Y; S3) + p3$
pm
with the constraint maxY −
p3
pm
≥ 0
5) CollateralizedBorrow: A collateralizes all owned Y to borrow X according to the price given by the constant product AMM M (i.e., the exchange rate
er =
1
PY(M;S2)
)
$B(A; Y; S5) = 0$
$B(A; X; S5) = B(A; Y; S4) \times cf \times PY(M; S2)$
with the constraint
$zY - B(A; Y; S4) \times cf \times PY(M; S2) \geq 0$
6) Repay: A repays the flash loan
$B(A; X; S6) = B(A; X; S5) - p1 - p2 - p3$
with the constraint $B(A; X; S5) - p1 - p2 - p3 \geq 0$
The objective function is the remaining balance of X after repaying the flash loan (cf. Equation 21).
$O(S0; p1; p2; p3) = B(A; X; S6)$
$= B(A; X; S5) - p1 - p2 - p3$
$= B(A; Y; S4) \times cf \times PY(M; S2)$
$- p1 - p2 - p3$
(21)

F Extended Discussion

In the following, we extend our discussion in Section 7.
Responsible disclosure: It is somewhat unclear how to perform responsible disclosure within DeFi, given that the underlying vulnerability and victim are not always perfectly clear and that there is a lack of security standards to apply. We plan to reach out to Aave, Kyber, and Uniswap to disclose the contents of this paper.
Does extra capital help: The main attraction of flash loans stems from them not requiring collateral that needs to be raised. One can, however, wonder whether extra capital would make the attacks we focus on more potent and the ROI greater. Based on our results, extra collateral for the two attacks of Section 3 would not increase the ROI, as the liquidity constraints of the intermediate protocols do not allow for a higher impact.
Potential defenses: Here we discuss several potential defenses. However, we would be the first to admit that these are not foolproof and come with potential downsides that would significantly hamper normal interactions.

– Should DEX accept trades coming from flash loans?
– Should DEX accept coins from an address if the previous block did not show those funds in the address?
– Would introducing a delay make sense, e.g., in governance voting, or price oracles?
– When designing a DeFi protocol, a single transaction should be limited in its abilities: a DEX should not allow a single transaction triggering a slippage beyond 100%.
Looking into the future: In the future, we anticipate DeFi protocols eventually starting to comply with a higher standard of security testing, both within the protocol itself, as well as part of integration testing into the DeFi ecosystem.
We believe that eventually, this may lead to some form of DeFi standards where it comes to financial security, similar to what is imposed on banks and other financial institutions in traditional centralized (government-controlled) finance.

We anticipate that either whole-system penetration testing or an analytical approach to modeling the space of possibilities like in this paper are two ways to improve future DeFi protocols.

Generality of the optimization framework: We show in Section 5 that our optimization framework performs efficiently on a given attack vector. To discover new attacks on a blockchain state with the framework, we may need to iterate over all the combinations of DeFi actions. The search space thus explodes as the number of DeFi actions increases. Our optimization framework requires to model every DeFi action manually. This, however, makes the framework less handy for users who are unfamiliar with the mathematical formulas of the DeFi actions. To make the framework more accurate, we can build gas consumption and block gas limit into the models, which requires to comprehend every DeFi action explicitly. We leave the automation of modeling for future work.