# Blockchain attacks

**Kos, Paolo**

**Master's thesis / Diplomski rad**

**2022**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **University of Rijeka / Sveučilište u Rijeci**

*Permanent link / Trajna poveznica:* https://urn.nsk.hr/urn:nbn:hr:195:997383

University of Rijeka – Faculty of Informatics and Digital Technologies

Graduate study – Information Communication Systems module

Paolo Kos

# Blockchain attacks

Master's thesis

Mentor: Assist. Prof. Dr. Vedran Miletić

Rijeka, 01.09.2022.

FAKULTET INFORMATIKE I DIGITALNIH TEHNOLOGIJA
SVEUČILIŠTE U RIJECI

Rijeka, 13. travnja 2022.

# Zadatak za diplomski rad

Pristupnik: Paolo Kos

Naziv diplomskog rada: Napadi na blockchain

Naziv diplomskog rada na eng. jeziku: Blockchain attacks

Sadržaj zadatka:

Kriptovalute su postale dio svakodnevice, bez obzira govorimo li o njima kao o novcu ili kao o investicijskom dobru. Zbog njihove popularnosti brojni se autori odlučuju okušati u izradi vlastitih kriptovaluta i ponudi istih na tržište, ali pritom riskiraju napade zlonamjernih aktera na lanac blokova koji je temelj njihove valute. Cilj rada je opisati napade i ponuditi rješenja za obranu od tih napada.

Mentor:
Doc. dr. sc. Vedran Miletić

Voditeljica za diplomske radove:
Prof. dr. sc. Ana Meštrović

Komentor:

Zadatak preuzet: 26. travnja 2022.

(potpis pristupnika)

Fakultet informatike i digitalnih tehnologija
Radmile Matejčić 2, 51000 Rijeka, Hrvatska
http://inf.uniri.hr ured@inf.uniri.hr
Tel: +385 (0)51584700 | Fax: +385 (0)51 584 749

# Summary

This paper aims to explain what cryptocurrencies are, how they work, and what security challenges exist in them. In doing so, it explains how blockchain together with smart contracts and liquidity pools are the fundamental parts of the cryptocurrency world. Paper describes two attacks on the blockchain in cryptocurrency trading called „Spoofing“ and „Momentum ignition strategy“. These problems are explained textual and graphically based on real-world events. The practical part of the paper gives pseudo-code solution proposals for those problems and shows graphically how they could work in real situations. The thesis was written with the support of the Juice d.o.o.



Keywords: cryptocurrency, blockchain, Ethereum, smart contract, liquidity pool, sniper bot, trading bot, spoofing, momentum ignition

# Table of Contents

1. Introduction....5
2. Technical understanding....6
2.1 Cryptocurrency....6
2.2 Blockchain....7
2.3 Proof-of-work (PoW) consensus mechanism....8
2.4 Ethereum....9
2.5 Smart contracts....9
2.6 Solidity....10
2.6.1 Truffle Suite....12
2.7 ERC20 Token Standard....13
2.8 Liquidity pools....16
2.9 Sniper bots....18
2.10 Trading Bots....18
2.11 Liquidation Level....18
2.12 Position....18
3. Problem description....20
3.1 Spoofing....20
3.1.1 Understanding the market....21
3.1.2 Fake buy wall....21
3.1.3 Canceling the order....22
3.1.4 Spoofing in real life....23
3.2 Momentum ignition....24
3.2.1 Understanding the market....24
3.2.2 Making a buy/sell trade....25
3.2.3 Selling on profit....26
3.2.4 Momentum ignition in real life....27
4. Problem solutions proposals....29
4.1 Solution 1: “Buffer”....29
4.1.1 Creating a buffer....29
4.1.2 Calculating the average price....30
4.1.3 Calculating the delta of the average price....30
4.1.4 Code....31
4.1.5 Interface....34
4.2 Solution 2: „Forbid high orders“....36
4.2.1 Code....36
4.2.2 Interface....38
5. Conclusion....39
6. Literature....40
7. Table of Figures....43

# 1. Introduction

Transactions via the Internet have become people's everyday life. Paying with "real" money or online is no longer a significant difference. Connecting the card to online providers and paying directly through them is an increasingly common concept every day. With the advancement of online transactions and the desire for anonymity when conducting them, the development of cryptocurrencies also began. As the popularity of cryptocurrencies grows, so does the number of people who want to make money quickly and easily, usually by fraud. This thesis will deal with the topic of recognizing the dangers when buying and selling cryptocurrencies and proposing solutions to deal with them. The thesis explains the basic concepts related to cryptocurrencies, shows 2 problems, and proposes a possible solution to prevent them. The thesis was written with the support of the Juice d.o.o.

The second chapter of the work, "Technical understanding", describes the concepts used in the work (such as, for example, what are cryptocurrencies, liquidity pools, etc.) Each sub-chapter describes one concept.

The third chapter of the work "Problem description" describes the problems found in cryptocurrency trading and is supported by pictures and examples from real life.

The fourth chapter of the paper proposes a solution for given problems written in pseudo-code using JavaScript, but also graphically, to make it more understandable for general readers.

# 2. Technical understanding

## 2.1 Cryptocurrency

Cryptocurrency, sometimes known as "crypto" is a type of virtual currency protected by cryptography, making it almost impossible to forge or double-spend. It's an encrypted and decentralized medium of exchange that has no central authority that manages and maintains its value of it. Individual coin ownership records and the creation of additional coins are kept in a computerized database known as the distributed ledger. [1], [2]

Distributed ledgers are databases that are consensually shared and synchronized across numerous sites and institutions accessible by multiple individuals. Any modifications or additions to the ledger are immediately reflected and duplicated to all participants. [3]

The first cryptocurrency is Bitcoin, which Satoshi Nakamoto initially described in principle in a paper titled "Bitcoin: A Peer-to-Peer Electronic Cash System" published in 2008. It's described as „an electronic payment system based on cryptographic proof instead of trust”. [1]

The two ways to obtain cryptocurrency are by mining and purchasing. [4]

Crypto mining is the process by which Bitcoin and several other cryptocurrencies use to create new coins and validate new transactions. Anyone who can buy specialized hardware and connect to the internet can be a miner. [2]

Cryptocurrency can be bought through a crypto exchange or a crypto broker. A cryptocurrency exchange is an online marketplace where buyers and sellers meet to trade cryptocurrencies for generally minimal fees. [4]

The most popular kind of exchange platforms are centralized exchanges that function as a middleman between a buyer and a seller. It offers users security and monitoring to help them find trading partners. Centralized cryptocurrency exchanges are convenient and dependable but come with transaction costs and hacking risks. [5], [6]

Examples of centralized cryptocurrency exchanges are: [6]

- Binance
- Coinbase
- Kraken

Decentralized cryptocurrency exchanges use blockchain technology or distributed ledgers, the currency isn't held by a third party, and transactions are done peer-to-peer using smart contracts. Decentralized crypto exchanges have less hacking risk and do not require customers to fill out

„know-your-customer” forms offering privacy and anonymity to users but are more complex than centralized crypto exchanges. [6]

Examples of decentralized cryptocurrency exchanges are: [7]

- Uniswap
- PancakeSwap
- Venus

A cryptocurrency broker is a company or person that simplifies the process of buying cryptocurrency by providing users with simple interfaces that communicate with exchanges. Crypto brokers enable buying and selling for customers at prices set by them and are useful if a user wants to buy a small number of cryptocurrencies. [8]

Examples of crypto brokers are: [8]

- Robinhood
- SoFi
- Webull

## 2.2 Blockchain

A blockchain is a distributed database that is updated and shared across many computers in a network. It stores information in digital format and is best known for its crucial role in cryptocurrency systems for maintaining a secure and decentralized record of transactions. [9]

Word “blockchain” can be split into two parts: “Block” and “Chain”. [9]

- “Block” means that data and state are being stored in groups known as “blocks” which are used in transactions. Each block contains data, a hash, and a hash of the previous block. Data stored inside of the block depends on the type of the blockchain (e.g., data of sender, receiver, and an amount of token). Hash is a unique block identifier calculated when the block is created. If data inside the block is changed, the hash changes too and the block is not the same as before. The hash of the previous block is used for chaining blocks. [9]
- “Chain” refers to the fact that each block has its parent meaning that all the blocks are chained together (if one block is changed, all subsequent blocks are changed too). [9]

The connection between blocks in the blockchain using hashing is shown in the figure below.

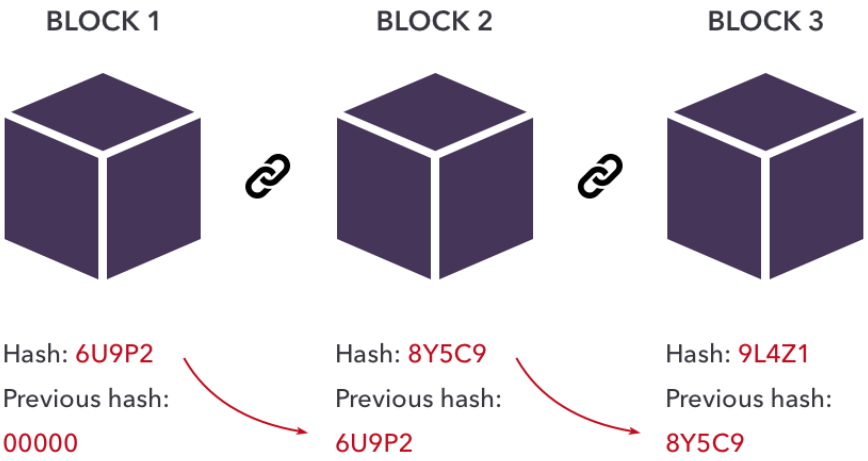



*Figure 1: Block connection in Blockchain [51]*

To maintain any cryptocurrency blockchain uses nodes. Nodes keep track of the distributed ledger and serve as communication hubs for various network tasks meaning that they ensure everyone interacting with the blockchain has the same data. Nodes must agree upon each new block and the chain. To accomplish that agreement, blockchains need a consensus mechanism. The consensus mechanism used by Ethereum is called „Proof-of-work". [10]

## 2.3 Proof-of-work (PoW) consensus mechanism

First published in 1993 by Cynthia Dwork and Moni Naor and later applied by Satoshi Nakamoto in 2008. Proof of Work consensus is the mechanism for most cryptocurrencies that exist. It allows the nodes of the network to agree on the state of all information recorded on the blockchain which prevents certain kinds of economic attacks. [11]

PoW works in a way that miners on a network compete against each other in solving complex computational puzzles. Despite puzzles being hard, their solutions are very easy to verify. Once a miner has found the solution, the block is broadcasted to the network, where all the other miners then verify that the solution is correct. Once the block is broadcasted to the network, the miner gets a reward. [10]

PoW helps to protect the network against numerous attacks. Because of PoW, a successful attack requires a lot of computational power and a lot of time to do the calculation, and therefore it's inefficient since the cost incurred is greater than the potential reward. [10]

The main issue with the PoW is that mining requires expensive computer hardware that consumes a big amount of power, so the cost of mining can sometimes be bigger than the potential reward. [10]

## 2.4 Ethereum

Published in 2015, Ethereum is a decentralized blockchain platform that creates a peer-to-peer network for safely executing and validating smart contract application code. Ethereum is programmable so anyone can build and deploy decentralized apps on its network. [12]

Users can build apps that store data on the blockchain and control what an app can do. Using the native Solidity scripting language and Ethereum Virtual Machine, Ethereum provides a very flexible environment for developing decentralized apps. [13]

Participants have complete ownership and visibility over transaction data thanks to immutable, verifiable, and securely distributed transaction records that are distributed throughout the network. As a cost of processing transactions on the network, a sender must sign transactions using the private account key and spend Ether, Ethereum's native cryptocurrency. [12]

To work for applications, Ethereum offers the computationally complete Ethereum Virtual Machine (EVM). EVM executes scripts across its network of distributed public nodes which provides the processing power for decentralized applications to run on the network. EVM compiles programming scripts into the bytecode and executes them on the blockchain. These scripts, known as Smart contracts, can be used as a type of digital contract to store the ledger of transactions for the Ether cryptocurrency. [13]

## 2.5 Smart contracts

Smart contracts are simple self-executed computer programs that run on the Ethereum blockchain. It is a set of data and code that is stored at a particular Ethereum blockchain address. One of the main parts of many blockchain-based ecosystems as well as one of the basic technology components of decentralized applications is smart contracts. [14]

A smart contract is a sort of Ethereum account meaning that it can send transactions over the network and has a balance. Instead of being controlled by a customer, they are distributed on the network and run as programmed. Like a standard contract, a smart contract can specify rules and use code to automatically enforce them. [14]

Smart contracts are fast, efficient, and accurate meaning that once all of the conditions are met, the contract is executed immediately. There is no documentation to sort through or time to spend fixing mistakes. Because there is no third party involved and the blockchain transaction records are encrypted, smart contracts are also transparent and safe. They function by using standard "if/when...then" statements that are written into code on a blockchain, limiting their functionality to one sort of transaction ("if something happens, then something else happens"). These procedures could include releasing funds to the appropriate parties, registering a vehicle, etc. The blockchain is updated after the transaction is finished. This indicates that the transactions are final and that only parties to whom permission has been granted can view the outcomes. [15], [16]

The most popular languages to write smart contracts are Solidity, Rust, JavaScript, Vyper, and Yul. [15]

## 2.6 Solidity

Solidity is an object-oriented programming language for implementing smart contracts proposed in 2014. by Gavin Wood. Designed to target the EVM and influenced by C++, Python, and JavaScript, Solidity belongs to the curly-bracket programming languages. It gives developers the possibility to create contracts for users such as voting, crowdfunding, multi-signature wallets, etc. [17]

Solidity is the primary language on Ethereum as well as on other private blockchains. To better understand the syntax of Solidity for creating smart contracts, the next code represents a simple Ethereum smart contract that only concatenates two strings. Code is written in Remix – Ethereum IDE which allows developing and administering smart contracts for Ethereum-like blockchains and it can help users to learn Solidity. [17]

The first line of code represents the license by which is the code represented and is written in a comment format. In this example, the license used is the MIT open source, no restriction license. The next line of code represents which solidity version is used in the code. The best practice is to use the most recent version of Solidity. *Pragma* keyword is used to enable certain compiler features and for compilers to know how to treat the code. Now, the newest version is 0.8.15. [18]

By adding a „^” symbol, it is ensured that Solidity can use any version above 0.8.15 up to 0.9.0 so the contract doesn't work if the new (breaking) compiler version is released. [18]

The next step is to define the contract name. As explained before, the contract is a collection of code and data and is defined by using a „contract” keyword and the name of the contract. Variables *initialText* and *name* are called state variables and they are a type string. State variables are variables whose values are permanently stored in contract storage. The keyword public allows users to access the value of the state variable from outside the contract. In this example, *initialText* has an initially set value, and the variable *name* yet has no defined value. The name is set in the constructor function. Constructor is a function that executes during the creation of the smart contract and can't be called later. It is typically used to initialize the token’s state using parameters passed by the contract’s deploying account. Keyword “memory” is used to temporarily hold values. After the function is called, memory is erased. [19], [20]

Under the constructor there are two functions named *declareName()* and *getContractCreatorText()*. Function *declareName()* is a public function that can be called by the user and what it does is change the name to whatever user types. Function getContractCreatorText() is a public function with a keyword „view” which means that the function will not modify the state of the contract (won't modify variables, emit events, etc.). [19]

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;
contract ExampleContract {
     string public initialText = "This contract is written by: ";
     string public name;

     constructor(string memory firstName) {
          name = firstName;
     }

     function declareName(string memory newName) public {
          name = newName;
     }

     function getContractCreatorText() public view returns (string
memory) {
          return string(abi.encodePacked(initialText, name));
     }
}
```

The next figure shows the output of the deployed contract. Before deploying, *initialName* was set to „Paolo" so when the *getContractCreatorText()* function is called, the output is „This contract is written by: Paolo". Whenever a state variable is defined (in this case *initialText* and *name*) solidity automatically creates the getter function which returns the values of those variables.

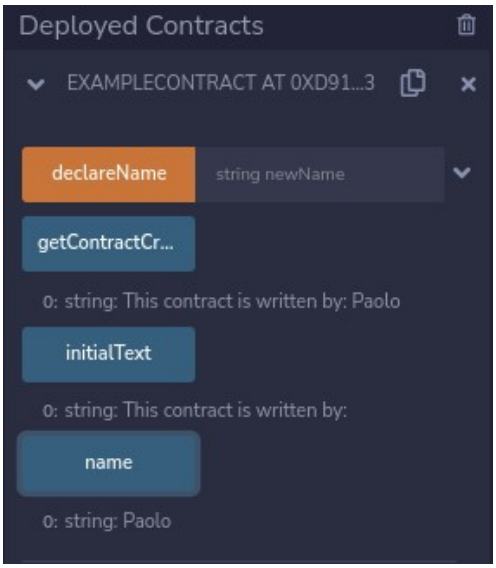



*Figure 2: Deployed contract (source: author)*

If a user wants to change the value of the *name* variable, there is an input field called *declareName* which will change it to whatever is typed in. Output with the name „Kos" is shown in the next figure.

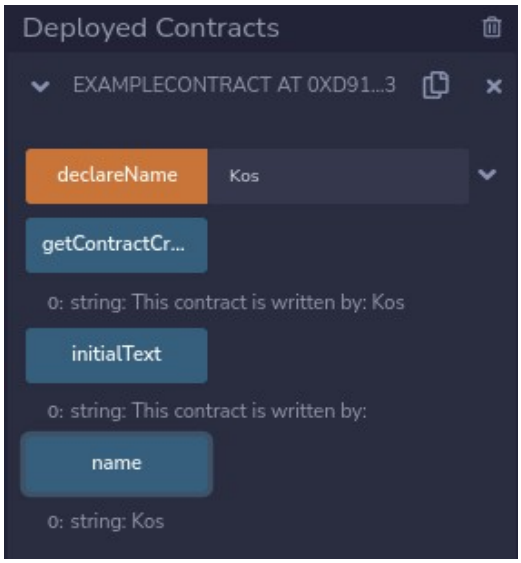



*Figure 3: Contract after changing name value (source: author)*

The code above is just a simple code written in Solidity, but writing real smart contracts is much more complex because it's used for transaction management which and how much funds are sent from one account to another, etc.

Programming smart contracts with Solidity has many advantages, one of them is that it has a large community because it was the first programming language for writing smart contracts so developers can get help whenever they need it. It's an easy-to-use, open-source, and object-oriented programming language which provides ABI (Application Binary Interface) to avoid syntax errors. [17]

### 2.6.1 Truffle Suite

Truffle Suite is a development environment based on Ethereum Blockchain and used for developing Distributed applications. It's used for compiling, deploying, and injecting smart contracts in Dapps. Using truffle developers get plenty of features to easily implement smart contracts, some of them are: [21]

- Built-in smart contract compilation and deployment
- Network management for deploying to public and private networks
- Instant asset rebuilding during development

- Scriptable deployment and migration of framework
- Automated contract testing

## 2.7 ERC20 Token Standard

Many Ethereum development standards help ensure that when a new project issues a token, it remains compatible with existing exchanges. Proposed in 2015 by Fabin Vogelstellar, and implemented in 2017, ERC-20 (Ethereum Request for Comments 20) is a way to standardize the tokens in smart contracts on the Ethereum blockchain. It's a standard interface for fungible tokens which are tokens that are interchangeable with other tokens like voting tokens, staking tokens, or virtual currencies. [22]

Functionalities that make smart contracts ERC-20 Token Contracts are: [22]

- Allows token transfers from one account to another
- Allows getting the current token balance of an account
- Allows getting a total supply of tokens available
- Approves whether a few tokens from an account can be spent by a third-party account

Once an ERC-20 token contract is deployed, it's responsible to keep track of the tokens on Ethereum. [23]

The token is just another word for „cryptocurrency” but a token can also be an asset or anything that is not unique and can be transferred. ERC-20 standard allows one token type to be exchanged for another token type i.e., one token that represents a lottery ticket can be exchanged for a token that represents an ounce of gold, etc. [22]

ERC-20 contains several functions and events that every token must implement. The minimum of functions needed in a token are: [23]

- totalSupply
- balaceOf
- transfer
- Transfer (an event)
- approve
- Approval (an event)

- allowance
- transferFrom

These functions assist in determining the number of tokens in circulation, storing and returning balances, making transfers, granting approvals, etc. To better understand each of these functions it will be explained through the next few lines of code which is a standard code for all smart contracts. [24]

Function *totalSupply()* returns the total number of tokens in existence regardless of owner. [24]

```
function totalSupply() public view returns (uint256) {

  return total;

}
```

Function b*alanceOf()* returns the number of tokens owned by an account. [24]

```
function balanceOf(address owner) public view returns (uint) {

  return balance[owner];

}
```

Function *transfer()* is used for transferring tokens from the owner's account, or to receive them. Functions get the owner's address and the number of tokens. When sending tokens, the function reduces the sender's balance by the number of tokens and when receiving tokens, functions add the received number of tokens. To ensure the account has a sufficient balance to execute the transfer, the function is using *require* keyword. If *require* fails, the transaction is rolled back. Before exiting the function, the *Transfer* event which allows registered listeners to react to its completion is fired. [24]

```
function transfer(address receiver, uint numberOfTokens) public
returns (bool) {

  require(numberOfTokens <= balance[msg.sender]);

  balance[msg.sender] = balance[msg.sender] — numberOfTokens;
```

```
  balance[receiver] = balance[receiver] + numberOfTokens;

  emit Transfer(msg.sender, receiver, numberOfTokens);

  return true;

}
```

Function *approve()* allows the owner to approve a delegate account (marketplace) to withdraw tokens from his account and transfer it to another account. It allows the marketplace to finalize transactions without prior approval. Before exiting the function it fires an *Approval* event. [24]

```
function approve(address delegate, uint numberOfTokens) public
returns (bool) {

allowed[msg.sender][delegate] = numberOfTokens;

  emit Approval(msg.sender, delegate, numberOfTokens);

  return true;

}
```

Function *allowance()* returns the currently approved number of tokens by an owner to a specific delegate which is set in an *approve()* function. [24]

```
function allowance(address owner, address delegate) public view
returns (uint) {

  return allowed[owner][delegate];

}
```

Function *transferFrom()* moves the number of tokens from *sender* to *recipient*. The *Require* statements verifies that the owner has enough tokens to transfer and that the delegate has the approval to withdraw. Before exiting the function, the *Transfer* event is fired. [24]

```
function transferFrom(address owner, address recipient, uint
numberOfTokens) public returns (bool) {
```

```
  require(numberOfTokens <= balance[owner]);

  require(numberOfTokens <= allowed[owner][msg.sender]);

  balance[owner] = balance[owner] — numberOfTokens;

    allowed[owner][msg.sender]    =    allowed[from][msg.sender]    —
numberOfTokens;

  balance[recipient] = balance[recipient] + numberOfTokens;

  Transfer(owner, recipient, numberOfTokens);

  return true;

}
```

The unctions above represent a valid but basic ERC20 implementation that is not fully secured but good enough to make simple token transfers. It ensures that Ethereum tokens of a different type will uniformly perform. [24]

## 2.8 Liquidity pools

To understand liquidity pools, it's important to know how the traditional stock market works and what liquidity is. The traditional stock market uses an order book model which is an electronic list of buy and sell orders organized by price level. That means that all the buyers and sellers set their orders of how much stock they want to buy and at what price. When the two buyers and sellers meet at the same price trade is transacted, the buyers get the desired stock, and the seller gets the cash. For example, if a person wants to sell his stocks immediately, he needs to sell them for the price someone else is willing to buy them for. If the person wants to sell it for a bigger price, he will need to wait for hours or maybe days for prices to match. If the prices never match in action come the market makers. Market makers are entities that are always willing to buy or sell an asset and by doing that, they provide liquidity so the users can always trade and don't have to wait for another counter-party. [25] [26]

Liquidity is the ease with which an asset can be converted to another asset without affecting its current market price. To better understand what liquidity is here is an example. [27]

If a person wants to buy a TV for $500, an asset that can most easily obtain it is cash. If a person doesn't have cash but has $500 worth of books, they are unlikely to find someone to trade the TV for books. The person will need to sell the books and use the cash they get to buy the TV. This will work if a person can wait for months or years to make the purchase, but if a person needs the TV as

soon as possible they may have to sell the books at a discount price. Books are an example of an illiquidity asset and cash is a liquid asset because it can most quickly be converted into another asset.

Reproducing this type of trading in decentralized finance (DEFI) is possible but it's slow and expensive. The reason for that is that the order book model relies on market makers and if there are no market makers, an exchange becomes illiquid making it unusable for users. The solution for the problem above is liquidity pools which are nothing more than a smart contract that is written to hold certain assets a person needs to trade. When it comes to decentralized exchanges (DEXs), liquidity pools play an essential role for users to pool their assets in a smart contract to provide asset liquidity for traders to swap between different currencies. [28]

A major component of the liquidity pool is automated market makers (AMMs). An AMM is a protocol that allows digital assets to be traded automatically and it supplies liquidity pools with tokens and the price of the tokens. To make liquidity pools work, in action come so-called liquidity providers (LP). LP is a user who deposits tokens into a liquidity pool and in return, they get provided with an award called a liquidity provider token. LP token represents the share of the liquidity pool that the LP owns, and the LP receives a percentage of liquidity pool trading fees. An example of how that works would be: [29]

1. The user goes to some decentralized crypto exchange i.e., Uniswap
2. The user finds the ETH – USDT liquidity pool
3. The user deposits a 50/50 split of ETH and USDT to that liquidity pool. In this case, $500 worth of ETH and $500 worth of USDT
4. As a reward, the user receives ETH - USDT liquidity provider tokens
5. Furthermore, the user can deposit LPTs to the ETH - USDT staking pool
6. The user get a token as a reward after the lockup period ends. The lockup period can last weeks or even months

The two biggest disadvantages of liquidity pools are that the AMM may reduce assets value and that one simple bug can lose users' funds forever. The AMM works by a simple formula: [30]

Balance of the token A x Balance of the token B = Balance of assets

If the balance of token A goes up, the balance of token B goes down. Therefore, if the purchasing power of an asset is too big, the user may be at loss. That behavior is called impermanent loss. [30]

Decentralized finance doesn't use third parties, so the custodian of users' assets is the smart contract. If there is a bug in a smart contract, users' funds will go down. [30]

## 2.9 Sniper bots

The term "Sniper bots" represents automated trading algorithms and software that monitor time-based activities and submit actions last moment so other people can't respond to them and by doing that it helps the user to buy and sell various cryptocurrencies in an automated way. [31]

## 2.10 Trading Bots

The term „Trading bots" represents simple programs that watch conditions on the market and place trades based upon predefined algorithms which allow users for automated and high-frequency trading. Most of the time, programs watch the market 24/7 and place trades following the defined algorithm, but the user still needs a solid strategy and a favorable market to do that. [32]

## 2.11 Liquidation Level

The liquidation level, which is typically represented as a percentage, is the threshold that, if met, will cause current holdings to be automatically closed. It is a security measure designed to stop investors from suffering substantial losses above a predetermined level and is often predetermined by the trader or the brokerage firm. [33]

## 2.12 Position

A position is the quantity of a security, asset, or piece of property that someone or some other entity owns (or has sold short). When a trader or investor places a buy order, indicating a bullish intention, or sells short securities, indicating a bearish intention, they are taking a position. [34]

Positions fall into two categories: [34]

- Long positions
- Short position

A long position is the most frequent one and involves holding a security or contract. Long positions profit from price increases and suffer losses from price declines. [34]

Short positions gain profit when the price of the underlying securities decreases. Frequently, a short position involves borrowing securities, selling them, and then buying them back, preferably at a lower price. [34]

# 3. Problem description

Even though using a sniper or trading bots sounds like a big money maker for the user, the damage that it can do to other users and the market itself is a large problem. Using computers to trade gives the user an advantage because it can respond thousands of times faster than a human ever could making it impossible for a human to compete in these trades and giving bot users an unfair advantage. [32]

Automated trading is not a new thing that started with cryptocurrencies, it was used long before through the history. An example is the famous Black Friday Wall Street crash in 1987. which was caused by the automated selling of stocks when they fell below a certain price causing a 22% fall in the index and thousands of people losing money. [35]

There are numerous ways how bot users can make money for themselves and at the same time lose money for other non-bot or even bot users. Those types of strategies are called “dirty strategies” because it manipulates the market so other users notice there is a chance to make money, make a transaction, and fall into a trap of those strategies. The most famous dirty strategies are Spoofing and Momentum Ignition. [32]

## 3.1 Spoofing

Term Spoofing (also called dynamic layering) indicates the process by which users attempt to influence the price of the crypto coin by creating fake orders. It creates an illusion of pessimism or optimism in the market. Spoofers place big buy or sell orders without the intention of filling them, so it gives other users the illusion of volume and active market manipulating them to buy or sell their coins. By doing so, the price of the cryptocurrency movies in their desired direction and they make a profit from it. Spoofing is done in just 3 steps: [36], [37]

1. Understanding the market
2. Fake buy wall
3. Canceling the order

To better understand spoofing next 3 paragraphs, recreate the whole process of it and show how spoofers make money by tricking the market. It’s important to understand that spoofing is illegal and that it’s a big problem on the market.

### 3.1.1 Understanding the market

First, the Spoofer must be an experienced trader which understands how the market (buying and selling) works. Spoofing is not beginner friendly and can cause money loss for new traders. The next figure represents the first step of Spoofing, “Understanding the market”. In this example, the coin used for spoofing will be an ORI token and the market is going down (price is dropping). There are 400 ORI tokens available for buying at the price of $20 (marked in the bottom red row), and 200 ORI tokens available for selling at the price of $19 (marked in the top green row). [38]–[40]

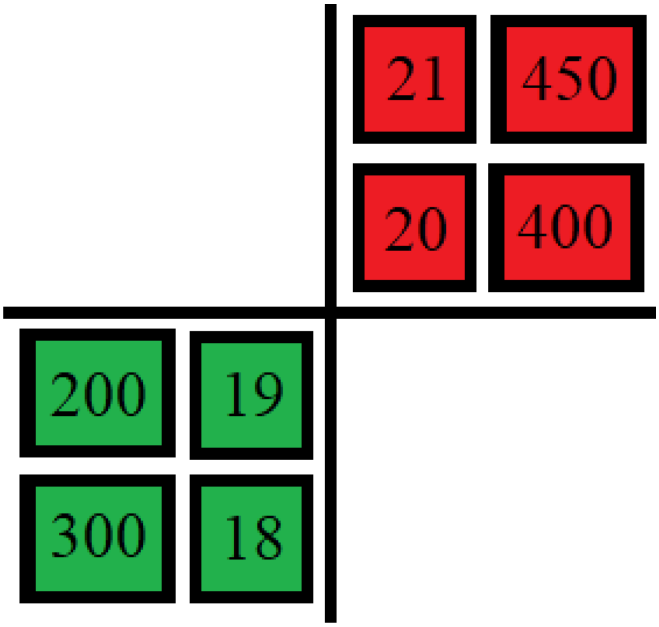



*Figure 4: Understanding the market (source: author)*

Spoofer usually holds a big number of coins and wants to sell them at the highest price possible. Seeing that the market is going down, Spoofer wants to sell his tokens as soon as possible not at the price of $19 but $20. To do so, the spoofer must move the market without placing a trade but tricking (spoofing) the market and other traders by putting a fake buy wall. [38]–[40]

### 3.1.2 Fake buy wall

Faking a buy wall means that the Spoofer puts a bunch of buy orders with no attention to buying crypto at those levels. In the example shown in the figure below, on the second level of the selling price (marked in the second green row), there are a lot of volumes incoming for the price of $18.

This volume indicates to other traders that someone (spoofer) is willing to buy crypto at that level and for that price. On the right side of the figure, the graph represents the movement of the token price. The blue marker displays what's happening with the price when the spoofer fakes buy wall. The market is tricked into thinking that it's come to the end of the fall because there is a big buy wall happening so what the market does is that they start buying in, which "eats up" the price of $20 with 400 coins available and moves the price of the token up. [38]–[40]

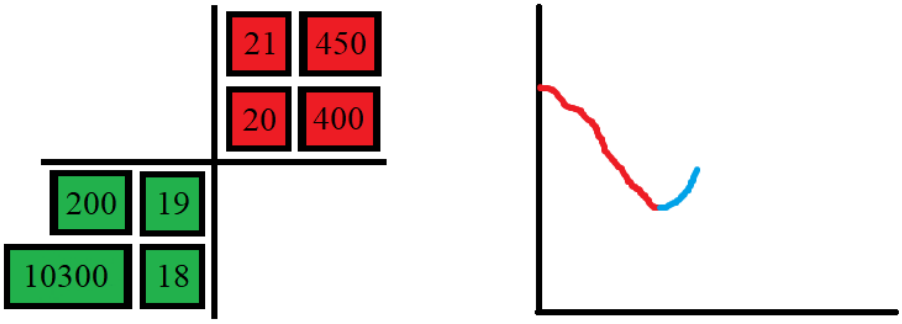



*Figure 5: Fake buy wall (source: author)*

### 3.1.3 Canceling the order

When the price of the token goes up, the spoofer immediately cancels his selling orders for the price of $18 and he can sell his tokens for the price of $20. The next figure represents what's happening with the market at that point. On the right side is the graph of the market price after spoofing. [38]–[40]

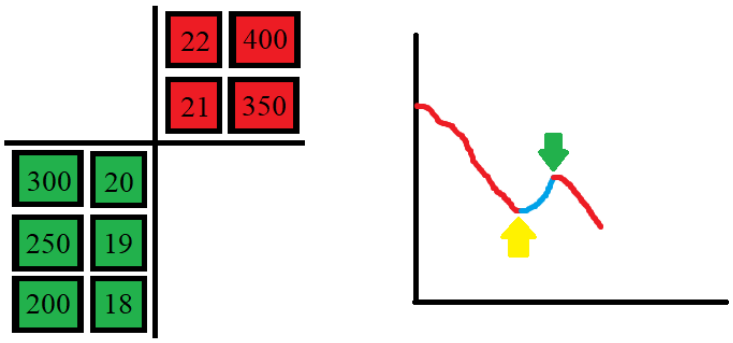



*Figure 6: Market after spoofing (source: author)*

As seen in the figure above, the highest selling price is now $20 (top green row) and after the spoofer sells all his tokes, what is happening with the market is that it crashes because of the high selling volume. The yellow arrow represents the price before spoofing and the green arrow represents the price after spoofing by which the spoofer sold his tokens. Users which are being spoofed lost their money because they thought the price will continue to rise, bought the token, and after spoofing is done the price crashes and they lose a lot of money. [38]–[40]

Spoofing can be done manually but it's harder to buy and sell that quickly, so it's mostly done automatically. The program detects the price of the token, sets fake buy orders, and waits for the market price to go up. After the price reaches a certain point, the program automatically cancels its orders and sells the token for the bigger price making the spoofer money and losing money for other traders. Spoofing can be done in both, buy and sell walls and it's hard to detect. [38]–[40]

### 3.1.4 Spoofing in real life

Best know spoofing attack happened on May 6th, 2010. and it's called "Flash Crash". Navinder Sarao, the UK-based financial trader, later pleaded guilty for spoofing, managed to "earn" $70 million from trading out of his bedroom. His spoofing method caused the 30-minute crash and trillion-dollar loss of the US market. [41]

According to the Commodity Futures Trading Commission, Sarao used automated programs to generate big sell orders and push the price down and then canceled orders at the lower market prices. As seen in the next figure, the blue line represents the big drop that happened because of spoofing. The market suffered a large drop in price in a very short time and later it slowly started to recover. [41]

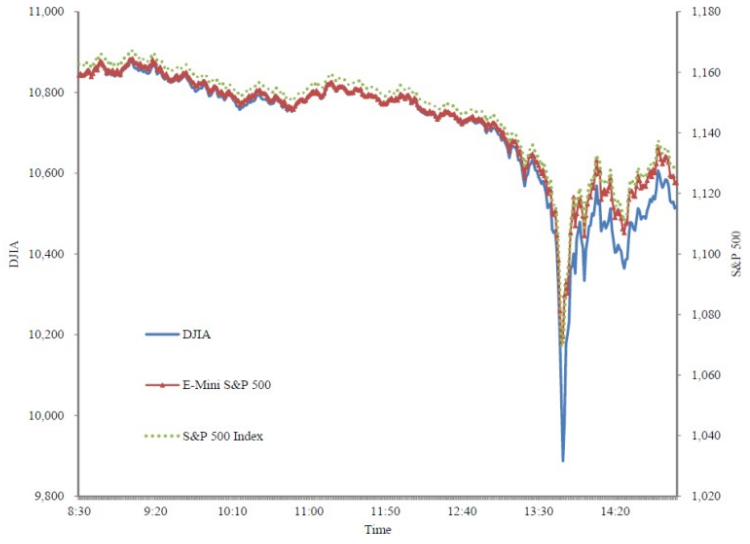



*Figure 7: Flash crash [41]*

The aftermath of the “Flash Crash”, except a trillion-dollar loss, is that for a short time, 8 major companies fell to one cent. Some of these companies were Accenture, CenterPoint Energy, and Exelon. On the other hand, companies like Apple Inc. and Hewlett-Packard increased in value to over $100 000 in price. [41]

## 3.2 Momentum ignition

The second “dirty strategy” is called “Momentum ignition” and it refers to the strategy or algorithm that causes sharp spikes on either sell or buy action. It can be done manually or automatically, and its purpose is to encourage other traders to trade quickly causing a rapid price move. Momentum ignition traders can spike the price up or down causing other algorithms or traders to indicate those moves and respond to them by buying or selling their tokens or even worse, losing their money by triggering liquidation levels. [42]

Momentum ignition is, the same as Spoofing, done in 3 steps: [43]

1. Understanding the market
2. Making a buy/sell trade
3. Selling on profit

### 3.2.1 Understanding the market

The next figure represents the starting point of the price. As seen, the price of the token is constant and doesn’t have many “spikes” in the positive or negative direction. That means that all other traders are trading equally and other algorithms/bots which are waiting for buying or selling opportunities are probably not triggered. The purple line represents the top price of the token. Assuming that the momentum ignition trader wants to make a profit while buying, he must buy a large number of tokens to make the price go up, closer to the top price. [42], [44], [45]

*Figure 8: Market price moving before the ignition (source: author)*

### 3.2.2 Making a buy/sell trade

Figure 9 represents the buy trade that the momentum ignition trader made. Because of his large trade, the price of the token starts to move up. Price moving up is represented with green color. [40]

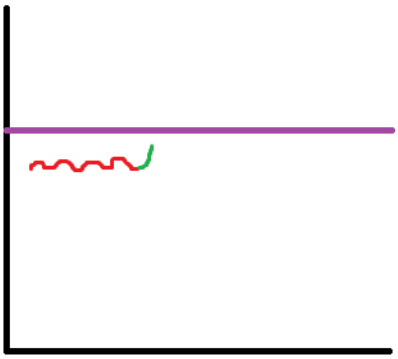


*Figure 9: Making a large buy trade (source: author)*

At that point, other traders see the price moving up quickly and decide to buy tokens so also they can make a profit. Also, other bot traders got the signal of the price moving up and their buying algorithms got triggered and bought more tokens. Those actions (represented in yellow color) cause the momentum ignition which in most cases causes the price to break through the top margin (figure 10). [42], [44], [45]

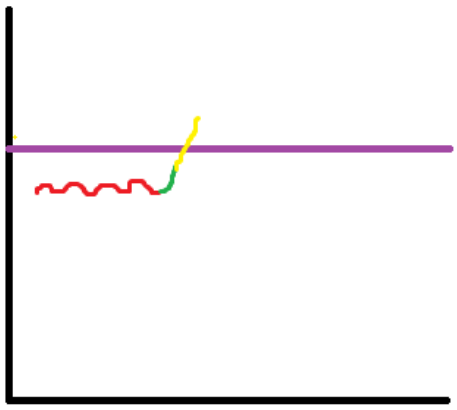


*Figure 10: Momentum ignition (source: author)*

### 3.2.3 Selling on profit

The last step of a successful momentum ignition strategy is selling the tokens for profit. Figure 11 represents the movement of the price after selling (marked blue). Because of the high buy volume, the price will surely quickly drop down to where it was before or even lower causing other regular or bot traders (if they don't sell the token) to lose money. The pink arrow represents the price at which the momentum ignition trader bought the token, and the orange arrow represents the price at which the momentum ignition trader can sell his tokens. Price stays on top for a while but suddenly makes a quick drop. [42], [44], [45]

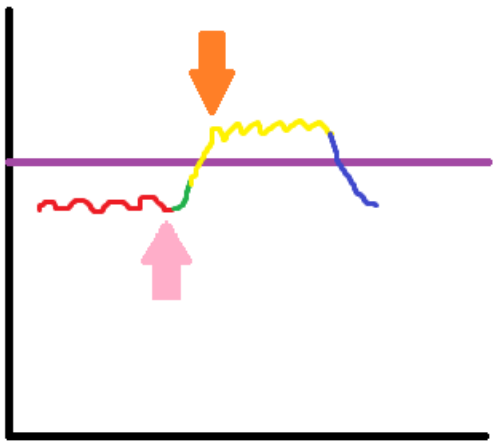


*Figure 11: Token price movement after momentum ignition strategy (source: author)*

If a momentum ignition trader wants to make a profit from selling his tokens, the process is the same as buying but much worse for other traders. In that scenario, if the price breaks the low margin, it will most probably make traders or bots sell tokens for the lower price, losing them money and creating a selling momentum ignition. Momentum ignition trader sells a large number of tokens and waits for the market response. Then the momentum ignition trader, knowing that the price will go back to where it was before, makes a buy trade and waits for the price to normalize so he can again sell his tokens. Figure 12 shows the process of selling momentum ignition strategy. [42], [44], [45]

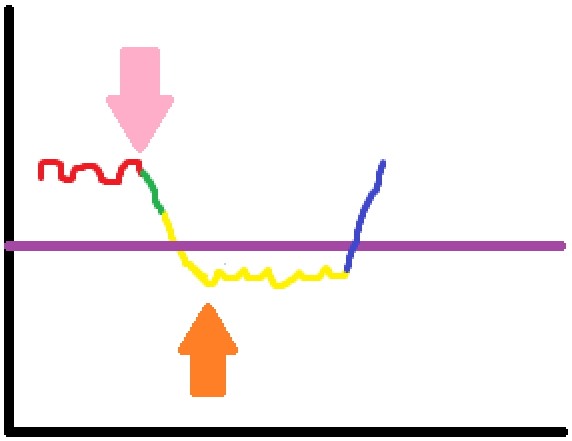


*Figure 12: Selling momentum ignition (source: author)*

### 3.2.4 Momentum ignition in real life

Even though the momentum ignition strategy is on the verge of being illegal, it's a problem that happens regularly. Credit Suisse AES Analysis reported that in the third quarter of 2012. there was an average of 1.6 times per day momentum ignition manipulation on the market with almost every stock exhibiting this pattern once a day or more. The next figure shows the price movement in momentum ignition represented by Credit Suisse AES Analysis. [46]


*Figure 13: Momentum ignition example [46]*

# 4. Problem solutions proposals

## 4.1 Solution 1: “Buffer”

As seen in the problem description, traders use their ability to buy or sell a larger amount of tokens to move the price in the desired direction. The first proposal to stop that behavior is a program that will check all the new incoming trade offers before they go to the market, calculate the average price of the trades when the new trade comes in, and compare it to the previous average using delta and based of that comparison, let the trade go through or stop it before it spikes the market price up or down.

### 4.1.1 Creating a buffer

Storing all the incoming trade offers can make the trading process very slow and therefore spoil the user experience and ruin the trading itself. Instead of storing all of the transactions in one place, the proposal is to use a buffer.

A buffer is nothing more than a container that will receive only a certain number of trade offers and use them exclusively without using “old” offers. For example, a buffer can store 500 trades. When the 501st trade comes in, the buffer will remove the 1st trade that was in it and store the 501st as the 500th and all other trades will be moved by 1 place up. By doing that, the buffer will always have 500 trade offers to use in the further process. The next figure represents the buffer loading and unloading trade offers.

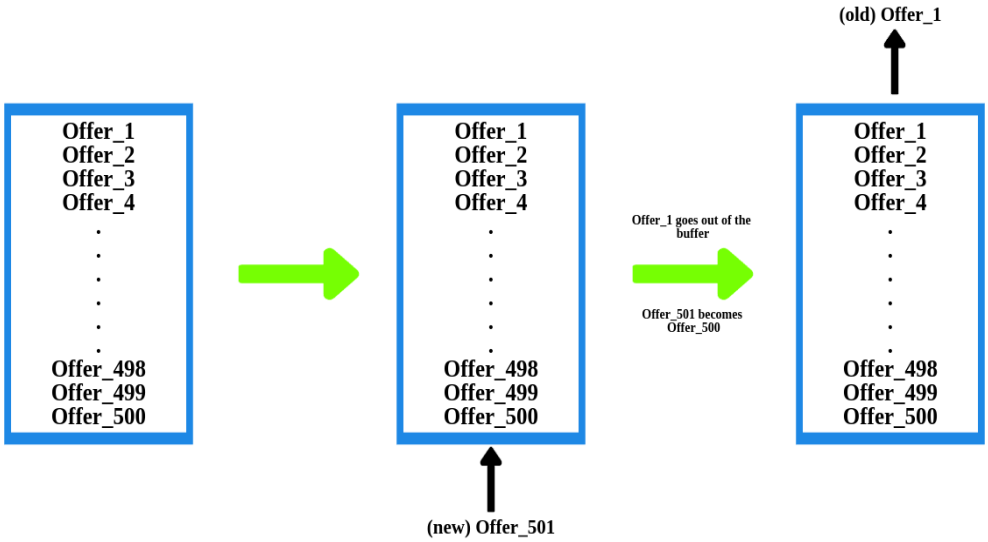



*Figure 14: Buffer (source: author)*

### 4.1.2 Calculating the average price

Every trade offer has a price that regulates the price of the token itself. When the buffer stores a trade offer, the price is stored too. The average price will be calculated each time a new trade offer comes into a buffer. When a new offer is added to the buffer, and the oldest one removed, it will take all the prices before the new offer, add their prices, add the price of the new offer to their sum price and calculate the average by dividing the sum of all the prices with the number of trade offers in the buffer. To calculate the average of all prices, the proposal is to use the following formula:

$$A(O) = \Sigma(P) / N$$

where A(O) is the average of the current offer, Σ(P) is the sum of all the prices in the buffer and N is the number of the trade offers in the buffer.

By calculating the average price, every offer will have its fixed average price so in the next step delta of the average price can be calculated.

### 4.1.3 Calculating the delta of the average price

The last step of determining whether a trade offer is secure to be executed or not is to calculate the delta of the averages. Delta can be calculated by the following formula:

$$\Delta O = A(O) - A(O-1)$$

where ΔO is the delta of the current trade offer, A(O) is the average price of the current trade offer, and A(O-1) average price of the trade offer before the current one.

When the delta is calculated, the result is a positive or negative number that adds up to the total price amount. Every token can decide what value of delta is top and bottom margin. If the incoming trade is large enough, the delta will go over the margin and the price will move up or down quickly. To prevent high trade offers, the proposal is to cancel them if they go over the margin of the delta.

### 4.1.4 Code

For showing how the solution works, what is created is a JavaScript simulation of how trade offers are getting into a buffer and delta is being calculated. At first, the buffer is empty, the token price is $50, and the maximum allowed number of trade offers in the buffer is 10.

The first 8 lines of code are variable declarations. Variable *tradesArray* is an array that represents a buffer. The array named a*veragesArray* is an array that contains all the calculated averages and *deltasArray* is an array that contains all of the calculated deltas. Variable *startingPrice* is a variable that is set to 50 and it represents the starting price of the token. Variable *endPrice* is also set to 50, but it represents the price that will change with each new trade offer. Variables *delta* and *average* represent each delta and average calculated which are later pushed to the arrays. The variable *id* represents the id of each trade offer and it's set to 0.

```
let tradesArray = [];
let averagesArray = [];
let deltasArray = [];
let startingPrice = 50;
let endPrice = 50;
let delta;
let average;
let id = 0;
```

The main functionality of the script is in the function called *executeTrade(trade)*. First, the function gets the value of the user input as a *trade* parameter. Then, it checks whether there is no user input or input equals 0, and trade is submitted. If it is, the browser sends an alert saying, "Enter a valid number bigger than 0" and exits the function immediately.

```
function executeTrade(trade) {
   if(!trade || trade == 0) {
    alert('Enter a valid number bigger than 0!')
    return false;
  }
```

The next if statement checks if the buffer contains more than 10 trade offers. If it does, it removes the first (oldest) trade offer with a *shift()* function. Function *shift()* removes the first element of an array and returns it. [47]

```
if(tradesArray.length > 9) {

     tradesArray.shift();

};
```

After the trade passes all the checks it's pushed to a buffer with *push()* function. Function *push()* adds an element to the end of an array and returns the new length of the array. [tu] After trade is pushed to an array, functions called are *calculateAverage()*, *calculateDelta(trade)*, and *calculateFinalPrice()*. At the end of the *executeTrade()* function, id is increased by 1.

```
tradesArray.push(trade);
calculateAverage();
calculateDelta(trade);
calculateFinalPrice();
id += 1;
```

The reason why the *executeTrade()* function is important is that it is a function that is called first and contains 2 if statements that regulate if the trade will happen or not and if the buffer is full so it can change the old offer with the new one. It also calls all the other functions for calculating the final price. Function *calculateAverage()* as the name of the function suggests, calculates the average price of all of the incoming trade offers. A variable *sum* is created and set to 0. It's a variable with the function of calculating the sum of the trade offers in the buffer. For statement loops through all the trades in an array of trades and takes each trades value and adds it to the *sum* variable. Every trade offer is added up to the *sum* after it's processed in the *parseInt()* function. The *parseInt()* parses a string argument (in this case a user's input) and returns it as an integer with a specified base (in this case base is 10). [48]

```
function calculateAverage() {
let sum = 0;
for(let item of tradesArray) {
     sum += parseInt(item, 10);
}
```

The function then calculates the average price of the trade offers. Variable *average* takes the value of the variable *sum* and divides it by the length of the *tradesArray* which is the number of the trades in a buffer (in this case, 10). Average is then formatted to 2 decimal places using the *toFixed()* function and pushed to the *averagesArray* as a number. The function returns the average price of each iteration.

```
average = sum / tradesArray.length;
average = average.toFixed(2)
averagesArray.push(Number(average))

return average;
}
```

The function to calculate delta is called *calculateDelta(trade)* and it takes the value of the trade as an argument. Variable delta gets the value of delta from *averagesArray* array by slicing it and then mapping through it. Method *slice()* returns the copy of a portion of an array into a new array. Number 1 in the *slice()* method represents the start index of the item in an array that is going to be excluded. [49] The *map()* method creates a new array which is populated with the results of calling a provided function on every element in an array. It returns the subtraction of the current element with the previous one. [50] Variable *delta* gets the value of the last element pushed to the array of deltas. When the first trade is submitted, an array of deltas will be empty because the average of one trade can't be calculated. If that's the case, the delta gets the value of the trade (the average price will also be the same as the first trade offer).

```
function calculateDelta(trade) {
  deltasArray = averagesArray.slice(1).map(function(n, i) {
     return n - averagesArray[i]; });

  delta = deltasArray[deltasArray.length-1]
  if(!deltasArray.length) {
    delta = trade;
  }
```

The last part of the *calculateDelta()* function is the validation of the trade. Both, the top and bottom margin can be calculated and move up and down depending on the token price or it can be hardcoded. For example, if the delta is bigger than 10, or smaller than -10, the browser will send an alert saying, "Trade too high", disable the submit button for that user, and exit the function.

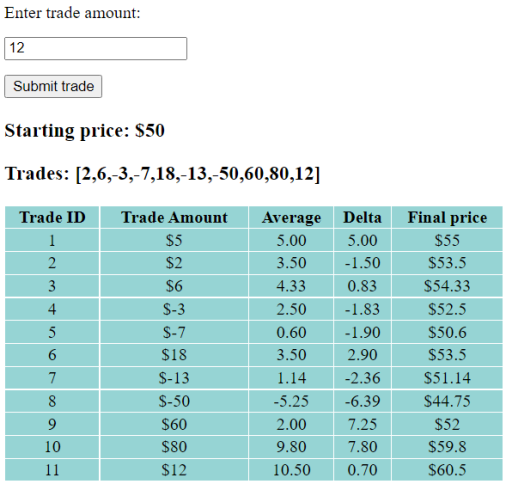
```
if(delta > 10 || delta < -10) {
     alert('Trade too high!');
     disableSubmitButton();
     return false;
}
```

Function *calculateFinalPrice()* takes the last calculated delta, adds it up to the end price, and saves it to the *endPrice* variable. In the end, the function returns the end price.

```
function calculateFinalPrice() {
  endPrice = endPrice + delta;
  return endPrice;
}
```

### 4.1.5 Interface

The testing interface is created in HTML and CSS. It consists of an input field where the user can type the positive or negative number (depending if he wants the buy or sell simulation), submit button to submit a trade offer, „Starting price“ to display what is the price of the simulated token before the buffer is started, and „Trades“ to display the functionality of the buffer and the table which displays every submitted trade offer. As seen in the next figure, the interface shows 11 trade offers, but the buffer consists only of 10 trades.




| Trade ID | Trade Amount | Average | Delta | Final price |
|---|---|---|---|---|
| 1 | $5 | 5.00 | 5.00 | $55 |
| 2 | $2 | 3.50 | -1.50 | $53.5 |
| 3 | $6 | 4.33 | 0.83 | $54.33 |
| 4 | $-3 | 2.50 | -1.83 | $52.5 |
| 5 | $-7 | 0.60 | -1.90 | $50.6 |
| 6 | $18 | 3.50 | 2.90 | $53.5 |
| 7 | $-13 | 1.14 | -2.36 | $51.14 |
| 8 | $-50 | -5.25 | -6.39 | $44.75 |
| 9 | $60 | 2.00 | 7.25 | $52 |
| 10 | $80 | 9.80 | 7.80 | $59.8 |
| 11 | $12 | 10.50 | 0.70 | $60.5 |

*Figure 15: Interface (source: author)*

If the offer is big enough to make the script block the next trade submission, the browser will give an alert, and the „Submit trade“ button will become unclickable. Trade offers won’t be displayed in the buffer or the table. Browser alert and forbidden trade are shown in the figures below.

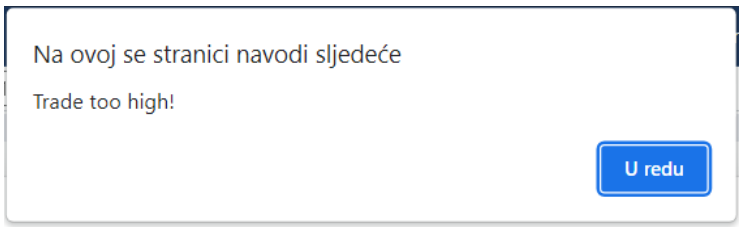
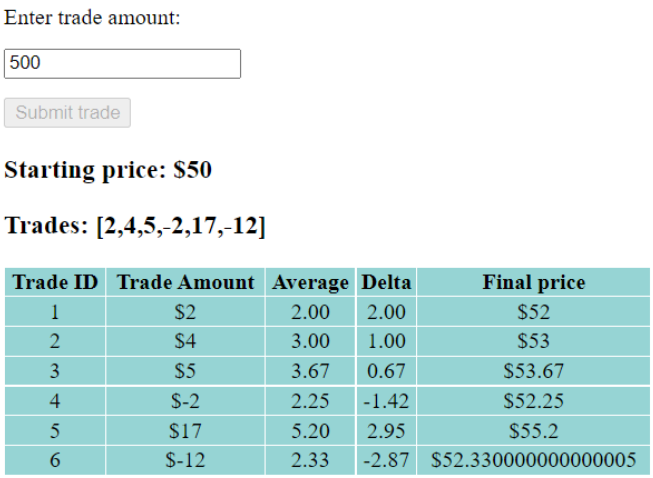



*Figure 16: Browser alert (source: author)*




| Trade ID | Trade Amount | Average | Delta | Final price |
|---|---|---|---|---|
| 1 | $2 | 2.00 | 2.00 | $52 |
| 2 | $4 | 3.00 | 1.00 | $53 |
| 3 | $5 | 3.67 | 0.67 | $53.67 |
| 4 | $-2 | 2.25 | -1.42 | $52.25 |
| 5 | $17 | 5.20 | 2.95 | $55.2 |
| 6 | $-12 | 2.33 | -2.87 | $52.330000000000005 |

*Figure 17: Forbidden trade (source: author)*

For the invalid input, the user gets the message shown in the figure below.

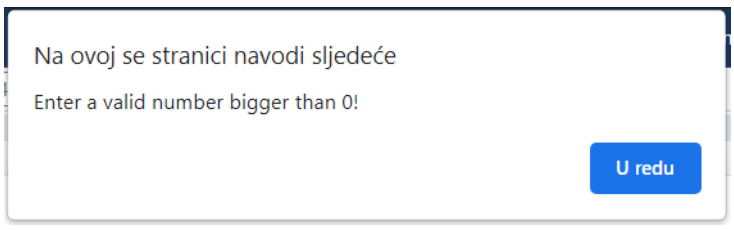



*Figure 18: Invalid input (source: author)*

## 4.2 Solution 2: „Forbid high orders“

The next solution proposal is suitable for contract owners that don’t want big trade offers on their tokens so they can avoid dirty strategies being used. The solution is that only acceptable trades are trades that are not larger than a certain percentage of a token price. For example, if the token price is $10000, the only acceptable trade offers are those up to $1000. The formula used for the calculation is:

$$Max(O) = P * 0.1$$

where Max(O) is the maximum offer acceptable and the P price of the token.

### 4.2.1 Code

The first part of the code is a variable declaration. Variable *ordersArray* is an array type variable that will contain all of the orders. Variable current*Price* is set to 10000 and it represents the current price of the token, *validOrder* is a boolean variable set to true and will be used to check if the submitted order is valid. The variable *id* represents the id of each order and is set to 0.

```
let ordersArray = [];

let currentPrice = 10000;

let validOrder = true;

id = 0;
```

Function *placeOrder(order)* is a function that gets the parameter order which is the value of the order submitted. Function first checks if the order submitted is valid, if it’s not, a message will be displayed and the function will be stopped. The function then calls the *checkOrder(order)* function which will be explained below. If an order is invalid, the function stops. Finally, if the order is valid it’s pushed to the array of orders and the id is increased by 1.

```
function placeOrder(order) {

  if(!order || order == 0) {
```

```
    alert('Enter a valid number bigger than 0!');

    return false;

  }
  checkOrder(order)

  if(!validOrder) {

    return false;

  }

  ordersArray.push(order);

  id += 1;

}
```

Function checkOrder(order), takes the value of the order as a parameter and it's used for checking if the order is higher or lower than 10% of the current price of the token. If it is, the order becomes invalid, the message is displayed and the function ends. If an order is in the allowed range, the order is valid.

```
function checkOrder(order) {

  if(order > currentPrice*0.1 || order < -currentPrice*0.1) {

    validOrder = false;

    alert('Invalid order!');

    return false;

  } else {

    validOrder = true;

  }

}
```

### 4.2.2 Interface

The next figure represents the interface for the solution written above. User inputs the order which is then displayed in a table. The table consists of the order id and order amount. Users can't enter invalid orders and can enter positive or negative numbers depending on if the user wants to simulate a buy or sell order.

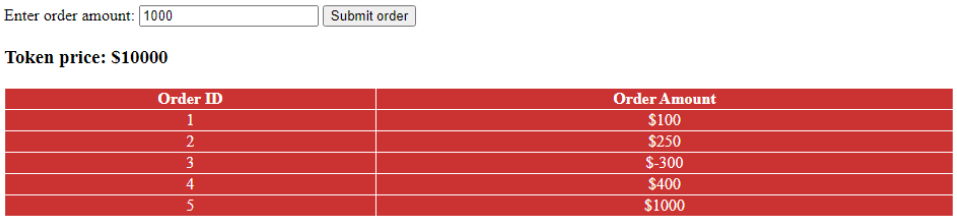



*Figure 19: „Forbid high orders" interface (source: author)*

If the order submitted is 10% larger than the token price (in the case above, larger than $1000), the message is displayed and the order is not pushed to the table. An invalid order message (for the order of $1001) is shown in the figure below.

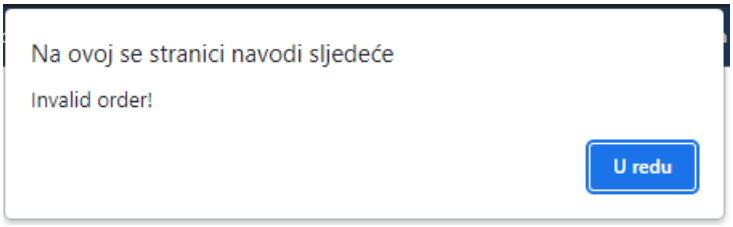



*Figure 20: Invalid order message (source: author)*

# 5. Conclusion

In the last few years, people are witnessing a rapid development of technologies. Traditional transactions are more and more being replaced by online payments, mostly cards, but since 2008 and the introduction of a cryptocurrency called Bitcoin, online transactions have reached a higher level. Since then, due to the anonymity of transactions, the possibility of earning money, and many other benefits provided by cryptocurrencies, millions of transactions are made every day using them.

Trading is a big part of cryptocurrencies and it brings great benefits to people who know how to use it. Despite the many positive things of trading, there are also negative ones, which a person can't influence. Two main problems addressed in this paper are "Spoofing" and "Momentum ignition strategy". Both use methods of buying or selling a high volume of tokens to manipulate the token price and by doing that make a profit for the user and lose money for other trades. The difference is that spoofers use fake orders offers, and momentum ignition traders place real buy or sell orders to manipulate the price.

Provided solutions for those problems, written in pseudocode are "Buffer" and "Forbid high orders". Both solutions are optional and can be used only for specific scenarios. The contract owner needs to decide whether he needs one or maybe both solutions and implement them in their smart contract. Solution "Buffer" works for any type of contract, and the "Forbid high orders" solution is good for those who don't want high buy orders to stop the easy price manipulation.

# 6. Literature


[1] “What Is Cryptocurrency?,” *Investopedia*. https://www.investopedia.com/terms/c/cryptocurrency.asp (accessed Aug. 01, 2022).
[2] K. Ashford, “What Is Cryptocurrency?,” *Forbes Advisor*, Nov. 20, 2020. https://www.forbes.com/advisor/investing/cryptocurrency/what-is-cryptocurrency/ (accessed Aug. 01, 2022).
[3] “Distributed Ledgers,” *Investopedia*. https://www.investopedia.com/terms/d/distributed-ledgers.asp (accessed Aug. 01, 2022).
[4] K. Tretina, “How To Buy Cryptocurrency,” *Forbes Advisor*, Jul. 12, 2021. https://www.forbes.com/advisor/investing/cryptocurrency/how-to-buy-cryptocurrency/ (accessed Aug. 01, 2022).
[5] “What Is a Cryptocurrency Exchange? A Beginner’s Guide for 2022,” *MintLife Blog*, Oct. 18, 2021. https://mint.intuit.com/blog/investments/what-is-a-cryptocurrency-exchange/ (accessed Aug. 01, 2022).
[6] “Cryptocurrency Exchanges,” *Corporate Finance Institute*. https://corporatefinanceinstitute.com/resources/knowledge/other/cryptocurrency-exchanges/ (accessed Aug. 01, 2022).
[7] H. B. Studio, “5 Best Decentralised Exchanges (DEXs) in 2022,” *mint*, May 27, 2022. https://www.livemint.com/brand-stories/5-best-decentralised-exchanges-dexs-in-2022-11653655894925.html (accessed Aug. 01, 2022).
[8] Bitpanda, “Difference between a cryptocurrency broker and an exchange.” https://www.bitpanda.com/academy/en/lessons/the-difference-between-a-cryptocurrency-broker-and-an-exchange (accessed Aug. 01, 2022).
[9] “Blockchain Facts: What Is It, How It Works, and How It Can Be Used,” *Investopedia*. https://www.investopedia.com/terms/b/blockchain.asp (accessed Aug. 01, 2022).
[10] “Proof of Work (PoW),” *Investopedia*. https://www.investopedia.com/terms/p/proof-work.asp (accessed Aug. 01, 2022).
[11] “Proof of work,” *Wikipedia*. Aug. 15, 2022. Accessed: Aug. 24, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Proof_of_work&oldid=1106420209
[12] “What is Ethereum?,” *ethereum.org*. https://ethereum.org (accessed Aug. 01, 2022).
[13] “What Is Ethereum? | AWS Blockchain,” *Amazon Web Services, Inc.* https://aws.amazon.com/blockchain/what-is-ethereum/ (accessed Aug. 01, 2022).
[14] “Introduction to smart contracts,” *ethereum.org*. https://ethereum.org (accessed Aug. 12, 2022).
[15] “What are smart contracts on blockchain? | IBM.” https://www.ibm.com/topics/smart-contracts (accessed Aug. 24, 2022).
[16] “Real World Examples of Smart Contracts,” *Gemini*. https://www.gemini.com/cryptopedia/smart-contract-examples-smart-contract-use-cases (accessed Aug. 24, 2022).
[17] “Solidity — Solidity 0.8.15 documentation.” https://docs.soliditylang.org/en/v0.8.15/ (accessed Aug. 18, 2022).
[18] “Layout of a Solidity Source File — Solidity 0.8.15 documentation.” https://docs.soliditylang.org/en/v0.8.15/layout-of-source-files.html#pragma (accessed Aug. 06, 2022).
[19] “Solidity - Variables.” https://www.tutorialspoint.com/solidity/solidity_variables.htm# (accessed Aug. 18, 2022).

[20] “Frequently Asked Questions — Solidity 0.2.0 documentation.” https://docs.soliditylang.org/en/v0.3.3/frequently-asked-questions.html#what-is-the-memory-keyword-what-does-it-do (accessed Aug. 05, 2022).
[21] “Home - Truffle Suite.” https://trufflesuite.com/ (accessed Aug. 05, 2022).
[22] “ERC-20 Token Standard | ethereum.org.” https://ethereum.org/en/developers/docs/standards/tokens/erc-20/ (accessed Aug. 09, 2022).
[23] “What Is ERC-20?,” *Investopedia*. https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/ (accessed Aug. 08, 2022).
[24] “How to Create an ERC20 Token the Simple Way,” *Toptal Engineering Blog*. https://www.toptal.com/ethereum/create-erc20-token-tutorial (accessed Aug. 07, 2022).
[25] “What Is the Stock Market and How Does It Work?,” *Investopedia*. https://www.investopedia.com/terms/s/stockmarket.asp (accessed Aug. 16, 2022).
[26] “Order Book,” *Investopedia*. https://www.investopedia.com/terms/o/order-book.asp (accessed Aug. 10, 2022).
[27] “What Is Liquidity?,” *Investopedia*. https://www.investopedia.com/terms/l/liquidity.asp (accessed Aug. 11, 2022).
[28] “What Is a Liquidity Pool? Crypto Market Liquidity,” *Gemini*. https://www.gemini.com/cryptopedia/what-is-a-liquidity-pool-crypto-market-liquidity (accessed Aug. 15, 2022).
[29] M. Antolin, “What Are Liquidity Pools?,” Jun. 07, 2022. https://www.coindesk.com/learn/what-are-liquidity-pools/ (accessed Aug. 09, 2022).
[30] “The Advantages and Disadvantages of Liquidity Pools,” *Tokenhell*, Jun. 26, 2021. https://tokenhell.com/the-advantages-and-disadvantages-of-liquidity-pools/ (accessed Aug. 10, 2022).
[31] “What Are Sniper Bots? | An Overview of Sniping Bots | Netacea.” https://www.netacea.com/glossary/sniper-bots/ (accessed Aug. 20, 2022).
[32] F. Edwood, “How bot trading influences the crypto market, explained,” *Cointelegraph*, Aug. 31, 2020. https://cointelegraph.com/explained/how-bot-trading-influences-the-crypto-market-explained (accessed Aug. 15, 2022).
[33] “Liquidation Level Definition,” *Investopedia*. https://www.investopedia.com/terms/l/liquidationlevel.asp (accessed Aug. 11, 2022).
[34] “Position,” *Investopedia*. https://www.investopedia.com/terms/p/position.asp (accessed Aug. 13, 2022).
[35] “Stock Market Crash: 1929 & Black Tuesday - HISTORY - HISTORY.” https://www.history.com/topics/great-depression/1929-stock-market-crash (accessed Aug. 22, 2022).
[36] “Cryptocurrency and Spoofing,” *CryptoCurrency Facts*. https://cryptocurrencyfacts.com/cryptocurrency-and-spoofing/ (accessed Aug. 12, 2022).
[37] “Spoofing - Definition, Understanding, and Why Spoofing is Important?” https://cleartax.in/g/terms/spoofing (accessed Aug. 22, 2022).
[38] “How to Catch a Spoofer,” *Bloomberg.com*. Accessed: Aug. 18, 2022. [Online]. Available: http://www.bloomberg.com/graphics/2015-spoofing/
[39] *What is Order Spoofing - Trading OrderFlow*. Accessed: Aug. 22, 2022. [Video]. Available: https://www.youtube.com/watch?v=R9MJ0MDlQjk&ab_channel=NinjaTrader
[40] *Crypto Trading: Market Manipulation (Spoofing)*. Accessed: Aug. 22, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=VU3HSjOf2BM&t=342s&ab_channel=CryptoWizards
[41] A. A. Kirilenko, A. S. Kyle, M. Samadi, and T. Tuzun, “The Flash Crash: The Impact of High Frequency Trading on an Electronic Market,” *SSRN Electron. J.*, 2011, doi: 10.2139/ssrn.1686004.

[42] “Kx Product Insights: Momentum Ignition Alert,” *KX*. https://kx.com/blog/kx-product-insights-momentum-ignition-alert/ (accessed Aug. 18, 2022).
[43] “Momentum Ignition Detection » Neurensic,” *Neurensic*, Sep. 13, 2016. https://neurensic.com/momentum-ignition-detection/ (accessed Aug. 22, 2022).
[44] *Dirty Strategies in Algorithmic Trading: Momentum Ignition Strategy*. Accessed: Aug. 18, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=aAYAh9e7KxA&t=165s&ab_channel=UKspreadbetting
[45] *Momentum Ignition: The Dark Art of Accumulation*. Accessed: Aug. 18, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=x68q1MWmID0&t=765s&ab_channel=BitcoinTradingChallenge
[46] BSIC, “Tips and tricks for manipulating the market (and getting punished for it) – BSIC | Bocconi Students Investment Club.” https://bsic.it/marketmanipulation/ (accessed Aug. 15, 2022).
[47] “Array.prototype.shift() - JavaScript | MDN.” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/shift (accessed Aug. 21, 2022).
[48] “parseInt() - JavaScript | MDN.” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt (accessed Aug. 21, 2022).
[49] “Array.prototype.slice() - JavaScript | MDN.” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/slice (accessed Aug. 21, 2022).
[50] “Array.prototype.map() - JavaScript | MDN.” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map (accessed Aug. 21, 2022).
[51] “What is blockchain technology?,” *IG*. https://www.ig.com/en/trading-strategies/what-is-blockchain-technology--200710 (accessed Aug. 01, 2022).

# 7. Table of Figures

Figure 1: Block connection in Blockchain [51].....8
Figure 2: Deployed contract (source: author).....11
Figure 3: Contract after changing name value (source: author).....12
Figure 4: Understanding the market (source: author).....21
Figure 5: Fake buy wall (source: author).....22
Figure 6: Market after spoofing (source: author).....22
Figure 7: Flash crash [41].....23
Figure 8: Market price moving before the ignition (source: author).....25
Figure 9: Making a large buy trade (source: author).....25
Figure 10: Momentum ignition (source: author).....26
Figure 11: Token price movement after momentum ignition strategy (source: author).....26
Figure 12: Selling momentum ignition (source: author).....27
Figure 13: Momentum ignition example [46].....28
Figure 14: Buffer (source: author).....29
Figure 15: Interface (source: author).....34
Figure 16: Browser alert (source: author).....35
Figure 17: Forbidden trade (source: author).....35
Figure 18: Invalid input (source: author).....35