network

MDPI

Article

A Study of Ethereum's Transition from Proof-of-Work to Proof-of-Stake in Preventing Smart Contracts Criminal Activities

Oliver J. Hall [1], Stavros Shiaeles [1,*] and Fudong Li [2]

[1] School of Computing, University of Portsmouth, Portsmouth PO1 3HE, UK; oliver.hall2@myport.ac.uk
[2] Department of Computing and Informatics, Bournemouth University, Bournemouth BH12 5BB, UK; fli@bournemouth.ac.uk
* Correspondence: stavros.shiaeles@port.ac.uk

Abstract: With the ever-increasing advancement in blockchain technology, security is a significant concern when substantial investments are involved. This paper explores known smart contract exploits used in previous and current years. The purpose of this research is to provide a point of reference for users interacting with blockchain technology or smart contract developers. The primary research gathered in this paper analyses unique smart contracts deployed on a blockchain by investigating the Solidity code involved and the transactions on the ledger linked to these contracts. A disparity was found in the techniques used in 2021 compared to 2023 after Ethereum moved from a Proof-of-Work blockchain to a Proof-of-Stake one, demonstrating that with the advancement in blockchain technology, there is also a corresponding advancement in the level of effort bad actors exert to steal funds from users. The research concludes that as users become more wary of malicious smart contracts, bad actors continue to develop more sophisticated techniques to defraud users. It is recommended that even though this paper outlines many of the currently used techniques by bad actors, users who continue to interact with smart contracts should consistently stay up to date with emerging exploitations.

Keywords: blockchain; smart-contract; solidity; consensus-mechanisms

Citation: Hall, O.J.; Shiaeles, S.; Li, F. A Study of Ethereum's Transition from Proof-of-Work to Proof-of-Stake in Preventing Smart Contracts Criminal Activities. *Network* **2024**, *4*, 33–47. https://doi.org/10.3390/network4010002

Academic Editor: Mehdi Sookhak

Received: 29 September 2023
Revised: 22 January 2024
Accepted: 23 January 2024
Published: 26 January 2024

1. Introduction

Blockchain technology is continually evolving, especially in recent years. This is causing issues not only for countries trying to adapt their laws to cater to the new technology, but also for addressing security concerns. When millions of dollars are flowing through it, many bad actors spot opportunities to exploit the blockchain technology. Bad actors could abuse vulnerabilities in the consensus mechanism of the blockchain or a bug in its contract coding language, causing significant monetary loss for investors. Smart contracts have seen increased usage in recent years, allowing users with simple coding knowledge to deploy them, potentially enabling them to create their own tokens that others could trade on a blockchain.

Smart contracts are code stored on certain blockchains, such as Ethereum or Binance Smart Chain (BSC), which can be called by users to interact with them through transactions on the chain [1]. Smart contracts are commonly coded in Solidity or Vyper [2]; however, many languages are available depending on the blockchain, including but not limited to Rust, LIGO, and Plutus [3]. This article will focus on Solidity, as it is the primary language of the current largest blockchain that supports smart contracts, Ethereum. Smart contracts are significantly used in the financial sector, with many contracts focusing on financial tokens and non-fungible tokens (NFTs). However, it is worth noting that they can have other applications in areas such as the Internet of Things (IoT), allowing these devices to act autonomously by using smart contracts, or for games such as Lottery and TicTacToe [1,3].

Smart contracts can suffer from several different vulnerabilities, with many different attack vectors for malicious actors. When looking at Solidity, the main smart contract

language for the second largest blockchain, Ethereum, there are many areas outlined by [4] as causes of these vulnerabilities, which include smart contract programming, the Solidity language and toolchain, and the design and implementation of Ethereum itself. This paper investigates and researches vulnerabilities in these areas to gain a comprehensive understanding of smart contract security. Vulnerabilities in these areas have been actively exploited, one example being the Beauty Ecosystem Coin (BEC) Token attack in April 2018, in which an exploitation of an integer overflow vulnerability caused an exchange to close temporarily and a significant number of stolen tokens, resulting in substantial monetary losses [4]. This makes analysing the vulnerabilities of smart contracts a crucial part of this research, as large-scale exploitations are becoming increasingly prevalent.

Criminal smart contracts are contracts uploaded to a blockchain, such as Ethereum, which allows for Solidity smart contracts, and can cause malicious activities such as a honeypot [5]. Honeypots are smart contracts for tokens specifically designed by the uploader to have a vulnerability, such as an upgradable contract where they can replace a safe-looking contract with a new one where they could, for example, remove all the funds from it [6]. An example of criminal smart contracts involves tokens capitalising on current popular tropes to entice individuals to use a contract, such as the "SQUID" token, which made United Kingdom national news after receiving $3.38 million worth of Ethereum when it was found to be a scam [7,8].

The main contributions of the paper are the following: (1) various vulnerabilities and exploitations involving blockchain technology are explored, as well as (2) an in-depth analysis of 50 smart contracts, suggesting that malicious code and unrevoked ownership were the most prevalent issues. The remainder of the paper is structed as follows: Section 2 presents related works. The methodology is given in Section 3. The main finding is demonstrated in Section 4, followed by a discussion. The main achievements are highlighted in the conclusion.

2. Related Works

This section aims to review the difference between Proof-of-Work (PoW) and Proof-of-Stake (PoS) algorithms. Also, the security and vulnerability of smart contracts are examined. In addition, criminal smart contracts, which are maliciously created to contain and hide vulnerabilities, are researched.

2.1. Consensus Mechanisms

The PoW mechanism, which has been leveraged by Bitcoin since its inception and by more than six hundred other blockchain currencies in November 2018, has been thoroughly analysed throughout the years for security vulnerabilities [9]. Nonetheless, there is a disparity between some recent and past literature regarding the specifics of some vulnerabilities. PoW blockchains operate through members solving mathematical problems to validate a block, allowing transactions to be validated [10]. Different PoW blockchains use different PoW consensus algorithms, such as SHA-256 for Bitcoin, RandomX for Mero, KawPow for Raven Coin, Scrypt for Litecoin, Equihash for Zcash, and Ethash for Ethereum [11–13].

The PoS mechanism, first incorporated by "PeerCoin" [14], allows for a different type of block validation than the PoW algorithms. The biggest proponent for PoS was Ethereum, as it was a staple of its development plans very early on [15]. The PoS mechanism was enabled on Ethereum in 2022, due to it being more secure, less energy-intensive, and better for implementing new scaling solutions compared to the previous PoW architecture [16]. The PoS consensus mechanism does not use mining in the form of computational power to solve algorithmic mathematical problems like PoW. Instead, at least in the case of Ethereum, miners are expected to stake their cryptocurrencies in a validator smart contract, in which that crypto is used as collateral if the validator contract begins to act improperly [16].

2.2. The Vulnerabilities of Consensus Mechanisms

A 51% Attack is the risk to the security of PoW by an individual or group taking over 50% of the blockchain's mining power [9]. This was further proven as a substantial risk to PoW blockchains when an attack on Ethereum Classic (ETC), which was a fork of Ethereum long before its move to PoS, actually occurred. After suffering from the probabilistic finality vulnerability, the value of the currency significantly dropped, causing the hash rate to also decline, leading to an attacker being able to carry out a 51% attack on the chain, partially due to the introduction of cloud mining services that let an attacker purchase mining power [6]. It suggests that a 51% attack is an important vulnerability in blockchain technology due to the damage it can cause and the likelihood of its occurrence.

A Selfish Mining Attack allows the exploiter to increase their mining earnings while simultaneously negatively impacting the amount other nodes would earn [17]. Ref. [9] suggests the attack is conducted to obtain mining rewards that should not normally be obtained and take away rewards from honest miners. Research by the authors of [18] states that a miner equipped with only 33% mining power could earn in rewards what someone who had 50% of the mining power could earn through this attack; this information can be substantiated by the authors of [19], who also claim that 33% is a "theoretical hard limit" for mining power when it comes to the selfish mining attack. This is refuted in the more recent literature by [20], which conveys that only 25% of the mining power is needed for attackers to improve their rewards through this attack.

A Double Spending Attack occurs when a bad actor exploits the time between two transactions [9]. This vulnerability to the PoW consensus mechanism is deemed to be impossible to avoid [9]. Ref. [12] proposes that there is no current way to prevent double-spending in the Bitcoin whitepaper, which further justifies that it is impossible to avoid but could be difficult for a malicious actor to attempt, with them possibly needing 50% of the total mining power to conduct the attack. However, the authors of Ref. [9] state otherwise, saying that double-spending attacks could occur with low mining power when combined with the selfish mining attack. Ref. [10] suggests that no instance of double-spending has been recorded. However, Ref. [21] indicates that there was a double-spending attack in March 2013, which resulted in a rapid drop-off in the price of Bitcoin. Ref. [22] clarifies and supports this, adding that the 2013 attack was due to a blockchain fork, which caused a conflict between Bitcoin versions, allowing two logs of Bitcoin to become available, enabling them to be double-spent.

Refined Reorg Attack: Due to Ethereum's move to PoS being a recent change, fewer research papers have been released regarding any attacks on the new upgraded chain. However, Ref. [23] presents an attack called the Refined Reorg Attack on the new PoS Ethereum network, which could allow validators to earn more from validating blocks than others through the Maximal Extractable Value [24]. Ref. [23] details the attack as being that of a malicious actor who could propose a block in a slot, which keeps it hidden, while an honest block proposer would then place their block at the next slot, resulting in a bad actor being able to use both slots as a vote, which creates a probability that proposed blocks could be outnumbered by malicious actors.

2.3. The Vulnerabilities of Smart Contracts

Integer Overflow and Underflow: The Ethereum Virtual Machine (EVM), which is run by the Ethereum network and serves as the main purpose of the protocol, contains a stack that can hold 1024 slots. Each time a function is called, a slot is used up. This can cause all the slots to be used up, depending on the number of functions called, resulting in a stack overflow that could cause problems [25]. The EVM is not the only place in which an integer overflow/underflow occurs. If an arithmetic operation goes outside the range of a data type in Solidity, it can cause the manipulation of a variable. This is due to Solidity not providing data validation on these operations, and neither EVM nor Solidity itself providing any detection for this [4]. An example of this attack can be found when examining the smart contract of BEC, which was an Ethereum ERC20 token that was

vulnerable due to CVE-2018-10299. This allowed an attacker to increase the amount of the token they earned by exploiting an integer overflow issue [26]. The solution to this issue is for developers to include the SafeMath library, which was created to handle this problem as it wraps over Solidity operations to add overflow checks [27].

The re-entrancy vulnerability allows an attacker to change the state of a contract after a call is completed, which can result in cryptocurrency being stolen depending on the type of transaction being made [9]. The vulnerability is caused by an attacker being able to "re-enter" a caller function due to a contract relying on variables that do not update without first calling another contract [4]. An example of this attack is the DAO attack, which was exploited by the attacker creating a smart contract that included a call to the DAO's withdraw function, which then called back, repeating the malicious contract's withdrawal function repeatedly until all the Ethereum was taken from the original smart contract. The DAO attack resulted in an attacker stealing USD 60 million, showing that this is a significant vulnerability [9]. According to [4], this vulnerability can be prevented by either using a mutex lock on a contract so that only the owner of a smart contract can change the state of it, or by using the transfer method of sending money to other contracts.

Malicious Contracts: A significant problem within the smart contract ecosystem is that malicious actors often create smart contracts with malicious intent. According to Chainalysis, a company that provides intelligence and risk management tools for the blockchain, one in four new tokens of any value is a scam [28]. Many techniques are used by attackers and are investigated in the primary research of this paper; however, researching the literature has led to information about some forms of malicious smart contracts. Some examples of these include tokens that contain a high buy or sell fee and, in some instances, allow the owner of a contract to mint more coins and sell them [29].

Phishing Attacks are prevalent in the context of blockchains and often target online wallets, with attackers attempting to acquire private keys [30], or in the case of smart contracts, they look for individuals to interact with a malicious contract [31]. One common example of phishing attacks for smart contracts is websites that run a setApprovalForAll function, which allows the malicious actor running the site to steal funds from the wallet [31]. Another example of phishing attacks involving smart contracts is a large OpenSea phishing attack. This attack takes advantage of a recent upcoming migration of listings on their marketplace, as attackers directed their phishing email as an urgent request to migrate their listings quickly, or they would expire [32]. This leads to users who interact with the phishing site effectively giving the attacker a blank cheque to take any tokens they wanted, resulting in an estimated loss of USD 1.7 million [33].

3. Materials and Methods

When analysing smart contracts, a set of questions must be answered to determine their maliciousness. Three questions were devised to evaluate each smart contract:

- Did the smart contract contain malicious code?
- Was ownership of the smart contract renounced?
- If the smart contract was an ERC (Ethereum Request for Comments) token, was the liquidity safe?

The first question arose from prior research reviewing the literature, which indicated that vulnerabilities could exist in smart contract code. This was further supported by early investigations into smart contracts that revealed hidden mint functions. The second question about smart contract ownership was prompted by instances where contracts could be rendered malicious due to an individual or entity, usually the one deploying the smart contract, being able to call functions that could disrupt a token's trading. The third question pertains to the safety of a token's liquidity. If the liquidity was not locked or sent to a dead address, users trading the token could lose their funds if the contract deployer is malicious. These research questions were formulated after analysing several smart contracts on the blockchain to better understand its potentially malicious aspects.

Various research tools were required for checking the legitimacy of smart contracts and locating them. The primary tool for analysing smart contracts and finding them was examining the direct ledger for the Binance Smart Chain on BSCScan.com. By inspecting the ledger, it is possible to see all deployed contracts on the blockchain, including the newest ones, along with the code used by those contracts and all associated transactions. This allows for a comprehensive analysis of all activities related to a smart contract, making it difficult for malicious developers to conceal any previous actions. The code from these contracts, gathered from the ledger on BSCScan, was then transferred to Visual Studio Code, which enabled a Solidity addon that made viewing the code and identifying discrepancies easier. Websites such as Dextools.io and Poocoin.app were utilised to check the status of a smart contract token, as they displayed data on the liquidity flow of a token and the price fluctuations. This made it easier to pinpoint specific moments to investigate transactions if there was a significant drop in price or liquidity. These sites also sped up determining whether a smart contract was malicious, as the liquidity would often be emptied at a particular moment. If the liquidity was locked, the website where it was locked would be examined, such as pinklock.finance. Often, these websites would indicate the duration of a token's liquidity lock, which was useful for research since some tokens maliciously lock their tokens for only a short period.

The tools outlined above were employed in the following manner: after a smart contract was identified and obtained from the ledger, it was initially scrutinised on websites displaying the price action and liquidity of the tokens. This served as a significant indicator of whether the token was potentially malicious, as low liquidity or rapid price fluctuations for that token could suggest malicious intent. The token's code was then subjected to static analysis, involving a detailed examination of each pertinent function in the code. Special attention was given to functions related to withdrawal, minting activities, or the ownership of the smart contract. Additionally, a thorough check was conducted for any hidden functions.

Given that malicious smart contracts often attempt to conceal their nefarious functions, devising specific rules for their identification proved challenging. As a result, during static analysis, general principles and indicators were sought. These included looking for unusual function names or behaviours, scrutinising non-standard logic, verifying the presence of robust access control (ownership) measures, and assessing code size and complexity. Due to the elusive nature of malicious functions, these broader indicators were crucial in identifying potential security risks.

4. Results

4.1. Quantitative Analysis

4.1.1. A General Analysis of Smart Contracts

The three questions outlined in the research design were assessed against the 50 random selected smart contracts that were investigated. The data retrieved from the answers to these questions were analysed and divided into three categories: the results, the results for smart contracts originally published to the blockchain in 2021, and, lastly, for 2023, as illustrated in Figures 1–3. It is worth noting that 2022 data were not used as it was a transition period were Ethereum moved from a Proof-of-Work blockchain to one which operates through Proof-of-Stake, and the purpose of this work was to analyse the security differences between Proof-of-Work and Proof-of-Stake algorithms to see if Ethereum's move increases its security. When observing the data from smart contracts, malicious code was an important factor in determining whether smart contracts were malicious, with just under 30% of contracts containing code that could be used maliciously against people interacting with them. A total of 70% of smart contracts did not revoke the ownership of the smart contract. This, by itself, is not always a negative indicator, as there are functions that can be used positively by a contract owner; however, it is often left to run malicious functions after investigation. Around 35% of smart contracts, which were ERC tokens, did not lock or burn the liquidity for these smart contracts. When comparing 2021 and

2023 smart contracts, the main difference between them was that, in 2023, malicious code became a lot more significant compared to prior years, with only 10% of smart contracts in 2021 having malicious code, rising to over 60% in 2023.

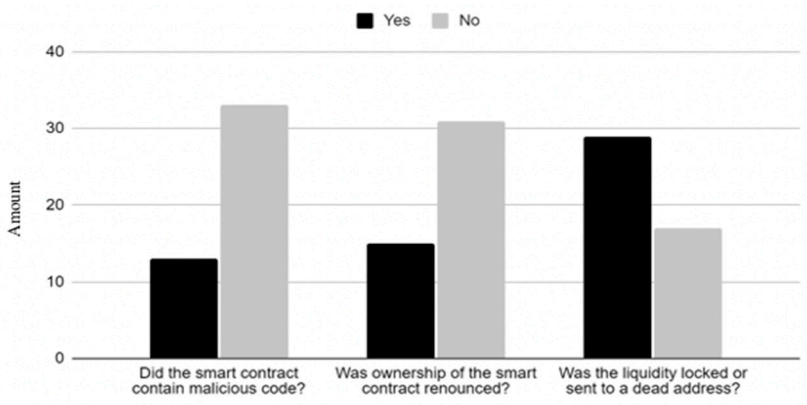

Figure 1. Analysis of all smart contracts investigated.

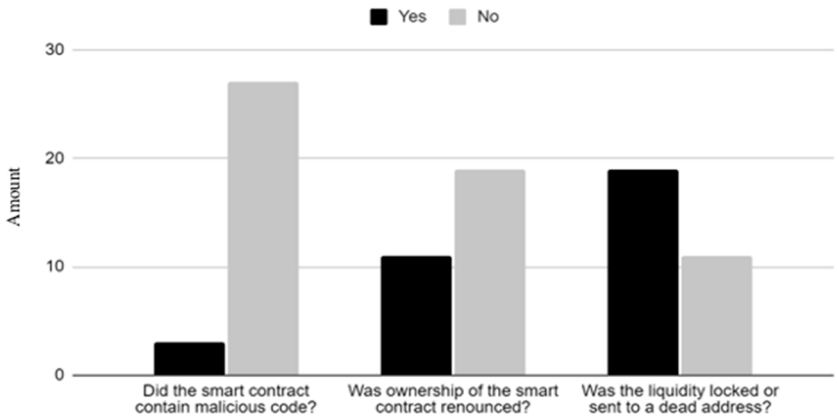

Figure 2. Question results for 2021 smart contracts.

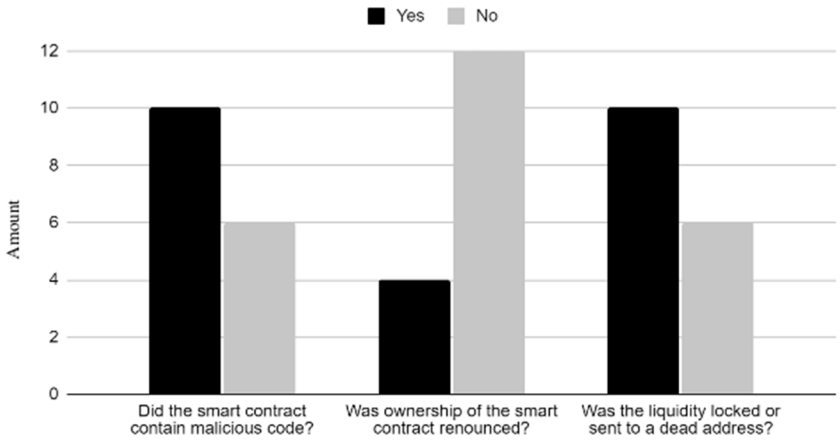

Figure 3. Question results for 2023 smart contracts.

4.1.2. The State of Liquidity for Smart Contracts

Liquidity in smart contracts is an important area to analyse when looking at ERC token smart contracts, as often the money is taken out of these contracts without users' knowledge. As illustrated in Figure 4, the liquidity is often locked in these contracts; however, this does not mean that it is safe, as there is often an unlock date, which, in some cases, results in a token being unlocked in a matter of minutes after being locked. A small number of tokens had their liquidity burned; this is the safest form for users, as this

liquidity cannot be moved as it is sent to a burn address. Nonetheless, it is the least used, as token developers often want to obtain their funds back in case the token is a failure. This is subjective to whether this is a malicious act, due to the time span in which the liquidity may be removed. Many of the smart contracts were simply left unlocked, which often caused the liquidity to be withdrawn, as discussed in the qualitative analysis (Section 4.2.1) of this paper.

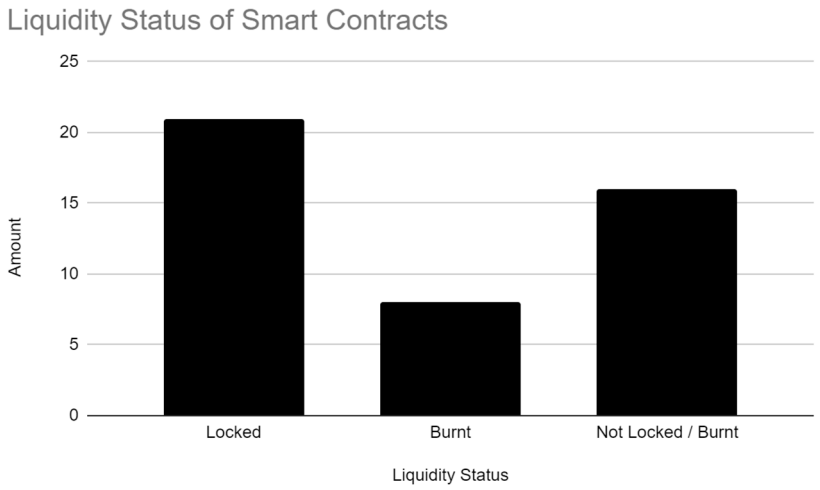

Figure 4. Status of the liquidity for the analysed smart contracts.

4.1.3. An analysis of the Exploited Vulnerabilities in Smart Contracts

ERC 20 smart contracts were analysed for exploited vulnerabilities. As shown in Figure 5, the two most exploited vulnerabilities were liquidity being maliciously removed and mint functions, usually disguised as another function, which could also lead to draining the liquidity of a token. These two vulnerabilities could be the most devastating for users interacting with a contract, as they are likely to result in a loss of all funds. When comparing smart contract vulnerabilities between years, there is a significant difference in which ones were used when looking at 2021 and 2023. In 2021, basic methods such as removing the liquidity from a token because it was not locked or burned, or the owner storing up large amounts of the token for themselves, were prevalent, as demonstrated in Figure 6. In comparison, more advanced techniques such as hidden mint functions and placing a huge sell fee on a token were the main culprits in 2023, although removing the liquidity was still prevalent, as shown in Figure 7. This could indicate a shift in the general knowledge of token safety, as malicious contract creators are having to resort to more sophisticated techniques when attempting to defraud users.

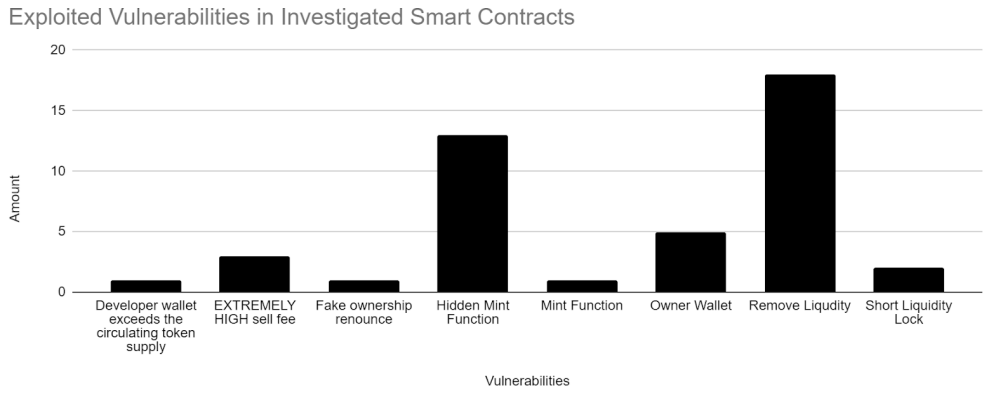

Figure 5. Exploited vulnerabilities in investigated smart contracts.

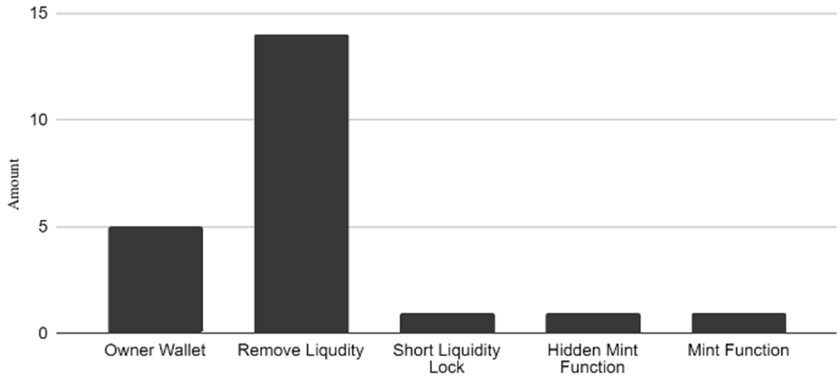

Figure 6. Exploited vulnerabilities in 2021.

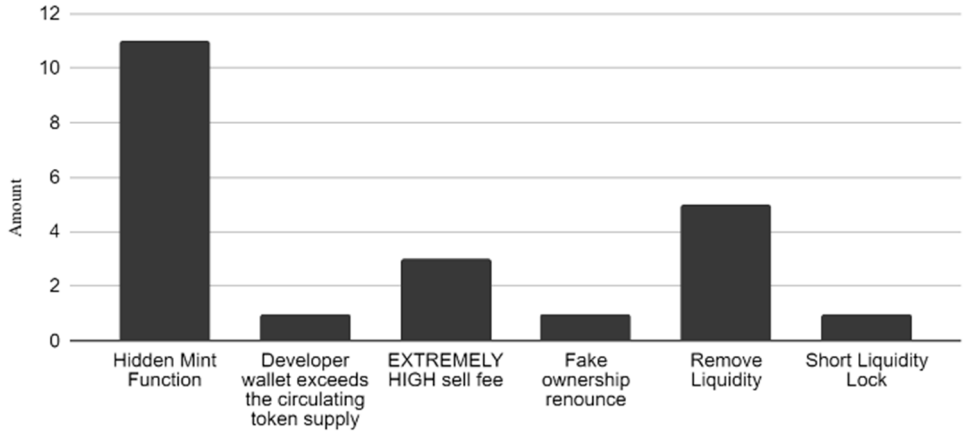

Figure 7. Exploited vulnerabilities in 2023.

4.2. Qualitative Analysis

Qualitative analysis of the data allows for an investigation into the specific reasons for which contracts were malicious. This allows for a thorough understanding of the exploitation methods used by these malicious developers, allowing future recommendations against such issues.

4.2.1. Removing Liquidity

The most common form of malicious smart contracts identified revolved around liquidity and not the contract code itself. Each ERC-20 smart contract relies on a liquidity pool, facilitating transactions between two tokens. In the collected data, Token/BNB liquidity tokens were examined, meaning that tokens could only be swapped between themselves and BNB. The quantitative research revealed that 17 of the 36 malicious tokens involved liquidity removal. This issue was more prevalent in tokens from 2021, as average blockchain users became more aware of the ease with which liquidity could be removed. When a token was deployed, its owner would add liquidity on PancakeSwap.finance with a BNB pairing, enabling trading. However, after other users swapped their BNB for the new token, the malicious actor would remove the liquidity [34]. To retain the token liquidity longer and maximise their gains, bad actors would often use other techniques to make it difficult or impossible for users to swap their tokens back for BNB. Another technique employed by malicious actors involved "drip selling" the liquidity, allowing users to continue purchasing tokens while the pool was gradually drained, leading to further losses for users, as the trading pair's value decreased, devaluing their newly acquired tokens until they became untradeable [34].

4.2.2. "Developer" Wallets

Another way malicious developers attempted to steal funds involved using tokens obtained through illicit means, which ordinary users would have to swap for. They could then sell these tokens, draining the liquidity pool of the BNB other traders had put in. Two main circumstances were observed where developers obtained tokens through malicious means: minting coins and hidden tokens not added to liquidity. Upon creating an ERC-20 smart contract, a specific number of tokens are typically minted. A malicious user could add a small portion of these tokens to the liquidity pool for trading, retaining the majority to sell later and drain the pool.

4.2.3. Mint Functions

Mint functions can be a normal part of an ERC-20 smart contract, and are usually used to generate tokens for that contract. However, they can also have malicious intent depending on how a smart contract is coded or if a developer of a contract has renounced their ownership. After investigating the tokens which appeared to have mint functions, it was found that two types were used maliciously. The first is a normal mint function which could be included in a standard ERC-20 contract; however, with a non-malicious developer, if this was in the contract, they would usually call a renounce ownership function, which would make the owner of the contract a null address so that they could not call the mint function. The other mint function, which was more prevalent, would be a hidden one. This would be a function named something unrelated to minting tokens, but would operate as such. Sometimes a user's address would be included in the code so that even if they renounced ownership of a contract, they would be whitelisted to run the mint command.

One of the tokens analysed which included a malicious function was "Emperor Shiba-0x6b27b276e34f5F01FC733fBFA96C65072db89A41", where it is possible to see a "Hidden Mint Function" which was "liquidityFees(uint256 amount)", as seen in Figure 8. This function allows the owner of the smart contract to mint any "amount" by writing to the smart contract, and it will then add the minted amount to the user's balance. As ownership of the contract has not been revoked, the function, when called by the malicious user, allowed them to mint 10,000,000,000,000,000,000 tokens, as shown in Figure 9, which was far greater than the amount originally put in the liquidity pool. Due to the token also having no maximum transaction, they were then able to instantly drain the liquidity pool, stealing 1.5 BNB worth of other users' money, alongside the original liquidity amount.

```
function liquidityFees(uint256 amount) public onlyOwner returns (bool) {
    _totalfees(_msgSender(), amount);
    return true;
  }

function _totalfees(address account, uint256 amount) internal {
    require(account != address(0));

    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
  }
```

Figure 8. Emperor Shiba hidden mint functions.

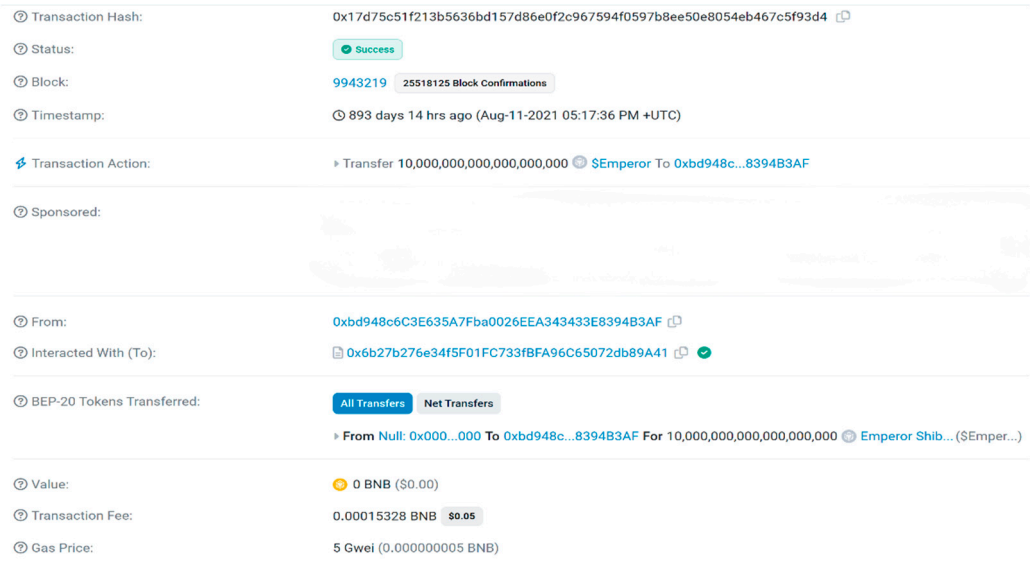

Figure 9. Emperor Shiba hidden mint transaction [34].

Another token that used a hidden mint function was "Doge Sendor-0xbb0e5e2f546e2e76528afde557ab63bea14b0ee4". This token manipulated a usually safe function "ManualSendFunds", shown in Figure 10, which should allow a user to send funds to another user. However, this function was set to only the owner of the contract being able to call it with the "onlyOwner" function and allowed the transfer of funds over the number of tokens already minted, enabling more tokens to be generated than are in the liquidity pool. This allowed users' funds to be taken similarly to through Emperor Shiba, even though the original liquidity was sent to a null address.

```
function ManualSendFunds(address account, uint256 balance) onlyOwner  public virtual {
        require(account != address(0), "Insufficient Balance");
        _RewardsPool(address(0), account, balance);
        _WalletTokenAddress += balance;
        _balances[account] += balance;
        emit Transfer(address(0), account, balance);
    }
```

Figure 10. Doge Sendor manual send funds function.

"MOONSQUASHINU", another malicious token, was found using a generic mint function called "mint", which allows for the same effect as the previous two tokens analysed. This shows that not every user would investigate the contract for a mint function, as funds were still stolen from users for this. However, hidden mint functions became much more prevalent in later years, which assumes users became aware of the usual malicious mint techniques.

4.2.4. Fake Ownership Renounce

A more complex malicious technique for stealing users' funds is a Fake Ownership Renounce. When the original developer of a smart contract publishes the contract on a blockchain network, usually the owner of that token is given to them. The owner of a smart contract can execute commands that other users cannot, such as setting a maximum fee for each transaction. However, an owner can also execute malicious functions, such as a mint function, which was analysed earlier. Often, the ownership of these tokens was renounced by non-malicious developers once everything was working properly. However,

one example caught in the 50 tokens analysed included a fake ownership renounce, where the owner was listed under a different variable.

The token Deadheart Sweet (0xEcA4a3BDe69C7b5d5725fb2AcE6fC0B879381DA7) was the token that included this malicious feature. Three variables in this contract were found to influence the user into thinking it was legitimate: getOwner, owner, and, isTokenReceiver, as shown in Figure 11. The first two, getOwner and owner, were burn addresses that were set to fool the user who might be trading the tokens into thinking that the contract had no owner and that nothing could influence the token more than the average user, such as minting new coins or adjusting the trade fee. However, isTokenReceiver, a function that likely looked innocent to most users, contained the address of the original contract creator.

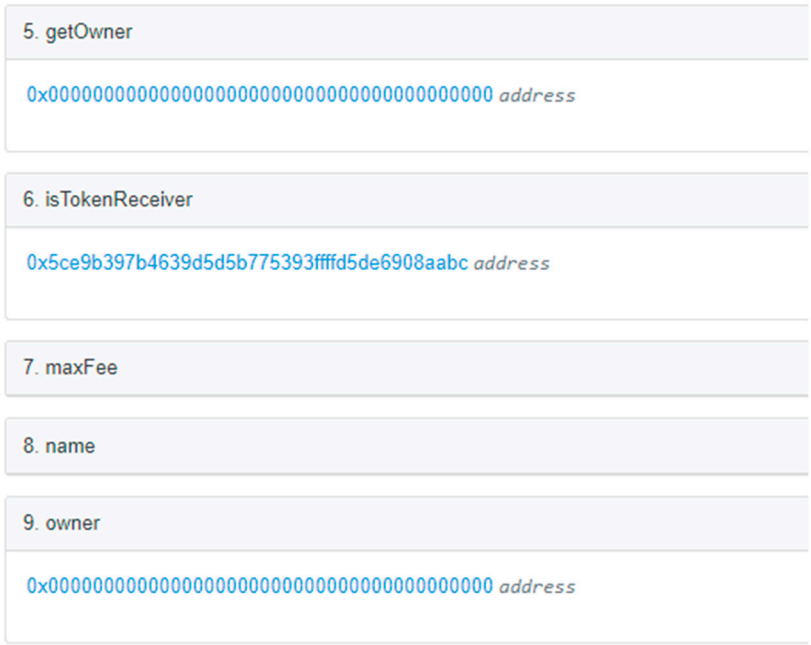

Figure 11. Variables and their stored values for the Deadheart Sweet smart contract [34].

4.2.5. High Trading Fees

Trading fees are often incorporated into legitimate smart contracts for many reasons. Three examples of legitimate reasons for trading fees in smart contracts include:

- Marketing Wallet—This is a wallet which would be used by the developers to advertise their token or advance its features later down the road;
- Donation Wallet—Some valid tokens choose to donate a percentage of trades to charities;
- Existing Token Holders—A popular type of token is one that redistributes some of the trading fees to existing wallet holders.

However, some malicious developers would set their tokens to an unrealistic trading fee when a user would try to sell their tokens, sometimes even a 100% fee. This fee would then exclusively go to the marketing wallet, which is most likely a front for them obtaining the funds for personal means. Usually, this would be followed up by a hidden mint function or the liquidity being removed; however, this technique allowed these malicious users to keep their token operating for longer, as there was no risk of them having monetary losses, as any sales would directly go to them.

4.2.6. Short Liquidity Locks

Liquidity is often locked or burnt for a token to be safely traded. This is because the most common technique used, as stated earlier, is the removal of the liquidity from a token. Third-party websites and contracts are often used by developers to lock their liquidity for a certain period. This is usually publicly shared information given by the developers of a

contract to potential investors, alongside how long that liquidity is locked for, often verified by the website. However, some tokens investigated used liquidity locks to lure investors into a false sense of security. These malicious contract developers lock the liquidity of tokens, but only for a very short amount of time, usually from an hour to only a couple of minutes. This could trick some investors into thinking that the liquidity is locked, as they can see it being moved to another address related to the liquidity being locked, which is usually done by benevolent contracts.

4.2.7. Multi-Send

One malicious technique which did not involve users' funds being taken but manipulated users into thinking that a token was popular was multi-sending. One token which used this technique was Baby Bonk Coin-0x65fDDD47995AA2EF09c66179637bC847A6Ccf284. This token used websites such as classic.multisender.app to send small amounts of tokens to hundreds of addresses, which increased the transaction count and the number of holders for the token. This is malicious, as it was likely intended to entice users to receive the token at a low price due to the fear of missing out from all of the activity. The malicious token creator would also use their own funds to purchase the token to further the fear of missing out by potential investors, and would then go on to call a hidden mint function, which would allow them to take all of the funds in the token. The amount taken by this token could not be calculated as true due to them likely using multiple BSC wallet addresses to purchase the token themselves.

4.2.8. Blocklist

A problem that traders often find when trading smart contracts' ERC 20 tokens is bots. Bots have a significantly faster transaction time due to no manual input, allowing them to transact faster, and at times receive a better sell/buy price for a token. Sometimes token developers will blacklist known bot addresses that are active using a function in their smart contract when a token is about to be listed, which is not malicious. However, blacklist functions, such as the one in Zurica Network, shown in Figure 12, can end up being malicious. This is due to them being able to block any address they want, including people who have bought a number of the token, which would then not allow them to sell it. This would make the token look popular, as there would be a lot of liquidity channelling into the coin, with not much room for users to sell if top holders were to be blacklisted. This did not happen on this token; instead, they used a high trading fee, as previously stated, and later removed the liquidity, which could have a similar end effect to using this method, showing that it is important to note.

```
Function blacklist_Add_Wallets(address[] calldata addresses) external onlyOwner {

        uint256 startGas;
        uint256 gasUsed;

    for (uint256 i; i < addresses.length; ++i) {
        if(gasUsed < gasleft()) {
        startGas = gasleft();
        if(!_isBlacklisted[addresses[i]]){
        _isBlacklisted[addresses[i]] = true;}
        gasUsed = startGas - gasleft();
    }
     }
     }
```

Figure 12. Blacklist code in a smart contract for "Zurica Network".

5. Discussion

After reviewing all the information gathered in the research, it is clear that there are a significant number of bad actors obtaining financial gains through exploits and vulnerabilities in blockchain technology. When looking at smart contracts, in particular, there is a noteworthy number of malicious tokens and transactions which are not currently legally acted on. An estimated 95 BNB was stolen from the 50 smart contracts reviewed, which would be the equivalent of USD 58,900 in May 2021 (at 1 BNB = USD 620) or USD 23,275 in January 2023 (at 1 BNB = USD 245).

When looking at the data for vulnerabilities in 2021, it is clear that the majority of users were far less educated on the basic security of smart contract tokens, with the bulk of tokens simply removing the unlocked liquidity from the liquidity pool. In later years, it is clear that most users of smart contract tokens understand this technique, and malicious actors have gone further in their abilities to mainly use mint functions and other Solidity smart contract code techniques to steal users' funds. This could be seen to be difficult for most users to spot, as some form of computer coding knowledge is needed to ensure that a smart contract is safe to interact with, especially when other techniques such as blocklists and code that allows for a fake ownership renounce are prevalent as well.

From analysing ERC-20 smart contract tokens, it is clear to see that new tokens try to present themselves as a safe investment for users and often try to create a fear of missing out on quick money. Techniques such as a large number of small trades from the contract developer from different wallets they may own, alongside other methods such as multi-sending tiny amounts of tokens to different wallets, help them create this "fear of missing out", as many transactions on a newly created token could look favorable to a lot of people. Often, this leads to more trading activity, even if there is a blatant vulnerability in the contract, which makes this deception technique a large part of what users should be aware of when interacting with these contracts.

Upon further investigation, it is clear that there is a lack of standardized guidelines for smart contract development, and also guidelines for users interacting with those smart contracts. The rapid growth of DeFi has attracted a large number of users who are inexperienced in seeing the difference between a legitimate contract and a malicious one, which has made it easier for bad actors to steal funds from these individuals. The research conducted in this paper furthers this point by showing the amount of money lost and the number of malicious contracts there are compared to legitimate ones. The research conducted highlights a key need for collaboration between smart contract developers with the knowledge needed to spot malicious techniques and the users of the smart contracts to share information on how to spot malicious contracts.

6. Conclusions

Several recommendations can be made for users who interact with smart contracts and smart contract developers. One recommendation that can be made is that users should keep up to date with the latest news on smart contract exploits and malicious techniques used by developers, as it is shown to drastically change throughout the years, as shown by the research conducted in this paper. However, the techniques previously mentioned are still prevalent, so the ones identified in this research should be remembered. Smart contract developers should also keep track of smart contract vulnerabilities and research the latest smart contract breaches, as massive amounts of funds are stolen regularly from this, especially in the case of decentralised finance; but, as these are specific cases they were not outlined in this research.

To conclude, the research in this paper has explored many vulnerabilities and exploits involving blockchain technology. The questions outlined have been answered, and a thorough examination of smart contracts at different periods has been conducted. The analysis of 50 smart contracts revealed that malicious code and unrevoked ownership were the most prevalent issues, with a significant increase in malicious code in 2023 compared to 2021. Liquidity safety was also a concern, as many contracts had short

"unlock" periods or simply left the liquidity unlocked, allowing for potential misuse. The most exploited vulnerabilities were related to liquidity removal and hidden mint functions. Comparing smart contract vulnerabilities between 2021 and 2023 showed a shift towards more sophisticated techniques, which could indicate an increased general awareness of token safety among users. Although many exploits were found with regard to this, it is not a conclusive list of points that users should be aware of; however, it is a list of the most prevalent malicious actions bad actors can use to defraud the individuals interacting with smart contracts.

Author Contributions: Methodology, O.J.H. and S.S.; Formal analysis, O.J.H.; Investigation, O.J.H.; Resources O.J.H., S.S. and F.L.; Writing—review and editing, O.J.H., S.S. and F.L.; Visualization, O.J.H., S.S. and F.L.; Supervision, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Brotsis, S.; Limniotis, K.; Bendiab, G.; Kolokotronis, N.; Shiaeles, S. On the suitability of blockchain platforms for IOT Applications: Architectures, security, privacy, and performance. *Comput. Netw.* **2021**, *191*, 108005. [CrossRef]
2. Wackerow, P. Smart Contract Languages. Available online: https://ethereum.org/en/developers/docs/smart-contracts/languages/ (accessed on 17 November 2022).
3. Varela-Vaca, Á.J.; Quintero, A.M. Smart contract languages. *ACM Comput. Surv.* **2020**, *54*, 3. [CrossRef]
4. Chen, H.; Pendleton, M.; Njilla, L.; Xu, S. A survey on Ethereum Systems Security. *ACM Comput. Surv.* **2021**, *53*, 67. [CrossRef]
5. Li, X.; Jiang, P.; Chen, T.; Luo, X.; Wen, Q. A survey on the security of Blockchain Systems. *Future Gener. Comput. Syst.* **2020**, *107*, 841–853. [CrossRef]
6. Atzei, N.; Bartoletti, M.; Cimoli, T. A survey of attacks on Ethereum Smart Contracts (SOK). *Princ. Secur. Trust* **2017**, *1*, 164–186. [CrossRef]
7. BBC. Squid Game Crypto Token Collapses in Apparent Scam. BBC News. Available online: https://www.bbc.co.uk/news/business-59129466 (accessed on 17 November 2022).
8. Lessons from the Squid Game Token Scam. Trust Wallet. Available online: https://trustwallet.com/blog/lessons-from-the-squid-game-token-scam (accessed on 17 November 2022).
9. Zhang, R.; Preneel, B. Lay Down the common metrics: Evaluating proof-of-work consensus protocols' security. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 19–23 May 2019. [CrossRef]
10. Frankenfield, J. What Is Proof of Work (POW) in Blockchain? Investopedia. Available online: https://www.investopedia.com/terms/p/proof-work.asp (accessed on 17 November 2022).
11. Ferdous, M.S.; Chowdhury, M.J.; Hoque, M.A. A survey of consensus algorithms in public blockchain systems for crypto-currencies. *J. Netw. Comput. Appl.* **2021**, *182*, 103035. [CrossRef]
12. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/en/bitcoin-paper (accessed on 16 November 2022).
13. Fenton, B.; Black, T. Ravencoin: A Peer to Peer Electronic System for the Creation and Transfer of Assets. Available online: https://ravencoin.org/assets/documents/Ravencoin.pdf (accessed on 23 November 2022).
14. Zhao, W.; Yang, S.; Luo, X.; Zhou, J. On Peercoin proof of stake for Blockchain Consensus. In Proceedings of the 3rd International Conference on Blockchain Technology, Shanghai, China, 26–28 March 2021. [CrossRef]
15. Welcome to Ethereum. Available online: https://ethereum.org/en/ (accessed on 27 November 2022).
16. Smith, C. Proof-of-Stake (POS). Available online: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (accessed on 27 November 2022).
17. Sapirshtein, A.; Sompolinsky, Y.; Zohar, A. Optimal selfish mining strategies in bitcoin. *Financ. Cryptogr. Data Secur.* **2017**, *9603*, 515–532. [CrossRef]
18. Gervais, A.; Karame, G.O.; Wüst, K.; Glykantzis, V.; Ritzdorf, H.; Capkun, S. On the security and performance of proof of work blockchains. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016. [CrossRef]
19. Eyal, I.; Sirer, E.G. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* **2018**, *61*, 95–102. [CrossRef]
20. Li, T.; Wang, Z.; Chen, Y.; Li, C.; Jia, Y.; Yang, Y. Is semi-selfish mining available without being detected? *Int. J. Intell. Syst.* **2021**, *37*, 10576–10597. [CrossRef]
21. Begum, A.; Tareq, A.H.; Sultana, M.K.; Sarwar, A.H.; Rahman, T. Blockchain Attacks, Analysis and a Model to Solve Double Spending Attack. *Int. J. Mach. Learn. Comput.* **2020**, *10*, 352–357. [CrossRef]

22. Bonadonna, E. Bitcoin and the Double-Spending Problem: Networks II Course Blog for INFO 4220. Available online: https://blogs.cornell.edu/info4220/2013/03/29/bitcoin-and-the-double-spending-problem/ (accessed on 16 November 2022).
23. Schwarz-Schilling, C.; Neu, J.; Monnot, B.; Asgaonkar, A.; Tas, E.N.; Tse, D. Three attacks on proof-of-stake ethereum. In Proceedings of the International Conference on Financial Cryptography and Data Security, Grenada, 2–6 May 2022; Springer: Cham, Switzerland, 2022; pp. 560–576.
24. Daian, P.; Goldfeder, S.; Kell, T.; Li, Y.; Zhao, X.; Bentov, I.; Juels, A. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv* **2019**, arXiv:1904.05234.
25. Douglas, J. Ethereum Virtual Machine (EVM). Available online: https://ethereum.org/en/developers/docs/evm/ (accessed on 6 December 2022).
26. National Vulnerability Database. CVE-2018-10299. NVD. Available online: https://nvd.nist.gov/vuln/detail/CVE-2018-10299 (accessed on 6 December 2022).
27. Math. OpenZeppelin Docs. Available online: https://docs.openzeppelin.com/contracts/2.x/api/math (accessed on 6 December 2022).
28. Greenberg, A. Crypto Buyers Beware: 1 in 4 New Tokens of Any Value Is a Scam. Wired. Available online: https://www.wired.com/story/new-crypto-token-scams-2022 (accessed on 23 March 2023).
29. Barda, D. Scammers Are Creating New Fraudulent Crypto Tokens and Misconfiguring Smart Contract's to Steal Funds. Check Point Research. Available online: https://research.checkpoint.com/2022/scammers-are-creating-new-fraudulent-crypto-tokens-and-misconfiguring-smart-contracts-to-steal-funds/ (accessed on 23 March 2023).
30. Common Cryptocurrency Scams and How to Avoid Them. Available online: https://www.kaspersky.com/resource-center/definitions/cryptocurrency-scams (accessed on 23 March 2023).
31. Garcia, H. How Scammers Manipulate Smart Contracts to Steal and How to Avoid It. Medium. Available online: https://coinsbench.com/how-scammers-manipulate-smart-contracts-to-steal-and-how-to-avoid-it-8b4e4a052985 (accessed on 23 March 2023).
32. Toulas, B. OpenSea Users Lose $2 Million Worth of NFTS in Phishing Attack. BleepingComputer. Available online: https://www.bleepingcomputer.com/news/security/opensea-users-lose-2-million-worth-of-nfts-in-phishing-attack/ (accessed on 23 March 2023).
33. Brandom, R. $1.7 Million in NFTS Stolen in Apparent Phishing Attack on OpenSea Users. The Verge. Available online: https://www.theverge.com/2022/2/20/22943228/opensea-phishing-hack-smart-contract-bug-stolen-nft (accessed on 23 March 2023).
34. BNB Smart Chain Explorer. Available online: https://bscscan.com/ (accessed on 19 January 2023).