

**Blockchain Programming**

Flash Loan Attacks: Manipulating Liquidity Pools and options to prevent this kind of attacks.

Maximilian Kiefer (XX-XXX-XXX)

Pascal Emmenegger (XX-XXX-XXX)

Nicola Crimi (12-748-612)

Affiliation

Date: 01.12.2021

**ABSTRACT**

The goal was to successfully conduct a flash loan Attack within the UZHETH Network. Out of several types of flash loan attacks, the team decided to conduct an oracle manipulation. The intention was to manipulate the price calculation function of a DeFi protocol that could execute collateralized loans. Finally, the question of how to prevent these kinds of attacks should be answered within the scope of this project.

## Keywords

flash loan attack, oracle manipulation, liquidity pools, prevention

# Introduction

Flash loan attacks (FLA’s) are a very important issue as many blockchain networks are subject to these kinds of attacks and are losing millions of US dollars. A FLA occurs when the borrower uses the markets as the loan is taking place, driving the value of a token underwater (there are several ways of doing so, which is explained later in this paper), and then allowing the attacker to buy back the token at a depressed amount. These flash loan attacks take benefit of the leverage provided by flash loans to allow an attacker to develop weaknesses within DeFi Protocols' smart contracts. In many cases, these exploits allow the attacker to totally drain a project's liquidity pools, racking up massive losses for the protocols' clients. Conventional lenders take on two types of risk. The initial one is default risk: if the borrower runs off with the money, that clearly is terrible. But the second risk to a lender is the illiquidity risk: if a lender lends out too many of its assets at the wrong times or doesn’t obtain judicious repayments, the lender may be suddenly illiquid and not be able to meet its own commitments. In other words, a flash loan functions as the following “I will lend you as much money as you want for this one transaction. But by the close of this transaction, you must pay me at the slightest as much as I lent you. If you are incapable to do that, I will roll back your contract”.

In addition, there are major security issues in blockchain transactions, which makes the flash loan attack and its varying types so attractive to cybercriminals. All flash attacks should eventually be obtained by miners. This will serve as a warning against flash attacks since it will leave attackers powerless to mould their discoveries of these vulnerabilities. Flash loans are used non-spitefully to take advantage of arbitrage prospects across various exchanges. Flash loans have been increasingly used in attacks on DeFi protocols such as with Cheese bank and Harvest.

# Set UP for A Flash Loan attack

## Programming and smart contracts

We are using UZHETH network smart contracts, and the programming being done is with the Solidity language. Furthermore, the Metamask extension is used as a wallet. All the programming has been done within the remix IDE.

## Deploying a decentralized exchange (DEX)

In order to conduct a FLA within the UZHETH network it is necessary to have at least one working DEX. The team decided on deploying the Uniswap V2 (without GUI). In consecutive steps, this DEX is used to manipulate its liquidity pools. The Uniswap V2 code is openly accessible on GitHub.

Smart Contract Name/address: xyz.sol/XXXXXXX

## Deploying DeFi Protocols

Besides a DEX, the team needed a (at least one) DeFi Protocol which was able to issue un-/collateralized (flash) loans. For the sake of simplicity, it decided on deploying two separate protocols. One protocol which was able to issue flash loans (FlashLender.sol) and one protocol which issues loans only against collaterals (CollateralLoan.sol). This protocol bases its price calculation on the DEX (see 2.2).

Smart contract name/address: xyz.sol/XXXXXXX

Smart contract name/address: xyz.sol/XXXXXXX

## Implementing and deploying a flash loan attack

The attack itself is implemented within the FlashBorrorw.sol smart contract which is responsible for the following steps:

1. Obtaining a flash loan from the FlashLender.sol contract
2. Swap a portion of the flash loan on the DEX and shift the liquidity ratio to the favour of the attacker (FlashBorrower.sol)
3. The tokens obtained by the swap (in our case USTs) swapping against a collateral (DOTs)
4. Pay back the flash loan which was initiated in step 1
5. Transfer the (positive) delta into the Metamask

Smart contract name/address: xyz.sol/XXXXXXX

## Minting ERC20 Tokens

For the oracle manipulation two tokens are needed. The group minted two ERC20 tokens (UZHDOT.sol, UZHUST.sol) with a total supply of 1’000’000.

Smart contract name/address: xyz.sol/XXXXXXX

Smart contract name/address: xyz.sol/XXXXXXX

# Conceptual overview

This section is dedicated to a general overview of the domain of flash loan attacks, about the recent history of attacks and about possibilities to prevent these kinds of attacks.

## Definition of a Flash Loan

A flash loan is a relatively new possibility of uncollateralized lending offered by a DeFi protocol. Furthermore, a flash loan is only valid within one blockchain transaction. Flash loans fail, if the borrower does not repay its debt before the end of the transaction. That is, because a blockchain transaction can be reverted during its execution. (Source: <https://preventflashloanattacks.com/>)

## Types of flash loan attacks

Since every FLA is slightly different than the other, there is not a sharp line when it comes to a classification. Roughly speaking, there are three different categories: pump & arbitrage, re-entrancy and oracle attacks.

Since most of the conducted flash loan attacks in 2020 were oracle attacks, this project focuses as well on this type.

Even within the category of oracle manipulations there is not a single pattern of attacks. On a high level the following steps are executed when it comes to these types of manipulations:

1. Taking out a massive loan (e.g. token A) from a protocol supporting flash loans. In our case from the CollateralLoan.sol contract
2. Swapping token A for token B on a DEX (Uniswap), dumping the price of token A
3. Deposit the purchased token B as collateral on a DeFi protocol that uses the above DEX as its sole price feed and borrow even more with this manipulated price
4. Use a portion of borrowed token A to fully pay back the original flash loan and keep the remaining tokens.

## Historic overview of FLAs

Since DeFi protocols allow flash loans (ca. end of 2019/early 2020) many protocols suffered from all kinds of flash loan attacks. In the following a brief overview:

Table 1 Overview of biggest FLAs in 2020

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Protocol** | **Value (in $)** | **Date** | **Type** | **Fix** |
| bZx (1) | 350’000 | Feb. 2020 | Pump & Arbitrage | ? |
| bZx (2) | 600’000 | Feb. 2020 | Oracle Attack | Chainlink Integrtion |
| Origin Protocol | 7’000’000 | Nov. 2020 | Re-entrancy Attack | ? |
| Harvest.Finance | 24’000’000 | Oct. 2020 | Oracle Attack | ? |
| Value Defi | 6’000’000 | Nov. 2020 | Oracle Attack | Chainlink Integrtion |
| Akropolis | 2’000’000 | Nov. 2020 | Re-entrancy Attack | Re-Entry Security |
| Cheese Bank | 6’000’000 | Nov. 2020 | Oracle Attack | ? |
| Compound | 89’000’000 | Nov. 2020 | Oracle Attack | ? |
| MakerDAO | Unknown | Nov. 2020 | Oracle Attack | ? |
| Warp Finance | 7’760’000 | Dec. 2020 | Oracle Attack | ? |

The project focuses on the bZx (2) attack which was one of the first oracle attacks and which is later fixed with the help of a Chainlink integration.

# Hands-on Flash loan attack

# Prevention methods

# Figures and tables

## General appearance

Make sure that all figures, tables, graphs and line drawings are clear and sharp and of the highest quality. Lines should be thick enough to allow proper reproduction. **Also in figures: use embedded arial font type only.**

Diagrams, graphics and photographs should be in **gray scale or in colour** of excellent quality with good contrast.

When preparing figures and tables, make sure that all lettering inside the figure is no smaller than the specified size of the paper text, i.e., **10 point**. Do not include any headlines in the diagrams, graphics or tables. All headlines should be written separately. See the examples below. Do not use different colours in diagrams. If you use a bar graph, please use a pattern that will appear clearly in black and white. Use different patterns instead of colours, as the colours will not provide sufficient contrast when printed in black and white.

If necessary add a source below the diagram. Do not add any kind of background color in the graph. The background should always be white.

## Numbering, captions and positioning

Number the figures separately from the maps and tables e.g., Figure 1, Figure 2, Figure 3; Table 1, Table 2, Table 3. Map 1, Map 2, Map 3 etc. Use (a), (b), (c) to distinguish individual subjects in a composite figure. See Figures 1 and 2 for examples of figure and caption placement

the paper. Begin the caption with a capital letter and end with a full stop. Always refer to figures as ‘Figure’ and not Fig. Place the figure or table on the text page as close to the relevant citation as possible, preferably at the top of a column. If a figure or table is too large to fit into one column, it may be centred across both columns at the top or the bottom of the page.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Country | 65–79 | | Over 80 | |
| Females | Males | Females | Males |
| Sweden | 45 | 34 | 15 | 9 |
| United Kingdom | 34 | 25 | 10 | 5 |
| France | 45 | 38 | 19 | 11 |
| Germany | 28 | 32 | 21 | 17 |
| Spain | 31 | 24 | 19 | 12 |

**Table 1** is an example of how a table or figure may be placed in a column, preferably in the beginning of a column.

# summary

The summary may be placed in the beginning of the article or in the end before the references.

**8 AUTHOR CONTRIBUTIONS**

The report must have an authorship statement at the end. An example is the following

AUTHOR CONTRIBUTIONS

All authors conceived and designed the project idea. P.M. and C.J.T. performed the literature review and wrote the introduction. B.S. performed the data collection. Y.Z. and X.Y. analysed the data. B.S. wrote the bulk of the text. All authors discussed and reached the conclusions. All authors revised and accepted the final version of this document.

# 9 References

Arenas, Alex, Albert Díaz Guilera, Jurgen Kurths, Yamir Moreno, and Changsong Zhou (2008) “Synchronization in complex networks”. In: *Physics Reports* 469, pp. 93–153.

Boccaletti, S., Vito Latora, Yamir Moreno, M Chavez, and D.-U Hwang (2006) “Complex networks: Structure and dynamics”. In: *Physics Reports* 424.4- 5, pp. 175–308.

Dijkstra, Edsger W. (1974) “Selected writings on Computing: A Personal Perspective”. In: Springer. Chap. On the role of scientific thought, pp. 60–66.

Geipel, M.M. (2009) “Dynamics of Communities and Code in Open Source Software”. PhD thesis. ETH Zurich.

Max, Muster (2014) “Test title”. In: *Journal* 12.3, pp. 54–60.

Merlo, F., S.A. Slaughter, and C. Francalanci (Aug. 2009) “The co-evolution of social networks and software structures: a study of open source and closed source projects”. In: *Academy of Management Annual Meeting*. Chicago, Illinois, USA

Willinger, W., D. Alderson, and J.C. Doyle (2009)

*Mathematics and the internet: A source of enormous confusion and great potential*. Defense Technical Information Center.

Zanetti, Marcelo Serrano (2012) “The Co-Evolution of Socio-Technical Structures in Sustainable Software Development: Lessons from the Open Source Software Communities”. Submitted.