Attacking the DeFi Ecosystem with Flash Loans for Fun and Profit

Kaihua Qin
Imperial College London

Liyi Zhou
Imperial College London

Benjamin Livshits
Imperial College London

Arthur Gervais
Imperial College London

***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 an 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 have been exploited recently (February 2020).

This paper is the first to explore the implication of flash loans for the nascent decentralized finance (DeFi) ecosystem. We analyze two existing attacks vectors with significant ROIs (beyond 500k%), and then go on to formulate finding flash loan-based attack parameters as an *optimization problem* over the state of the underlying Ethereum blockchain as well as the state of the DeFi ecosystem. Specifically, we show 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.

I. 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 may be perceived as a positive force. An abuse of credit, however, i.e. when borrowers take on more debt than they're able to repay, would necessary entail negative future consequences. Excessive debt may 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 nearly infinitive volume — or lend to individuals independently of demographics and geographic location, effectively giving access to capital to rich and poor alike.

Given the peculiarities of blockchain-based smart contracts, so-called *flash loans* emerged. A flash loan is a loan that is only valid within one 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 a repayment is not satisfied. Such instant loan yields three novel properties, absent in centralized financial economies:

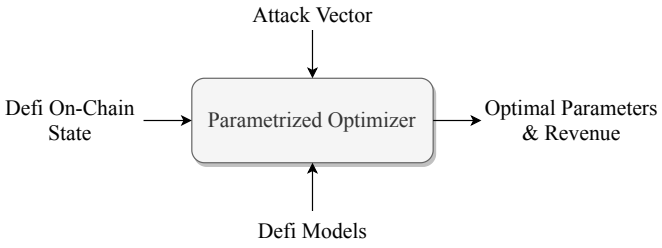

Fig. 1. Overview of our parametrized optimizer. Given a blockchain state, DeFi models, and attack vector, it solves for the parameters yielding the best revenue.

- **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 are taken from a public smart contract-governed liquidity pool. Any borrower can borrow the entire pool at any point in time. As of March 2020, the two largest flash loan pools [5], [15] each offer in excess of 20M USD.

To the best of our knowledge, this is the first paper which investigates flash loans. We categorize their use cases and explore their dangers. We meticulously dissect two events where talented traders realized a profit of each about 350k USD and 600k USD with two independent flash loans. We show how these traders however, 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 (cf. Figure 1).

This paper makes the following contributions:

- **Flash loan usage analysis.** We provide a comprehensive overview of how and where the technique of

[1]Besides the risk of smart contract vulnerabilities.

flash loans can and is utilized.

- **Post mortem of existing attacks.** We provide a detailed analysis of two existing attacks that used flash loans and generated an ROI beyond 500k%: 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 several DeFi systems covering exchanges, credit/lending and margin trading systems, we provide a framework to determine the parameters that yield the maximum revenue a trader can achieve when utilizing a particular flash loan strategy.
- **Opportunity loss.** We analyze previously proposed and executed attacks to quantify the *opportunity loss* for the attacker given their optimal behavior, as determined by the framework above. We experimentally validate the opportunity loss of both aforementioned attacks on their respective blockchain state.

Paper organization: The remainder of the paper is organized as follows. Section II elaborates on the DeFi background. Section III outlines flash loan use cases. Section IV, dissects two known flash loan attacks and Section V shows how to optimize their revenue. Section VI provides a discussion. We outline related work in Section VII. We conclude the paper in Section VIII.

II. Background

Decentralized ledgers, such as Bitcoin [36], enable the performance of transactions among peers without trusting third parties. At its core, a blockchain is a chain of blocks [8], [36], extended by miners by crafting new blocks that contain transactions. Smart contracts [39] allow the execution of complicated transaction types enabling DeFi.

A. Decentralized Finance (DeFi)

Decentralized Finance is a conglomerate of financial cryptocurrency-related protocols defined by open-source smart contracts. These protocols for instance allow to lend and borrow assets [31], [18], exchange [15], [38], margin trade [15], [4], short and long [4], and allow to create derivative assets [18]. At the time of writing, the DeFi space accounts for over 1bn 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).

B. Reverting EVM State Transitions

The Ethereum blockchain is in essence a replicated state machine. To achieve a state transition, one applies as input transactions which modify the EVM state following rules encoded within deployed smart contracts. The EVM state is *only* altered if the transaction execute successfully. Otherwise, the EVM state is reverted to the previous, non-modified state. Transactions can fail due to three reasons: (i) insufficient transaction fees (i.e. due to an out-of-gas exception), (ii) due to a conflicting transaction (e.g. using the same nonce) or (iii) due to a particular condition within the to be executed transaction that cannot be met. State reversion hence appears to be a necessary feature.

C. Flash Loans

Reversing EVM state changes, allows for an intriguing new financial concept: *flash loans*. A flash loan is only valid within a single transaction (cf. Figure 2).

```
1. Take a flash loan
2. Use the lent assets
3. Pay back the loan plus interests
```

Fig. 2. Flash loan approach, summarized.

Flash loans rely on the atomicity of blockchain (and, specifically, Ethereum) transactions within a single block. Atomicity has two important implications on flash loans. First, non-collateralized lending: A lender does not need to provide upfront collateral to request a loan of any size, up to the flash loan liquidity pool amount. Any lender, willing to pay the required transactions fees (which typically amount to a few USD) is an eligible lender. Second, risk-free lending: If a lender is not able to pay back the loan, the flash loan transaction fails. Besides smart contract, and more generally blockchain vulnerabilities, the lender is hence not exposed to the risks of a debt default.

D. DeFi Actors

In the following, we define the on-chain actors that we consider within this work and focus on a single blockchain.

Trader a trader possesses a private/public key pair and is eligible to sign and send transactions towards other accounts and smart contracts.

Liquidity Provider a trader with surplus capital may chose 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 to accessing the available capital.

E. DeFi Platforms

We briefly summarize relevant DeFi platforms, such as exchanges [38], [26], margin trading [15], [4], credit/lending [31], [18] DeFi platforms. Within this paper, we are not covering alternative DeFi platforms such as stablecoins [31], prediction markets and insurance systems.

Exchanges: We observe the following DeFi exchanges.

Limit order book (LOB) DEX: An *order book* is a collection of bid and ask orders. *Traders* post buy/bid or sell/ask orders for an asset of the market to a LOB. A bid order positions the trader as a buyer, while an ask positions the trader as a seller. Buyers aim to purchase an asset at the lowest price possible, while sellers aim for the highest possible selling price. When a trader specifies an

order with a fixed or better price, the trader issues a so-called *limit order* [2]. Once buyers and sellers post orders with compatible prices, their orders can be matched. A liquidity provider contributes bid and asks, to facilitate a match (i.e. *market making*). Several blockchain exchanges operate a LOB within a smart contract [32], [27], [26].

Automated market maker (AMM) DEX: An alternative exchange design is to collect funds within a liquidity pool, e.g. two pools for an AMM asset pair X/Y. 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 pools X and Y to in/decrease liquidity. AMM DEX support endpoint such as `SwapXforY` to trade an asset X for Y. 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. A number of DEX operate under the AMM model [38], [26].

When trading on an exchange, *price slippage* may occur, i.e. the change in the price of an asset during a trade. The greater the quantity to be traded, the greater the slippage.

Margin trading: Trading on margin offers the opportunity for traders to borrow assets from the trading platform (or broker) and trade with these borrowed assets. The trader typically must provide collateral and the trading platform then enables the trader to borrow several multipliers of the collateral for margin trading. Multiple DeFi platforms offer margin trading [4], [15].

Credit and lending: With over 900M USD locked capital, credit represents one of the most significant recent use-cases for blockchain based DeFi systems. Because borrowers are only represented with weak identities (e.g. public keys), they must provide between 125% [15] to 150% [31] collateral of an asset x to borrow 100% of another asset y. Different DeFi lending platforms exist, ranging from user-to-user lending, to pooled lending [18] and lending that enable decentralized stable coins.

III. Use Cases for Flash Loan

In this section, we are analyze the possible use cases for flash loans. It is in general difficult to qualify these activities as fully benign or malicious—it depends on the intent of the people orchestrating these transactions.

A. Arbitrage

The value of an asset is typically determined by 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 each exchange. Related work compared Bitcoin, Ethereum and Ripple price variation across 14 exchanges in Europe, Korea, Japan and the US (excluding China) from 1st January 2017 to 28th February 2018 [30]. The study found twice, price deviations beyond 50% during several hours. *Arbitrage* is the process of exploiting price differences among exchanges for a financial gain [3]. In fact, arbitrage helps synchronizing exchanges by incentivizing traders to equate the price of the same asset across exchanges. To perform arbitrage, a trader needs a reserve of an asset at different exchanges — i.e. arbitrage requires an extensive portfolio and volatility risk management.

How flash loans change arbitrage risks: Given flash loans, a trader can perform arbitrage on different DEX, without the need to hold a monetary position or being exposed to volatility risks. The trader can simply open a loan, perform an arbitrage trade and pay back the loan plus interests. One may argue that flash loans render arbitrage risk-free, the risks of smart contract vulnerabilities, however, remain.

Arbitrage example: On $18th$ Jan 2020, a flash loan borrowed 3,137.41 DAI from Aave [5] to make an arbitrage trade on the AMM DEX Uniswap[2]. To prepare the arbitrage, DAI is converted to 3137.41 SAI using MakerDAO's migration contract[3]. The arbitrage converts SAI for 18.16 ETH using SAI/ETH Uniswap, and then immediately converts 18.16 ETH back to 3,148.39 DAI using DAI/ETH Uniswap. After the arbitrage, 3,148.38 DAI is transferred back to Aave to pay the loan plus fee. This transaction costs 0.02 ETH of gas (about 5.63 USD at the time of writing). Note that even though the transaction sender gains 3.29 DAI from the arbitrage, this particular transaction is not profitable.

B. Wash Trading

The trading volume of an asset, is a metric indicating the trading popularity of an asset. The most popular assets therefore, are supposed to be traded the most — e.g. Bitcoin to 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 to attract interests. In September 2019, 73 out of the top 100 exchanges on Coinmarketcap [9] were wash trading over 90% of their volumes [1]. 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 comes at a cost of the loan interest, trading fees, and (blockchain) transaction fees, e.g. gas. A wash trading endeavour 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 3). We visualize in Figure 3 the required cost to create fake volumes in Uniswap markets. At the time of writing, the

[2]transaction id: 0x4555a69b40fa465b60406c4d23e2eb98d8aee51def21faa28bb7d2b4a73ab1a9

[3]address: 0xc73e0383F3Aff3215E6f04B0331D58CeCf0Ab849

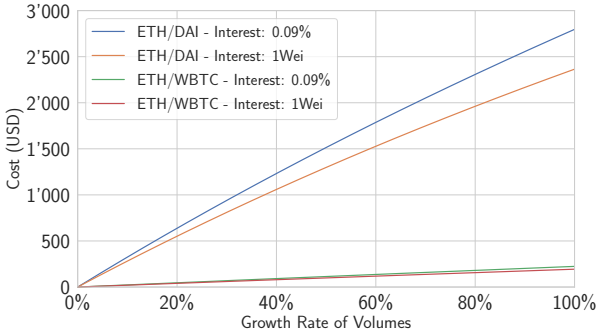

Fig. 3. Wash trading cost on two Uniswap markets with flash loans costing 0.09% (Aave) and a constant of 1 Wei (dYdX) respectively. The 24-hour volumes of ETH/DAI and ETH/WBTC market were 963′786 USD and 67′690 USD respectively (1st of March, 2020).

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).

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[4]. 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.

C. Collateral Swapping

We classify DeFi platforms that rely on users providing cryptocurrencies [15], [5], [31] 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 [31]) and (ii) a DeFi system where long-term loans are offered and assets are aggregated within liquidity pools (e.g. margin trading [4] or long term loans [5]). 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 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 20*th*, 2020, a flash loan borrowed 20.00 DAI (from Aave) to perform a collateral swap (on MakerDAO)[5]. 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 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.

[4]transaction id: 0xf65b384ebe2b7bf1e7bd06adf0daac0413defeed42fd2cc72a75385a200e1544

[5]transaction id: 0x5d5bbfe0b666631916adb8a56821b204d97e75e2a852945ac7396a82e207e0ca

```
contract FlashMintableCoin is ERC20 { [...]
    function flashMint(uint256 amount) {
        // mint coins and transfer them
        mint(msg.sender, amount);
        // borrower uses the loan
        Borrower(msg.sender).execute(amount);
        // reverts if not have enough to burn
        burn(msg.sender, amount);
}}
```

Fig. 4. Flash mint example.

D. 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 4) 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[6]:

IV. Flash Loan Post-Mortem

In this section we investigate how flash loans are used and outline in depth two malicious flash loan transactions which yielded an ROI beyond 500k%. To our knowledge, flash loans only appeared in the beginning of 2020.

A. Flash Loan Uses in the Wild

We first consider flash loans offered by the Aave [5] on the Ethereum blockchain, which started operating on the 8th of January 2020. To our knowledge this 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). At the time of writing, Aave charges a constant 0.09% interest fee for flash loans and amassed a total liquidity of 22M USD.

We collect flash loan data between the 8th of January 2020 and the 26th of February 2020 with a full archive Ethereum node gathering all event logs of the Aave smart contract[7]. We then map the transaction data to a known

[6]cf. https://etherscan.io/address/0x09b4c8200f0cb51e6d44a1974a1bc07336b9f47f#code

[7]address: 0x398eC7346DcD622eDc5ae82352F02bE94C62d119

Actions within DeFi	Flash loan txs	Total amount(DAI)	Mean gas used
0x, Aave, Dai, Oasis, USDC, WETH9	2	5227.47	593075.0 ± 47342.21
0x, Aave, Dai, Oasis, WETH9	1	1051.00	437015.0
0x, Aave, Dai, WETH9	2	49.96	493656.0 ± 9981.52
Aave, BAT, CollateralSwap, DSProxy, Dai, MakerDAO, Uniswap, WETH9	13	371.31	991763.85 ± 14030.38
Aave, BAT, CollateralSwap, Dai, MakerDAO, Uniswap, Unkown, WETH9	2	40.03	982763.0 ± 3357.34
Aave, Bancor, Dai, MakerDAO, OneLeverage, cDai, cEther	5	78.13	2205191.8 ± 71491.59
Aave, Compound, Dai, Kyber, MakerDAO, OneLeverage, cDai, cEther	6	151.27	2330717.83 ± 56046.77
Aave, Compound, Dai, MakerDAO, Oasis, OneLeverage, WETH9, cDai, cEther	9	2778.95	2096669.44 ± 31230.68
Aave, Compound, Dai, MakerDAO, Oasis, USDC, Unkown, cDai	1	0.00	4615916.0
Aave, Compound, Dai, MakerDAO, Oasis, Unkown, WETH9, cDai, cEther	1	9.13	4284296.0
Aave, Compound, Dai, MakerDAO, OneLeverage, Uniswap, cDai, cEther	8	425.66	2015270.0 ± 24679.46
Aave, Compound, Dai, MakerDAO, Uniswap, Unkown, cDai	1	0.00	4827060.0
Aave, Dai	9	5679.00	202890.78 ± 3058.56
Aave, Dai, Kyber, MakerDAO, OneLeverage, cDai, cEther	12	2554.08	2020557.33 ± 63103.89
Aave, Dai, MakerDAO, Oasis, OneLeverage, WETH9, cDai, cEther	6	1220.93	1729013.83 ± 55009.27
Aave, Dai, MakerDAO, OneLeverage, Uniswap, cDai, cEther	11	117.50	1634849.55 ± 25608.84
Aave, Dai, MakerDAO, SAI, Uniswap	1	3137.41	447779.0
Aave, Dai, MakerDAO, Uniswap, cDai, cEther	8	1368.71	1013177.0 ± 142538.07
Aave, Dai, Unkown	1	0.10	567382.0
Aave, Unkown	6	0.00	205047.83 ± 6077.03

Fig. 5. Classifying the usage of flash loans in the wild, based on an analysis of transactions between 8th of January, 2020 and the 26th of February, 2020 on Aave [5]. `Unknown` indicates a private contract we could not attach an owner to.

list of projects (cf. Appendix A). In Figure 5 we show our analysis of Aave flash loans, and manually label with which platforms the flash loans interacts with. We observe that most flash loans interact with lending/exchange DeFi systems and that the flash loan's transaction costs (i.e. gas) appears significant (at times beyond 4M gas, compared to 21k gas for regular Ether transfer).

B. Pump and Arbitrage

A flash loan transaction[8], followed by 74 transactions, yielded a profit of 1′193.69 ETH (350k USD) given a transaction fee of 132.36 USD (cumulative 50′237′867 gas, 0.5 ETH). We show in Section V-E that the parameters chosen by the adversary are not optimal, the adversary could have earned a profit exceeding 829.5k USD.

Attack intuition: The core of this trade utilises a margin trade on a DEX (bZx) to increase the price of WBTC/ETH on another DEX (Uniswap) and thus creates an arbitrage opportunity. The trader then borrows WBTC using ETH as collateral (on Compound), and then purchases ETH at a "cheaper" price on the distorted (Uniswap) DEX market. To maximise the profit, the adversary then converts the "cheap" ETH to purchase WBTC at a non-manipulated market price over a period of two days after the flash loan. The adversary then returns WBTC (to Compound) to redeem the ETH collateral. As demonstrated in Figure 6, this trade mainly consists of two parts. For simplicity, we omit the conversion between WETH (the ERC20-tradable version of ETH) and ETH.

Flash Loan (one block): The first part of the attack (cf. Figure 6) consists of 7 steps within a single transaction.

[8]executed on the 15th of February, 2020, transaction id: 0xb5c8bd9430b6cc87a0e2fe110ece6bf527fa4f170a4bc8cd032f768fc5219838, 264.71 USD/ETH

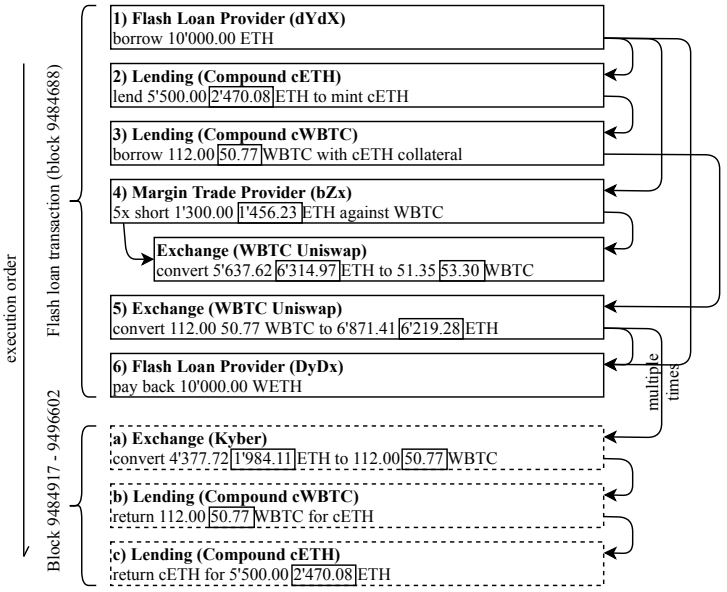

Fig. 6. Procedure diagram of the pump and arbitrage attack. Solid box represents a single state change operation, and dashed box represents aggregated state change operation. The attack consists of two parts a single flash loan transaction and several loan redemption transactions. The numbers within rectangles represent the optimal parameters found by our parametrized optimizer (cf. Section V).

In step (1), the adversarial trader borrows a flash loan of 10,000.00 ETH from a flash loan provider (dYdX). In step (2) and (3), the adversarial trader uses 5,500.00 out of the 10,000.00 ETH as collateral, to borrow 112.00 WBTC on a lending platform (Compound). More specifically, the adversary first deposits 5,500 ETH to Compound, in exchange of 274,843.68 cETH (cTokens) as a proof of owning this liquidity. The adversary then borrows WBTC (on Compound) using the cETH tokens as collateral. 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 forced liquidation against the 274,843.68 cETH collateral if the price fluctuates. In steps (4), the trader opens 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 239.84% (cf. Equation 1).

$$\frac{124.41 - 36.55}{36.55} = 239.84\% \quad (1)$$

Both DEXes, Uniswap and bZx, allowed for such high slippage to occur. In step (5), the trader converts 112.00 WBTC borrowed from lending platform (Compound) to 6′871.41 ETH on DEX (Uniswap) (at 61.35 ETH/WBTC). Similarly, the slippage can be calculated per Equation 2.

$$\frac{\frac{1}{61.35} - \frac{1}{36.55}}{\frac{1}{36.55}} = -40.42\% \quad (2)$$

In step (6), the trader pays back the loan, paying a 1×10^{11} Wei fee. Note that dYdX only requires a fee of 1 Wei. After the flash loan transaction (i.e. the first part of this pump and arbitrage trade), the trader gained 71.41 ETH, and has an over-collateralized loan of 5,500 ETH for 112 WBTC (49.10 ETH/WBTC). If the ETH/WBTC market price is above this loan exchange rate, the adversary can redeem the loan's collateral as follows.

Loan redemption: The second part of the trade consists of three recurring steps, (step (a) - (c)), between Ethereum block 9484917 and 9496602. Those transactions aim to redeem ETH by paying back 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.

Finding the victim: We investigate who of the participating entities is losing money. Note that in step (4) of Figure 6, 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 (4) requires to purchase WBTC at a price of 328.49 ETH/WBTC, with both, the adversary's collateral and the pool funds of the liquidity provider. 328.49 ETH/WBTC does not correspond to the market price of 36.55 ETH/WBTC prior to the attack, hence the liquidity provider overpay by nearly a magnitude of the WBTC price.

How much are the victims losing: We now quantify the losses by 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 in total.

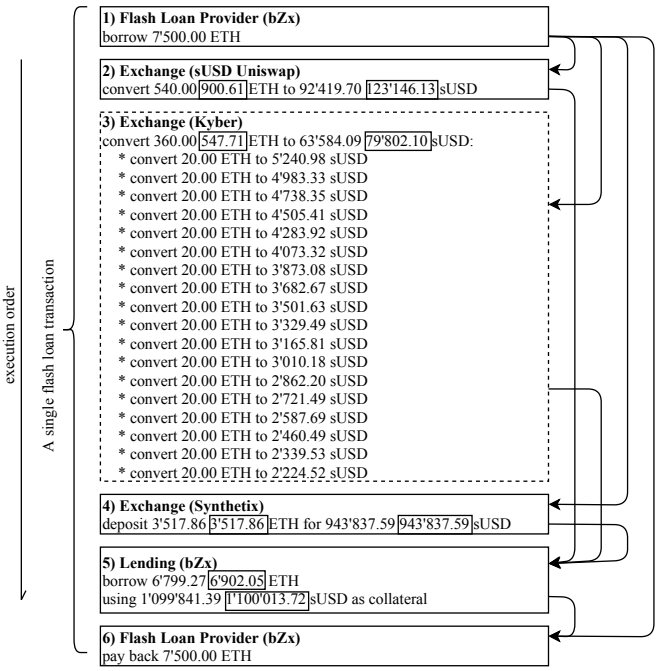

Fig. 7. Procedure diagram of the oracle manipulation attack. Solid box represents a single state change operation, and dashed box represents aggregated state change operation. The numbers within rectangles represent the optimal parameters found by our parametrized optimizer (cf. Section V).

Arbitrage: is more money left on the table: Due to the attack, Uniswap's price 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).

C. Oracle Manipulation

In the following, we discuss the details of a second flash loan trade, which yields a profit of 2,381.41 ETH (c. 650k USD) within a single transaction[9] given a transaction fee of 118.79 USD. Before diving into the details, we cover additional required background knowledge. We again show how the chosen attack parameters were suboptimal and present in Section V-E attack parameters that would yield a profit of 1.1M USD instead. For this attack, the adversary involves three different exchanges for the same sUSD/ETH market pair (the Kyber-Uniswap reserve, Kyber, and Synthetix). Two of these exchanges (Kyber, Kyber-Uniswap) act as price oracle for the lending platform (bZx) from which the adversary borrows assets.

[9]executed on the 18*th* February 2020, transaction id: 0x762881b07feb63c436dee38edd4ff1f7a74c33091e534af56c9f7d49b5ecac15, 282.91 USD/ETH

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). One drawback of this approach, is the danger of a DEX price manipulation.

Attack intuitionn: The core of this trade is an oracle manipulation using a flash loan on the asset pair sUSD/ETH. The manipulation lowers the price of sUSD/ETH (from 268.30 sUSD/ETH to 106.05 sUSD/ETH on Uniswap and 108.44 sUSD/ETH on Kyber Reserve). In a second step, the adversary benefits from this sUSD/ETH price decrease by borrowing ETH with sUSD as collateral.

Adversarial oracle manipulation: We identify a total of 6 steps steps within this transaction (cf. Figure 7). In step (1), the trader 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 943,837.59 sUSD (at an average of 213.64 sUSD/ETH). Step (2) purchases sUSD with ETH at 171.15 sUSD/ETH (on the Kyber-Uniswap reserve) and step (3) purchases sUSD with ETH at 111.23 sUSD/ETH (on Kyber). The third involved party is the lending platform bZx, which uses the DEX Kyber as a price oracle. Step (2) and (3) allow the adversary to borrow more sUSD with ETH, because the price of sUSD/ETH perceived by the lending platform decreased by over 58% since the beginning of the attack. Step (4) converts ETH to sUSD on a third exchange market (Synthetix), which is yet unaffected by the previous trades. This exchange is not serving as price oracle for the lending platform (bZx).

The adversarial trader then uses the sum of the purchased sUSD (1,099,841.39) as collateral to borrows 6,799.27 ETH (at $\frac{\text{exchange rate}}{\text{collateral factor}} = \max(106.05, 108.44) \times 1.5 = 162.66$ sUSD/ETH on bZx). Now the adversary possesses 6,799.27 + 3,082.14 ETH and in the last step pays back 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.

Finding the victim: The adversary distorted the price oracle (i.e. Uniswap and Kyber) from 268.30 sUSD/ETH to 107.83 sUSD/ETH, while other DeFi platforms remain unaffected with 268.30 sUSD/ETH. Similar to the Pump and Arbitrage attack, the lenders on bZx are the victims losing cryptocurrency 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.

V. Optimal DeFi Attack Parameter Generation

In light of the complexity of the aforementioned DeFi attacks (cf. Section IV), in this section we propose a *constrained optimization framework* that allows to efficiently discover the optimal trade parameters to maximize the resulting expected revenue.

A. System Model and Assumptions

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, such as out-of-gas, or insufficient funds returned). Our system comprises of regular users, or traders, which do hold at least one private/public key pair to denote their blockchain address. The private key enables users to transfer cryptocurrency assets and interact/invoke smart contracts.

We assume that the underlying blockchain is not compromised by a malicious adversary. We therefore assume that the share of consensus participants corrupted by the adversary is bounded by the threshold required to maintain safety and liveness of the underlying blockchain. In the Nakamoto consensus-based blockchains, for example, we assume that the fraction of the computational power of the adversary does not exceed 1/3 [21], [20]. Similarly, in Byzantine fault-tolerant systems, (e.g. Proof-of-Stake based), we assume that the number of faulty processes does not exceed 1/3 of the number of consensus participants. The previous assumptions guarantee the *chain quality* and *common-prefix* properties [20]. We consider a transaction to be *securely included* within the blockchain after k confirmations, where k depends on the transaction value [21] and the *chain-growth* property [20].

Importantly, flash loans only apply to a single transaction and hence we limit our analysis to what may happen within a single blockchain block.

B. Threat and Network Model

Foremost, we assume that the cryptographic primitives of the considered blockchain are secure. We also assume the presence of at least one computationally-bounded and economically rational adversary $\mathbb{A}$. $\mathbb{A}$ attempts to exploit the availability of flash loans for financial gain. $\mathbb{A}$ may perform any action that maximizes its economic revenue, such as censor or delay transactions, observe unconfirmed transactions on the network layer or the memory pool, and mount Sybil attacks [14]. For the network layer we follow related work [17], [13] in assuming that honest nodes are well-connected, and that communication channels are semi-synchronous. Importantly, we assume that transactions broadcast by users are received by honest users within an upper bound time. The adversary may collude with other adversaries. While $\mathbb{A}$ 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.

C. Modelling the State of DeFi

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 $\mathsf{S}' = \mathcal{T}(\mathsf{S}; p)$ with the constraints $\mathcal{C}(S; p)$, 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 Ethereum Virtual Machine (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). Note that when quantifying profits, we ignore the loan interest/fee payments and Ethereum 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 $\mathcal{B}(\mathbb{E}; \mathsf{X}; \mathsf{S})$ to denote the balance of currency X held by entity $\mathbb{E}$ at a given state S. The constraint of Equation 3 must always be satisfied.

$$\forall(\mathbb{E}, \mathsf{X}, \mathsf{S}),\ \mathcal{B}(\mathbb{E}; \mathsf{X}; \mathsf{S}) \geq 0 \tag{3}$$

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 $\mathbb{F}$ with z_{X} amount of asset X, which the adversary $\mathbb{A}$ 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 $\mathbb{A}$, i.e. $\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S})$.

Transitions: We define the transition functions of `Loan` in Equation 4 and `Repay` in Equation 5, where the parameter b_{X} denotes the loaned amount.

$$\begin{aligned} \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}') &= \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}) + b_{\mathsf{X}} \\ \text{s.t.} \quad z_{\mathsf{X}} - b_{\mathsf{X}} &\geq 0 \end{aligned} \tag{4}$$

$$\begin{aligned} \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}') &= \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}) - b_{\mathsf{X}} - \text{interest}(b_{\mathsf{X}}) \\ \text{s.t.} \quad \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}) - b_{\mathsf{X}} - \text{interest}(b_{\mathsf{X}}) &\geq 0 \end{aligned} \tag{5}$$

Fixed price trading: We define the endpoint `SellXforY` that allows the adversary $\mathbb{A}$ to trade q_{X} amount of X for Y at a fixed price p_m. maxY is the maximum amount of Y available for trading.

State: We consider the following state variables:

- Balance of asset X held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S})$
- Balance of asset Y held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S})$

Transitions: Transition functions of `SellXforY` are defined in Equation 6.

$$\begin{aligned} \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}') &= \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}) - q_{\mathsf{X}} \\ \mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}') &= \mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}) + \frac{q_{\mathsf{X}}}{\mathsf{p}_m} \\ \text{s.t.} \quad \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}) - q_{\mathsf{X}} &\geq 0 \\ \mathsf{maxY} - \frac{q_{\mathsf{X}}}{\mathsf{p}_m} &\geq 0 \end{aligned} \tag{6}$$

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 current DeFi ecosystem [38]. We denote by $\mathbb{M}$ 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: $u_{\mathsf{X}}(\mathsf{S})$, which equals to $\mathcal{B}(\mathbb{M}; \mathsf{X}; \mathsf{S})$
- Amount of Y in AMM liquidity pool: $u_{\mathsf{Y}}(\mathsf{S})$, which equals to $\mathcal{B}(\mathbb{M}; \mathsf{Y}; \mathsf{S})$
- Balance of X held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S})$
- Balance of Y held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S})$

Transitions: Among the endpoints of $\mathbb{M}$, we focus on `SwapXforY` and `SwapYforX`, which are the relevant endpoints for the DeFi attacks discussed within this work. p_{X} is a parameter that represents the amount of X the adversary intends to trade. $\mathbb{A}$ inputs p_{X} amount of X in AMM liquidity pool and receives o_{Y} amount of Y as output. The constant product rule [38] requires that Equation 7 holds.

$$u_{\mathsf{X}}(\mathsf{S}) \times u_{\mathsf{Y}}(\mathsf{S}) = (u_{\mathsf{X}}(\mathsf{S}) + (1 - \mathsf{f})p_{\mathsf{X}}) \times (u_{\mathsf{Y}}(\mathsf{S}) - o_{\mathsf{Y}}) \tag{7}$$

We define the transition functions and constraints of `SwapXforY` in Equation 8 (analogously for `SwapYforX`).

$$\begin{aligned} \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}') &= \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}) - p_{\mathsf{X}} \\ \mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}') &= \mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}) + o_{\mathsf{Y}} \\ u_{\mathsf{X}}(\mathsf{S}') &= u_{\mathsf{X}}(\mathsf{S}) + p_{\mathsf{X}} \\ u_{\mathsf{Y}}(\mathsf{S}') &= u_{\mathsf{Y}}(\mathsf{S}) - o_{\mathsf{Y}} \\ \text{where} \quad o_{\mathsf{Y}} &= \frac{p_{\mathsf{X}} \times (1 - \mathsf{f}) \times u_{\mathsf{Y}}(\mathsf{S})}{u_{\mathsf{X}}(\mathsf{S}) + p_{\mathsf{X}} \times (1 - \mathsf{f})} \\ \text{s.t.} \quad \mathcal{B}(\mathbb{M}; \mathsf{X}; \mathsf{S}) - p_{\mathsf{X}} &\geq 0 \end{aligned} \tag{8}$$

Because an AMM DEX $\mathbb{M}$ transparently exposes all price transitions on-chain, it can be used as a price oracle by the other DeFi platforms. The price of Y with respect to X given by $\mathbb{M}$ at state S is defined in Equation 9.

$$\mathsf{p}_{\mathsf{Y}}(\mathbb{M}; \mathsf{S}) = \frac{u_{\mathsf{X}}(\mathsf{S})}{u_{\mathsf{Y}}(\mathsf{S})} \tag{9}$$

Automated price reserve: The automated price reserve is another type of AMM that automatically calculates the exchange price depending on the assets hold in inventory. We denote a reserve holding the asset pair X/Y with $\mathbb{R}$. A minimum price minP and a maximum price maxP is set

when initiating $\mathbb{R}$. $\mathbb{R}$ relies on a liquidity ratio parameter lr to calculate the asset price. We assume that $\mathbb{R}$ holds $k_{\mathsf{X}}(\mathsf{S})$ amount of X at state S. We define the price of Y in Equation 10.

$$\mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S}) = \mathsf{minP} \times e^{\mathsf{lr}\times k_{\mathsf{X}}(\mathsf{S})} \tag{10}$$

The endpoint `ConvertXtoY` provided by $\mathbb{R}$ allows the adversary $\mathbb{A}$ to exchange X for Y.

State: We consider the following state variables:

- The inventory of X in the reserve: $k_{\mathsf{X}}(\mathsf{S})$, which equals to $\mathcal{B}(\mathbb{R};\mathsf{X};\mathsf{S})$
- Balance of X held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S})$
- Balance of Y held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S})$

Transitions: We denote as h_{X} the amount of X that $\mathbb{A}$ inputs in the exchange to trade against Y. The exchange output amount of Y is calculated by the following formulation.

$$j_{\mathsf{Y}} = \frac{e^{-\mathsf{lr}\times h_{\mathsf{X}}} - 1}{\mathsf{lr} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S})}$$

We define the transition functions within Equation 11.

$$\begin{aligned} k_{\mathsf{X}}(\mathsf{S}') &= k_{\mathsf{X}}(\mathsf{S}) + h_{\mathsf{X}} \\ \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}') &= \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}) - h_{\mathsf{X}} \\ \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}') &= \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}) + j_{\mathsf{Y}} \\ \text{where} \quad j_{\mathsf{Y}} &= \frac{e^{-\mathsf{lr}\times h_{\mathsf{X}}} - 1}{\mathsf{lr} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S})} \\ \text{s.t.} \quad \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}) - h_{\mathsf{X}} &\geq 0 \\ \mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S}') - \mathsf{minP} &\geq 0 \\ \mathsf{maxP} - \mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S}') &\geq 0 \end{aligned} \tag{11}$$

Collateralized lending & borrowing: We consider a collateralized lending platform $\mathbb{L}$, which provides the `CollateralizedBorrow` endpoint that requires the user to collateralize an asset X with a collateral factor cf (s.t. $0 < \mathsf{cf} < 1$) 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 $\mathbb{L}$ for simplicity.

- Balance of asset X held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S})$
- Balance of asset Y held by $\mathbb{A}$: $\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S})$

Transitions: The parameter c_{X} represents the amount of asset X that $\mathbb{A}$ aims to collateralize. Although $\mathbb{A}$ is allowed to borrow less than his collateral would allow for, we assume that $\mathbb{A}$ makes use the entirety of his collateral. Equation 12 shows the transition functions of `CollateralizedBorrow`.

$$\begin{aligned} \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}') &= \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}) - c_{\mathsf{X}} \\ \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}') &= \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}) + b_{\mathsf{Y}} \\ \text{where} \quad b_{\mathsf{Y}} &= \frac{c_{\mathsf{X}} \times \mathsf{cf}}{\mathsf{er}} \\ \text{s.t.} \quad \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}') - c_{\mathsf{X}} &\geq 0; z_{\mathsf{Y}} - b_{\mathsf{Y}} \geq 0 \end{aligned} \tag{12}$$

$\mathbb{A}$ can retrieve its collateral by repaying the borrowed asset through the endpoint `CollateralizedRepay`. We show the transition functions in Equation 13 and for simplicity ignore the loan interest fee.

$$\begin{aligned} \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}') &= \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}) + c_{\mathsf{X}} \\ \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}') &= \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}) - b_{\mathsf{Y}} \\ \text{s.t.} \quad \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}) - b_{\mathsf{Y}} &\geq 0 \end{aligned} \tag{13}$$

Margin trading: A margin trading platform $\mathbb{T}$ allows the adversary $\mathbb{A}$ 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 $\mathbb{A}$ shorts Y with respect to X on $\mathbb{F}$. 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 $\mathbb{F}$ that is available for the short margin. $\mathbb{A}$ is required to over-collateralize at a rate of ocr in a margin trading. In our model, when a short margin (short Y with respect to X) is opened, $\mathbb{F}$ 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 $\mathbb{A}$: $\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S})$
- The locked amount of Y: $\mathcal{L}(\mathbb{A};\mathsf{Y};\mathsf{S})$

Transitions: We assume $\mathbb{F}$ transacts from an external market at a price of emp. The transition functions and constraints are specified in Equation 14.

$$\begin{aligned} \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}') &= \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}) - c_{\mathsf{X}} \\ \mathcal{L}(\mathbb{A};\mathsf{Y};\mathsf{S}') &= \mathcal{L}(\mathbb{A};\mathsf{Y};\mathsf{S}) + l_{\mathsf{Y}} \\ \text{where} \quad l_{\mathsf{Y}} &= \frac{d_{\mathsf{X}} \times \ell}{\mathsf{ocr} \times \mathsf{emp}} \\ \text{s.t.} \quad \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}) - c_{\mathsf{X}} &\geq 0 \\ w_{\mathsf{X}} + d_{\mathsf{X}} - \frac{d_{\mathsf{X}} \times \ell}{\mathsf{ocr}} &\geq 0 \end{aligned} \tag{14}$$

D. Parametrized Optimization

Our parametrized optimizer (cf. Figure 1) is designed to solve the optimal parameters that maximizes the revenue given an on-chain state, Defi models (cf. Section V-C) 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 15).

$$\mathsf{S}_i = \mathcal{T}_i(\mathsf{S}_{i-1}; p_i) \tag{15}$$

Description	Variable	Value
Maximum Amount of ETH to flash loan	v_{X}	10,000
Collateral Factor	cf	0.75
Collateralized Borrowing Exchange Rate	er	36.48
Maximum Amount of WBTC to Borrow	z_{Y}	155.70
Uniswap Reserved ETH	$u_{\mathsf{X}}(\mathsf{S}_0)$	2,817.77
Uniswap Reserved WBTC	$u_{\mathsf{Y}}(\mathsf{S}_0)$	77.08
Over Collateral Ratio	ocr	1.153
Leverage	ℓ	5
Maximum Amount of ETH to leverage	w_{X}	4,858.74
Market Price of WBTC	p_m	39.08

Fig. 8. Initial on-chain states of the pump and arbitrage attack.

By nesting transition functions, it is trivial to obtain the cumulative state transition functions $\mathcal{ACC}_i(\mathsf{S}_0; p^{1:i}); p_i)$ that satisfy Equation 16, where $p^{1:i} = (p_1, ..., p_i)$.

$$\begin{aligned}\mathsf{S}_i &= \mathcal{T}_i(\mathsf{S}_{i-1}; p_i)\\ &= \mathcal{T}_i(\mathcal{T}_{i-1}(\mathsf{S}_{i-2}; p_{i-1}); p_i)\\ &= \mathcal{T}_i(\mathcal{T}_{i-1}(...\mathcal{T}_1(\mathsf{S}_0, p_1)...; p_{i-1}); p_i)\\ &= \mathcal{ACC}_i(\mathsf{S}_0; p^{1:i})\end{aligned} \tag{16}$$

Therefore the constraints generated in each step can be expressed as Equation 17.

$$\mathcal{C}_i(\mathsf{S}_i; p_i) \iff \mathcal{C}_i(\mathcal{ACC}_i(\mathsf{S}_0; p^{1:i}); p_i) \tag{17}$$

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).

$$\mathcal{O}(\mathsf{S}_0; \mathsf{S}_N) \iff \mathcal{O}(\mathsf{S}_0; \mathcal{ACC}(\mathsf{S}_0; p^{1:N})) \tag{18}$$

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 19).

$$\begin{aligned}\text{maximize} \quad & \mathcal{O}(\mathsf{S}_0; \mathcal{ACC}(\mathsf{S}_0; p^{1:N}))\\ \text{s.t.} \quad & \mathcal{C}_i(\mathcal{ACC}_i(\mathsf{S}_0; p^{1:i}); p_i) \quad \forall i \in [1, N]\end{aligned} \tag{19}$$

E. Optimizing the Pump and Arbitrage

In the following, we evaluate our parametrized optimization framework on the existing attacks described in Section IV. Figure 8 summarizes the on-chain state when the attack was executed (i.e. S_0). We use these blockchain records as the initial state in our evaluation. X and Y denote ETH and WBTC respectively. For simplicity, we ignore the trading fees in the constant product AMM (i.e. $\mathsf{f} = 0$ for $\mathbb{M}$). The endpoints executed in the pump and arbitrage attack 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 in the pump and arbitrage attack vector, we intend to tune the following two parameters, *(i)* p_1: the amount of X collateralised to borrow Y in the endpoint 2) and *(ii)* p_2: the amount of X collateralised to short Y in the endpoint 3). Following the procedure of Section V-D, we proceed with detailing the construction of the constraint system.

0): We assume the initial balance of X owned by $\mathbb{A}$ is B_0 (cf. Equation 20), and we refer the reader to Figure 8 for the remaining initial state values.

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_0) = \mathsf{B}_0 \tag{20}$$

1) `Loan`**:** $\mathbb{A}$ gets a flash loan of X amounts $p_1 + p_2$ in total

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_1) = \mathsf{B}_0 + p_1 + p_2$$

with the constraints

$$p_1 \geq 0, p_2 \geq 0, v_{\mathsf{X}} - p_1 - p_2 \geq 0$$

2) `CollateralizedBorrow`**:** $\mathbb{A}$ collateralizes p_1 amount of X to borrow Y from the lending platform $\mathbb{L}$

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_2) = \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_1) - p_1 = \mathsf{B}_0 + p_2$$

$$\mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}_2) = \frac{p_1 \times \mathsf{cf}}{\mathsf{er}}$$

$$\text{with the constraint } z_{\mathsf{Y}} - \frac{p_1 \times \mathsf{cf}}{\mathsf{er}} \geq 0$$

3) `MarginShort` **&** `SwapXforY`**:** $\mathbb{A}$ opens a short margin with p_2 amount of X at a leverage of ℓ on the margin trading platform $\mathbb{T}$; $\mathbb{T}$ swaps the leveraged X for Y at the constant product AMM $\mathbb{M}$

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_3) = \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_2) - p_2 = \mathsf{B}_0$$

$$u_{\mathsf{X}}(\mathsf{S}_3) = u_{\mathsf{X}}(\mathsf{S}_0) + \frac{p_2 \times \ell}{\mathsf{ocr}}$$

$$u_{\mathsf{Y}}(\mathsf{S}_3) = \frac{u_{\mathsf{X}}(\mathsf{S}_0) \times u_{\mathsf{Y}}(\mathsf{S}_0)}{u_{\mathsf{X}}(\mathsf{S}_3)}$$

$$\mathcal{L}(\mathbb{A}; \mathsf{Y}; \mathsf{S}_3) = u_{\mathsf{Y}}(\mathsf{S}_0) - u_{\mathsf{Y}}(\mathsf{S}_3)$$

$$\text{with the constraint } w_{\mathsf{X}} + p_2 - \frac{p_2 \times \ell}{\mathsf{ocr}} \geq 0$$

4) `SwapYforX`**:** $\mathbb{A}$ dumps all the borrowed Y at $\mathbb{M}$

$$\mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}_4) = 0$$

$$u_{\mathsf{Y}}(\mathsf{S}_4) = u_{\mathsf{Y}}(\mathsf{S}_3) + \mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}_2)$$

$$u_{\mathsf{X}}(\mathsf{S}_4) = \frac{u_{\mathsf{X}}(\mathsf{S}_3) \times u_{\mathsf{Y}}(\mathsf{S}_3)}{u_{\mathsf{Y}}(\mathsf{S}_4)}$$

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_4) = \mathsf{B}_0 + u_{\mathsf{X}}(\mathsf{S}_3) - u_{\mathsf{X}}(\mathsf{S}_4)$$

5) `Repay`**:** $\mathbb{A}$ repays the flash loan

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_5) = \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_4) - p_1 - p_2$$

$$\text{with the constraint } \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_4) - p_1 - p_2 \geq 0$$

6) `SellXforY` **&** `CollateralizedRepay`**:** $\mathbb{A}$ buys Y from the market with the market price p_m and retrieves the collateral from $\mathbb{L}$

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_6) = \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_5) + p_1 - \mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}_2) \times \mathsf{p}_m$$

Objective function	$u_{\mathsf{X}}(\mathsf{S}_0) + \frac{p_2 \times \ell}{\mathsf{ocr}} - u_{\mathsf{X}}(\mathsf{S}_4) - p_2 - \frac{p_1 \times \mathsf{cf} \times \mathsf{p}_m}{\mathsf{er}}$
Constraints	$p_1 \geq 0,\ p_2 \geq 0$ $v_{\mathsf{X}} - p_0 - p_1 \geq 0$ $z_{\mathsf{Y}} - \frac{p_1 \times \mathsf{cf}}{\mathsf{er}} \geq 0$ $w_{\mathsf{X}} + p_2 - \frac{p_2 \times \ell}{\mathsf{ocr}} \geq 0$ $\mathsf{B}_0 + u_{\mathsf{X}}(\mathsf{S}_0) + \frac{p_2 \times \ell}{\mathsf{ocr}} - u_{\mathsf{X}}(\mathsf{S}_4) - p_1 - p_2 \geq 0$

Fig. 9. Constraints generated for the pump and arbitrage attack. We remark that $u_{\mathsf{X}}(\mathsf{S}_4)$ is a nonlinear component with respect to p_1 and p_2.

The objective function is the adversarial ETH revenue (cf. Equation 21).

$$\begin{aligned}\mathcal{O}(\mathsf{S}_0; p_1; p_2) &= \mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_6) - \mathsf{B}_0 \\ &= u_{\mathsf{X}}(\mathsf{S}_3) - u_{\mathsf{X}}(\mathsf{S}_4) - p_2 - \mathsf{p}_m \times \mathcal{B}(\mathbb{A}; \mathsf{Y}; \mathsf{S}_2) \\ &= u_{\mathsf{X}}(\mathsf{S}_0) + \frac{p_2 \times \ell}{\mathsf{ocr}} - u_{\mathsf{X}}(\mathsf{S}_4) - p_2 \\ &\quad - \frac{p_1 \times \mathsf{cf} \times \mathsf{p}_m}{\mathsf{er}}\end{aligned} \tag{21}$$

Constraints: We summarize the constraint in Figure 9, five linear constraints and one nonlinear constraint, which implies that the optimization can be solved efficiently.

F. Optimizing the Pump and Arbitrage Attack

We apply the Sequential Least Squares Programming (SLSQP) algorithm from SciPy[10] to solve the optimization problem. Our program is evaluated on a Ubuntu 18.04.2 machine, 16 CPU cores and 32GB RAM. We repeated our experiment for $1'000$ times, the optimizer spent 6.1ms on average converging to the optimum.

Optimal pump and arbitrage parameters: The optimizer provides a maximum revenue of $2,778.94$ ETH when setting the parameters $(p_1; p_2)$ to $(2,470.08; 1,456.23)$, while in the original attack the parameters $(5,500; 1,300)$ only yield $1,171.70$ ETH. Note, 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 IV). This is a 829.5k USD gain over the attack that took place, using the price of ETH at that time.

Optimal parameter validation: We experimentally validate the optimal pump and arbitrage attack by forking the Ethereum blockchain with Ganache[11] at block 9484687 (one block prior to the original attack transaction). We then implement the pump and arbitrage attack in solidity v0.6.3. In the Pump and Arbitrage attack, revenues are divided into two parts: part one from the flash loan transaction, and part two which is a follow-up operation in later blocks (cf. Section IV) to repay the loan. For simplicity, we chose to only validate the first part, abiding

[10] https://www.scipy.org/. We use the `minimize` function in the `optimize` package.

[11] https://www.trufflesuite.com/ganache

Description	Variable	Value
Maximum Amount of ETH to flash loan	v_{X}	$7,500$
Uniswap Reserved ETH	$u_{\mathsf{X}}(\mathsf{S}_0)$	879.757
Uniswap Reserved sUSD	$u_{\mathsf{Y}}(\mathsf{S}_0)$	$243,441.12$
Liquidity Rate	lr	0.00252
Minimum sUSD Price of Kyber Reserve	minP	0.0037
Maximum sUSD Price of Kyber Reserve	maxP	0.0148
Inventory of ETH in Kyber Reserve	$k_{\mathsf{X}(\mathsf{S}_0)}$	0.90658
Market Price of sUSD	p_m	0.00372719
Maximum Amount of sUSD to Buy	maxY	$943,837.59$
Collateral Factor	cf	0.667
Maximum Amount of ETH to Borrow	z_{Y}	$11,086.29$

Fig. 10. Initial on-chain states of the oracle manipulation attack.

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 part one revenue (as opposed to 71.41 ETH for the original attack). Executing our attack consumes a total of 3.3M gas. We note that these cumbersome manual parameter adjustments would be unnecessary with a more precise DeFi model.

G. Optimizing the Oracle Manipulation Attack

In the oracle manipulation attack, X denotes ETH and Y denotes sUSD. Again, we ignore the trading fees in the constant product AMM (i.e. $\mathsf{f} = 0$ for $\mathbb{M}$). The initial state variables are presented in Figure 10. We assume that $\mathbb{A}$ owns zero balance of X or Y. We list the endpoints involved in the oracle manipulation attack vector as follows.

1) `Loan`(bZx)
2) `SwapXforY`(Uniswap)
3) `ConvertXtoY`(Kyber reserve)
4) `SellXforY`(Synthetix)
5) `CollateralizedBorrow`(bZx)
6) `Repay`(bZx)

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* the amount of X used to swap for Y in step 3); *(iii)* the amount of Y used to exchange for Y in step 4). We construct the constrained optimization problem as follows.

1) `Loan`: $\mathbb{A}$ gets a flash loan of X amounts $p_1 + p_2 + p_3$

$$\mathcal{B}(\mathbb{A}; \mathsf{X}; \mathsf{S}_1) = p_1 + p_2 + p_3$$

with the constraints

$$p_1 \geq 0, p_2 \geq 0, p_3 \geq 0, v_{\mathsf{X}} - p_1 - p_2 - p_3 \geq 0$$

2) `SwapXforY`**:** $\mathbb{A}$ swaps p_1 amount of X for Y from the constant product AMM $\mathbb{M}$

$$\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_2) = \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_1) - p_1 = p_2 + p_3$$
$$u_{\mathsf{X}}(\mathsf{S}_2) = u_{\mathsf{X}}(\mathsf{S}_0) + p_1$$
$$u_{\mathsf{Y}}(\mathsf{S}_2) = \frac{u_{\mathsf{X}}(\mathsf{S}_0) \times u_{\mathsf{Y}}(\mathsf{S}_0)}{u_{\mathsf{X}}(\mathsf{S}_2)}$$
$$\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_2) = u_{\mathsf{Y}}(\mathsf{S}_0) - u_{\mathsf{Y}}(\mathsf{S}_2)$$

3) `ConvertXtoY`**:** $\mathbb{A}$ converts p_2 amount of X to Y from the automated price reserve $\mathbb{R}$

$$\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_3) = \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_2) - p_2 = p_1$$
$$k_{\mathsf{X}}(\mathsf{S}_3) = k_{\mathsf{X}}(\mathsf{S}_0) + p_2$$
$$\mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S}_3) = \mathsf{minP} \times e^{\mathsf{lr} \times k_{\mathsf{X}}(\mathsf{S}_3)}$$
$$\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_3) = \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_2) + \frac{e^{-\mathsf{lr} \times p_2} - 1}{\mathsf{lr} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S}_0)}$$
$$\text{s.t.} \quad \mathsf{maxP} - \mathsf{P}_{\mathsf{Y}}(\mathbb{R};\mathsf{S}_3) \geq 0$$

4) `SellXforY`**:** $\mathbb{A}$ sells p_3 amount of X for Y at the price of p_m

$$\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_4) = \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_3) - p_3 = 0$$
$$\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_4) = \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_3) + \frac{p_3}{\mathsf{p}_m}$$

$$\text{with the constraint } \mathsf{maxY} - \frac{p_3}{\mathsf{p}_m} \geq 0$$

5) `CollateralizedBorrow`**:** $\mathbb{A}$ collateralizes all owned Y to borrow X according to the price given by the constant product AMM $\mathbb{M}$ (i.e. the exchange rate $\mathsf{er} = \frac{1}{\mathsf{P}_{\mathsf{Y}}(\mathbb{M};\mathsf{S}_2)}$)

$$\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_5) = 0$$
$$\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_5) = \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_4) \times \mathsf{cf} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{M};\mathsf{S}_2)$$

with the constraint

$$z_{\mathsf{Y}} - \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_4) \times \mathsf{cf} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{M};\mathsf{S}_2) \geq 0$$

6) `Repay`**:** $\mathbb{A}$ repays the flash loan

$$\mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_6) = \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_5) - p_1 - p_2 - p_3$$

$$\text{with the constraint } \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_5) - p_1 - p_2 - p_3 \geq 0$$

The objective function is the remaining balance of X after repaying the flash loan (cf. Equation 22).

$$\begin{aligned}\mathcal{O}(\mathsf{S}_0;p_1;p_2;p_3) &= \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_6)\\ &= \mathcal{B}(\mathbb{A};\mathsf{X};\mathsf{S}_5) - p_1 - p_2 - p_3\\ &= \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_4) \times \mathsf{cf} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{M};\mathsf{S}_2)\\ &\quad - p_1 - p_2 - p_3\end{aligned} \tag{22}$$

Constraints: We summarize the produced constraints of the oracle manipulation attack vector in Figure 11. Five constraints are linear and the other two are nonlinear.

H. Finding Optimal Oracle Manipulation Parameters

We execute our optimizer 1,000 times on the same Ubuntu 18.04.2 machine with 16 CPU cores and 32 GB RAM. The average convergence time is 12.9ms.

Objective function	$\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_4) \times \mathsf{cf} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{M};\mathsf{S}_2) - p_1 - p_2 - p_3$
Constraints	$p_1 \geq 0,\ p_2 \geq 0,\ p_3 \geq 0$ $v_{\mathsf{X}} - p_1 - p_2 - p_3 \geq 0$ $\mathsf{maxP} - \mathsf{minP} \times e^{\mathsf{lr} \times (k_{\mathsf{X}}(\mathsf{S}_0) + p_2)} \geq 0$ $\mathsf{maxY} - \frac{p_3}{\mathsf{p}_m} \geq 0$ $z_{\mathsf{Y}} - \mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_4) \times \mathsf{cf} \times \mathsf{P}_{\mathsf{Y}}(\mathbb{M};\mathsf{S}_2) \geq 0$

Fig. 11. Constraints generated for the oracle manipulation attack. We remark that $\mathcal{B}(\mathbb{A};\mathsf{Y};\mathsf{S}_4)$ and $\mathsf{P}_{\mathsf{Y}}(\mathbb{M};\mathsf{S}_2)$ are nonlinear components with respect to p_1, p_2 and p_3.

Optimal oracle manipulation parameters: The optimizer discovers that setting $(p_1;p_2;p_3)$ to $(898.58;546.80;3,517.86)$ results in about $6,323.93$ ETH in profit for the adversary. This results in a gain of 1.1M USD instead of about 600k USD.

Optimal parameter validation: We fork the Ethereum blockchain with Ganache at block 9504626 (one block prior to the original adversarial transaction). We then implement the oracle manipulation attack solidity v0.6.3. We validate that executing the adversarial smart contract with parameters $(p_1;p_2;p_3) = (898.58;546.8;3,517.86)$ 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 exceeds the block gas limit (9.7M) on the Ethereum main network). By analyzing the adversarial validation contract, we find 460 is the maximum value of p_2 that makes the gas consumption under the block limit. Following the similar methodology in Section V-F, we add the new constraint to the optimizer, which then gives the optimal parameters $(714.3;460;3,517.86)$. The augmented validation contract makes a profit of $4,167.01$ ETH and consumes 9.6M gas.

VI. 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 [28], [23], [11], [40], [37], and to avoid various forms of market manipulation, etc. [33], [34], [7], 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 attack in Section IV. 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 effectively 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, in a certain way, democratizes the

attack, opening this strategy to the masses. However, it is quite likely that there will be other mechanisms invented that will enable further, potentially even more devastating, attacks in the near future.

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.

Determining what is malicious: An interesting question remains whether we can qualify the use of flash loans (cf. Section III), as clearly malicious (or clearly benign). We believe this is a difficult question to answer and prefer to withhold the value judgement. The two attacks in Section IV are clearly malicious: pump and arbitrate 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 an phenomenon, with good and bad consequences.

Does extra capital help: The main attraction of flash loans stems from them not requiring a 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 IV 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 may 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 of modeling the space of possibilities like in this paper are two ways to improve future DeFi protocols.

VII. Related Work

There is a growing body of work focusing of various forms of manipulation and financially-driven attacks in cryptocurrency markets. Because some of the phenomena presented in this paper are so new, there is a paucity of directly related work. However, existing work can be divided into the following categories.

Crypto manipulation: A thorough crypto manipulation study by Daian *et al.* [12] analyses the behaviour of competitive arbitrage bots. Gandal *et al.* [19] demonstrate that the unprecedented spike in the USD-BTC exchange rate in late 2013 was possibly caused by price manipulation. Makarov *et al.* [29] probe arbitrage opportunities in crypto markets. Many scholars use GARCH models to fit the time series of Bitcoin price. Dyhrberg *et al.* [16] explore the financial asset capabilities of Bitcoin and suggests categorizing Bitcoin as something between gold and US Dollar; Katsiampa [25] emphasizes modelling accuracy and recommends the AR-CGARCH model for price retro-fitting. Bariviera *et al.* [6] compute the Hurst exponent by means of the Detrended Fluctuation Analysis method and conclude that the market liquidity does not affect the level of long-range dependence. Corbet *et al.* [10] demonstrate that Bitcoin shows characteristics of an speculative asset rather than a currency also with the presence of futures trading in Bitcoin. Some recent papers focus on the phenomenon of pump-and-dump for manipulating crypto coin prices [41], [24].

Governance Attacks: DeFi protocols such as MakerDAO [31] operate with a decentralized governance mechanism. Holders of a voting token, are eligible to propose and vote on changes to the protocol. By design, an entity that is capable of amassing a sufficient number of voting tokens is eligible to perform unilaterally significant changes (e.g. to liquidate and receive all collateral). Governance attacks [35] could be aggravated with flash loans [22].

VIII. Conclusion

This paper is the first one to present a detailed exploration of 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.

References

[1] Bti market surveillance report - september 2019 - bti. https://www.bti.live/bti-september-2019-wash-trade-report/. (Accessed on 02/24/2020).

[2] SEC Glossary, 2019.

[3] Arbitrage, 2020.

[4] Bzx network, 2020.

[5] Aave. Aave Protocol. https://github.com/aave/aave-protocol, 2020.

[6] Aurelio F. Bariviera, MarÃŋa JosÃĺ Basgall, Waldo Hasperué, and Marcelo Naiouf. Some stylized facts of the Bitcoin market. *Physica A: Statistical Mechanics and its Applications*, 484:82–90, 2017.

[7] Iddo Bentov, Yan Ji, Fan Zhang, Yunqi Li, Xueyuan Zhao, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-Time Cryptocurrency Exchange using Trusted Hardware. *Conference on Computer and Communications Security*, 2019.

[8] 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 *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.

[9] CoinMarketCap. Bitcoin market capitalization, 2019.

[10] Shaen Corbet, Brian Lucey, Maurice Peat, and Samuel Vigne. Bitcoin Futures – What use are they? *Economics Letters*, 172:23–27, 2018.

[11] Crytic. Echidna: Ethereum fuzz testing framework.

[12] 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. *IEEE Security and Privacy 2020*, 2020.

[13] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[14] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[15] dYdX. dYdX. https://dydx.exchange/, 2020.

[16] Anne Haubo Dyhrberg. Bitcoin, gold and the dollar - A GARCH volatility analysis. *Finance Research Letters*, 16:85–92, 2015.

[17] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016.

[18] Compound Finance. Compound finance, 2019.

[19] Neil Gandal, JT Hamrick, Tyler Moore, and Tali Oberman. Price manipulation in the Bitcoin ecosystem. *Journal of Monetary Economics*, 95(4):86–96, 2018.

[20] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[21] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[22] Lewis Gudgeon, Daniel Perez, Dominik Harz, Arthur Gervais, and Benjamin Livshits. The decentralized financial crisis: Attacking defi, 2020.

[23] Bo Jiang, Ye Liu, and WK Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 259–269. ACM, 2018.

[24] Josh Kamps and Bennett Kleinberg. To the moon: defining and detecting cryptocurrency pump-and-dumps. *Crime Science*, 7, 12 2018.

[25] Paraskevi Katsiampa. Volatility estimation for Bitcoin: A comparison of GARCH models. *Economics Letters*, 158:3–6, 2017.

[26] Kyber. Kyber. https://kyber.network/, 2020.

[27] Aurora Labs. Idex: A real-time and high-throughput ethereum smart contract exchange. Technical report, January 2019.

[28] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269, 2016.

[29] Igor Makarov and Antoinette Schoar. Trading and Arbitrage in Cryptocurrency Markets. 2018.

[30] Igor Makarov and Antoinette Schoar. Trading and arbitrage in cryptocurrency markets. *Journal of Financial Economics*, 135(2):293–319, 2020.

[31] Maker. Makerdao. https://makerdao.com/en/, 2019.

[32] 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.

[33] Vasilios Mavroudis. Market Manipulation as a Security Problem. *arXiv preprint arXiv:1903.12458*, 2019.

[34] Vasilios Mavroudis and Hayden Melton. Libra: Fair Order-Matching for Electronic Financial Exchanges. *arXiv preprint arXiv:1910.00321*, 2019.

[35] Micah Zoltu. How to turn $20M into $340M in 15 seconds. https://medium.com/coinmonks/how-to-turn-20m-into-340m-in-15-seconds-48d161a42311, 2019. [Online; accessed 9-Feburary-2020].

[36] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[37] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–82. ACM, 2018.

[38] Uniswap.io. accessed November, 2019.

[39] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[40] Valentin Wüstholz and Maria Christakis. Harvey: A greybox fuzzer for smart contracts. *arXiv:1905.06944*, 2019.

[41] Jiahua Xu and Benjamin Livshits. The anatomy of a cryptocurrency pump-and-dump scheme. In *Proceedings of the Usenix Security Symposium*, August 2019.

Appendix

```
category_map = {
    "0x6B175474E89094C44Da98b954EedeAC495271d0F": "Dai",
    "0x61935CbDd02287B511119DDb11Aeb42F1593b7Ef": "0x",
    "0x197E90f9FAD81970bA7976f33CbD77088E5D7cf7": "MakerDAO",
    "0x7a3370075a54B187d7bD5DceBf0ff2B5552d4F7D": "Kyber",
    "0x9759A6Ac90977b93B58547b4A71c78317f391A28": "MakerDAO",
    "0xad37fd42185Ba63009177058208dd1be4b136e6b": "MakerDAO",
    "0x2a1530C4C41db0B0b2bB646CB5Eb1A67b7158667": "Uniswap",
    "0x398eC7346DcD622eDc5ae82352F02bE94C62d119": "Aave",
    "0x3d9819210A31b4961b30EF54bE2aeD79B9c9Cd3B": "Compound",
    "0x5d3a536E4D6DbD6114cc1Ead35777bAB948E3643": "cDai",
    "0xc0829421C1d260BD3cB3E0F06cfE2D52db2cE315": "Bancor",
    "0x8007aa43792A392b221DC091bdb2191E5fF626d1": "Kyber",
    "0x5bcA0f6cD5F9a74895d66005acEf969342F301A0": "CollateralSwap",
    "0x20a1d01e03D65495AE157d47E4519EceACb608f6": "OneLeverage",
    "0x5ef30b9986345249bc32d8928B7ee64DE9435E39": "MakerDAO",
    "0x3D0B1912B66114d4096F48A8CEe3A56C231772cA": "MakerDAO",
    "0x3Ab6564d5c214bc416EE8421E05219960504eeAD": "Bancor",
    "0x201b704Ae89b31fB795F5EF41E62461b9302E1BA": "DSProxy",
    "0x65bF64Ff5f51272f729BDcD7AcFB00677ced86Cd": "Kyber",
    "0x4Ddc2D193948926D02f9B1fE9e1daa0718270ED5": "cEther",
    "0x2F0b23f53734252Bda2277357e97e1517d6B042A": "MakerDAO",
    "0x23401C7811411f40008CE9688EdB293D8fe507bc": "DSProxy",
    "0x06f7Bf937Dec0C413a2E0464Bb300C4d464bb891": "Bancor",
    "0x3dfd23A6c5E8BbcFc9581d2E864a68feb6a076d3": "Aave",
    "0x63825c174ab367968EC60f061753D3bbD36A0D8F": "Kyber",
    "0x794e6e91555438aFc3ccF1c5076A74F42133d08D": "Oasis",
    "0xC9A4AEF09fD9ae835A0c60A0757C8dd748116781": "OneLeverage",
    "0x1F573D6Fb3F13d689FF844B4cE37794d79a7FF1C": "Bancor",
    "0xFa8C4B17ac43A025977F5feD843B6c8c4EA52F1c": "DSProxy",
    "0x2E642b8D59B45a1D8c5aEf716A84FF44ea665914": "Uniswap",
    "0xE03374cAcf4600F56BDDbDC82c07b375f318fc5C": "Bancor",
    "0x309627af60F0926daa6041B8279484312f2bf060": "Bancor",
    "0x09cabEC1eAd1c0Ba254B09efb3EE13841712bE14": "Uniswap",
    "0x0D8775F648430679A709E98d2b0Cb6250d2887EF": "BAT",
    "0x207737F726c13C1298B318D233AAa6164EE6b712": "DSProxy",
    "0x818E6FECD516Ecc3849DAf6845e3EC868087B755": "Kyber",
    "0x35D1b3F3D7966A1DFe207aa4514C12a259A0492B": "MakerDAO",
    "0x39755357759cE0d7f32dC8dC45414CCa409AE24e": "Oasis",
    "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48": "USDC",
    "0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2": "WETH9",
    "0x7778d1011e19C0091C930d4BEfA2B0e47441562A": "OneLeverage",
    "0x89d24A6b4CcB1B6fAA2625fE562bDD9a23260359": "SAI",
    "0xd3ec78814966Ca1Eb4c923aF4Da86BF7e6c743bA": "Bancor",
    "0x19c0976f590D67707E62397C87829d896Dc0f1F1": "MakerDAO",
    "0x35A679A2A63F774BBEc5E80E32aE436BC3b5d98e": "DSProxy",
}
```