Security Issues in aDEX

Adrian Goh Jun Wei
U1721134D
School of Computer Science and Engineering
Nanyang Technological University
Singapore
jgoh070@e.ntu.edu.sg

***Abstract*—I explain the security flaws that centralized exchange has and why there is a need for a decentralized exchange for the trading and exchanging of cryptocurrencies. With a focus on the security of decentralized exchanges, I deep dive into the issues in adriangohjw Decentralized Exchange (aDEX), how they have been addressed and proposing some solutions to fix them if otherwise.**

***Keywords*—blockchain, Ethereum, smart contracts, security, tokens, decentralization, decentralized exchanges (DEX), reentrancy attack, overflow attack**

I. INTRODUCTION

The rise of blockchains has enabled any individual to own and transfer assets across a decentralised network without needing to trust any external parties, leading to the rapid adoption of the ownership of digital assets that exist on the blockchain. Some examples include cryptocurrencies and tokens such as Bitcoin, Ethereum, Litecoin, Siacoin etc.

The rapid adoption of digital assets also quickly led to the launch of many platforms that exist to facilitate the efficient exchange of these assets. Similar to the traditional financial stock exchange like the New York Stock Exchange (NYSE), these platforms provide assets liquidity to users by allowing them to trade these assets as a fee. Examples of such trading platforms include Binance, Liquid.com, Bitfinex and even Singapore-based KuCoin.

However, while stock exchanges can be secure due to government regulation and being platforms to trade centralised company shares, this was not the case for exchanges and platforms facilitating the trade of cryptocurrencies.

Even before the mainstream adoption of cryptocurrencies, there have been multiple centralized cryptocurrency exchange hacks. Some of these hacks include:

a. In 2014, Mt. Gox, the largest Bitcoin exchange handling 70% of transactions at the time, lost ~850K Bitcoin (USD450M), which represented over 6% of bitcoins in circulation at that time. [1]
b. In 2016, Bitfinex lost ~120k Bitcoin (~$72M). [2]
c. In 2019, Binance lost $40M worth of Bitcoins. [3]
d. In 2020, Singapore-based exchange KuCoin lost ~$280M worth of crypto assets. [4]

In the case of Mt Gox, the news of the hack resulted in the price of bitcoin plunging by 20% [1] due to a loss in investors confidence in not just cryptocurrency exchanges, but cryptocurrencies as a whole.

Besides, in the months leading up to Mt. Gox hack, users expressed increasing frustration and difficulties in withdrawing their assets. Technical limitations posed a challenge and stopped the company from having a concrete understanding of transaction details, including uncertainty on whether bitcoins have been transferred to users' wallets. [5]

These multiple incidents have proven that using and entrusting centralized exchanges to facilitate crypto trades posed a significant risk of fund loss and thefts due to their centralized functioning. In many cases, most of them serve as a custodian of users' funds. What this means is that users do not actually have access to the private keys of their exchange wallet's account, but instead hold the keys to accounts on the exchange. For example, when you buy Ether on these exchanges, even though it appears in your account, you don't own the Ether. These companies could only be updating their internal non-transparent system to reflect the changes, and not recording these transactions on the blockchain. Therefore fundamentally, these assets are held and managed under the companies that operate these exchanges, and still exists a high level of trust that these companies will not lose/misuse the funds. However, we have seen how this assumption that we can trust these centralized exchanges are false - in fact, this is the main reason behind many of such hacking incidents.

In 2018 alone, over $1 billion has been stolen from centralised cryptocurrency exchanges. [6] Due to the lack of security that centralized exchanges have demonstrated, a new wave of blockchain projects that aimed to solve this problem has emerged: decentralized exchanges (DEXs). Most DEXs aim to solve the main reason for the downfall of many centralized exchanges, which is being the single point of failure in losing users' assets. They enabled safer trading by leveraging the technology behind cryptocurrencies themselves. This is of utmost importance since the theft of funds from exchanges is a huge risk-factor for customers.

In my development project, I will be developing a DEX named aDEX (adriangohjw Decentralized Exchange). We will be building this DEX on the Ethereum blockchain, and it will be able to support the trading of Ether and other ERC20 tokens.

While these blockchain projects to replace centralized exchanges are indeed visionary and revolutionary, most of them face the Blockchain Trilemma [7], a term coined by Vitalik Buterin, the co-founder of Ethereum. It addresses the challenges that developers face in creating a blockchain that

can be scalable, decentralized and secure without compromising on any facet.

Blockchains are often forced to make trade-offs that prevent them from achieving all three aspects:

a. Decentralization: the degree in which a blockchain can facilitate transactions effectively without having to rely on a central point of control, and this is done by diversifying its ownership, influence and value amongst multiple parties.
b. Scalability: the capability of a blockchain to handle an increasingly growing amount of transactions, and is usually measured using TPS (transactions per second), block height (the number of transactions per block), and transaction size.
c. Security: the defensibility of a blockchain against attacks, bugs, and other unforeseen issues. A fastened crypto protocol needs to be able to prevent and recover from short-term attacks (resilience) in the short term without making changes to previous states of the distributed ledger (immutability).

In this paper, I will be addressing the security aspects of the Blockchain Trilemma when I am analyzing aDEX. I will also be looking into what has and will be done to address these security issues that I am highlighting.

II. MOTIVATION

The most important reason for the birth of DEXs and why there is a preference over centralized exchanges is the security it provides through the trustless custody of users' assets. Regardless of how scalable and many transactions an exchange can facilitate, it will quickly lose the trust and support of users if it is unable to ensure that users assets are securely managed and are not at risk of being stolen. Therefore, security is inarguably one of the largest factors that users consider when it comes to deciding if they should proceed to use a particular platform to their trading.

It is worthwhile to note that most DEXs are only partially decentralized. What this means is that in most cases, they still host order books in their server or as an off-chain to scale the exchange to handle higher traffic. We understand from our lectures that off-chain solutions do not come at the expense of security. As such, it is an implementation that we can accept.

Besides, although hosting and processing the order books outside the blockchain can result in frontrunning and denial-of-service (DoS) that might result in a less than ideal trading positions and outcomes for the end-users, they do not increase the risk of users losing their funds and assets. Furthermore, the issue of frontrunning and DoS can also be present in stock trading firms, albeit it being illegal. As such, I will not be discussing and addressing them in my development project.

Fundamentally, I developed aDEX with the intended goal of solving the security flaws of exchanges. I will be deep-diving into the security issues in aDEX in the blockchain and smart contract aspect.

However, while security is the key concern when I analyze aDEX, I do also acknowledge that there is a strong correlation between the degree of decentralization of an exchange and its level of security. This is apparent from the many past hacking incidents. Nevertheless, decentralization by itself is not the actual goal of a DEX, as it is merely just the means to the end and one of the many solutions to achieve security.

III. OBSERVATIONS AND ANALYSIS

While entrusting funds to a smart contract or protocol, instead of a centralized entity, can remove many risk factors, we should be aware that it is not a 100% risk-free solution to the problem we are solving. We will be looking into some vulnerabilities in decentralized applications (dApps) that could still raise security concerns.

A. False decentralization

In 2018, Bancor, a crypto company that touts a DEX service was hacked and lost $23.5M of cryptocurrency tokens belonging to its users. [8] This happened because although the operation and interaction on the exchange are done through the smart contracts, the hacked Bancor wallet could withdraw cryptocurrencies out of the smart contracts. This is possible because the smart contracts deployed have a deliberate built-in permissioned backdoor [9] that allows the Bancor team to arbitrarily issue, freeze and even destroy any Bancor Network Token (BNT) tokens whenever they want. Hackers were able to make use of this flaw to steal funds out of a BNT's connector balance that served as a reserve.

So while the actual smart contracts were indeed safe and not compromised, it is undeniable that having an overwriting backdoor also meant that hackers were able to drain the funds out of the smart contracts as long as they got hold of the Bancor wallets. As a result, it was also the same reason that resulted in Bancor being a single point of failure when it comes to losing funds.

Also in 2018, there was some controversy around the Singapore-based company Soar Labs, which operated the cryptocurrency exchange soarexchange.io. Soar Labs initially issued $5M worth of Soarcoin to Australian-based Byte Power Group in exchange for a stake in the company. [10] After some dispute in the following few months, Soar Labs decided to withdraw these Soarcoins from Byte Power Group's wallet through a backdoor function in the smart contract. This backdoor gives them the ability to transfer a balance from any party to any party.

While this is not a hacking incident, Soar Labs' decision to reverse the tokens did indeed raise a lot of questions and backlashes from the crypto community regarding the centralised power and authority that they possessed. The trustless nature of blockchain was compromised and trust towards the company was depleted. As such, the company has since shut down in 2019.

The main reasons for backdoors in smart contracts are due to a gap between the fully decentralized idea and the real-world constraints arising from operating the blockchain and possibly regulatory reasons.

In the case of Bancor, the reasoning was that while their main smart contract is still undergoing testing, they should retain full control in case anything goes wrong. As such, the backdoor was intentional without any ill intent to allow them to respond to and remedy any unforeseen issues that may occur during the crowd sale and in the early days of operation.[11]

In the case of Soar Labs, the co-founder claimed that they only use the backdoor in exceptional situations. For example, they assisted a compromised cryptocurrency exchange in recovering Soarcoin that was stolen away by a malicious party. [12]

Another potential use case of a backdoor includes having to meet regulatory requirements and working with the government bodies to freeze or destroy tokens should it be discovered that they fell into the hands of criminal organizations by any means.

While all these reasons for the inclusion of a backdoor in smart contracts are valid, we cannot deny that they will inevitably diminish the public's trust in dApps, especially since trust and security are crucial when it comes to evaluating a DEX.

B. Risk of reentrancy attack

Reentrancy bug happens which a malicious external contract can take over the control flow of the contract by calling back into the contraction before the first invocation of the function is finished, thus making changes to your data that the calling function wasn't expecting.

In 2017, The DAO was hacked and 3.6M Ether (~$50M) were stolen using the first reentrancy attack. It was one of the highest-profile reentrancy attacks in Ethereum's history. The hack was so impairing that the Ethereum Foundation resorted to a very controversial hard fork to revert the transfer to recover the lost investor funds. While most supported the hard fork, there were some within the community that thought it violated the core immutability principle of blockchain and continued to use the compromised chain, which we now know as Ethereum Classic. [13]

In aDEX, the components that we should be wary of the potential for a reentrancy bug are those that involve updating the ether/token balance of an account in the contract. The four main components identified are:

1. Depositing Ether/tokens - Ether/tokens are sent to the contract and the balance of these assets increases
2. Withdrawing Ether/tokens - request for ether/tokens to be withdrawn from the contract and the balance of these assets decreases
3. Making a buy order - deducting Ether from one's balance to add an order in the buy order book, in exchange for other ERC20 tokens
4. Making a sell order - deducting ERC20 tokens from one's balance to add an order in the sell order book, in exchange for Ether

Upon further inspection, only 3 components (*depositToken*, *withdrawToken*) are susceptible to such an attack as they call external function(s). In these components, below is the wrong way to implement the methods as it will result in them being exposed to a reentrancy bug.

a. In the vulnerable implementation of *depositToken*, the malicious external function could call *depositToken* to be executed again before the function has been executed completely. As such, step 2 of increasing the account's token balance will run again, leading the exchange to think that they have more tokens than they should have. As such, the attacker could subsequently withdraw more tokens than they own and drain the exchange.

Pseudo design of a vulnerable depositToken

```
function depositToken(...) {
   (1) Initiating an external contract
   (2) Increasing account's token balance
   (3) Calling an external function
}
```

b. In the vulnerable implementation of withdrawToken, the malicious external function could withdraw tokens from the exchange, then call execute *withdrawToken* again before the initial function has been executed completely. Since step 3 of decreasing the account's token balance has not occurred yet, the exchange will be tricked into thinking that the attacker still has sufficient amounts of tokens to be withdrawn. Thus, the withdrawal of tokens will still be permissible and the attacker can drain more tokens from the exchange than the amount they own.

Pseudo design of a vulnerable withdrawToken

```
function withdrawToken(...) {
   (1) Initiating an external contract
   (2) Calling an external function
   (3) Decreasing account's token balance
}
```

In the cases mentioned above, if a reentrance bug is present and exploited, token balances of an account will be wrongly inflated, thus making it possible to withdraw more tokens than they rightfully own. The accounts which deposited these additional assets will be the victim and lose them.

C. Risk of Overflow attack

In 2018, an unusually large amount of BeautyChain (BEC) tokens were hacked and sold in the market, resulting in the value of the digital currency to drop to almost zero. After investigation, they discovered that it was due to a lack of a check for the possibility of an integer underflow/overflow during arithmetic operations within the contract transfer functions. Hackers were able to exploit the data overflow to

attack the smart contract of BEC and drain the contract of a large number of tokens. This specific sub-category of the overflow attack was termed as BatchOverflow. [14]

The Exchange contract in aDEX should not be at risk of facing an overflow attack from an external malicious party. Nevertheless, I have identified two components of the Exchange contract that might be affected by the overflow bug due to the internal operations and implementation, which can have a devastating impact on the exchange.

a. *tokenIndex* overflowing and resetting back to 0
b. *etherBalanceForAddress/tokenBalanceForAddress* overflowing and updated to 0, or underflowing and updated to a extremely large value

Firstly, *tokenIndex* is initialized with the value 1 upon the contract deployment. Also, it has an integer type of uint8, which means it has a range of 0 to 255. Thus, this exchange can only accumulate 255 different tokens. If we attempt to add more ERC20 tokens to the exchange after that, tokenIndex will overflow and be reset to 0. tokenIndex is a crucial component of the Exchange contract and most of the other components of the contract depend on it for some form of derivation. Thus, what this means is that if the tokenIndex overflows and reset to 0, it will cause a huge portion of the contract to malfunction.

In our contract, we use the function getTokenIndex to derive the index (unique identifier) of a token from a symbol name. Let's say that a particular token ABC in the exchange has a unique index of 13. If overflow happens, we will be unable to retrieve the index of token ABC and thus, unable to do any form of deposit, withdrawal or trades. Similarly, this applies to all other tokens in the exchange. As a result, every token deposited into the contract will be frozen and inaccessible to their owner.

Extract of the getTokenIndex function

```
function getTokenIndex(...) public view returns (...)
{
   for (uint8 i = 1; i <= tokenIndex; i++) {
      if (keccak256(bytes(symbolName)) ==
         keccak256(bytes(tokens[i].symbolName))) {
            return i;
      }
   }
   return 0;
}
```

Not fixing this overflowing bug might appear to only freeze all the tokens in the exchange on the surface. However, if the contract owner of this Exchange contract is malicious, he/she will be able to retrieve a significant portion of the tokens stored indirectly through a loophole.

For example, let us better explain the risk with the following scenario:

- TokenA was assigned the *tokenIndex* of *x*
- TokenB was assigned the *tokenIndex* of *y*
- Owner has deposited 10 TokenA into the exchange
- Owner does not have any TokenB in the exchange
- 1 TokenB is a lot more valuable (based on fiat currency value) as compared to 1 TokenA

After the overflowing bug occurs, the owner of the contract can add tokens to the exchange such that TokenB is assigned the tokenIndex of x. He/she can then proceed to withdraw 10 TokenB (that others have deposited) from the contract into his/her wallet even though he/she has not deposited them into the exchange. As such, even though 10 TokenA will be locked up in the contract, the contract owner will be financially better off.

Secondly, in the case where the Ether/ERC20 tokens balance of a particular account is too large, it will overflow and become 0. As such, even though this account should rightfully have access to these assets, he/she will be unable to access it. The account will lose all these assets as they are frozen in the contract. However, as an integer type of uint256 is used, it is very much so unlikely that this balance amount will be reached and thus, the overflowing bug is not too big of a concern.

However, adding on to the second pointer, on the other hand, the underflowing bug could be devastating because a particular account will end up having way too many Ether/tokens than he/she should rightfully have. This will result in the account being able to withdraw Ether/token that was deposited by the other accounts. If that happens, those other accounts will then lose the assets they have in the exchange. Therefore, we should be wary of the underflowing bug and check for it whenever there is a deduction of the asset balance, such as when a user is withdrawing Ether/token from his/her account or is trying to make a buy/sell order.

IV. PROPOSED SOLUTIONS

A. Eliminating false decentralization

The inclusion of a backdoor in a smart contract gave a false sense of security and decentralization. The proposed solution to eliminate false decentralization is a rather simple one - do not implement any backdoor that allows the contract owner to overwrite and manipulate the ownership of any tokens. As much as it is tempting to do so for practical reasons like ease of operation, these backdoors will inevitably be the downfall of the exchange in the future.

As such, I implemented aDEX without any backdoor and the contract owner is unable to transfer, create, destroy and manipulate the tokens in the contract after deployment, unlike that of Bancor's BNT or Soar Labs' Soarcoin.

The only power that the contract owner has is to add more tokens (*addToken*) that can be traded into the exchange, which will not compromise the integrity of the contract.

B. Eliminating risk of reentrancy attack

In the following segment, I will be explaining how we can eliminate the risk of reentrancy attack in the two components pointed out. In general, we should only increase the account's

balance after calling an external function, and decrease the account's balance before calling external functions.

Firstly, when implementing depositToken, we should only increase the token balance for an account after executing the external function. As such, attackers will not be able to artificially inflate the number of tokens the exchange thinks they have. Launching a reentrance attack will only cause them to deposit more tokens into the exchange without them being recognized and reflected.

Simplified extract of a guarded depositToken function

```
function depositToken(...) public returns (...) {
   ...
   // (1) initiating an external contract
   ERC20Interface token = ERC20Interface(
      tokens[_tokenIndex].contractAddress
   );

   // (2) calling an external function transferFrom
   require(
      token.transferFrom(
         msg.sender, address(this), amount
      ) == true
   );

   // (3) increasing account's token balance
   tokenBalanceForAddress[msg.sender][_tokenIndex]
      += amount;
   ...
}
```

Similarly, when implementing withdrawToken, we should only decrease the token balance for an account before executing the external function. As such, attackers will not want to launch a reentrance attack as their token balance will constantly be decreasing while their assets get frozen up in the exchange.

Simplified extract of a guarded withdrawToken function

```
function withdrawToken(...) public returns (...) {
   ...
   // (1) initiating an external contract
   ERC20Interface token = ERC20Interface(
         tokens[_tokenIndex].contractAddress
   );

   // (2) decreasing account's token balance
   tokenBalanceForAddress[msg.sender][_tokenIndex]
      -= amount;

   // (3) calling an external function transfer
   require(
      token.transfer(msg.sender, amount) == true
   );
   ...
}
```

C. *Eliminating risk of Overflow attack*

To prevent the overflow bug from crippling our exchange, we should always check for overflow/ underflow before any arithmetic operations in our contract, which I have already implemented, as seen in the following code extracts.

a. Overflow check before adding tokens

```
function addToken(...) {
   // (1) overflow check
   require(tokenIndex + 1 >= tokenIndex);

   // (2) increasing tokenIndex
   tokenIndex++;
   ...
}
```

b. Overflow check before increasing token balance

```
function depositToken(...) public returns (...) {
   // (1) overflow check
   require(
      getBalanceForToken(symbolName) + amount
         >= getBalanceForToken(symbolName)
   );

   // (2) increasing token balance
   tokenBalanceForAddress[msg.sender][_tokenIndex]
      += amount;
   ...
}
```

However, there is still room for improvement. Instead of implementing our checks, we can use the SafeMath library [15], as shown in the following examples (assuming we have imported SafeMath.sol):

a. Overflow check before adding tokens (SafeMath)

```
using SafeMath for uint8;
function addToken(...) {
   // overflow check + increasing tokenIndex
   tokenIndex = tokenIndex.add(1);
   ...
}
```

b. Overflow check before increasing token balance (SafeMath)

```
using SafeMath for uint;
function depositToken(...) public returns (...) {
   // overflow check + increasing token balance
   tokenBalanceForAddress[msg.sender][_tokenIndex]
      = getBalanceForToken(symbolName).add(amount);
   ...
}
```

V. CONCLUSION

In this brief paper, I have analyzed the security issues of aDEX. I have also outlined the steps that I have taken to address them, as well as proposed some solutions to fix the existing issues moving forward.

Beyond the security issues mentioned in this paper, it is worthwhile to note that I built aDEX on top of the Ethereum blockchain network. As such, it inherits every risk and challenges that could occur to the Ethereum network itself, such as the famous 51% Attack (and Eclipse Attack) which could compromise the network integrity.

Besides, even with a very secure smart contract, DEXs are still susceptible to attacks that are not about the blockchain. In 2017, EtherDelta, a fully decentralized exchange, lost 308ETH ($270k) in a phishing attack. While the smart contracts that govern EtherDelta's operation weren't compromised, the hackers managed to take over EtherDelta's DNS server and serve a fake version of the site to visitors. While it is undeniable that security breaches can still happen from such attacks, the same can be said about any platforms, regardless of its degree of decentralization. [16]

REFERENCES

[1] "*Mt. Gox*", Wikipedia, n.d. Accessed on: Nov. 24, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Mt._Gox

[2] "*Bitfinex*", Wikipedia, n.d. Accessed on: Nov. 24, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Bitfinex

[3] E. Lam, *Hackers Steal $40 Million Worth of Bitcoin From Binance Exchange*, Bloomberg, May 8, 2019. Accessed on: Nov. 24, 2020. [Online]. Available: https://www.bloomberg.com/news/articles/2019-05-08/crypto-exchange-giant-binance-reports-a-hack-of-7-000-bitcoin

[4] A. Hui, W. Zhao, *Over $280M Drained in KuCoin Crypto Exchange Hack*, CoinDesk, Oct. 09, 2020. Accessed on: Nov. 24, 2020. [Online]. Available: https://www.coindesk.com/hackers-drain-kucoin-crypto-exchanges-funds

[5] J. Frankenfield, *Mt. Gox*, Investopedia, Feb. 2, 2020. Accessed on: Nov. 24, 2020. [Online]. Available: https://www.investopedia.com/terms/m/mt-gox.asp

[6] Y. Khatri, *Nearly $1 Billion Stolen In Crypto Hacks So Far This Year: Research*, CoinDesk, Oct. 18, 2018. Accessed on: Nov. 24, 2020. [Online]. Available: https://www.coindesk.com/nearly-1-billion-stolen-in-crypto-hacks-so-far-this-year-research

[7] CertiK, *The Blockchain Trilemma: Decentralized, Scalable, and Secure?*, Medium, Oct. 4, 2019. Accessed on: Nov. 24, 2020. [Online]. Available: https://medium.com/certik/the-blockchain-trilemma-decentralized-scalable-and-secure-e9d8c41a87b3

[8] J. Russel, *The crypto world's latest hack sees Bancor lose $23.5M*, TechCrunch, Jul. 10, 2018. Accessed on: Nov. 24, 2020. [Online]. Available: https://techcrunch.com/2018/07/10/bancor-loses-23-5m/

[9] A. Baydakova, *$13.5 Million Hack Ignites Fresh Debate Over Crypto Project Bancor*, CoinDesk, Jul.16, 2018. Accessed on: Nov. 24, 2020. [Online]. Available: https://www.coindesk.com/13-5-million-hack-ignites-fresh-debate-over-crypto-project-bancor

[10] A. Munro, *Warning: Soarcoin (SOAR) has a backdoor*, finder.com, Jun. 8, 2018. Accessed on: Nov. 24, 2020. [Online]. Available: https://www.finder.com.au/warning-soarcoin-soar-has-a-backdoor

[11] U. Wertheimer, *Bancor Unchained: All Your Token Are Belong To Us*, Medium, Jun. 21, 2017. Accessed on: Nov. 24, 2020. [Online]. Available: https://medium.com/unchained-reports/bancor-unchained-all-your-token-are-belong-to-us-d6bb00871e86

[12] J. Kirk, *Exclusive: Aussie Firm Loses $6.6M to Backdoored Cryptocurrency*, BankInfoSecurity, Jun. 5, 2018. Accessed on: Nov. 24, 2020. [Online]. Available: https://www.bankinfosecurity.com/exclusive-aussie-firm-loses-5m-to-backdoored-cryptocurrency-a-11057

[13] "*The DAO (organization)*", Wikipedia, n.d. Accessed on: Nov. 24, 2020. [Online]. Available: https://en.wikipedia.org/wiki/The_DAO_(organization)

[14] PeckShield, *New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018–10299)*, Medium, Apr. 23, 2018. Accessed on: Nov. 24, 2020. [Online]. Available: https://peckshield.medium.com/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536

[15] J. Dourlens, *Using Safe Math library to prevent from overflows*, EthereumDev, Apr. 5, 2020. Accessed on: Nov. 24, 2020. [Online]. Available: https://ethereumdev.io/using-safe-math-library-to-prevent-from-overflows/

[16] K. Sedgwick, "*One Week On from the Etherdelta Hack, Funds Are Still Being Stolen*", Bitcoin.com, Dec. 26, 2017. Accessed on: Nov. 24, 2020. [Online]. Available: https://news.bitcoin.com/one-week-etherdelta-hack-funds-still-stolen/