A Survey of DeFi Security: Challenges and Opportunities

Wenkai Li, Jiuyang Bu, Xiaoqi Li*, Hongli Peng, Yuanzheng Niu and Yuqing Zhang

School of Cyberspace Security, Hainan University, Renmin Avenue 58, Haikou, 570228, China

ARTICLE INFO

Keywords:
Blockchain
Cryptocurrency
Decentralized Finance
Smart Contract

ABSTRACT

DeFi, or Decentralized Finance, is based on a distributed ledger called blockchain technology. Using blockchain, DeFi may customize the execution of predetermined operations between parties. The DeFi system use blockchain technology to execute user transactions, such as lending and exchanging. The total value locked in DeFi decreased from $200 billion in April 2022 to $80 billion in July 2022, indicating that security in this area remained problematic. In this paper, we address the deficiency in DeFi security studies. To our best knowledge, our paper is the first to make a systematic analysis of DeFi security. First, we summarize the DeFi-related vulnerabilities in each blockchain layer. Additionally, application-level vulnerabilities are also analyzed. Then we classify and analyze real-world DeFi attacks based on the principles that correlate to the vulnerabilities. In addition, we collect optimization strategies from the data, network, consensus, smart contract, and application layers. And then, we describe the weaknesses and technical approaches they address. On the basis of this comprehensive analysis, we summarize several challenges and possible future directions in DeFi to offer ideas for further research.

1. Introduction

The blockchain concept originated from the research of Haber and Stornetta (1990) added timestamps to text, audio, and video files in digital form to guarantee their authenticity. When Nakamoto (2008) refined the blockchain concept for the first time, blockchain had begun to serve as a decentralized network with numerous properties, attracting considerable research. At the same time, the application of cryptography principles (Nakamoto, 2008) and the promotion of consensus mechanisms (Jakobsson and Juels, 1999) have enabled digital currencies with blockchain as the core to allow mutually untrusting parties to complete transactions securely.

Suppose blockchain-based Bitcoin transactions represent the blockchain 1.0 era. In that case, the combination of smart contracts and blockchain signifies the era of blockchain 2.0. Szabo (1996) first introduced the concept of the smart contract, which denoted a promise or agreement in digital form. Buterin et al. (2014) proposed Ethereum, which updates and verifies blockchain data via the state. Ethereum is currently a significant platform for smart contracts and decentralized applications.

In addition, Decentralized Finance (DeFi) is a decentralized application that uses blockchain in the financial domain to implement pre-defined financial protocols. Blockchain technology is widely used in various fields, such as education, health, and finance. Moreover, because Ethereum blockchain technology integrated with finance better during the Bitcoin period, DeFi technology in the financial field is gaining more attention.

Moreover, the blockchain serves as the foundation of the DeFi application and enables transactions on DeFi to be completed securely. Blockchain's consensus mechanism ensures the integrity of DeFi transactions. The consensus mechanism selects the ledger nodes for the blockchain. The nodes with bookkeeping rights incorporate the DeFi application's transactions into a new block. The proper execution of the financial logic of the DeFi application relies on smart contracts (Jensen et al., 2021). The smart contract isolates from the outside world and cannot be modified once deployed on the blockchain. In detail, to get reliable real-world asset price information, DeFi introduces an oracle (Werner et al., 2021), which is a system to provide real-world financial asset price information.

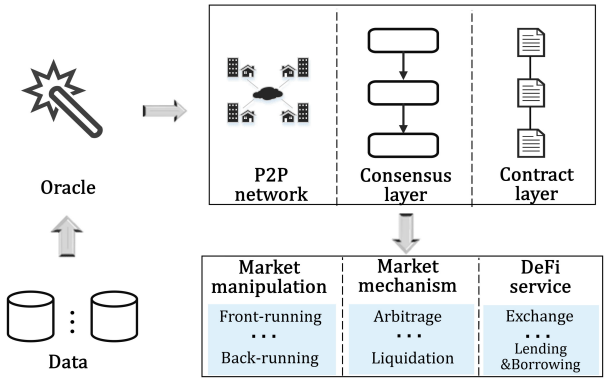

Figure 1: Overview of Research Ideas and Analysis Paths on DeFi Security.

With the rapid development of DeFi, it can be divided into stablecoin, Decentralized Exchange (DEX), cryptocurrency market, and insurance. Additionally, it had locked in $200 billion until April 2022 (Shaman et al., 2022). However, the value locked up in the entire DeFi dropped by around $85 billion in July 2022, causing us to ponder the security of DeFi.

While some studies about the risk of DeFi in Table 1, they paid more attention to financial issues. Werner et al. (2021) classified attacks according to risk categories from an economic perspective. Qin et al. (2021b) systematically and quantitatively compared various lending systems and mea-

Email: csxqli@gmail.com
*Corresponding author.

Table 1
Comparison of Our Study and Other DeFi Security Related Literature.

Reference	Contributions	Date	Categories
Jensen et al. (2021)	It focuses on the analysis of financial services. It classifies the risks of users, liquidity providers, arbitrageurs and application designers separately.	April 2021	Financial Risk
Werner et al. (2021)	It focuses on the economic aspects and classifies the financial risks encountered by DeFi. And it analyzes the DeFi protocol and ecosystem.	September 2021	Financial Risk
Qin et al. (2021b)	It first introduces the breadth of the lending market (a DeFi service). It quantifies the instability of lending protocols.	November 2021	Financial Risk
Gudgeon et al. (2020)	It introduces a new type of Flash Loan attack and demonstrates the weaknesses and price fluctuations of the DeFi protocol.	June 2020	Technical Risk
Qin et al. (2021a)	It compares the differences between traditional CeFi and DeFi, including legal, economic, and security.	June 2021	Financial Risk
Amler et al. (2021)	It classifies DeFi services through the economics dimension, highlighting the advantages of DeFi compared to traditional finance.	September 2021	Financial Risk
Bartoletti et al. (2021)	It formalizes DeFi theory in order to analyze various DeFi incentive mechanisms and design principles.	September 2021	Technical Optimization
Liu et al. (2020)	Markov Chain and volatility prediction risk management are proposed. Loss distribution reduces mortgage rates, and VaR calculates external risks.	October 2020	Technical Optimization
Wang et al. (2021a)	It proposes a DeFi attack detection system that collects and analyzes transactions using symbol execution and transaction monitoring.	March 2021	Technical Optimization
Bekemeier (2021)	It presents the first systematic risk and is the first empirical guide to stylized facts both at the technical level and economic level.	December 2021	Technical and Financial Risk
Our study	It is the first to provide a systematic summary of DeFi security incidents and systematically analyze the vulnerabilities. We also provide future directions.	____	Systematic Review

sured the risks that participants may encounter. Gudgeon et al. (2020) described the design flaws in lending protocols and DeFi losses due to price volatility. Qin et al. (2021a) systematically compared Centralized Finance (CeFi) and DeFi, including legal, economic, and market. Bartoletti et al. (2021) formalized the DeFi theory, which was used to understand systematically and analyze the incentives in DeFi to balance interest rates and prices. Other studies proposed by Jensen et al. (2021) and Amler et al. (2021) were used to analyze the risk of assets in DeFi on Ethereum.

In addition to the research of financial risks in DeFi, optimization schemes were also widely studied, as shown in Table 1. Liu et al. (2020) used a mathematical-statistical approach to the market for four types of assets and clearing to construct MovER, a framework for controlling the risk of the system. Wang et al. (2021a) proposed Blockeye, which constructed state dependencies from smart contracts and used the collected transactions to analyze whether it is subject to a DeFi attack. Even though there are some optimized solutions to vulnerabilities, attacks keep appearing, such as the Ronin Bridge incident (Network, 2022).

Similar work to ours was proposed by Bekemeier (2021), it discussed systemic risk, both at the technical level of the blockchain and the economic level and provided experience analysis. The difference is that our work is more comprehensive. Our work in this paper systematically summarizes vulnerabilities at all technical levels, following the analytical path shown in Figure 1. In addition, we analyze the attack events caused by the vulnerabilities. Most importantly, we also summarize the most state-of-the-art optimizations at each layer. Finally, we conclude with some challenges and possible future directions.

The main contributions of this paper are as follows:

(i) To the best of our knowledge, we conducted the *first* systematic examination of the security issues of the DeFi ecosystem built on blockchain.

(ii) We systematically summarize the vulnerabilities of the Ethereum-based DeFi system, investigate real-world attack events related to DeFi and classify them according to their vulnerability principles.

(iii) We survey the security optimizations in DeFi from the system level and conclude the challenges to suggest future research directions in this area.

The rest of the paper is structured as follows. Section 2 presents the background of the paper. In Section 3, we examine some vulnerabilities in DeFi, and in Section 4, we analyze real-world attacks. Section 5 provides several security optimization strategies, while Section 6 highlights DeFi's challenges and future directions. Finally, Section 7 concludes the paper.

2. Background

2.1. Ethereum

Ethereum is a public blockchain system initialized using the Proof-of-Work (PoW) consensus mechanism, in which miners fight for control of blocks using computing power in

exchange for incentives (Li et al., 2020c). However, it has subsequently shifted to the Proof-of-Stake (PoS) algorithm, which is based on the quantity and age of stakes held (Wahab and Mehmood, 2018). It first uses the Turing-complete programming language Solidity, Vyper and others to develop smart contracts (Chen et al., 2020c; Li et al., 2020b). Anyone can deploy Decentralized Applications (DAPPs) on the Ethereum chain that can communicate with others. The most popular application in the financial field is DeFi, which provides a wide range of financial services.

2.1.1. Layers of Dapp on Ethereum

Dapps, like traditional software architectures, may be separated into six layers as follows (Duan et al., 2022):

(i) The data layer handles off-chain data before passing it on to the network layer.
(ii) The network layer is peer-to-peer, assuring network node autonomy.
(iii) The consensus layer guarantees that miners wrap network layer requests into blocks.
(iv) The incentive and consensus layers are interrelated, and the incentive layer ensures that miners do not behave maliciously.
(v) The smart contract layer connects the consensus with application layers and exchanges data between them.
(vi) The application layer binds the information from the smart contract layer and shows it to the user after processing.

2.1.2. Transaction Process on Ethereum

When a user interacts with the applications and begins a transaction request using the interfaces provided by the smart contract, the transaction request broadcasts to all nodes on the P2P network chain. When the miner gets the request, it selects and packages the transaction into blocks. The miner adds blocks to the chain using the consensus algorithm and synchronizes them with all nodes on the network. Simultaneously, the smart contract changes the state variables depending on transaction data and visualizes them in the application.

2.1.3. Geth

Go-Ethereum (Geth) is an official Ethereum client implemented in the go programming language (Adam et al., 2013). It includes instructions for several tasks, such as creating an Ethereum private chain and interacting with the network environment.

2.1.4. Gas

To avoid the overuse of network resources, all transactions on Ethereum are paid a cost called `gas` (Chen et al., 2017a), and the transaction fee equals the amounts of `gas` multiplied by `gasPrice` (Chen et al., 2020b). The user who proposes transactions sets the `gasPrice`, and miners with high computing resources would conduct the transaction earlier if the `gasPrice` is high. There is also a concept called `gaslimit`, which is used to limit the maximum amount of `gas` that can be used for a transaction (Chen et al., 2017b). It means that the maximum charge for a transaction is `gaslimit` multiplied by `gasPrice`.

2.1.5. Consensus Mechanism of Ethereum

The fundamental technology of blockchain is the consensus mechanism, which ensures the blockchain's secure, stable, and efficient operation. At the same time, the consensus mechanism enables the "mistrustful" parties on Ethereum to complete the verification and confirmation of transactions. Researchers (Lashkari and Musilek, 2021) are continuously improving various consensus mechanisms such as PoW, PoS, Delegated-Proof-of-Stake (DPoS), and Practical Byzantine Fault Tolerance (PBFT).

Ethereum still uses PoW as its consensus mechanism. Nakamoto (2008) proposed poW to prevent double-spending in cryptocurrencies. The core idea of PoW is to compete among nodes for the bookkeeping rights and rewards of each block through their computing power (Mingxiao et al., 2017). All miner nodes in the network use the information in the previous block, such as previous block hash, timestamp, and nonce, to determine the next block. In PoW, miner nodes find the hash value by continuously trying random number nonce, which is difficult to calculate but simple to verify.

By PoW's high consumption of resources, so Ethereum intends to use PoS as the new consensus mechanism. In 2011, quantum mechanical proposed POS, whose core idea is that the greater the ownership of a node to a specific amount of cryptocurrency, the greater the equity of the node (Mingxiao et al., 2017). In PoS, it filters nodes by calculating the number of currencies in the nodes as a percentage of the total currencies and the time of holding currencies. This approach selects nodes first and then performs arithmetic operations, which means that many computational resources are not wasted.

2.1.6. Maximal Extractable Value (MEV)

The Ethereum consensus shift caused several definitions to be updated. Initially, MEV was the miner extractable value, but now the maximum extractable value makes more sense.

Miner extractable value refers to the profit miners make by performing a series of operations on the blocks they mine (Qin et al., 2022). For example, miners reorder transactions to optimize the initial ordering of transactions and earn additional Ordering Optimization (OO) fees (Daian et al., 2020). And the phenomenon that miners sell priority in blocks to make users keep raising the cost of gas is called Priority Gas Auctions (PGA).

Maximal Extractable Value is the maximum value that the validator V can extract by reordering, inserting, or not executing the transactions $T_{i,...,j} = \{t_i, ..., t_j\}$ in the block. In addition, we assume that the balance in V before the transaction is $b(s)$ and $b(s')$ is after the transaction. So the value obtained by sequential execution $EV(V, T_{i,...,j})$ equals $b(s') - b(s)$, and $R(T_{i,...,j})$ means the order of transactions is in full array. Thus the maximal extractable value MEV can be de-

fined as $MEV = max(EV(V, R(T_{i,...,j})))$.

2.2. DeFi

2.2.1. Development of DeFi

The introduction of blockchain technology (Nakamoto, 2008) has changed the traditional financial ecosystem. With the advent of Ethereum, smart contracts became the basis for the development and implementation of DeFi. Since the landing of MakerDAO in 2014 which is the first Ethereum-based DeFi project, several DeFi protocols have emerged to implement functions of traditional CeFi, such as lending platforms, exchanges, derivatives, and margin trading systems (Wang et al., 2022). As liquidity mining mentioned in 2020, DeFi was pushed into high gear with the emergence of decentralized exchanges such as Compound, which are entirely managed by smart contracts. Money Legos brings unlimited creativity to DeFi products. It means that a new financial product can be realized by combining the underlying DeFi protocols (Popescu et al., 2020). In 2022, regulated Decentralized Finance (rDeFi) becomes the new trend in DeFi development (Coinchange, 2022).

2.2.2. DeFi Service

As seen in Figure 1, DeFi applications can be made up of DeFi services, also known as protocols, such as exchange, lending, and asset operation. Blockchain will wait for assets or data to be processed through protocols before uploading them to the application layer, which is the market (Schär, 2021). The DEX serves as a forum for asset suppliers and buyers to engage, it can separate into two types: centralized order system and Automated Market Maker (AMM) (Zhou et al., 2021b). The former is comparable to a regular exchange in that customers produce trade orders following transactions start. The latter is accomplished quickly by initiating a transaction using a previously constructed asset price algorithm.

2.2.3. Market Mechanism

In addition to technological issues, DeFi has an economic mode of operation, which is the market mechanism. Users can control and alter numerous assets using the DeFi service normally. However, attackers can benefit by manipulating the asset through market-based strategies at the economic level.

3. Analysis of Vulnerabilities

From the proposal of Ethereum to 2022, various vulnerabilities have emerged to promote the ecological development of DAPPs. Therefore, studying the vulnerabilities related to DeFi helps understand the defense methods of attacks. To summarize threats in DeFi, we focus on data, consensus, contract, and application layers.

3.1. Data Security Vulnerabilities

For the data layer, if attackers change the data under the chain during the uploading process to the chain, it will result in irreversible mistakes due to the immutability of the blockchain. Figure 2 shows that it could encounter oracle mechanism vulnerability and inappropriate key management.

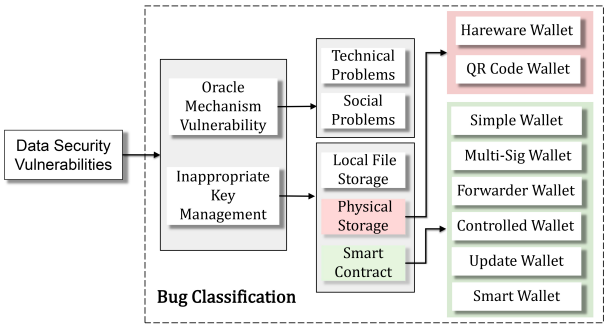

Figure 2: Classification of Data Security Vulnerabilities.

3.1.1. Oracle Mechanism Vulnerability

The oracle is an automated service mechanism that allows the system to obtain the off-chain asset price data as input (Werner et al., 2021). And smart contracts rely on the exchange rates of prices provided by oracle for proper operation. However, as Figure 1 shows, the risk to oracle grows drastically when a single point of failure occurs. For example, over 3 million sETH were arbitrated due to the oracle errors in SYNTHETIX, a protocol that converts entity into synthetic (Synthetix, 2019). Oracle risks can be divided into technical and social problems.

Technical oracle problems may be defined as a process of passing data with three key elements:

(i) How to collect all the data accurately?
(ii) How to process the data with as few errors as possible?
(iii) How to upload the processed data to smart contracts?

Furthermore, the current oracle form may be centralized and distributed. Centralized oracle uses trusted third parties to collect, process, and transfer data to smart contracts. Distributed oracle consists of numerous nodes that take data from multi-sources and process it using an algorithm, such as a consensus (Kumar et al., 2020) or weighted voting method (Angeris and Chitra, 2020). Finally, the oracle system assesses the chain information.

There are not only technical problems but also social problems in oracle (Caldarelli and Ellul, 2021; Egberts, 2017). Assuming such a game where there exists an Oracle O_i. The O_i picks the off-chain data and processes it as $D_n = (d_i, ..., d_j)$. The contract $S_i = (f_{i1}, ..., f_{im})$ uses D_n for transactions Tx_i, where f_{ii} is the i_{th} function in the contract S_i. If an attacker a_i pays c to modify d_i in D_n, and obtains benefits b_{ii}. When the cost c by the attacker is less than the benefits b_{ii}, the attacker gets a profit that would be attractive to other attackers. While the c cannot be measured directly from technical methods, it requires analysis of specific social situations, so the oracle problem is controversial in terms of social issues.

3.1.2. Inappropriate Key Management

In the DeFi ecosystem, wallets are used to manage private keys, and asset authentication is based on keys in most

Table 2
Comparison of Different Key Storage Methods on Ethereum.

Wallets	Descriptions	Features				
		Flex	Sec	Sca	TP	TxC
Local Storage	Keys are stored centrally in the file system by default	-	-	-	-	-
Hardware Wallet	Hardware devices can isolate external networks and transport operations	×	✓	×	×	-
QR Code Wallet	QR code generated from the address and scanned to obtain the address	×	✓	×	×	-
Simple Wallet	It can simply handle cryptocurrencies and tokens for raw transactions	✓	-	✓	-	✓
Multi-Sig Wallet	The transaction process requires multiple owners to sign to ensure users' security	×	✓	✓	✓	✓
Forwarder Wallets	Forwarding assets to a master wallet and users only need to preserve the subkey	✓	✓	✓	×	✓
Controlled Wallets	The third party keeps the key and anyone who uses the key needs authorization	×	✓	×	×	✓
Update Wallets	Users can customize the update by selecting some parts to be updated	✓	✓	✓	×	✓
Smart Wallets	Wallets with enhanced functionality that achieve expansion of normal functions	✓	×	✓	×	✓

cases. However, similar to Bitcoin, the DeFi system suffers from the problem of improper key management. Existing key management methods, such as physical storage (Shbair et al., 2021; Dabrowski et al., 2021), offline wallets (Khan et al., 2019; He et al., 2018), and password-derived wallets (Kaliski, 2000), have some drawbacks. In Table 2, we summarize nine forms of wallets, where local storage is the initial form of local file storage, hardware wallets and QR code wallets both belong to physical storage wallets. The remains belong to smart contract wallets. Moreover, Flex in Table 2 is the Flexibility, above the Local Storage is a ✓, and vice versa is a ×. The same applies to Sec, Sca, TP, and TxC, representing Security, Scalability, Transparency, and Transaction Costs, respectively.

In Ethereum, users can access the Ethereum chain by using Geth. When a user creates an account a_i, the client generates a file to be stored locally, which contains the unique key key_i associated with the account a_i. Before the account initiates a transaction Tx_i or mining, the client reads the key_i in the file. However, anyone without restricted access can read the file and even falsify $(key_i, ..., key_j)$ for profit.

There are three types of wallets, software, hardware, and paper, depending on the form in which they exist (Suratkar et al., 2020). Hardware and paper-based storage, which are physical storage, are more secure because they store keys in a way that isolates them from multi-user interaction. Nevertheless, it also has the weaknesses of poor scalability (Arapinis et al., 2019) and the inability to have a single point of failure caused by the architecture design (Dabrowski et al., 2021).

Di Angelo and Salzer (2020) divided smart contract wallets into six types. They restrict the direct access to assets and provide some Application Binary Interfaces (APIs) for manipulating data.

- **Simple Wallet:** It is the initial form of wallet, offering simply raw transaction capability and storing all keys in files. When a malicious parity obtains file system permissions, keys can be read or even manipulated.
- **Multi-signature Wallet:** It requires the co-signature of many owners for increased protection. The combination of many signatures dilutes the individual's influence, providing decentralization. And the public multiple signature combination could enhance transparency.
- **Forwarder Wallet:** It adds forwarding operations to the signing process, such as password-derived wallets, which allow users to customize the master key and then derive sub-keys from controlling the asset. The forwarding operation faces a balance between transparency and security. If the derivation algorithm is publicly available, attackers who got the master key in some ways will reproduce the derivation process to obtain all sub-keys.
- **Controlled Wallet:** The custodial wallet is an example of a controlled wallet since it keeps ownership of the account and grants access to users. It offers some protection by centralized management, but the non-transparent action also tests managers' credibility.
- **Update Wallet:** Update wallets permit users to modify updates depending on features, allowing for greater flexibility in wallet operation. However, compatibility across many versions might result in worse security.
- **Smart Wallet:** Smart wallets include some sophisticated features, such as key recovery. As a result, the smart contract enables wallets to execute a range of services in addition to transferring money, but it adds to the dangers involved with smart contracts in 3.3.

3.2. Consensus Mechanism Vulnerabilities

Blockchain, such as Ethereum, is consensus-based. Up to now, many significant works have already been done in design, testing, auditing, and maintenance. So there aren't many consensus flaws, but we gather the consensus bugs that occurred in Geth according to (NVD, 2022; Yang et al., 2021; Luu et al., 2015) in Table 3 and we classify them from three aspects in Figure 3. There are four severity categories, with low suggesting that the developer resolved before they occur. The middle level was deployed to the test network before discovered, while the High was in the main chain. The

Table 3
Summarization of Consensus Vulnerabilities on Geth that have Endangered DeFi.

Brief Explanation	Descriptions	Date	Severity
Journaling Mechanism	Geth can't restore a deleted empty account due to out-of-gas	November 2016	High
EVM Stack Underflow	SWAP, DUP, and BALANCE underflow the EVM stack	February 2017	High
Stack Elements	In a static environment with fewer than three stack elements	October 2017	Low
Encryption Algorithm	The elliptic curve algorithm was not fully validated	February 2018	High
Timestamp Overflow	Timestamp, state variables in blocks, overflow	March 2019	High
Shallow Copy	Pre-compile contract, making Geth inconsistent with memory	July 2020	High
Ether Shift	Transfering the balance of the deleted account to the new account	August 2020	High
Certain Sequences	Certain transaction sequences can lead to the failure of consensus	December 2020	Middle
Incorrect Requirements	Failure to properly authorize timestamp leads to double spending	February 2021	High
Memory Corruption	RETURNDATA corruption due to data replication, resulting in forking	August 2021	High
Denial of Service (DoS)	Combination of short-term restructuring and delayed consensus decision	October 2021	Critical
Bignum Overflow	Some large values in consensus specification overflow leads to a fork	April 2022	High

critical one implies that the vulnerability is widely available and has a significant impact on the integrity of the network.

Certain malicious behaviors utilize consensus rules to affect the sequences of transactions. There are a variety of attacks combined with MEV, such as flash loans (Qin et al., 2021c; Zhou et al., 2021a), sandwich attacks (Zhou et al., 2021b; Qin et al., 2022), and forking attacks (Daian et al., 2020). As Figure 3 depicted, we classify this part into three segments: 1) Transaction Order Vulnerability; 2) Forking Vulnerability; 3) State Management Vulnerability.

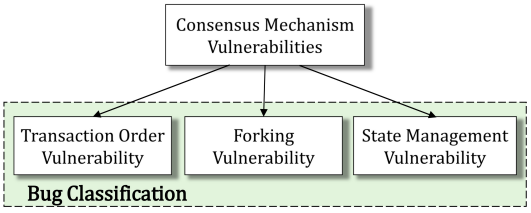

Figure 3: Classification of Consensus Mechanism Vulnerabilities

3.2.1. Transaction Order Vulnerability

Transaction order vulnerability describes that an attacker alters the initial sequence of transactions by leveraging the miner's desire for profit. The sandwich attack is a typical example. The attacker predicts that the victim will buy asset A, and pays a higher gas fee to acquire it before the victim at a lower price. And then, they sell A at a higher price for arbitrage since the victim's purchase boosts the price (Zhou et al., 2021b).

3.2.2. Forking Vulnerability

Forking events in DeFi are generally associated with transaction fee-based forks and time-bandit attacks (Daian et al., 2020). Mining revenue incentivizes miners to perform honestly, but the OO fee motivates them to reorder transactions in the block, enhancing the income. Most bugs contain forking vulnerabilities in Table 3, for example, memory corruption, incorrect requirements, shallow copy, and certain sequences.

3.2.3. State Management Vulnerability

Transactions in Ethereum are based on updating states between blocks (Wood et al., 2014). According to the consensus rules in Ethereum, the confirmation between the old and new blocks needs to be completed within 12 minutes. Therefore, if attackers complete the extraction of the state variables within the block, then they can attack the transaction within the specified time. For example, timestamp overflow and incorrect requirements are in Table 3. The former is because the timestamp exceeds the representation of `uint64`, resulting in a hash error in the block (Yang et al., 2021). The latter is that the timestamp in a block gets the permission mistake, which means the block to be refused by the chain permanently, causing a chain fork and the execution of a double-spending attack(NVD, 2022).

3.3. Smart Contract Vulnerabilities

There are 20 types of smart contract vulnerabilities in Ethereum defined in (Chen et al., 2020a), of which Table 4 shows the weaknesses that attackers might use to make a profit. We searched Common Vulnerabilities & Exposures (CVE) and summarized over 500 vulnerabilities (CVE, 2022). In Figure 4, we describe the classification of smart contract vulnerabilities in this paper. And Table 4 shows that bugs written by Solidity were categorized into several types as detailed below:

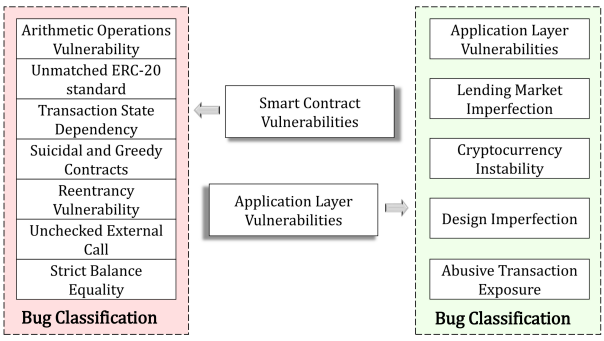

Figure 4: Classification of Smart Contract and Application Layer Vulnerabilities

Table 4
Summarization of Smart Contract Vulnerabilities in DeFi

Categories	Causes	Categories	Causes
Unchecked External Calls	Without checking return values	Reentrancy	Repeated calls before completed
Unexpected Permission Check	Failure to check permissions	Nested Call	Unrestricted call depth
DoS Under External Influence	External exceptions inside loops	Missing Return	Denote return but no value
Unmatched ERC-20 Standard	Not follow the standard	Greedy Contracts	Receive but do not withdraw Ethers
Strict Balance Equality	Balance check failed	Block Info Dependency	Status in blocks leakage
Misleading Data Location	Incorrect `storage` type	Missing Interrupter	No backdoor to handle crises
Transaction State Dependency	Error using `tx.origin`	Arithmetic Bugs	Unmatched type to values

3.3.1. Arithmetic Operations Vulnerability

In Solidity, bugs such as integer overflow, float lack of precision, and division by zero are common during arithmetic data operations.

An upward overflow can occur if a `memory` integer exceeds the maximum range, e.g., `uint256` is a default type of integer that can express the number from 0 to $2^{256} - 1$. In Listing 1, the function allows the `owner` to add tokens to the user, but a sufficient `amount` on line 3 can make the balance in `balance[target]` vanish.

Since Solidity lacks the float type of data structure, the phenomenon in which the float result of an operation might lose coins. When one integer is divided by a larger integer, the result is always 0. For example, 1 ETH divided by 10 Eth equals 0. Even some contracts do not restrict the operation of division by zero, which results in code logic errors as the result of the calculation becomes big infinitely.

```
1  function mintToken(address target, uint256
       amount) onlyOwner{
2    require(target != 0x0);
3    balance[target] += amount;
4    totalSupply += amount;
5    Transfer(0, this, amount);
6    Transfer(this, target, amount);
7  }
```

Listing 1: Integer Overflow Instance

3.3.2. Unmatched ERC-20 Standard

Ethereum provides various APIs for developers to implement certain functions, such as transferring money, but some developers may not adhere to all standards, resulting in problems in smart contracts. The ERC-20 standard is one of the APIs used to manipulate cryptocurrencies, including how to transfer tokens between addresses and access token data (Richards et al., 2022a). When transferring tokens, for example, `transfer()`, `transferFrom()`, and `approve()` will return a `boolean` value to indicate whether the function succeeded, and many smart contracts cause transfer mistakes since they do not verify the return value.

3.3.3. Transaction State Dependency

Contracts should check the permissions of certain sensitive invocations that use the global variable `tx.origin`, which points to the address in the entire call stack where the transaction was originally sent (Chen et al., 2021). Assume the `Wallet` contract in Listing 2 sends a transaction to the `Attack` contract, and then the `attack()` function invokes the `transfer()` function in the `Wallet` contract, at which point `tx.origin` meets the detection in line 6, making the success of the attack.

```
1  contract Wallet{
2    address public owner;
3    constructor() payable{
4      owner = msg.sender; }
5    function transfer(address to, uint amount)
       public{
6      require(tx.origin == owner);
7      (bool sent,) = to.call.value(amount);
8      require(sent,"Failed to send Ether");
9    }
10 }
11 contract Attack{
12   address payable public owner;
13   Wallet w;
14   constructor(Wallet wal){
15     w = Wallet(wal);
16     owner = payable(msg.sender); }
17   function attack() public{
18     w.transfer(owner,address(w).balance); }
19 }
```

Listing 2: Transaction Dependency Instance

3.3.4. Suicidal and Greedy Contracts

Smart contracts usually include a provision enabling the owner to commit `suicide` if the contract is challenged. The SELFDESTRUCT Operational Code (opcode) in a suicidal contract can ignore all contract code logic, even the `fallback()` function (Li et al., 2021). However, attackers utilize this feature to corrupt the logic of some contracts, which leads to restrictions on all other operations that depend on the contracts. For example, the Parity wallet was attacked by a suicidal contract in 2017 (Li et al., 2020a), which resulted in a permanent lock of all cryptocurrencies that transferred to the wallet before the wallet maintainer fixed the vulnerability.

Similar to the suicidal contract, the greedy contract locks up the ether, but it is alive. Greedy contracts do not have instructions related to the withdraw and send (Nikolić et al.,

2018), such as `send`, and `transfer`, so it locks all ethers and cannot withdraw. Therefore, making sure there are means to get ether out before transferring it to a contract (Chen et al., 2020a).

```
1  function payOut(address recipient, uint amount)
       returns(bool){
2    if(msg.sender != owner || msg.value>0 ||(
       payOwnerOnly && recipient != owner))
3      throw;
4    if(recipient.call.value(amount)()){
5      payOut(recipient, amount);
6      return true;
7    }else{
8      return false;
9    }
10 }
```

Listing 3: Reentrant Vulnerability Instance

3.3.5. Reentrancy Vulnerability

The concept of threads does not exist in Solidity, so it cannot execute more than two operations concurrently. This means that when a contract initiates a call via `call()`, it must wait for the completion of the call before making the next call. However, it would be attacked if the callee contracts change the global state during the waiting (Luu et al., 2016). The DAO attack leverages the recursive invocations to make the system keep cycling until internal assets run out. It exits in line 4 of Listing 3 (Daian, 2016), where the original `recipient` continues executing `call.value()` after a successful transfer.

3.3.6. Unchecked External Call

The return value or the arguments of an external call can affect the states of the code, and many contracts do not check the return value leads to vulnerabilities. The mode of logic used in this bug is similar to that of misuse ERC-20 standard. When a function calls code logic outside the contract, it is equivalent to the entire runtime in a black box. At this point, failure to check the return value of the external call may cause the logic of the contract to break. For example, when multiple functions are nested, and the external call does not check the return value of the internal call in time can go wrong (Chen et al., 2020a).

Smart contracts in the DeFi trade by using external call functions including `delegatecall()`, `call()`, `send()`. More crucially, a failed external call in these methods results in a transaction not being rolled back, which can cause logical effects.

3.3.7. Strict Balance Equality

Equations are commonly used in programs to make decisions concerning contract logic. When an attacker employs some methods, such as a suicide transfer ether, to alter the state of the variables utilized in the equation, rendering the judgments of the equation incorrect, the attack affects the logic of the code that follows the equation. For example, in Listing 4, when the balance in the account is 1 ether and it passes the check in line 2. In line 3, the attacker transfers ether into the account, causing the judgment to fail, so the transfer in line 4 does not follow the normal logic. It is a loophole caused by not fully checking the judgment conditions of the equation.

```
1 function receive(address a) payable{
2   if(msg.value > 1 ether) throw;
3   if(this.balance == 1 ether){
4     a.send(1 ether);
5   }
6 }
```

Listing 4: Strict Balance Equality Instance

3.4. Application Layer Vulnerabilities

The application layer visualizes the state in the chain and interacts directly with the user. In this paper, we focus on DAPPs in the financial domain. DeFi applications, in general, suffer from price manipulation attacks similar to traditional centralized financial applications. With the current development, the problems in the application layer could be divided into lending market imperfection, cryptocurrency instability, design imperfection, and abusive transaction exposure in Figure 4.

3.4.1. Lending Market Imperfection

When the prices in the market are out of balance, it will result in bad debts for one of the participants in the lending market. To get more loans, attackers can boost the cryptocurrency exchange rate on the oracle by modifying the real-time price-related status before the loan is made. For example, an attacker can gain a larger quantity of tokens by directly manipulating token prices in the asset pool or increasing the price of collateral before lending (Wu et al., 2021), putting the borrower in danger of bad debt.

3.4.2. Cryptocurrency Instability

The large fluctuations of cryptocurrencies are caused by a variety of factors, including pump-and-dump schemes. The instability can easily trigger liquidation procedures. DEX have chosen stablecoins, which are tied to the price of real-world money, as the pricing standard to minimize losses, but they still exist as a risk. For example, a 99.98 % plunge in May 2022 in the price of the luna coin, whose value is tied to a stablecoin called Terra, left the entire crypto market with over $700 million in collateral liquidated (Lyanchev, 2022).

3.4.3. Design Imperfection

The attackers utilize incorrectly configured functionality or specific convenience features of DeFi platform exchanges (Wang et al., 2021b). Flash loan is designed as risk-free loans to be a convenient improvement to the loan that needs to borrow the flash loan, exchange it for currency, and repay the loan in an atomic transaction. For example, attackers borrow the flash loan to receive collateral at a premium and make a profit in this atomic transaction (Yazdanparast,

2021), which results in bad debts for the users who borrow money from attackers.

3.4.4. Abusive Transaction Exposure

Exchanges disclose all transactions as soon as feasible to ensure completely behavioral transparency because off-chain matching services are not automated. Unfortunately, exchanges can restrict access to select users and launch denial of service attacks (Baum et al., 2021) to dominate the market, audit transactions, and even front-run orders.

4. Analysis of Attack Events

In this section, we investigate real-world attacks in the DeFi ecosystem (CryptoSec, 2022; Bouteloup, 2022) and analyze the vulnerabilities exploited in the attacks with the classification shown in Figure 5.

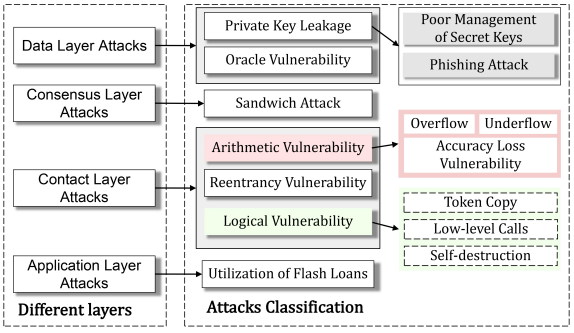

Figure 5: Classification of Attack Events at Each Layer

4.1. Date Layer Vulnerabilities

4.1.1. Private Key Leakage

The developer deploys DeFi applications on the blockchain through private keys managed in the wallet. Also, users confirm and initiate transactions on the DeFi app through the private key. We summarize real-world DeFi security events due to private key leaks in Table 5. We believe that there are two reasons for these security incidents:

(i) The keys owned by DeFi application developers lack a secure storage environment and well key management.
(ii) The developer triggered a phishing attack.

- **Poor Management of Secret Keys.** In the Meerkat Finance (Obelisk, 2021) incident, the administrator of the project used a private key and a false time lock in the contract. It transferred about $30 million worth of BNB tokens from the BNB Vault. In Listing 5 (Bscscan, 2021a), the administrator used the visual ambiguity of the number "0" and the letter "o" to make the variable slot values in the `_admin()` and `_setAdmin()` functions differently. This means that the time lock of BNB Vault is false, and the administrator can achieve the transfer of BNB tokens through this backdoor.

```
1  function _admin() internal view returns (address
       adm) {
2      bytes32 slot = ADMIN_SLOT;
3      assembly {
4        adm := sload(slot) }
5  }
6  function _setAdmin(address newAdmin) internal {
7      bytes32 slot = ADMIN_SL0T;
8      assembly {
9        sstore(slot, newAdmin) }
10 }
```

Listing 5: Meerkat Finance Instance

- **Phishing Attack.** The scripts embedded in the DeFi website interact with the wallet via API, which may give opportunities for phishing attacks (Winter et al., 2021). In the BadgerDAO (BadgerDAO, 2021) incident, the attackers stole the Badger developer's secret keys and injected malicious scripts into BadgerDAO's web pages. The scripts intercepted the user's transactions and prompted the user to allow the attacker to operate on the ERC-20 tokens in their wallets.

  The transparent nature of DeFi allowed the attacker to easily gather information about the developers. The attacker sent malicious emails to bZx developers, stealing the private management key of bZx deployed on the BSC and Polygon chains. The attackers used the management private key to upgrade the contract to mint unlimited tokens (bZx Contributor, 2021).

4.1.2. Oracle Vulnerability

The DeFi ecosystem relies heavily on oracle to provide off-chain or on-chain asset data, and cannot verify the accuracy of the data. This means that if the DeFi protocol uses only a single DEX as the source of asset prices, then the DeFi protocol will assume that it is true and accurate regardless of the movement of its asset price data.

In Table 5, oracle attacks have caused significant damage to DeFi applications. Most of the oracle attacks are based on the following steps (Wang et al., 2021c):

(i) **Preparation of Funds.** The attacker borrows a large number of assets unsecured through various Flash Loan providers, e.g., bZx, dYdX. He/She intends to inject the assets into other DeFi agreements to inflate their prices while hoarding the target assets.

(ii) **Raising the Price of Target Assets.** The attacker manipulates the oracle by balancing the target assets stored in the liquidity pool, i.e., by exchanging a large number of tokens back and forth between different liquidity pools. Since a single oracle is used, it passes the manipulated price data into the DeFi protocol.

(iii) **Profiting.** The attacker exchanges the target asset for money borrowed by the Flash Loan, a service provided by DeFi, e.g., collateralized borrowing. As the

Table 5
Summarization of Real-world Attacks Exploiting Different Types of Vulnerabilities

Vulnerabilities	Features	Victims	Date	Amount (million USD)
Private Key Leakage	The private keys of DeFi deployers are under threat due to poor private key management or phishing attacks. The key authorizes and verifies the transactions of the user. When an attacker utilizes the key, it is simple to tamper with the transaction, putting the trader's interests at risk. The attacker alters the website's Application Programming Interface (API) and embeds the vulnerability to get the user's personal information, including the user's key.	Meerkat Finance	March 2021	31
		Paid Network	March 2021	160
		Roll	March 2021	5
		EasyFi	April 2021	80
		bZx	November 2021	55
		8ight Finance	December 2021	1
		BitMart	December 2021	150
		AscendEX	December 2021	77
		Vulcan Forged	December 2021	140
		LCX	January 2022	6
		Ronin Bridge	March 2022	624
Oracle Attacks	The oracle price data feed can be manipulated by the attackers who change the asset data for the smart contracts. When an oracle is attacked, real-world data posted to the blockchain changes. It mismatches on-chain data with the real world, harming users.	bZx	February 2020	0.9
		Harvest Finance	October 2020	24
		Cheese Bank	November 2020	3
		PancakeBunny	July 2021	2
		Vee Finance	September 2021	35
		Vesper Finance	December 2021	1
Arithmetic Vulnerability	The attacker passes in specific parameters that cause the arithmetic operations in the contract to overflow.	Uranium Finance	April 2020	50
		Compound	September 2021	80
		Pizza DeFi	December 2021	5
		Umbrella Network	March 2022	0.7
Reentrancy Vulnerability	When a function calls an untrusted contract and that contract recursively calls the original function, it's reentrant.	dForce	April 2020	24
		Akropolis	November 2020	2
		Grim Finance	December 2021	30
Logical Vulnerability	The adversary employs unique methods to alter the contract program logic inadvertently and cause the loss of the DeFi application. It comprises possessing a token copy, low-level calls, and Self-destroying.	Furucombo	February 2020	14
		bZx	November 2020	8
		BurgerSwap	May 2021	7
		Eleven Finance	June 2021	4
		Punk Protocol	August 2021	3
		Starstream Finance	April 2022	4
Flash Loan	It allows users to borrow and settle loans in real-time in a single transaction without providing any collateral.	Warp Finance	December 2020	7
		Alpha Homora	February 2021	37
		Elephant Money	April 2022	11

attacker inflates the price of the target asset, it can exchange the target asset for a larger amount of other assets. By this step, the attacker will gain much profit.

(iv) **Loan Repayment.** The attacker restores the assets in the liquidity pool to their initial state to avoid losses caused by price slippage (Wang et al., 2021c), and repays the loan.

The bZx attack (PeckShield, 2020) happened in February 2020, and it was through the above attack steps that the attackers made a profit of about $0.9 million. The attacker borrowed lots of ETH through the bZx platform. At KyberSwap AMM, a portion of the ETH was exchanged for sUSD tokens to drive up the price of sUSD. Next, the attacker bought the sUSD from the Synthetic Depot contract at the normal price. The attacker pledged the sUSD in the account into the bZx protocol in exchange for ETH. As the price of sUSD in bZx was inflated, it could be exchanged for more ETH. Finally, the attacker repaid the loan.

In 2021, Vee Finance lost 35 million USD due to the oracle vulnerability. It had only one oracle as a price input source. At the same time, the attackers profited by using errors in the contract to bypass the slippage protection checks. Similarly, the Harvest protocol used the USDT price in Curve as the price data. Since the USDT price became lower at this point, the attacker could pledge more USDT with the same assets. The attacker performed 32 attacks and profited 24 million USD from the protocol (Werner et al., 2021).

4.2. Consensus Layer Vulnerabilities

4.2.1. Sandwich Attack

Currently, the blockchain is based on the consensus of Proof of Work, which gives bookkeeping rights to the node

that calculates the required hash value. Nodes with bookkeeping rights can specify the order of transactions according to their own rules. The dependency on transaction order vulnerability is one of the factors affecting the security of smart contracts, and it also applies to DeFi applications. For example, the sandwich attack is now widely studied.

The Sandwich attack applies to AMMs like Uniswap and takes advantage of a special feature of AMMs, such as the fact that for every token swap that occurs on an AMM like Uniswap, the price of its swapped tokens changes. The steps of the Sandwich attack are as follows:

(i) **Network Spy.** There are some spy nodes deployed on the network to collect all the transactions for asset exchanges. If attackers consider that a transaction that exchanges token A for token B is profitable, they will create two transactions for racing to control the transaction and make a profit. It means that the price of token B in the liquidity pool will be increased.

(ii) **Transaction Creation.** The attacker creates a front-running transaction to exchange token A for token B, and the price of token B in the liquidity pool will be raised. Suppose the price of token B rises too much. In that case, the slippage detection may be triggered, and the attack will be failed, so the attacker will generally control the number of tokens purchased. The victim is also exchanging token A for token B, which causes the price of token B to continue to rise. As the attacker's front-running trading causes the price of token B to rise, the victim can only obtain less than the expected amount of token B. Finally, after the victim's transaction, the attackers would create a back-running transaction that converts token B into token A, thus making a profit.

According to our research, sandwich attacks often occur on AMMs, such as Uniswap, Linch, and SushiSwap. About 30,000 Sandwich attacks have occurred on Ethereum, which allowed the attackers to generate a profit of 2 million USD (EigenPhi, 2022).

4.3. Contract Layer Vulnerabilities

Smart contracts are the basis for implementing decentralized financial instruments. When DeFi applications were deployed on the blockchain, some errors in the smart contract might cause irreparable damage to DeFi (Torres et al., 2018).

4.3.1. Arithmetic Vulnerability

Almost all DeFi applications involve arithmetic operations on currencies. These operations consist of adding or subtracting from account balances and converting exchange rates between different tokens (Werner et al., 2021). There have been overflow and precision loss vulnerabilities in the DeFi ecosystem. These arithmetic vulnerabilities have caused significant damage to DeFi applications.

- **Overflow.** In April 2018, there were multiple DeFi applications, e.g., OKEx, that suffered huge losses and were forced to shut down due to an overflow vulnerability in the ERC-20 token contract, e.g., MESH and UGToken. There is a commonality in this overflow event, such as the problem caused by the `transferProxy()` function in the contract in Listing 6 (Etherscan, 2018).

  The overflow vulnerability appears in line 2 of Listing 6 (Etherscan, 2018). Since `_fee` and `_value` are both input parameters, they can be controlled artificially. Then an attacker can design the incoming parameters so that their size exceeds the storage range of `uint` type and an overflow occurs. When an overflow occurs, it causes the unsigned integer to flip to 0 as a whole. This means that the attacker can bypass the check of the `if` statement in the second line and make it transfer the tokens to an address with no balance (Billy, 2018).

```
1 function transferProxy(address _from, address
      _to, uint256 _value, uint256 _fee,uint8 _v,
      bytes32 _r, bytes32 _s) public
      transferAllowed(_from) returns (bool){
2   if(balances[_from] < _fee + _value) revert();
3   ...
4   Transfer(_from, _to, _value);
5   balances[msg.sender] += _fee;
6   Transfer(_from, msg.sender, _fee);
7   ...
8   return true;
9 }
```

Listing 6: Snippets of MESH Token

```
1 function _setCompSpeed(CToken cToken, uint
      compSpeed) public {
2   ...
3   setCompSpeedInternal(cToken, compSpeed);
4 }
5   ...
6   if (supplierIndex.mantissa == 0 && supplyIndex
      .mantissa > 0) {
7     ... }
8   Double memory deltaIndex = sub_(supplyIndex,
      supplierIndex);
9   ...
```

Listing 7: Snippets of COMPOUND Contract

- **Underflow.** The larger loss in arithmetic vulnerability is Compound Finance. Its reward payouts `CompSpeed` could be set to 0, indicating the suspension of reward payouts, and the market award index `supplyIndex` was 0. For new users, their award index `supplierIndex` was initialized to `CompInitialIndex` presented by Compound as 10^{36}. An underflow vulnerability occurred in Listing 7 (Flatow et al., 2021) at line 8. This caused the formula for calculating the difference in the reward index `deltaIndex = sub_( supplierIndex = 0, supplierIndex=`10^{36}`)` to underflow and became a very

large value, while the Compound Finance reward calculation relied on the value of `deltaIndex`.

There was no attacker in this security incident, but rather an overpayment of rewards due to an underflow vulnerability in the contract. This incident caused the Compound 80 million USD in damages. In 2022, Umbrella NetWork also lost 0.7 million USD due to an underflow vulnerability.

- **Accuracy Loss Vulnerability.** The Uranium Finance contract allowed users to borrow money using Flash Loan. However, the contract suffered from accuracy handling errors when calculating the amount to be returned, resulting in a calculated amount that was 100 times larger than the actual amount (SlowMist, 2021). It means that the attacker only needs to return a small portion of the loan to pass the check of the `require` statement in Listing 8 (Bscscan, 2021b) and pays off the loan.

```
1 uint balance0Adjusted = balance0.mul(10000).sub(
      amount0In.mul(16));
2 uint balance1Adjusted = balance1.mul(10000).sub(
      amount1In.mul(16));
3 require(balance0Adjusted.mul(balance1Adjusted)
      >=uint(_reserve0).mul(_reserve1).mul
      (1000**2), 'UraniumSwap: K');
```

Listing 8: Snippets of UraniumPair Contract

4.3.2. *Reentrancy Vulnerability*

A contract executing a transaction invokes a malicious contract account, and the malicious contract account invokes a function in the contract before the contract state changes (Rodler et al., 2019). The most significant reentrancy attack in Ethereum was the DAO attack (Buterin, 2016) that caused a hard fork of Ethereum. Reentrancy attacks were applied to the DeFi protocol with its development. In Table 5, 54 million USD was lost to DeFi due to a reentrancy vulnerability.

In April 2020, the dForce protocol suffered a reentrancy attack with a loss of about 24 million USD. The attackers exploited the ERC-777 (Richards et al., 2022b) compliant imBTC tokens. Compared to the ERC-20 token standard, the ERC-777 token standard has one feature. When ERC-777 tokens were sent or received, they would go through Hook in the form of a callback to notify the sender or recipient. The attacker in the incident took advantage of this feature and re-entered the dForce contract to increase the amount of imBTC collateral and get a higher yield (Werner et al., 2021).

Grim Finance on the Fantom (Cronje et al., 2022) chain lost 30 million USD due to a re-entry vulnerability. First, the attacker created a contract to inject the cryptocurrency borrowed from the service of Flash Loan into Spirit Swap (SpiritSwap, 2022) to obtain Spirit-LP certificates. Next, the Spirit-LP certificates were pledged to the GrimBoostVault contract in exchange for the GB-BTC-FTM, which was a token, via the `depositFor()` function in Listing 9 (FTMScan, 2021). Since the legitimacy of the token contract was not verified, the attacker re-called the `depositFor()` function in the `safeTransferFrom()` function of the malicious contract, implementing reentrancy attack to collateralize more tokens for profit. Finally, the attacker returned the borrowed funds.

```
1  function depositFor(address token, uint _amount,
       address user) public {
2    uint256 _pool = balance();
3    IERC20(token).safeTransferFrom(msg.sender,
       address(this), _amount);
4    earn();
5    uint256 _after = balance();
6    _amount = _after.sub(_pool);
7    uint256 shares = 0;
8    if (totalSupply() == 0) {
9      shares = _amount;}
10   else{ shares = (_amount.mul(totalSupply())).
       div(_pool); }
11   _mint(user, shares); }
12 }
```

Listing 9: Snippets of GrimBoostVault Contract

4.3.3. *Logical Vulnerability*

According to our investigation, a large number of vulnerabilities in the DeFi application stem from simple programming errors in the smart contract (Werner et al., 2021). Due to the tamper-evident nature of the blockchain, these errors can cause significant damage to the DeFi application.

- **Token Copy.** This was the third attack on bZx in 2020. The attackers exploited a vulnerability in the contract by passing the same address to the sender parameter `_balancesFrom` and the receiver parameter `_balancesTo` in the bZx contract, thus copying the balance in the account (Kistner, 2021).

- **Low-level Calls.** Starstream Finance is a DeFi project on the Metis Andromeda network. As seen in Listing 10 (Baranov, 2022), the vulnerability was due to the public function `execute()` of the DistributorTreasury contract using an unchecked external call `to.call()`, allowing anyone to make an external call. It meant that an attacker could use the function to generate a call to the `withdrawTokens()` function to extract the STAR Token in the StarstreamTreasury contract.

```
1 function execute(address to,uint256 value,bytes
      calldata data) external returns (bool,
      bytes memory) {
2   (bool success, bytes memory result) = to.call{
      value: value}(data);
3    return (success, result);
4 }
```

Listing 10: Snippets of DistributorTreasury Contract

- **Self-destruction.** Self-destruction of contracts and destruction of tokens in contracts are both common operations in the DeFi ecosystem. Usually, attackers will transfer stolen valuable cryptocurrency into the contract under their control. To avoid being traced, the attackers will destroy the attack contract after transferring the tokens in their contracts.

  The Eleven Finance attack (REKT, 2021) was caused by the fact that the attacker could not destroy the proof of assets when withdrawing them from the contract, thus enabling the withdrawal of the deposit twice. The specific reason for this attack was that the function `emergencyBurn()` in the ElevenNeverSellVault insurance contract allowed the attacker to withdraw the deposited assets without destroying their proofs. The attacker then called the `withdraw()` function in Listing 11 to perform the normal process of withdrawing the assets (Eleven, 2021). This incident caused a loss of approximately 4 million USD to Eleven Finance.

```
1  function withdraw(uint256 _shares) public {
2    ...
3    if(avai<_shares) IMasterMind(mastermind).
       withdraw(nrvPid, (_shares.sub(avai)));
4      token.safeTransfer(msg.sender, _shares);
5      ...}
6  function emergencyBurn() public {
7    ...
8    if(avai<balan)
9      IMasterMind(mastermind).withdraw(nrvPid, (
       balan.sub(avai)));
10   token.safeTransfer(msg.sender, balan);
11   ...}
```

Listing 11: Snippets of ElevenNeverSellVault Contract

4.4. Application Layer Vulnerabilities

4.4.1. Utilization of Flash Loans

Flash loan is a type of unsecured lending that relies on the atomicity of blockchain transactions at the point of execution (Qin et al., 2021c) and adds dynamism to DeFi. Unfortunately, attackers can use Flash Loan to reduce the cost of the attack. According to our survey, most DeFi attacks involved Flash Loan services.

At the same time, Flash Loan provides asset protection for arbitrageurs to realize price manipulation of traditional finance on DeFi. The arbitrageur uses part of the borrowed assets to raise or lower the price of the assets in the AMM liquidity pool, that is, the asset exchange ratio. The arbitrageurs then use the remaining loan to make another trade for profit (Wu et al., 2021). Finally, they repay the loan. The Elephant Money attack event that occurred in April 2022 was based on a traditional price manipulation attack, which resulted in a loss of 11 million USD. The attackers minted the TRUNK stablecoin, increasing the price of the ELEPHANT token (Greig, 2022). They finished by making a profit by buying more WBNB and BUSD tokens with ELEPHANT and TRUNK.

5. Analysis of Security Optimization

Even though numerous attacks exploit various vulnerabilities, many research efforts have succeeded in detecting and defending against these attacks, which fuels the rapid development of blockchain. We analyze the vulnerabilities in the Section 3 according to the hierarchy shown in Figure 6.

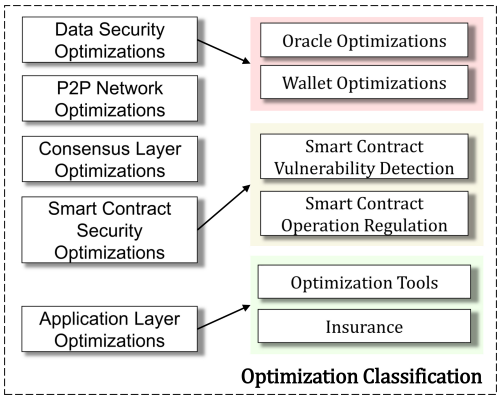

Figure 6: Classification of Optimization at Each Layer

5.1. Data Security Optimizations

As the oracle mechanism and key management flaws are mentioned in 3.1, optimizing the data layer can effectively prevent attacks from exploiting the flaws to tamper with the authenticity of the data.

5.1.1. Oracle Optimization Schemes

Due to the necessity for off-chain asset information such as pricing, there is an expanding demand for superior oracles (Kaleem and Shi, 2021). There have been decentralized and centralized related studies for this event, as summarized in Table 6, where the 'C' means centralized, 'D' means decentralized, 'On' represents the method is on-chain and 'Off' is off-chain.

Town Crier (TC) (Zhang et al., 2016) employs SGX technology, a Trusted Execution Environment (TEE) offered by Intel, to address the issue of safe communication between blockchain smart contract $SC = (SC_1, SC_2)$ and the network layer. We assume that TC has one extra TCP layer than the HTTPS network protocol for providing dependable data sources. When SC_1 launches a transaction to SC_2, the program *prog* created in preparation in TC receives the transaction's datagram request, then obtains the data source through the external network's HTTPS protocol, and finally delivers the request with a digital signature to the requester SC_1. It separates the hostile network operation and host process.

Uniswap (Adams et al., 2021) automatically reconciles the required cryptocurrency pricing information by using the Automated Market Maker (AMM), using an on-chain smart contract to set up the symbolic equation. We assume a centralized asset container *pool* with a large number of two cryptocurrencies a and b, which correspond to stocks of (a) and

Table 6
Summarization of Different Oracle Optimization Schemes.

Reference	Key Technologies	Features	Categories				
			C-On	C-Off	D-On	D-Off	Others
Zhang et al. (2016)	TEE	Operations are handled in TEE to invalidate malicious requests at cost of throughput	✔	×	×	×	×
Breidenbach et al. (2022)	Node Network	The median value of multi-parties stored in the node network to prevent tampering	×	×	✔	×	×
Ritzdorf et al. (2017)	Signature Verification	Transport protocol needs to be modified before using to validate server information	×	✔	×	×	×
Adler et al. (2018)	Incentivize Voting	Nash equilibrium can be satisfied in rational participants but does not guarantee authenticity	×	×	✔	×	×
Bin et al. (2022)	Sidechain Extension	It increases the scalability and throughput as a chain, but it can also be attacked	×	×	×	×	✔
Kenton (2020)	Committee Voting	Ability to handle lots of request transactions, but they cannot be processed immediately	×	×	×	✔	×
Zhang et al. (2020)	Zero-Knowledge Proof	The price request can be proved to be correct and does not need to modify the server	×	×	×	✔	×
Adams et al. (2021)	Symbolic Equation	Trading frequently by AMM, but rivals can profit from economic attacks	✔	×	×	×	×
Wang et al. (2021c)	Track Comparison	The modified EVM runs historical data to detect if the oracle is vulnerable to attacks	×	×	×	×	✔
Nick et al. (2022)	TEE	Several trusted computations used to prove the data and transport it to the smart contract	×	✔	×	×	×

(b). Uniswap allows $(a) \times (b) = k$, where k is a constant. When a portion of a is borrowed, (b) grows due to the drop of (a).

TLS-N (Ritzdorf et al., 2017) is an extension of Transport Layer Security (TLS). TLS generates an encrypted network channel for data exchange to ensure that the adversary does not access the data of the conversation between the two parties. TLS-N is proposed to solve the requirement of the third trusted party. It starts evidence generation and collection after the traditional handshake, which means that the evidence generator signs the handshake state with the private key immediately after the handshake. And TLS-N records all the handshake operations. All handshake records and proof signatures are utilized to assure non-repudiation of the conversation.

Provable (Nick et al., 2022) is similar to Chainlink (Breidenbach et al., 2022) in that it sends currency information to Ether via operational nodes. The distinction is that the former uses trusted computing to ensure the accuracy of information across the network and transport layers, while the latter operates outside the chain. The latter uses median value computations to store multi-party data in the decentralized oracle network, ensuring safe data exchange. And it is worth noting that Chainlink uses a reputation system to stimulate each node.

Astraea (Adler et al., 2018) provides a voting oracle approach using the game analysis. It selects the submitters of the funding allocation scheme, the voters who vote on the funding, and the validators who verify the correctness of the scheme. The advantage of this strategy is that it ensures the accuracy but not the legitimacy of the data for reasonable participants.

Meter (Bin et al., 2022) is a sidechain built in parallel to the blockchain using PoW consensus, increasing the main chain's throughput and scalability. The cryptocurrency MTR on Meter is an autonomous and distributed coin passed to other blockchains, such as Ethereum, through sidechain technology, making the price of the cryptocurrency stable. It improves not only the performance but also expansion.

MakerDAO (Kenton, 2020) collects a variety of asset price information from several oracle sources, then the committee votes to select one set of trusted feeds and pass them to the smart contract. However, to prevent the attacked information from being uploaded to the chain, the oracle security mechanism delays the input of the prices for one hour after being obtained. Finally, the committee voted to set up an emergency oracle to freeze a malicious oracle to prevent the oracle crash.

DECO (Zhang et al., 2020) is similar to TLS-N, they are both interested in evidence generation. TLS-N needs advanced modifications to the server protocol to generate a proof. In contrast, DECO generates a proof in this Section using the zero-knowledge proof, which means there is no a revelation of encryption keys. P and V are given a shared key by the prover P, the verifier V, and the TLS server S. P initiates a query request using the shared key, S responds and transmits data to P. V detects the request, respond, and then announces its shared key. Finally, P proves the returned data.

ProMutator (Wang et al., 2021c) detects whether an oracle is vulnerable by analyzing normal and abnormal transactions against the price oracle. The transactions of price

Table 7
Summarization of Wallet Security Optimization Schemes.

Reference	Key Scheme
Dabrowski et al. (2021)	Improving collaborative key generation and signature
Khan et al. (2019)	Combination with a cold and hot wallet for privacy
He et al. (2018)	A practical way to public key encryption
He et al. (2019b)	A p2p wallet scheme with a routing protocol
Dai et al. (2018)	A lightweight wallet based on TEE
Rezaeighaleh and Zou (2019)	A new scheme for creating sub-wallet keys
Jian et al. (2021)	The shared key generated by T-ECDSA for signing
Han et al. (2021)	Combine signatures into one based on the bloom filter

oracle attacks are executed in the modified EVM according to predefined rules. And then the comparison of the original and mutating traces generates reports, which has analyzed the differences.

5.1.2. Wallet Optimization Schemes

Users initiate a transaction and sign it using the key pair, the assets in the account are risky when the key leaks to an adversary. In Table 7, some studies (Dabrowski et al., 2021; Khan et al., 2019; He et al., 2018, 2019b; Dai et al., 2018; Rezaeighaleh and Zou, 2019; Jian et al., 2021; Han et al., 2021) proposed specific solutions for wallet management and wallet architecture.

According to (Dabrowski et al., 2021), existing hardware wallets migrated from the PC wallet architecture, resulting in a bad design that does not fundamentally fix the problem when just utilizing authentication and communication encryption. Interactive authentication adds several signatures and keys to the original wallet structure, which prevents attackers from manipulating the keys for transactions using a malfunctioning wallet.

Combined with software and hardware, two android applications created in (Khan et al., 2019) provide privacy protection. It contains a cold wallet with key storage in the form of QR codes and a hot wallet for sending transactions, respectively.

He et al. (2018) presented a new practical way of public key cryptography deployment. It formalizes the user's interactions with the management server m, the central server c, and the proxy p. This method provides five protocols, the first of which initializes all parameters in preparation for secure channels and verification operations, such as producing keys and verifying digital signatures. The second process is registration, in which u is bound to the m and produces valid login credentials. The registration data is then sent to c. The third is a backup, and the fourth is a verification that is utilized to perform a transaction in p. The final service provided by it is wallet recovery with the help of registration and backup.

He et al. (2019b) used a routing protocol to convert the C/S architecture into a Peer-to-Peer (P2P) structured wallet management scheme. It solves the multi-constraint disorder problem and distributes the data through a P2P network. In addition, it proposes a new key sharing strategy called SKN to improve the availability of keys. Users interact with each other through a streaming network, which is a fully connected network, to ensure that the same key is not stored in the same node.

Dai et al. (2018) designed the SBLWT, a lightweight wallet architecture which is based on TEE. The insecure storage module in the normal environment only has read access to the encrypted block header in the trusted environment, while encrypted messages can be proactively stored in the insecure space, thus ensuring security between message exchanges.

Unlike traditional forwarding wallets, Rezaeighaleh and Zou (2019) shared the keys of the main wallet and sub-wallets to enable the generation of various sub-wallet addresses in a transaction. And the address generation process in the sub-wallets is designed to secure its keys. Most importantly, there is unnecessary to back up the sub-wallets because the master-sub structure can derive the sub-wallets from the main wallet.

To defend against a single point of failure, Jian et al. (2021) proposed a Threshold-based ECDSA scheme, also called T-ECDSA. When the number of participants is within the threshold, they created shared private keys to sign transactions. Participants outside the threshold take turns signing transactions. There is also another program (Han et al., 2021) that uses T-ECDSA to design wallets. It combines multiple signatures into one signature. However, the design of the bloom filter protects the information of the participants on a small scale.

5.2. P2P Network Optimizations

The transactions initiated by each node in Ethereum are transmitted through P2P networks to achieve self-governance without a third party; however, the lack of authentication and other features leads to a series of attacks, such as the eclipse attack(Wüst and Gervais, 2016; Marcus et al., 2018; Xu et al., 2020; Henningsen et al., 2019) in Table 8. An information eclipse attack occurs when an aggressor removes nodes from a network to restrict access to information from nodes.

Wüst and Gervais (2016) proposed a novel eclipse bug, when the block height is n, a malicious node can obtain the $(n+1)_{th}$ block by preventing a regularly functioning node N from receiving it. It invalidates subsequent blocks, even if node N may receive them. However, it offers some countermeasures. If there is a block request that is not corresponding, the block is requested from multiple peers instead of just one peer. Through multi-party collaborative governance, this approach can solve denial of service attacks

Table 8
Summarization of Network Security Optimization Schemes.

Reference	Key Scheme
Wüst and Gervais (2016)	Sending requests to multiple peers
Marcus et al. (2018)	Ensure that node ids always exist in the query table
Xu et al. (2020)	Analyze packets using a random forest model
Henningsen et al. (2019)	Increasing the default number of peers

caused by eclipse attacks.

However, Marcus et al. (2018) suggested a series of protection methods against eclipse attacks on Ethereum, two of which are also adopted by Geth. When a node restarts, the client's `seeding` is triggered every hour, or `lookup()` is called on an empty `table` which stores the information in `memory`, but the `seeding` is available only if the `table` is empty. However, node IDs should always be inserted into the `table` to prevent attacks. Specifically, Geth runs a `lookup()` on three random targets during `seeding` to add more legitimate nodes from the `db` which stores the information on disk to the `table` to prevent attackers from inserting their node IDs into an empty `table` during `seeding`.

Xu et al. (2020) provided the ETH-EDS model for analyzing packets and identifying three features in packets for the random forest model, which is a classification model based on decision trees. It detects malicious nodes that isolate users, allowing users to defend their networks in real-time.

Henningsen et al. (2019) proposed an optimization solution for the false friend eclipse attack that exists in Geth. The attack can be deployed with just 2 IPs and loaded on the node immediately. The peer is the number of network nodes that can be connected, default by 25. It is similar to Wüst and Gervais (2016) in that it increases the peers to 50 to increase the probability of survival of spare nodes.

5.3. Consensus Layer Optimizations

The consensus layer and the incentive layer are interdependent. The design of the consensus mechanism directly affects the behavior of miners. Although many consensus mechanisms have been proposed, such as PoW and PoS, there is little regulation of the consensus or incentive levels. Table 9 summarizes the following optimized solutions.

Yang et al. (2021) used the tool called Fluffy to discover two vulnerabilities, "Shallow copy" and "Ether Shift" in Table 3. It first picks test cases and then changes the transactions in those cases. Next, it puts these transactions into multiple EVMs and collects all the state and coverage at the end of the execution. Finally, it repeats the above steps until the state no longer changes.

As described in 3.2, fork attacks might affect the security of blockchain in terms of consensus mechanism, Zhou et al. (2021a) developed DeFiPoser to monitor fork behaviors. Figure 7 shows the process of DeFiPoser, it prunes the

Table 9
Summarization of Consensus Security Optimization Schemes.

Reference	Key Scheme
Yang et al. (2021)	Execute variant transactions with several EVMs
Zhou et al. (2021a)	Build the graph and analyze it to find fork attacks
Swende et al. (2017)	Randomly select an EVM version to test transactions
Lattner et al. (2017)	Fuzz engine for parallel extraction of functions
Fu et al. (2019)	Test EVMs with contracts and analyze results

patches after building the DeFi graph based on the heuristic approach (Dwivedi, 2020) and then does a greedy search of the negative cycle in the directed transaction flow graph, which means finding all possible profitable cycles in the trade flow graph, to detect arbitrage transactions in cyclic or more complicated scenarios. A binary search of all the paths finds the most profitable one. If it is within the quantization threshold quantified by the Markov decision process, there is a chance to motivate a fork attack by miners using MEV.

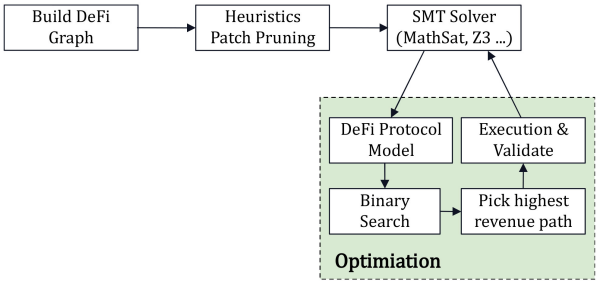

Figure 7: Diagram of DeFiPoser core process.

Ethereum exists an EVMLab (Swende et al., 2017) library for interacting with Ethereum Virtual Machines, also called EVMs, which are officially used to analyze the bytecode of smart contracts. In this way, one version of the EVM is randomly selected for a single transaction on a smart contract, and then finds bugs.

Another library called LibFuzzer (Lattner et al., 2017), was developed based on a tool chain LLVM written in C++. It generates N concurrent processes for functions in the contract and randomly assigns subsets to them, where one of the subsets will merge its generated corpus into the main set in the end. These corpora are used to find bugs using fuzz.

EVMFuzzer (Fu et al., 2019) is a tool for testing and evaluating EVM. It takes the target EVM and its API as input and then creates an execution environment for them to test and evaluate the EVM. In this process, multiple EVMs receive some of the same quality contracts that have been selected and output the results in the same format. It discovered a DOS attack on Geth, which was recorded as CVE-2018-19184.

Table 10
Summarization of Methods for Smart Contract Vulnerabilities Detection.

Reference	Key Technologies	Target Vulnerability	Features
Luu et al. (2016)	Symbolic Execution	Transaction State Dependency Block Info Dependency Unhandled Exception Reentrancy	Control Flow Graph (CFG) Construction Symbolic Execution Core Analysis
Chen et al. (2018)	Machine Learning Static Analysis	Ponzi Scheme	Obtain account features from transactions Obtain code features from OPCode XGBoost
Amani et al. (2018)	Formal Verification	Logical Vulnerabilities	Formal EVM extensions with Isabelle/HOL Logical verification at bytecode level
He et al. (2019a)	Machine Learning Fuzz	Greedy Contract Ether Leaking Suicidal Contract Block Info Dependency Unhandled Exception Controlled Delegatecall	Symbolic Execution GRU Fully Connected Network (FCN) Fuzz
Gao et al. (2019)	Machine Learning	Overflow Block Info Dependency Reentrancy Greedy Contract Bad Randomness	Clone Detection Code Embedding Similarity Checking
Xue et al. (2020)	Static Analysis Path Protection Technology (PPT)	Reentrancy	Cross-contract CFG Static Taint + PPT
So et al. (2020)	Static Analysis	Arithmetic Bug Assertion Violation ERC-20 Standard Violation	Insert assertions generate many queries and invariants, queries are validated with Solver
So et al. (2021)	Machine Learning Symbolic Execution	Ether Leaking Suicidal Contract	Hunt sequences with symbol execution Train language model with sequences Guide symbolic execution with the model
Huang et al. (2021)	Machine Learning	Overflow Reentrancy Unexpected Permission Check Bad Randomness	CFG Construction Slicing Graph Embedding Similarity Checking
Chen et al. (2021)	Static Analysis	Transaction State Dependency DoS Under External Influence Strict Balance Equality Reentrancy Nested Call Greedy Contract Unchecked External Calls Block Info Dependency	CFG Construction Feature Detection Core analysis
Choi et al. (2021)	Static Analysis Fuzz	Assertion Violation Unexpected Permission Check Block Info Dependency Ether Leaking Greedy Contract Arithmetic Bug Unchecked External Call DoS Under External Influence Reentrancy Suicidal Contract	Collection of bytecode data streams Select semantic information as seeds Fuzz based on data stream feedback
Nam and Kil (2022)	Formal Verification Game Theory	Interactive Contract Vulnerabilities	Convert source files to MCMAS files Construct contract game structure Examine ATL properties in the structure
Jin et al. (2022)	LLVM IR Symbolic Execution	Arithmetic Bug Reentrancy Suicidal Contract Unchecked External Calls	Convert the source code to LLVM IR Locating Vulnerabilities Collect ordered sets of transactions Symbolic Execution Verification

5.4. Smart Contract Security Optimizations

The smart contract, which is a part of the DeFi project connecting the data and the application layer, might alter the state of a transaction, and cause errors. Therefore, it is critical to improve the security of contracts.

5.4.1. Smart Contract Vulnerability Detection

In this Section, we will introduce the methods in the Table 10 which were proposed for detecting the vulnerabilities in the smart contract. Much research(Chen et al., 2021; Luu et al., 2016; Amani et al., 2018; He et al., 2019a; Gao et al., 2019; Xue et al., 2020; So et al., 2020, 2021; Huang et al.,

2021; Choi et al., 2021; Nam and Kil, 2022; Jin et al., 2022) has been undertaken to discover contract vulnerabilities using various technological tools, such as formal verification, symbolic execution, and machine learning.

Combined with dynamic testing extends the ability of symbolic execution techniques to detect unknown vulnerabilities, thus improving the robustness of programs. Figure 8 shows an overview of ILF (He et al., 2019a) that combines fuzzing, machine learning, and symbolic execution. The system used the symbolic execution for a portion of the contracts to generate transaction sequences as the training dataset for a new model consisting of GRU, which is a type of neural network and a fully connected network so that the model can learn the fuzzing in the state after the symbolic execution to test contracts with high coverage.

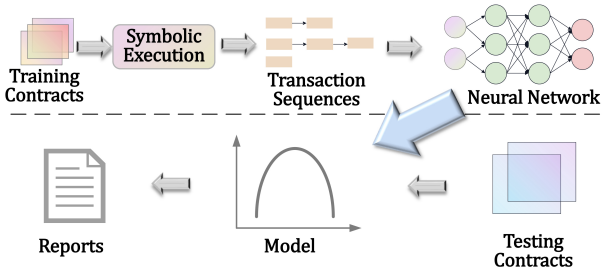

Figure 8: Schematic Diagram of ILF Process Framework.

As our best known, Oyente (Luu et al., 2016) is the first detection tool using symbolic execution for smart contracts. It examines the logic of the contract code and generates control flow graphs. It then instructs the Ethereum initial state simulation run to construct feasible data flow operations. After that, the appropriate analysis methods discover various vulnerabilities.

Chen et al. (2018) counted the frequency of opcodes in the contract and analyzed the features of malicious accounts through the ether flow graph. Then, for vulnerability identification, the XGBoost model (Chen and Guestrin, 2016), which is a gradient model based on the decision tree, was built using the features. This strategy can be utilized before contract deployment because it does not need attributions about the transaction.

Amani et al. (2018) formalized the EVM in terms of bytecode using Isabelle/HOL and built a double verifier logic on reasoning about the program. They then demonstrated the safety of the system. However, owing to the development of Ethereum, it cannot describe the complete semantics of the smart contract in several versions.

DefectChecker (Chen et al., 2021) uses the commands provided by Geth to disassemble the contract bytecode into opcodes and then split the opcodes into multiple base blocks, executing different instructions for each block and defining eight types of rules to detect vulnerabilities. It analyzes specific features of the vulnerability, and then different generic specifications are set for detecting the vulnerability based on Chen et al. (2020a).

SmartEmbed (Gao et al., 2019) consists of two main aspects, structured code embedding that converts code into word vectors and similarity checking that can detect the similarity of different vectors. It marks Solidity codes, and a new code embedding is generated using word embedding techniques, and finally, bugs can be found within a threshold by comparing the similarity between vectors.

Clairvoyance (Xue et al., 2020) designed several path protection techniques for reentrancy vulnerability and used taint analysis techniques to reduce the phenomenon of false positives from other tools. More importantly, this lightweight approach allows analysis of cross-contract behavior.

Up to May 2022, the compiler of Solidity has over 90 versions, each with significant updates (Beregszaszi et al., 2022). As a result, the compiled bytecodes with the same logic are diverse and noisy. To solve this problem, Huang et al. (2021) labelled the data and reordered the opcodes. This process ignores all irrelevant instructions, then analyzes the bytecode execution process and slices the data by the label to reduce the noise impact of meaningless code. Subsequently, this method uses an unsupervised graph embedding algorithm to deal with the smart contracts, each slice of code encoded as a vector and their similarity compared for vulnerability detection.

Smartian (Choi et al., 2021) just started statically analyzing the contract bytecode and collecting the data stream. The seed pool initialization predicts the sequence of transactions in some data stream and considers them as seeds for initialization. Finally, the seeds, which are the sequences of generated transactions, are used to guide the fuzz of the data flows.

Another methodology similar to ILF uses high-quality transaction sequences to direct symbolic execution in order to discover susceptible contracts. SmarTest (So et al., 2021) based on VeriSmart (So et al., 2020), first performs symbolic executions on the contracts in the training dataset, each lasting long enough to gather all the fragile transaction sequences, and then utilizes these fragile transaction sequences as the collection of training sequences for the language model. The goal of training the language model is to create a training corpus Y from which counts of n tuples are gathered to guide the symbolic execution to find fragile transaction sequences.

Nam and Kil (2022) examined smart contracts using the Alternating-time Temporal Logic (ATL) model, a formal verification approach that determines if Ethereum smart contracts meet certain features. Additionally, it converts Solidity to MCMAS, an ATL checker that requires input from the user. It enables developers to validate the attributes they would like to examine.

EXGEN (Jin et al., 2022) transforms the contract source code or bytecode to an Abstract Syntax Tree (AST), which is subsequently converted into an LLVM representation of the unified form. Afterward, all restrictions are eliminated utilizing symbolic execution techniques. The system then employs a solver to resolve the constraints and establish the order of vulnerabilities. After that, the sequence is reviewed for reliability before being added to the blockchain.

Table 11
Summarization of Methods for Smart Contract Vulnerability Regulation.

Reference	Target Vulnerability	Key Technologies	Features
Rodler et al. (2019)	Reentrancy	Dynamic Taint Technique	It traces the `storage` Variable and writes them into the lock during the transaction, when an attack is detected, the transaction would be rolled backed
Ferreira Torres et al. (2020)	Reentrancy Unexpected Permission Check	Domain-Specific Language (DSL) Vulnerability Patterns Voting Mechanism	ÆGIS modified the EVM to revert the codes written by DSL, which described the vulnerability patterns. Therefore, it can prohibit malicious control flow and data flow by comparing the patterns
Cao et al. (2020)	Reentrancy Block Info Dependency Strict Balance Equality Unchecked External Calls Unexpected Permission Check Missing Return Greedy Contracts	Online Framework	SODA is an online framework that can be divided into two layers. The lower layer collects EVM information, and the higher layer provides interfaces for developing detection apps
Rodler et al. (2021)	Reentrancy Arithmetic Bug Unexpected Permission Check	Patch	It first makes the vulnerability detection utilizing other detection tools, and then to fix the contract, it rewrites bytecode. Finally, EVMPatch tests the fixed contract with the historical transactions to verify whether the patch is correct
Ferreira Torres et al. (2021)	Reentrancy Arithmetic Bug Unchecked External Calls Unexpected Permission Check Unmatched ERC-20 Standard	Mapping Knowledge Domain Dynamic Taint Analysis	It extracts information from tracked transactions, and builds a graph with the nodes and transactions. Horus identified attacks with the graph and the queries, and finally loaded the tracing assets into a graph database
Nguyen et al. (2021)	Reentrancy Arithmetic Bug Transaction State Dependency	Automatic Contract Transformation	Several runtimes in SGUARD are used to identify control dependency, and it uses symbolic trace generations for each loop to get more data dependencies. When it finds and fixes bugs based on their definitions

5.4.2. Smart Contract Operation Regulation

Smart contracts can be more secure by detecting vulnerabilities, according to Torres et al. (Ferreira Torres et al., 2021). However, the number of assaults has not decreased, which indicates that contract regulation needs to be improved. It has been studied in (Rodler et al., 2019; Ferreira Torres et al., 2020; Cao et al., 2020; Rodler et al., 2021; Ferreira Torres et al., 2021; Nguyen et al., 2021) in Table 11, and we briefly introduce Sereum (Rodler et al., 2019) in Figure 9, a tool focused on runtime monitoring and verification of the reentrancy bug.

The transaction manager converts all control flows into conditional jump instructions in the bytecode interpreter. The taint engine identifies data flows in conditional jump instructions, tagging storage variables as the key variables and writing into the lock. The attack detector detects the variables. If the modification occurs, the whole transaction rolls back to the point where the variable was marked, which is the starting point of the entire transaction.

Sereum defends online smart contracts from reentrancy attacks, but such systems are difficult to expand to additional weaknesses. EVMpatch (Rodler et al., 2021) intended to address this issue by providing a bytecode rewriting engine that updates contracts currently on the chain, and the patch program is readily scaleable to other flaws. Automated analysis tools and vulnerability revelations detect and generate reports on vulnerabilities, and bytecode rewriters receive vulnerability reports and patch the contract at the byte level. The testing module then verifies that the patch will work with the previous transactions. After the test passes, the deployer uploads the patched contract to Ethereum.

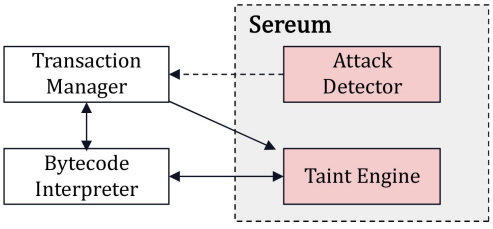

Figure 9: Diagram of Sereum System Architecture.

Table 12
Summarization of Methods for DeFi Optimization.

Reference	Target	Key Technologies	Features
Wang et al. (2021a)	Data Flow Dependency Violation on Invariant	Symbolic Reasoning Transaction Monitor	Two sections track data flow dependencies and monitor malicious transactions respectively
Wu et al. (2021)	Price Manipulation Attack	Semantic Lifting Pattern Detection	By collecting historical transaction data, a Cash Flow Tree (CFT) is constructed, which is used to recover high-level semantic information
Popescu (2020)	Unexpected Code in Smart Contract	Pricing Decentralized Underwriting	Tokens are used to relate membership relationships, and members work together to maintain contracts
Mussenbrock et al. (2017)	Crypto wallets Collateral	Market-based Approach	It is a centralized platform used for covering multiple risks, and if the user suffers enough damage, all losses can be refunded
Ivanov et al. (2021)	Protocol Failure Stablecoin Yield DEX	Multiple-chain Covers Portfolio	Integrate Various insurance and liquidity provision can improve the performance and maximize insurance revenues
Peaster et al. (2022)	Derivatives Options	Automated Squeeth Strategy Liquidity Provider	Code audits and establishing reliable protocols for bounty payments
Ivanov (2022)	Smart Contract Stablecoin Decentralized Exchange (DEX)	Managed by DAO Discretionary Coverage	Coverage provision is added by users, so the coverage can be transparent, and there are many asset pools to share risks in the smart contract, stablecoin, and DEX

Many tools only detect bugs but do not quantify and track stolen assets. To monitor stolen assets, the Eye of Horus (Ferreira Torres et al., 2021) employs knowledge graph technology. It extracts data streams of vulnerable transactions using taint analysis. Then it analyzes the input data relationships and finds attacks from the generated logs. Finally, it obtains the attacker's addresses and timestamps, then loads the transaction information into a graphical database for access to the asset flow.

ÆGIS (Ferreira Torres et al., 2020) constructed some control flow and data flow patterns describing the vulnerabilities and modified the EVM so that it could revert to transactions against patterns during contract operation. Anyone can submit a pattern when discovering a new vulnerability. All voters in the chain then determine whether the pattern can be added to the list.

SODA (Cao et al., 2020) is an online detection framework for a smart contract that includes a manager, information collector, and logger. It provides registered and unregistered APIs to APPs and requires the APP to send operational information, block numbers, and different functions to the manager. The information collector collects all blocks, transactions, and contract structure information for vulnerability detection, and the logger issues an alert if it finds anomalies. This framework with detection APPs is compatible with multiple blockchains.

Nguyen et al. (2021) proposed SGUARD, which evaluates control dependencies to discover malicious opcodes once all traces are enumerated at the bytecode level and located to external callers. Finally, the flaw is corrected through patching at the source code level. Correspondingly, it analyzes all types of data dependencies, including memory, storage, and stack. However, the limitation of this work is that there is no explicit number of iterations to obtain all data dependencies in smart contracts.

5.5. Application Layer Optimizations

There are some market manipulations at the application level that can lead to damage to user assets, but some research and works in Table 12 exist to safeguard various applications, including optimization tools and insurance.

5.5.1. Optimization Tools

Although there is a correlation between the various layers, methods for lower levels can not fully recognize the attacks against the application layer. Some research(Wang et al., 2021a; Wu et al., 2021) makes contributions.

Wang et al. (2021a) designed Blockeye to divide the detection work into two phases. In Figure 10, the first phase uses symbolic execution analysis in oracle to check whether state data streams are externally manipulated to detect vulnerable DeFi. During the second phase, transaction monitors under the chain collect transactions to extract the features and further analysis to monitor the attack.

DeFiRanger (Wu et al., 2021) is a price manipulation checker that first collects transactions on Ethereum and constructs a tree structure on the flow of cash. Then it defines the style based on DeFi attack behavior and recovers the low-level semantics to high-level semantics. Finally, the system detects attacks and analyzes them to generate a report based on the style and high-level semantics.

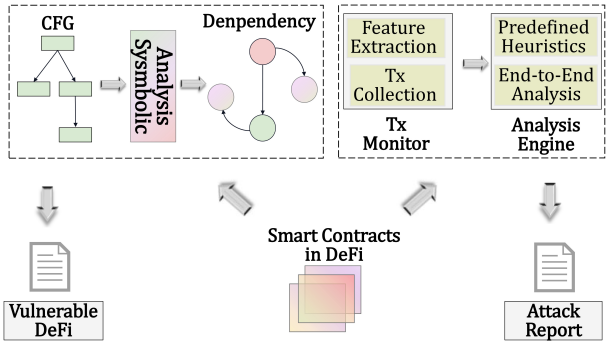

Figure 10: Diagram of BLOCKEYE Core Process.

5.5.2. Insurance

As the DeFi market expands, insurance is critical to ensuring its stability (Popescu, 2020). Our research divides risks in DeFi into market risks, technical risks, and credit risks. However, the enormous damages experienced by regular users result from technical or credit risks. So there require insurance systems to safeguard the properties of users, and they can be classified as centralized and decentralized.

For example, Smart Contract Cover, which offers smart contract insurance, is evaluated by the Nexus Mutual internal assessors (Popescu, 2020) to determine the cost of the insurance. It is self-governed by the members who own the NXM token of the pool built with "mutual", making the risk evenly shared.

Etherisc (Mussenbrock et al., 2017) is a centralized platform that offers several insurance programs, including crypto wallets and collateral. As for crypto wallets, it guarantees wallets against hacking and the risk of theft. Another one refers to the protection of the price of the collateral provided by the borrower within a certain range. For example, when the price of the collateral is reduced by more than 90% due to a big drop in the market, the insurance will pay the original price to the borrower.

Opyn (Peaster et al., 2022), which focuses on insurance for option trading products, enables users to choose options to hedge risks based on ERC20 tokens, and the protocol is automatically performed by smart contracts for multi-party governance. It also provides audit services and publishes all contracts, offering bounties for contract optimization.

Bridge Mutual (Ivanov, 2022) allows users to add insurance to products of their choice, and the decentralization makes the entire coverage process transparent for insurance purposes. To achieve risks shared, thousands of pools covering diverse platforms, stable currencies, and exchanges are coefficients.

Bright Union (Ivanov et al., 2021) integrates most insurance in the industry and licenses to encrypt them to enable mutual coverage without a license. It can improve cost performance for higher complete coverage by combining various insurance.

6. Challenges and Future Directions

6.1. Challenges

DeFi is a mechanism built through blockchain technology as the underlying technology that does not rely on the operation of a centralized service. The most significant advantage of DeFi is that funds are automatically traded through technical protocols. Thus it eliminates the need for human intervention, which increases liquidity in the market and thus facilitates financial transactions. However, DeFi still has challenges, and the following issues can be solved to facilitate the ecosystem's development better.

(i) Data issues remain a challenge. Technologies such as oracle are already being used to solve the problem of data synchronization, and wallets protect the key to ensure the encryption process. The improved transmission protocols are designed to secure the data transmission process. However, under the complete autonomy of blockchain, the data cannot be modified after it is transmitted on the chain in principle. An error in the data transmission process or the data source will lead to problems in the system. The technology in these aspects is not mature enough. It cannot fully guarantee the correctness of data without affecting the other performance of the system, such as scalability and throughput.

(ii) Key technologies of blockchain are not mature enough. Starting with Bitcoin, the consensus mechanism is one of the fundamental technologies of blockchain. Even though it has few vulnerabilities, it still has a series of actions against consensus rule flaws that may result in forking. More importantly, the loss it causes is significant since forking is irreversible. The smart contract is the basis of various DAPPs, and the vulnerabilities that exist in it are also endless. Due to the rapid development of the smart contract, there is still a lack of practical tools for potential vulnerabilities in it.

(iii) The technology features are not fully utilized, but there are just migrating the existing mechanisms. The various existing economic instruments lead to technical indicators' failure. Because DeFi is not fully integrated with the natural world, it currently relies on various virtual indicators for the prices of financial products. However, when some data beyond the normal range enters the system, it can lead to errors. Furthermore, since there is no trusted third party to guarantee DeFi, it has to rely on various protocol stacks, which leads to fragmentation and uncertainty of assets.

(iv) The DeFi account has numerous security issues. Accounts are the entities that perform transactions, and in the real world, users use them to send transactions to the blockchain. The new users tend to choose DeFi apps with high transaction volumes and user activity. Many Bot Accounts in a DeFi application could create a risk of fraud against real users. Bot accounts are

program-controlled accounts that initiate transactions to the DeFi app regularly, ensuring that the DeFi app is active. DeFi lacks the means to detect bot accounts and is still in the manual detection stage.

(v) Some DeFi users engage in irresponsible financial conduct. DeFi evolved from traditional forms of finance, and it contains the same wrong behaviors in conventional finance, such as market manipulation and arbitrage. Most DeFi security work currently focuses on researching smart contract vulnerabilities. It lacks the research on wrong financial behaviors in DeFi. The immaturity of the related technologies and the complicated economic environment have led to the difficulty of behavior detection in the DeFi ecosystem.

6.2. Future Directions

In addition to cryptocurrencies, Ethereum and other systems in the 2.0 era of blockchain have a more comprehensive range of financial applications with the development of smart contracts. However, due to the combination of multiple protocol stacks and the development of blockchain technology, DeFi's security is also gaining attention. Therefore, we offer some possible future directions for enhancement.

(i) Oracle system that can adequately connect to the outside world is desperately needed. DeFi services are about quantifying the laws of the natural world through technical methods. However, when there is a large amount of asset price data beyond what the quantified system can rate, this can cause the original quantified system to run out of steam. So DeFi desperately needs a sound Oracle system to connect the objective external world to the DeFi system to achieve equal and reasonable coexistence between the data.

(ii) Secure DeFi protocol development experience guide is needed. As described in Section 5, many optimization options are available that improve security for developers and users. Moreover, with the growth of DeFi, more and more novices are getting into the DeFi protocol development work. And DeFi protocols' security optimization necessitates some prior knowledge. As a result, most novice developers cannot effectively use the optimization tools available. So to enhance the stable development of the industry, security guidance work for DeFi protocol development is urgently needed.

(iii) Mature sidechain technology is worth developing. Financial projects involve many money transactions, and DeFi is no exception. Blockchain technology is crucial for DeFi services. When users initiate transactions using DeFi services, the blockchain underlay processes all the transactions before returning to the user interaction interface. So the speed of transactions in blockchain is significantly correlated with the transaction speed of DeFi users. Moreover, without compromising security, the sidechain extension technology will better increase the system throughput of transactions to increase the number of transactions and transaction speed processed by the system.

(iv) The security of blockchain technology needs to be improved. As the underlying technology on which DeFi technology relies, blockchain technology needs more security research. As summarized in Section 3, many vulnerabilities still have caused real-world attacks. In Section 4, we can see that these attacks have caused severe damage, affecting people's confidence in DeFi technology. Although there have been many optimizations, the number of attacks against blockchain technology has not decreased. So it is worthwhile to continue studying security for all layers of blockchain structure, especially the smart contract layer that triggered most of the vulnerabilities.

(v) Effective multi-layer vulnerability detection tool is still lacking. The detection method designed for a particular layer cannot detect higher-level information. For example, detection tools designed for the smart contract use the information in the smart contract, so they cannot detect the features at the application level. Most application-level attacks must combine multiple layers, so joint detection of vulnerabilities between multiple layers is worth investigating.

(vi) The dynamic supervision techniques for each layer can be improved. There are many static analysis methods and detection techniques. However, these techniques cannot prevent the damage caused by the attack timely when the attack occurs. Efficient dynamic supervision technology could solve this dilemma. However, there is a lack of efficient supervision technology for various layers, including the data, network, and application layers.

(vii) Decentralized applications should be fully integrated with technical features. The current application design copies other existing frameworks, ensuring the application's usability but sacrificing security. For example, many DAPPs in DeFi simulate the real-world design of financial product data. However, they do not fully consider all the circumstances. A more significant number of assets than the existing assets suddenly entering the system will significantly shrink the existing proportion of assets. Thus, market participants cannot have the right to master the assets. Therefore, the DAPPs should take full advantage of the technical features instead of abandoning the design framework, thus leading to an imperfect combination.

7. Conclusion

DeFi is a new type of platform based on blockchain technology that may increase the number of financial transactions while also efficiently enhancing the development of finance. This paper is the first systematic analysis of all

levels of vulnerability, real-world attacks, and optimization schemes. Furthermore, based on our systematic analysis, we provide some DeFi challenges and future directions. First, we start with a systematic analysis of each layer, and a series of vulnerabilities are summarized. For each vulnerability, we investigate real-world attack cases and explore the vulnerabilities used in each case. We then summarize the studies on optimizing for these vulnerabilities at each layer. Finally, we summarize the dilemmas and security issues encountered. In terms of the future directions of optimization, we believe that comprehensive attack analysis and monitoring are critical to DeFi security.

References

Adam, S., Péter, S., Jeffrey, W., 2013. Go ethereum:official go implementation of the ethereum protocol. URL: https://geth.ethereum.org/. (accessed 3 August 2022).

Adams, H., Zinsmeister, N., Salem, M., Keefer, R., Robinson, D., 2021. Uniswap v3 core. URL: https://uniswap.org/whitepaper-v3.pdf. (accessed 24 July 2022).

Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N., Kastania, A., 2018. Astraea: A decentralized blockchain oracle, in: Proceedings of the IEEE international conference on internet of things (IThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData), pp. 1145–1152.

Amani, S., Bégel, M., Bortin, M., Staples, M., 2018. Towards verifying ethereum smart contract bytecode in isabelle/hol, in: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP), pp. 66–77.

Amler, H., Eckey, L., Faust, S., Kaiser, M., Sandner, P., Schlosser, B., 2021. Defi-ning defi: Challenges & pathway, in: Proceedings of the 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), pp. 181–184.

Angeris, G., Chitra, T., 2020. Improved price oracles: Constant function market makers, in: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies (AFT), pp. 80–91.

Arapinis, M., Gkaniatsou, A., Karakostas, D., Kiayias, A., 2019. A formal treatment of hardware wallets, in: Proceedings of the International Conference on Financial Cryptography and Data Security (FC), pp. 426–445.

BadgerDAO, O., 2021. Badgerdao exploit technical post mortem. URL: https://badger.com/technical-post-mortem. (accessed 24 July 2022).

Baranov, V., 2022. Distributortreasury code. URL: https://andromeda-explorer.metis.io/address/0x6f99b960450662d67bA7DCf78ac959dBF9050725/contracts/. (accessed 1 August 2022).

Bartoletti, M., Chiang, J.H.y., Lafuente, A.L., 2021. Towards a theory of decentralized finance, in: Proceedings of the International Conference on Financial Cryptography and Data Security (FC), pp. 227–232.

Baum, C., David, B., Frederiksen, T.K., 2021. P2dex: privacy-preserving decentralized cryptocurrency exchange, in: Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS), pp. 163–194.

Bekemeier, F., 2021. Deceptive assurance? a conceptual view on systemic risk in decentralized finance (defi), in: Proceedings of the 4th International Conference on Blockchain Technology and Applications (ICBTA), pp. 76–87.

Beregszaszi, A., Śliwak, K., et al., 2022. Solidity releases. URL: https://blog.soliditylang.org/category/releases/. (accessed 21 July 2022).

Billy, C., 2018. Integer overflow found in multiple erc20 smart contracts. URL: https://medium.com/@peckshield/integer-overflow-i-e-proxyoverflow-bug-found-in-multiple-erc20-smart-contracts-14fecfba2759/. (accessed 1 August 2022).

Bin, Q., Zhang, S., et al., 2022. Meter whitepaper. URL: https://www.allcryptowhitepapers.com/meter-whitepaper/. (accessed 1 August 2022).

Bouteloup, J., 2022. Rekt. URL: https://rekt.news/leaderboard/. [accessed on 27 July, 2022].

Breidenbach, L., Cachin, C., Chan, B., et al., 2022. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. URL: https://chain.link/whitepaper. (accessed 3 August 2022).

Bscscan, T., 2021a. Meerkat finance contract code. URL: https://bscscan.com/address/0x7e0c621ea9f7afd5b86a50b0942eaee68b04a61c#code. [accessed on 6 June, 2022].

Bscscan, T., 2021b. Uraniumpair code. URL: https://bscscan.com/address/0xa08c4571b395f81fbd3755d44eaf9a25c9399a4a#code/. (accessed 16 June 2022).

Buterin, V., 2016. Critical update re: Dao vulnerability. URL: https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/. (accessed 3 August 2022).

Buterin, V., et al., 2014. A next-generation smart contract and decentralized application platform. white paper , 2–1.

Caldarelli, G., Ellul, J., 2021. The blockchain oracle problem in decentralized finance—a multivocal approach. Applied Sciences 11, 7572. doi: https://doi.org/10.3390/app11167572.

Cao, R., Chen, T., Li, T., Luo, X., Gu, G., Zhang, Y., Liao, Z., Zhu, H., Chen, G., He, Z., Tang, Y., Lin, X., Zhang, X., 2020. Soda: A generic online detection framework for smart contracts, in: Proceedings of the 27th Network and Distributed System Security Symposium (NDSS), pp. 1–17.

Chen, J., Xia, X., Lo, D., Grundy, J., Luo, X., Chen, T., 2020a. Defining smart contract defects on ethereum. IEEE Transactions on Software Engineering , 327–345doi: https://doi.org/10.1109/TSE.2020.2989002.

Chen, J., Xia, X., Lo, D., Grundy, J., Luo, X., Chen, T., 2021. Defectchecker: Automated smart contract defect detection by analyzing evm bytecode. IEEE Transactions on Software Engineering doi: https://doi.org/10.1109/TSE.2021.3054928.

Chen, T., Feng, Y., Li, Z., Zhou, H., Luo, X., Li, X., Xiao, X., Chen, J., Zhang, X., 2020b. Gaschecker: Scalable analysis for discovering gas-inefficient smart contracts. IEEE Transactions on Emerging Topics in Computing 9, 1433–1448.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (KDD), pp. 785–794.

Chen, T., Li, X., Luo, X., Zhang, X., 2017a. Under-optimized smart contracts devour your money, in: Proceedings of the 24th IEEE international conference on software analysis, evolution and reengineering (SANER), pp. 442–446.

Chen, T., Li, X., Wang, Y., Chen, J., Li, Z., Luo, X., Au, M.H., Zhang, X., 2017b. An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks, in: Proceedings of the International conference on information security practice and experience (ISPEC), pp. 3–24.

Chen, T., Li, Z., Zhu, Y., Chen, J., Luo, X., Lui, J.C.S., Lin, X., Zhang, X., 2020c. Understanding ethereum via graph analysis. ACM Transactions on Internet Technology 20, 1–32. doi: https://doi.org/10.1145/3381036.

Chen, W., Zheng, Z., Cui, J., Ngai, E., Zheng, P., Zhou, Y., 2018. Detecting ponzi schemes on ethereum: Towards healthier blockchain technology, in: Proceedings of world wide web conference (WWW), pp. 1409–1418.

Choi, J., Kim, D., Kim, S., Grieco, G., Groce, A., Cha, S.K., 2021. Smartian: Enhancing smart contract fuzzing with static and dynamic data-flow analyses, in: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 227–239.

Coinchange, Financial, I., 2022. Regulated decentralized finance (rdefi) is paving the way for exponential growth. URL: https://www.globenewswire.com/news-release/2022/05/04/2435410/0/en/Regulated-Decentralized-Finance-rDeFi-is-Paving-The-Way-for-Exponential-Growth.html. (accessed 25 July 2022).

bZx Contributor, 2021. Preliminary post mortem. URL: https://bzx.network/blog/prelminary-post-mortem. (accessed 26 July 2022).

Cronje, A., Kong, M., Alexander, et al., 2022. Fantom. URL: https://fantom.foundation/. (accessed 1 August 2022).

CryptoSec, 2022. Documented timeline of defi exploits. URL: https://cryptosec.info/defi-hacks/. (accessed 27 July 2022).

CVE, 2022. Cve search results. URL: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=smart+contract. (accessed 21 July 2022).

Dabrowski, A., Pfeffer, K., Reichel, M., Mai, A., Weippl, E.R., Franz, M., 2021. Better keep cash in your boots-hardware wallets are the new single point of failure, in: Proceedings of the ACM CCS Workshop on Decentralized Finance and Security (DeFi), pp. 1–8.

Dai, W., Deng, J., Wang, Q., Cui, C., Zou, D., Jin, H., 2018. Sblwt: A secure blockchain lightweight wallet based on trustzone. IEEE Access 6, 40638–40648. doi: https://doi.org/10.1109/ACCESS.2018.2856864.

Daian, P., 2016. Analysis of the dao exploit. URL: https://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/. (accessed 3 August 2022).

Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A., 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability, in: Proceedings of the IEEE Symposium on Security and Privacy (SP), pp. 910–927.

Di Angelo, M., Salzer, G., 2020. Characteristics of wallet contracts on ethereum, in: Proceedings of the 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), pp. 232–239.

Duan, L., Sun, Y., Zhang, K., Ding, Y., 2022. Multiple-layer security threats on the ethereum blockchain and their countermeasures. Security and Communication Networks 2022. doi: https://doi.org/10.1155/2022/5307697.

Dwivedi, A.D., 2020. Security analysis of lightweight iot cipher: Chaskey. Cryptography , 22doi: https://doi.org/10.3390/cryptography4030022.

Egberts, A., 2017. The oracle problem-an analysis of how blockchain oracles undermine the advantages of decentralized ledger systems. Available at SSRN 3382343 doi: http://dx.doi.org/10.2139/ssrn.3382343.

EigenPhi, 2022. Sandwich arbitrage. URL: https://eigenphi.io/ethereum/sandwich/. (accessed 21 July 2022).

Eleven, F., 2021. Elevenneversellvault code. URL: https://bscscan.com/address/0x27dd6e51bf715cfc0e2fe96af26fc9ded89e4be8#code/. (accessed 3 August 2022).

Etherscan, 2018. Mesh token. URL: https://etherscan.io/address/0x3ac6cb00f5a44712022a51fbace4c7497f56ee31#code/. (accessed 1 August 2022).

Ferreira Torres, C., Baden, M., Norvill, R., Fiz Pontiveros, B.B., Jonker, H., Mauw, S., 2020. Ægis: Shielding vulnerable smart contracts against attacks, in: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (SIGSAC), pp. 584–597.

Ferreira Torres, C., Iannillo, A.K., Gervais, A., et al., 2021. The eye of horus: Spotting and analyzing attacks on ethereum smart contracts, in: Proceedings of the International Conference on Financial Cryptography and Data Security (FC), pp. 33–52.

Flatow, J., Hayes, G., et al., 2021. compound finance. URL: https://etherscan.io/address/0x75442Ac771a7243433e033F3F8EaB2631e22938f#code/. (accessed 21 July 2022).

FTMScan, 2021. Source of grimboostvault code. URL: https://ftmscan.com/address/0x660184ce8af80e0b1e5a1172a16168b15f4136bf#code. (accessed 2 August 2022).

Fu, Y., Ren, M., Ma, F., Shi, H., Yang, X., Jiang, Y., et al., 2019. Evmfuzzer: detect evm vulnerabilities via fuzz testing, in: Proceedings of the 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering (FSE), pp. 1110–1114.

Gao, Z., Jayasundara, V., Jiang, L., Xia, X., Lo, D., Grundy, J., 2019. Smartembed: A tool for clone and bug detection in smart contracts through structural code embedding, in: Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 394–397.

Greig, J., 2022. Hackers steal more than 11 million usd from elephant money defi platform. URL: https://therecord.media/hackers-steal-more-than-11-million-from-elephant-money-defi-platform/. (accessed 21 July 2022).

Gudgeon, L., Perez, D., Harz, D., Livshits, B., Gervais, A., 2020. The decentralized financial crisis, in: Proceedings of the Crypto Valley Conference on Blockchain Technology (CVCBT), pp. 1–15.

Haber, S., Stornetta, W.S., 1990. How to time-stamp a digital document, in: Proceedings of the Conference on the Theory and Application of Cryptography (TAC), pp. 437–455.

Han, J., Song, M., Eom, H., Son, Y., 2021. An efficient multi-signature wallet in blockchain using bloom filter, in: Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC), pp. 273–281.

He, J., Balunović, M., Ambroladze, N., Tsankov, P., Vechev, M., 2019a. Learning to fuzz from symbolic execution with application to smart contracts, in: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 531–548.

He, S., Wu, Q., Luo, X., Liang, Z., Li, D., Feng, H., Zheng, H., Li, Y., 2018. A social-network-based cryptocurrency wallet-management scheme. IEEE Access 6, 7654–7663. doi: https://doi.org/10.1109/ACCESS.2018.2799385.

He, X., Lin, J., Li, K., Chen, X., 2019b. A novel cryptocurrency wallet management scheme based on decentralized multi-constrained derangement. IEEE Access 7, 185250–185263. doi: https://doi.org/10.1109/ACCESS.2019.2961183.

Henningsen, S., Teunis, D., Florian, M., Scheuermann, B., 2019. Eclipsing ethereum peers with false friends. arXiv preprint arXiv:1908.10141 doi: https://doi.org/1908.10141/arXiv.1908.10141.

Huang, J., Han, S., You, W., Shi, W., Liang, B., Wu, J., 2021. Hunting vulnerable smart contracts via graph embedding based bytecode matching. IEEE Transactions on Information Forensics and Security 16, 2144–2156. doi: https://doi.org/10.1109/TIFS.2021.3050051.

Ivanov, K., 2022. Bridge mutual white paper. URL: https://uploads-ssl.webflow.com/5fac3e348dbd5932a7578690/60a7ffd8a8874fc955e580ac_Bridge%20Mutual%20WP%20v1.pdf. (accessed 1 August 2022).

Ivanov, K., Kool, J., et al., 2021. Bright union. URL: https://brightunion.io/. (accessed 21 July 2022).

Jakobsson, M., Juels, A., 1999. Proofs of work and bread pudding protocols, in: Secure information networks. Springer, pp. 258–272.

Jensen, J.R., von Wachter, V., Ross, O., 2021. An introduction to decentralized finance (defi). Complex Systems Informatics and Modeling Quarterly , 46–54doi: https://doi.org/10.7250/csimq.2021-26.03.

Jian, Z., Ran, Q., Liyan, S., 2021. Securing blockchain wallets efficiently based on threshold ecdsa scheme without trusted center, in: Proceedings of the Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS), pp. 47–51.

Jin, L., Cao, Y., Chen, Y., Zhang, D., Campanoni, S., 2022. Exgen: Cross-platform, automated exploit generation for smart contract vulnerabilities. IEEE Transactions on Dependable and Secure Computing doi: https://doi.org/10.1109/TDSC.2022.3141396.

Kaleem, M., Shi, W., 2021. Demystifying pythia: A survey of chainlink oracles usage on ethereum, in: Proceedings of the International Conference on Financial Cryptography and Data Security (FC), Springer. pp. 115–123.

Kaliski, B., 2000. PKCS# 5: Password-based cryptography specification version 2.0. Technical Report. RSA Laboratories.

Kenton, P., 2020. The maker protocol: Makerdao's multi-collateral dai (mcd) system. URL: https://makerdao.com/en/whitepaper/. [accessed on 31 July, 2022].

Khan, A.G., Zahid, A.H., Hussain, M., Riaz, U., 2019. Security of cryptocurrency using hardware wallet and qr code, in: Proceedings of International Conference on Innovative Computing (ICIC), IEEE. pp. 1–10.

Kistner, K.J., 2021. itoken duplication incident report. URL: https://bzx.network/blog/incident/. (accessed 21 July 2022).

Kumar, M., Nikhil, N., Singh, R., 2020. Decentralising finance using decentralised blockchain oracles, in: Proceedings of the International Conference for Emerging Technology (INCET), pp. 1–4.

Lashkari, B., Musilek, P., 2021. A comprehensive review of blockchain consensus mechanisms. IEEE Access , 43620–43652doi: https://doi.org/10.1109/ACCESS.2021.3065880.

Lattner, C., Topper, C., Pilgrim, S., et al., 2017. libfuzzer – a library for coverage-guided fuzz testing. URL: https://llvm.org/docs/LibFuzzer.html. (accessed 26 July 2022).

Li, X., Chen, T., Luo, X., Wang, C., 2021. Clue: towards discovering locked

cryptocurrencies in ethereum, in: Proceedings of the 36th Annual ACM Symposium on Applied Computing (SAC), pp. 1584–1587.

Li, X., Chen, T., Luo, X., Yu, J., 2020a. Characterizing erasable accounts in ethereum, in: Proceedings of the International Conference on Information Security (ISC), Springer. pp. 352–371.

Li, X., Chen, T., Luo, X., Zhang, T., Yu, L., Xu, Z., 2020b. Stan: Towards describing bytecodes of smart contract, in: Proceedings of the 20th IEEE International Conference on Software Quality, Reliability and Security (QRS), IEEE. pp. 273–284.

Li, X., Jiang, P., Chen, T., Luo, X., Wen, Q., 2020c. A survey on the security of blockchain systems. Future Generation Computer Systems 107, 841–853. doi: https://doi.org/10.1016/j.future.2017.08.020.

Liu, Q., Yu, L., Jia, C., 2020. Mover: stabilize decentralized finance system with practical risk management, in: Proceedings of the 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), pp. 55–56.

Luu, L., Chu, D.H., Olickel, H., Saxena, P., Hobor, A., 2016. Making smart contracts smarter, in: Proceedings of the ACM SIGSAC conference on computer and communications security (CCS), pp. 254–269.

Luu, L., Teutsch, J., Kulkarni, R., Saxena, P., 2015. Demystifying incentives in the consensus computer, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 706–719.

Lyanchev, J., 2022. Over $700 million liquidated as terra (luna) crashes below $1. URL: https://cryptopotato.com/over-700-million-liquidated-as-terra-luna-crashes-below-1/. (accessed 24 July 2022).

Marcus, Y., Heilman, E., Goldberg, S., 2018. Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive doi: https://eprint.iacr.org/2018/236.

Mingxiao, D., Xiaofeng, M., Zhe, Z., Xiangwei, W., Qijun, C., 2017. A review on consensus algorithm of blockchain, in: Proceedings of the IEEE international conference on systems, man, and cybernetics (SMC), pp. 2567–2572.

Mussenbrock, C., Konstantin, A., et al., 2017. Etherisc white paper. URL: https://uploads-ssl.webflow.com/6243075ff83d08a79dc7b307/624edb8a4b12178635a629c6_etherisc_whitepaper_1.01_en.pdf. (accessed 5 June 2022).

Nakamoto, S., 2008. Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review , 21260doi: http://dx.doi.org/10.2139/ssrn.3440802.

Nam, W., Kil, H., 2022. Formal verification of blockchain smart contracts via atl model checking. IEEE Access doi: https://doi.org/10.1109/ACCESS.2022.3143145.

Network, R., 2022. Community alert: Ronin validators compromised. URL: https://roninblockchain.substack.com/p/community-alert-ronin-validators?s=w. (accessed 27 July 2022).

Nguyen, T.D., Pham, L.H., Sun, J., 2021. Sguard: Towards fixing vulnerable smart contracts automatically, in: Proceedings of the IEEE Symposium on Security and Privacy (SP), pp. 1215–1229.

Nick, S., Thomas, B., Greg, K.e.a., 2022. Provable: blockchain-agnostic oracle service for dapps developers. URL: https://defiprime.com/provable. (accessed 1 August 2022).

Nikolić, I., Kolluri, A., Sergey, I., Saxena, P., Hobor, A., 2018. Finding the greedy, prodigal, and suicidal contracts at scale, in: Proceedings of the 34th annual computer security applications conference (ACSAC), pp. 653–663.

NVD, 2022. National vulnerability database. URL: https://nvd.nist.gov/vuln/. (accessed 26 July 2022).

Obelisk, T., 2021. Meerkat finance hack. URL: https://obeliskauditing.com/blog/articles/meerkat-rug-article. [accessed on 6 June, 2022].

Peaster, W., Kaur, G., et al., 2022. Opyn. URL: https://www.defipulse.com/projects/opyn. (accessed 12 May 2022).

PeckShield, 2020. bzx hack ii full disclosure. URL: https://medium.com/@peckshield/bzx-hack-ii-full-disclosure-with-detailed-profit-analysis-8126eecc1360/. [accessed on 20 July, 2022].

Popescu, A.D., 2020. Transitions and concepts within decentralized finance (defi) space. Research Terminals in the social sciences .

Popescu, A.D., et al., 2020. Decentralized finance (defi)-the lego of finance. Social Sciences and Education Research Review 7, 321–349.

Qin, K., Zhou, L., Afonin, Y., Lazzaretti, L., Gervais, A., 2021a. Cefi vs. defi-comparing centralized to decentralized finance. arXiv preprint arXiv:2106.08157 doi: https://doi.org/10.48550/arXiv.2106.08157.

Qin, K., Zhou, L., Gamito, P., Jovanovic, P., Gervais, A., 2021b. An empirical study of defi liquidations: Incentives, risks, and instabilities, in: Proceedings of the 21st ACM Internet Measurement Conference (IMC), pp. 336–350.

Qin, K., Zhou, L., Gervais, A., 2022. Quantifying blockchain extractable value: How dark is the forest?, in: Proceedings of the IEEE Symposium on Security and Privacy (SP), IEEE. pp. 198–214.

Qin, K., Zhou, L., Livshits, B., Gervais, A., 2021c. Attacking the defi ecosystem with flash loans for fun and profit, in: Proceedings of the International Conference on Financial Cryptography and Data Security (FC), pp. 3–32.

REKT, 2021. Eleven, finance. URL: https://rekt.news/11-rekt/. (accessed 21 July 2022).

Rezaeighaleh, H., Zou, C.C., 2019. Deterministic sub-wallet for cryptocurrencies, in: Proceedings of the IEEE International Conference on Blockchain (Blockchain), pp. 419–424.

Richards, S., Wackerow, P., Smith, C., et al., 2022a. Ethereum erc-20 standard. URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/. [accessed on 1 August, 2022].

Richards, S., Wackerow, P., Smith, C., et al., 2022b. Ethereum erc-777 standard. URL: https://ethereum.org/en/developers/docs/standards/tokens/erc-777/. [accessed on 1 August, 2022].

Ritzdorf, H., Wüst, K., Gervais, A., Felley, G., Capkun, S., 2017. Tls-n: Non-repudiation over tls enabling-ubiquitous content signing for disintermediation. Cryptology ePrint Archive doi: https://eprint.iacr.org/2017/578.

Rodler, M., Li, W., Karame, G., Davi, L., 2019. Sereum: Protecting existing smart contracts against re-entrancy attacks, in: Proceedings of the Network and Distributed System Security Symposium (NDSS), pp. 24–27.

Rodler, M., Li, W., Karame, G.O., Davi, L., 2021. Evmpatch: Timely and automated patching of ethereum smart contracts, in: Proceedings of the 30th USENIX Security Symposium (USENIX Security), pp. 1289–1306.

Schär, F., 2021. Decentralized finance: On blockchain-and smart contract-based financial markets. FRB of St. Louis Review doi: http://dx.doi.org/10.20955/r.103.153-74.

Shaman, R., Chemla, J., et al., 2022. Defillama. URL: https://defillama.com/. [accessed on 10 July, 2022].

Shbair, W.M., Gavrilov, E., State, R., 2021. Hsm-based key management solution for ethereum blockchain, in: Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–3.

SlowMist, 2021. Slowmist: Analysis of uranium finance's hacked event. URL: https://slowmist.medium.com/slowmist-analysis-of-uranium-finances-hacked-event-9c9d11af7b2b. [accessed on 27 May, 2022].

So, S., Hong, S., Oh, H., 2021. Smartest: Effectively hunting vulnerable transaction sequences in smart contracts through language model-guided symbolic execution, in: Proceedings of the 30th USENIX Security Symposium (USENIX Security), pp. 17–20.

So, S., Lee, M., Park, J., Lee, H., Oh, H., 2020. Verismart: A highly precise safety verifier for ethereum smart contracts, in: Proceedings of the IEEE Symposium on Security and Privacy (SP), pp. 1678–1694.

SpiritSwap, 2022. Spirit swap home page. URL: https://www.spiritswap.finance. (accessed 24 July 2022).

Suratkar, S., Shirole, M., Bhirud, S., 2020. Cryptocurrency wallet: A review, in: Proceedings of the 4th International Conference on Computer, Communication and Signal Processing (ICCCSP), pp. 1–7.

Swende, M.H., Ferrante, M.D., et al., 2017. Utilities for interacting with the ethereum virtual machine. URL: https://github.com/ethereum/evmlab. (accessed 21 July 2022).

Synthetix, 2019. Synthetix response to oracle incident. URL: https://blog.synthetix.io/response-to-oracle-incident/. (accessed 21 June 2022).

Szabo, N., 1996. Smart contracts: building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought,(16) 18, 28.

Torres, C.F., Schütte, J., State, R., 2018. Osiris: Hunting for integer bugs in ethereum smart contracts, in: Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC), pp. 664–676.

Wahab, A., Mehmood, W., 2018. Survey of consensus protocols. arXiv preprint arXiv:1810.03357 doi: https://doi.org/10.48550/arXiv.1810.03357.

Wang, B., Liu, H., Liu, C., Yang, Z., Ren, Q., Zheng, H., Lei, H., 2021a. Blockeye: Hunting for defi attacks on blockchain, in: Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering: Companion Proceedings (ICSE), IEEE. pp. 17–20.

Wang, D., Wu, S., Lin, Z., Wu, L., Yuan, X., Zhou, Y., Wang, H., Ren, K., 2021b. Towards a first step to understand flash loan and its applications in defi ecosystem, in: Proceedings of the Ninth International Workshop on Security in Blockchain and Cloud Computing (SBC), pp. 23–28.

Wang, S.H., Wu, C.C., Liang, Y.C., Hsieh, L.H., Hsiao, H.C., 2021c. Promutator: Detecting vulnerable price oracles in defi by mutated transactions, in: Proceedings of the IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 380–385.

Wang, Z., Qin, K., Minh, D.V., Gervais, A., 2022. Speculative multipliers on defi: Quantifying on-chain leverage risks. Financial Cryptography and Data Security doi: https://doi.org/10.1016/j.cose.2019.101604.

Werner, S.M., Perez, D., Gudgeon, L., Klages Mundt, A., Harz, D., et al, K., 2021. Sok: Decentralized finance (defi). arXiv preprint arXiv:2101.08778 doi: https://doi.org/10.48550/arXiv.2101.08778.

Winter, P., Lorimer, A.H., Snyder, P., Livshits, B., 2021. What's in your wallet? privacy and security issues in web 3.0. arXiv preprint arXiv:2109.06836 doi: https://doi.org/10.48550/arXiv.2109.06836.

Wood, G., et al., 2014. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151, 1–32.

Wu, S., Wang, D., He, J., Zhou, Y., Wu, L., Yuan, X., He, Q., Ren, K., 2021. Defiranger: Detecting price manipulation attacks on defi applications. arXiv preprint arXiv:2104.15068 doi: https://doi.org/10.48550/arXiv.2104.15068.

Wüst, K., Gervais, A., 2016. Ethereum eclipse attacks. Technical Report. ETH Zurich.

Xu, G., Guo, B., Su, C., Zheng, X., Liang, K., Wong, D.S., Wang, H., 2020. Am i eclipsed? a smart detector of eclipse attacks for ethereum. Computers & Security 88, 101604. doi: https://doi.org/10.1016/j.cose.2019.101604.

Xue, Y., Ma, M., Lin, Y., Sui, Y., Ye, J., Peng, T., 2020. Cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts, in: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1029–1040.

Yang, Y., Kim, T., Chun, B.G., 2021. Finding consensus bugs in ethereum via multi-transaction differential fuzzing, in: Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 349–365.

Yazdanparast, E., 2021. All you need to know about defi flash loans. URL: https://medium.com/coinmonks/all-you-need-to-know-about-defi-flash-loans-ca0ff4592d90. [accessed on 21 May, 2022].

Zhang, F., Cecchetti, E., Croman, K., Juels, A., Shi, E., 2016. Town crier: An authenticated data feed for smart contracts, in: Proceedings of the ACM SIGSAC conference on computer and communications security (CCS), pp. 270–282.

Zhang, F., Maram, D., Malvai, H., Goldfeder, S., Juels, A., 2020. Deco: Liberating web data using decentralized oracles for tls, in: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1919–1938.

Zhou, L., Qin, K., Cully, A., Livshits, B., Gervais, A., 2021a. On the just-in-time discovery of profit-generating transactions in defi protocols, in: Proceedings of the IEEE Symposium on Security and Privacy (SP), pp. 919–936.

Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A., 2021b. High-frequency trading on decentralized on-chain exchanges, in: Proceedings of the IEEE Symposium on Security and Privacy (SP), pp. 428–445.