
Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Are Smart Contract Attack Strategies Optimized?

Nguyen D. Trung

Bachelor's Thesis

Are Smart Contract Attack Strategies Optimized?

Bachelor's Thesis

Supervised by Prof. Dr. Phil. nat. Sebastian Steinhorst
Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Advisor Jens Ernstberger

Co-Advisor Liyi Zhou

Author Nguyen D. Trung
Lissi-Kaeser-Straße 08
80797 München

Submitted on December 05,2023

Declaration of Authorship

I, Nguyen D. Trung, declare that this thesis titled "Are Smart Contract Attack Strategies Optimized?" and the work presented in it are my own unaided work, and that I have acknowledged all direct or indirect sources as references.
This thesis was not previously presented to another examination board and has not been published.

Signed:

Date:

Acknowledgments

I would like to express my sincere gratitude to my dedicated supervisors, Jens Ernstberger and Liyi Zhou, for their unwavering support and guidance. They have consistently provided support and motivation, even amid their busy schedule.

My heartfelt appreciation to my partner, Dan Phuong, for standing by me through the challenges of my thesis and undergraduate journey. She has consistently provided support and motivation.

Additionally, I would like to thank the TUM Blockchain Club for allowing me multiple opportunities to expose to professionals in the industry which helped me to boarder my knowledge in the field.

Eventually, I would like to send a special acknowledgment to my family and friends whose encouragement has been a source of strength and an integral part during the course of my studies.

I would not have been able to accomplish this work without the support of each of these people.

List of Acronyms

ABI Application Binary Interface

API Application Programming Interface

BSC Binance Smart Chain

BTC Bitcoin

DeFi Decentralized Finance

DAO Decentralized Autonomous Organization

DEX Decentralized Exchange

EOA External Owned Account

EVM Ethereum Virtual Machine

ETH Ether

LLM Large Language Model

MEV Miner Extracted Value

RAM Random Access Memory

RPC Remote Procedure Call

TVL Total Value Locked

TRXhash Transaction Hash

Abstract

Since the first blockchain-based Decentralized Finance (DeFi) went online on the Ethereum main net in 2017, DeFi Ecosystems have reached their peak in Total Value Locked (TVL) of 258 billion USD. At the time of writing, although the market has been observing a bear market, TLV in DeFi protocols has been very stable with protocols in two Ethereum Virtual Maschine (EVM) compatible blockchains holding more than 60% of the market, Ethereum (33 billion USD, 52.94%) and BNB Smart Chain (4,475 billion USD, 7.17%). DeFi introduces a variety of protocols inspired by traditional finance, such as cryptocurrency exchanges, lending platforms, and derivatives. Additionally, it introduces innovative concepts like flash loans and atomic composability in DeFi trading. Considering DeFi protocols' transparency, they are prone to many exploitations from adversaries with monetary motives. Thus, an interesting question arises: Would the adversaries have been able to take advantage of the protocols differently using different strategies?

For this thesis, we have conducted a comprehensive dataset of 147 high-impact real-world DeFi incidents on Ethereum and BNB Smart Chain from May 01, 2022, to Oct 31, 2023, in which period the reported damage value amounted to over 1 billion USD. The analysis is supported by a State-of-the-Art Smart Contract fuzzer, which suggests that some of the attack strategies are only sub-optimal. Finally, based on our calculation using the dataset, we conclude that the rescue time frame for the defending party in a smart contract attack has decreased compared to prior years.

Contents

1

Introduction

"The blockchain is an incorruptible digital ledger of economic transactions that can be programmed to record not just financial transactions but virtually everything of value."

Don & Alex Tapscott

1.1 Motivation

Ever since the publication of Ethereum's white paper by Vitalik Buterin in 2014, and the subsequent official launch of the Ethereum Network, a realm of innovation has unfolded. The introduction of Smart Contracts on the Ethereum Network brought forth a paradigm shift, introducing a programmable blockchain whose potential transcends the inherent capabilities of a distributed ledger, as observed in Bitcoin. This opened up a new era of possibilities and functionalities within the blockchain space.

This breakthrough has empowered the creation of decentralized applications (DApps), enabling the execution of trustless transactions without reliance on a central authority. Moreover, these DApps harness the capabilities of smart contracts to fashion intricate financial protocols, as evidenced by the emergence of decentralized exchanges [1, 2], lending platforms [3, 4], staking protocols [5, 6], and derivatives [7, 8].Decentralized Finance ecosystems [9, 10] have been successfully attracting investors. At its peak, at the end of 2021, Total Value Locked (TVL) in DeFi surpassed 180 billion USD. Together with its inherent transparency, this has consequentially made DeFi Protocols attractive targets for adversaries with monetary motives.

The openness of these protocols makes them vulnerable to attacks from people who study and understand these weaknesses. The complexity of these systems, along with the potential for financial gains, creates a situation where attackers can use known problems to harm the

security of DeFi ecosystems. This highlights the need for continuous research and efforts to strengthen these protocols, ensuring they can withstand new threats and keep DeFi systems safe.

Previous research by Zhou et. al [11] has shown in a span of 4 years from Apr 30, 2018, to Apr 30, 2022, DeFi Ecosystems and their users have suffered from more than 3.24 billion USD loss due to adversarial attacks. Many of these attacks targeted Smart Contracts' vulnerabilities. According to the database by Zhou et. al [11], those attacks resulted in 1.54 billion USD loss (46.9% total loss). Based on real-world incidents, it yields two interesting questions: "Would the adversaries have been able to take advantage of the protocols differently using different strategies?" and "How much time did the defending parties have to react to malicious actors?". To stay up-to-date on the most recent developments in DeFi Ecosystem, it is necessary to follow the most recent incidents.

1.2 Contributions

We summarize our contributions as follows:

- Extending the DeFi Incident dataset and keeping the dataset up-to-date with the most recent incidents on Ethereum and BNB Smart Chain from May 01, 2022, to Oct 31, 2023
- Leverage a Bytecode-Level Hybrid Fuzzer for Smart Contracts to find out whether attacks are suboptimal, with a specific emphasis on monetary exploits.
- Analyzing the rescue timeframe for the entities defending against recent incidents from the dataset

By addressing these points, we want to deepen the understanding of Smart Contract Vulnerabilities, raise awareness of the risks associated with investing in DeFi markets, and provide a foundational reference for further research.

1.3 Paper Organization

For reference, below is how this paper is structured. In **Chapter 2**, we provide the fundamental knowledge about i) Ethereum, and all its most important aspects and extensions, such as the EVM, Ethereum Account, Smart Contracts, Gas ii) Decentralized Finance and its building blocks. iii) DeFi Security and the attack vectors. **Chapter 3** presents the important related works that are foundational for our current research. **Chapter 4** describes the methodology for collecting data and analyzing it. **Chapter 5** explores the optimality of some attacks using ItyFuzz, the SoTA smart contract fuzzer. In **Chapter 6**, we investigate the time frame for defending party in an under attack DeFi Protocol.

2

Background

"Bitcoin is just one example of something that uses the blockchain"
Melanie Swan

2.1 Decentralized Layer: Ethereum

Before the Ethereum Blockchain, the term blockchain was coined and widely used based on its technical characteristics. Bitcoin [12] is one of the most famous applications of the blockchain. According to Bitcoin's creator, it is a peer-to-peer electronic cash system. Essentially, Bitcoin Blockchains are co-hosted by computer networks called nodes, which are operated by miners. During a certain time frame, miners gather all transactions in the network, calculate whether the transactions are correct, and then include them in a "block", refer to Figure 5. If all other nodes agree that the fastest miner's block should be included in the blockchain's global state, the "block" will be included. Think of the Bitcoin blockchain as a ledger, exact copies of which are distributed to every participant in the network (nodes), who can keep track of every transaction from the first one ever made in the network, however, none is allowed to make any change or tamper with the record of the ledger. This allows Bitcoin to be: transparent, immutable, and as a result, trustworthy. However, acting only as a distributed ledger can only take it so far.

2.1.1 Backbone of the Ethereum Blockchain: the EVM

In 2014, Vitalik Buterin published a white paper on his original idea of a programmable blockchain system called Ethereum. This system allows developers to create and deploy smart contracts on the Ethereum network. Smart contracts are programs that run on the Ethereum Virtual Machine (EVM). The Ethereum Blockchain, like Bitcoin, can also be considered a ledger. However, at its core, it is more complex. Think of Ethereum as a transaction-based state-machine [13], as shown in Figure 1, which consists of several states, and the transitions

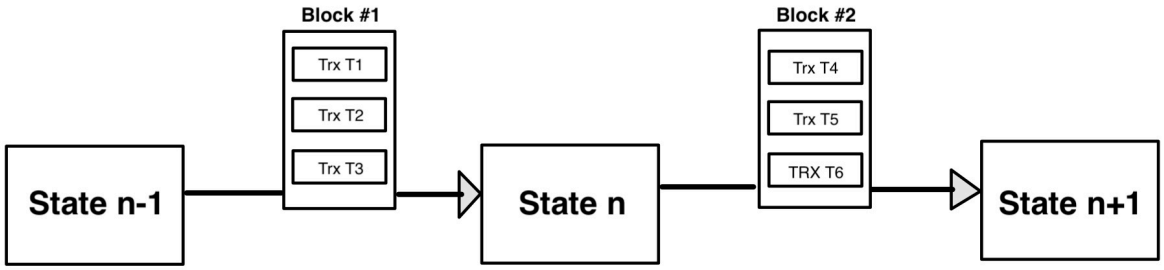

Figure 1: Ethereum as a state maschine

between those states. In Ethereum, states consist of objects called "accounts", 2.1.3. Each of these accounts has its identifier, a 20-byte address. State transitions are just the transfer of value between these accounts. In contrast to Bitcoin, Ethereum's accounts are more complex, as they can contain both code and data. Additionally, Ethereum accounts can interact with each other, allowing for more complex transactions.

The Ethereum Virtual Machine (EVM) is the key to Ethereum's distinction. On Ethereum, instead of simply adding to or subtracting from the amounts of tokens in accounts, network participants can construct sets of rules on how they want to manipulate the values on the Ethereum network. These sets of rules are referred to in the Ethereum white paper [14] as the "Smart Contracts", refer to 2.1.2. The EVM is a Turing-complete virtual machine that executes Ethereum's Smart Contracts. Executing Smart Contracts results in changes in values from Ethereum accounts. The EVM also recognizes several predefined exceptions such as Revert, OutOfGasError, InvalidParameter [15] which represent a failed execution due to logical, syntax, or resource insufficiency. When encountering exceptions, all changes made during the current transaction will be reverted.

Understanding "EVM-compatible": Being EVM (Ethereum Virtual Machine) compatible means that a blockchain or cryptocurrency adheres to the same standards and functionalities as the Ethereum blockchain. It allows decentralized applications (DApps) and smart contracts developed for Ethereum to be seamlessly executed on the compatible blockchain. Binance Coin (BNB), the native cryptocurrency of the Binance Smart Chain (BSC), is EVM-compatible. This means that developers can use the familiar Ethereum tools, programming languages, and existing smart contracts on BSC, offering a more flexible and interoperable environment for decentralized applications and fostering compatibility with the broader Ethereum ecosystem.

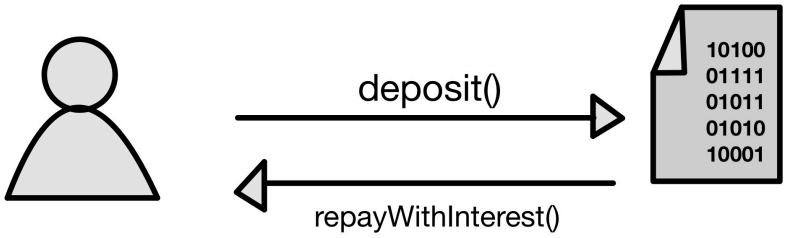

Figure 2: Interaction with a Smart Contract

2.1.2 What are Smart Contracts?

Smart Contracts are computer programs that live on the Ethereum blockchain. The term was first coined in 1994, by Nick Szabo [16]. Developed on Nick's idea of what Smart Contracts should do, these programs define how values should be handled or transferred between accounts on the Ethereum network.

Motivation example
Take a simple loan transaction into consideration. Following are the involved parties. A civilian, Alex, wants to borrow 1000 USD from a stranger, Bob. if Bob agrees to lend Alex only with 5% interest after 12 months, Alex will simply repay Bob 1050 USD, when the time dues. However, this is not always the case. There are times when Alex simply disappears with Bob's 1000 USD. A middleman, usually a lawyer or an institution with authority, should assure Alex keeps his word. Otherwise, a penalty should be imposed on Alex.

In the trustless world of Ethereum, the smart contract replaces the need for a middleman. Smart Contracts are computer programs that live on the Ethereum network, in which the logic of transactions is defined by deterministic code logic. Instead of having a third party, a "loan" Smart Contract will be deployed to the Ethereum network, in which it is defined that the lender will automatically receive 5% of interest at the end of the lending period. The contract is usually visible to the public which allows transparency of the system. From his External Owned Account (EOA) 2.1.3, network participants can trigger the "loan" contract, and the contract will take care of the repayment with interest after a pre-defined period. Figure 2, is the example of a smart contract that allows participants to deposit an amount of value to its Smart Contract Account 2.1.3 with the *deposit()* function, and when certain logic matches, the *repayWithInterest()* is triggered to repay the depositors.

Smart Contracts are usually written in Solidity, a high-level Turing complete language de-

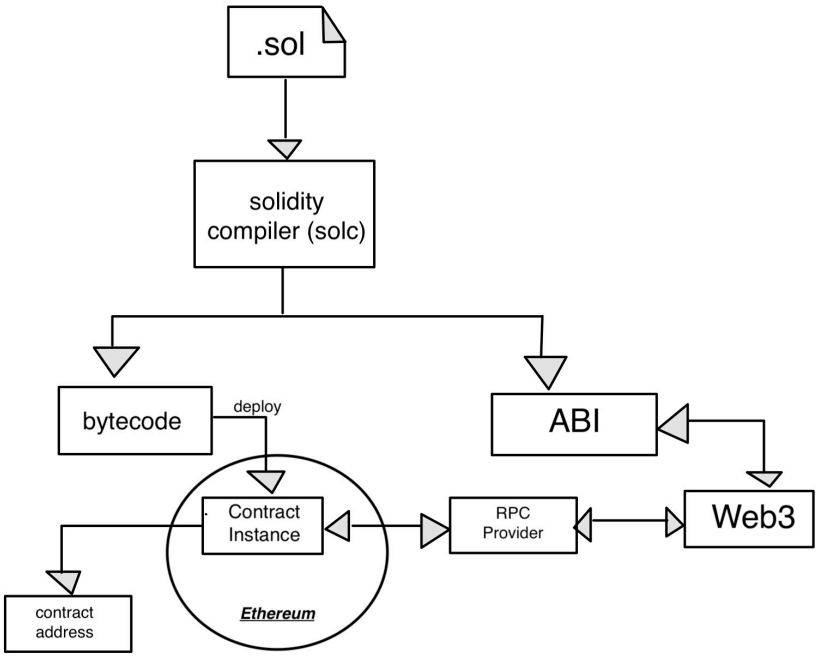

Figure 3: Smart Contract Compilation [19]

signed specifically to write Smart Contracts. This Solidity file should still be complied by a solidity compiler [17] into bytecode that EVM understands. The compilation process generates two main outputs: the bytecode and the Application Binary Interface (ABI). The bytecode is a set of opcode [18] that EVM understands and ABI is mainly used to interact with the Smart Contract like in Figure 3 [19].

Once deployed, Smart Contracts are technically: immutable, trustless, decentralized, and transparent [20]. This enables various services and applications we know today in DeFi ecosystems such as Decentralized Autonomous Organizations (DAO), Decentralized Exchanges, Lending and Borrowing services, and many more.

2.1.3 Own your token: Ethereum Account

Ethereum accounts are the primary objects of the Ethereum network. Essentially, as demonstrated by Figure 4, a state in the Ethereum network is simply the composition of many different accounts that store values, Ether, the native token of the Ethereum network or any token of type ERC-20 [21]. Every account is assigned a unique identifier in the form of a 20-byte address. An example of such an account address is **0x71C7656EC7ab88b098defB751B7401B5f6d8976F**.

There are two types of Ethereum accounts: External Owned Account (EOA) and Smart Contract Account. EOA is owned by an individual or entity, while the smart contract account is owned by its code. The EOA's address is usually referred to as its "public key", and only the

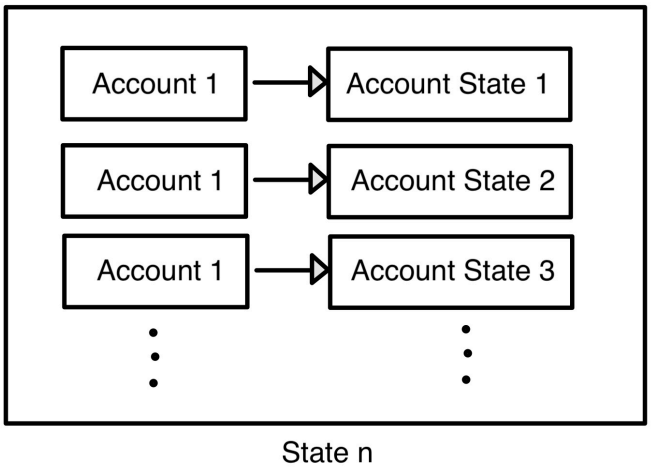

Figure 4: State in Ethereum

entity with the equivalent "private key" has access to its fund. Smart contract accounts are usually used to lock a certain amount of funds that serves the purpose of that certain contract.

For example, each liquidity pool, refer to 2.2.2 has in principle its Smart Contract that governs the operation of that pool, and its logic to send funds back to entities, or receive funds from another entity (can be an EOA or another Smart Contract Account). As a result, Smart Contract Accounts usually possess large amounts of funds, for instance, at the time of writing, Curve stETH/ETH pool of Lido holds 114 million USD worth of crypto tokens at the address **0xDC24316b9AE028F1497c275EB9192a3Ea0f67022** [22]. This makes Smart Contract Accounts an attractive target for adversarial parties.

2.1.4 Transaction's atomicity: all or nothing

In the Ethereum network, a transaction is a fundamental unit of activity that represents the execution of an operation on the Ethereum blockchain. Transactions are how participants interact with the Ethereum network to transfer value (Ether) or trigger the execution of smart contracts.

On a higher level, when a network participant sends tokens to another EOA or trigger functions in a Smart Contract, a transaction is emitted to the network under the name (address) of the participant. In Ethereum, before the new block is mined, all transactions will be gathered into a transaction pool, from which the miner (node operator) will pick out and add to his next block to be mined. The order of the transactions depends however totally on the sorting of the node operator. This allows miners to pick: a) transactions with the most gases included, b) transactions that can potentially allow them to exploit some arbitrage opportunities or to

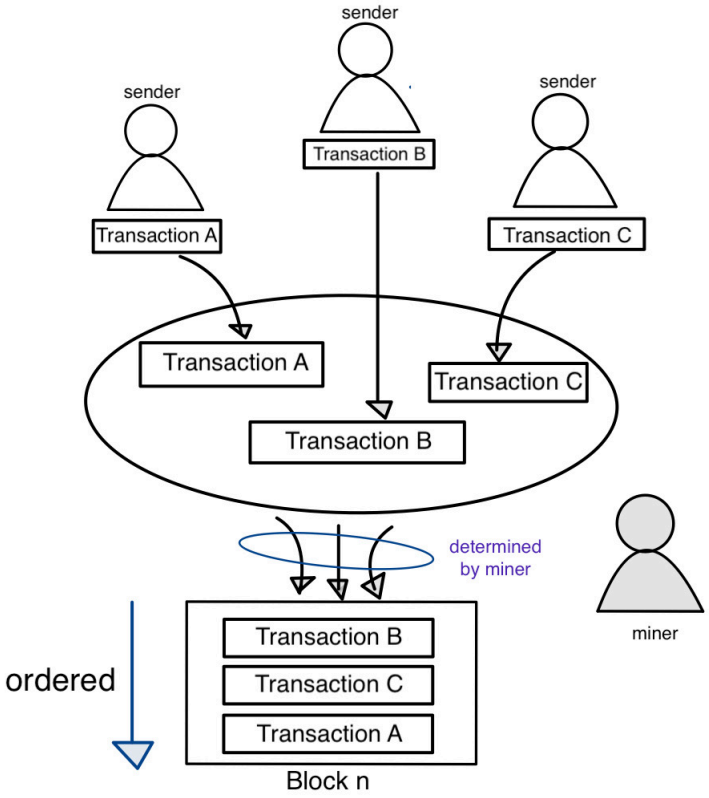

Figure 5: Ordering transactions [13]

front-run potentially profitable transactions. The gain that the miner can extract from this kind of behavior is introduced by Dain et al. [23] as the Miner Extracted Value (MEV). This yields a need for investors to technically censor their transactions from miners with tools like flashbot [24].

Another noticeable characteristic of a transaction is its atomicity. The atomic execution of a transaction in the Ethereum Virtual Machine (EVM) is a fundamental principle that ensures the integrity and consistency of smart contract transactions. In the context of EVM, atomicity means that either the entire transaction is successfully executed or none of it is, with no partial or incomplete results. Exactly this has enabled novel constructs such as flash loans, which can only exist on the blockchain.

2.1.5 Fueling your transactions with Gas

As much freedom as there is on the Ethereum blockchain, there is another factor that restricts participants from making whatever transactions as many times as they wish: the gas fee.
The Ethereum Virtual Machine (EVM) uses gas as a measure of computational work performed. With gas, the Ethereum blockchain calculates the amount of resources (computational power and storage) that are required for running programs and operations. Comparable to

general compiled programming language, each operation in a smart contract requires a specific amount of computational effort from the EVM that can be translated into gas. The Ethereum network witnesses fluctuations in its minimum gas price at different points in time, influenced by the prevailing network load. Moreover, network participants also have the option to specify the amount of Ether they are willing to pay for their transactions. A part of the gas fee is used to incentivize the node operators, while a part is consumed (burnt) by the network. Transactions with higher included gas have a higher chance of being added to the next block, while other transactions may have to wait even till the next blocks. Gas is denominated in a sub-unit of Ether (ETH) called "gwei." One Ether is equivalent to 1,000,000,000 gwei or 1×10^{18} gwei. The term "gwei" is derived from the name of Wei Dai, a computer scientist and cryptographer. Such a small sub-unit is used to help express smaller fractions of Ether more conveniently, given the divisibility of Ether into smaller units.

2.2 Building DeFi: The Crucial Role of Smart Contracts

A glimpse at history: Our financial system has been ever-evolving. From the Stone Age to Ancient Civilizations, we engaged in a barter system, trading goods for goods. Realizing the limitations of bartering due to market fluctuations in needs and supplies, we devised an early currency system using seashells or attractive stones to represent value. However, this solution proved susceptible to replication and lacked scarcity. Consequently, civilizations, such as the Roman Empire, transitioned to metallic currencies, particularly gold or silver, to establish more robust and widely accepted mediums of exchange. During these times, more primitive forms of financial services like lending already existed. Fast forward to the Medieval Period and Renaissance, we invented paper-based instruments like Promissory Notes used by merchants and traders in medieval Europe. A crucial turning point occurred in the 19th century with the adoption of the gold standard, backing paper currencies with tangible assets. This shift laid the foundation for our current financial system, marked by intricate global currency networks, advanced banking services, and complex financial instruments. This evolution reflects our ongoing pursuit of efficient and secure methods for economic transactions in a dynamic world.

Shifting to Decentralisation: With the introduction of blockchain technology, Ethereum emerged with its evolution cornerstone: EVM and Smart Contracts. In this section, we will discover how decentralization technology, Ethereum, and its Smart Contracts have been trying to imitate the traditional financial system, in which Decentralized Autonomous Organizations (DAOs), Decentralized Exchanges (Dex), Lending and Borrowing protocols, Staking protocol, and many other financial services emerged. Moreover, DeFi has also gone beyond to expand to more novel construct like flash-loan, which is only possible on the blockchain due to its atomicity in execution.

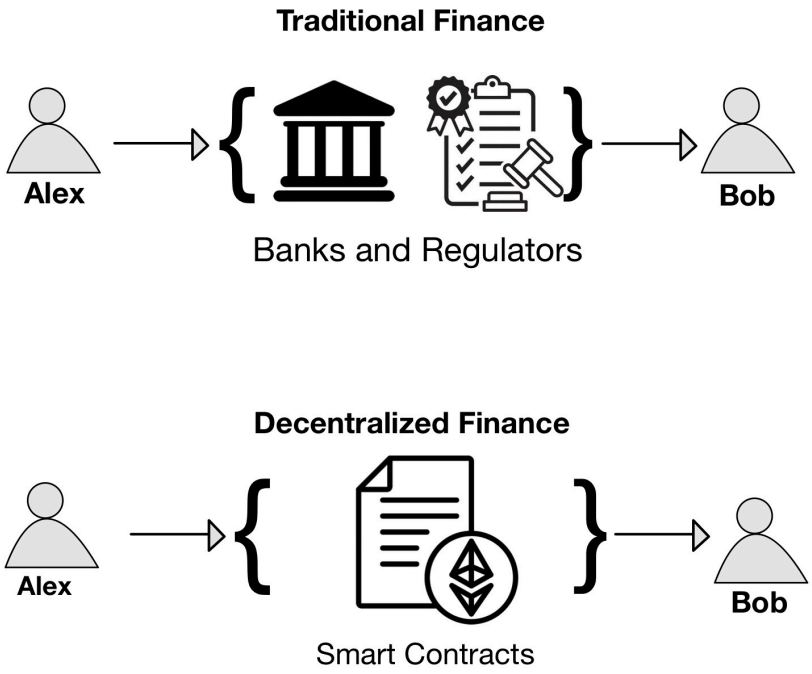

Figure 6: Traditional Finance vs DeFi

In the next parts, we will discover in depth the most prominent DeFi services and the Smart Contracts staked behind those DeFi protocols.

2.2.1 Stablecoins

As crypto tokens are known for their volatility, there is a need for an intermediary that is stable and reliable in order to provide an intermediary between crypto tokens. This led to the invention of stablecoins as a result [25, 26]. There is usually a peg to an external reference currency, fiat currency such as the USD or Euro, which determines the value of the currency. The use of these kinds of systems allows traditional finance systems to be seamlessly integrated with DeFi systems.

The three main Stablecoins are:

1. **Fiat-Collateralized Stablecoins:** These are backed by a reserve of fiat currency like the USD or Euro. Examples of these are USD Tether (USDT) issued by Tether [1], USD Coin (USDC) issued by Circle [2]. It is promised that each USDT or USDC is represented by a reserved USD in the issuer bank's reserve.
2. **Crypto-Collateralized Stablecoins:** This type of stablecoins are overcollateralized. In the case of DAI [27], a stablecoin backed by Ethereum on MakerDAO, everyone can

1 https://tether.to/en/
2 https://www.circle.com/en/

collateralize to an electric vault their asset to mint DAI. That vault is managed by smart contracts and uses external oracles to provide the value of the collateral asset.

3. **Algorithmic Stablecoins:** These are unconventional and not backed by any asset. Instead, there is an underlying algorithm responsible for maintaining the stability of that stablecoin, in which the algorithm adjusts the token supply. Examples of this type are: Ampleforth [3], Frax [28], and TerraSDR. Even before the downfall of Terra [29], there were already critics of the way this stablecoin works, it is argued that Terra failed by design [30] and algorithmic stablecoins have an unsustainable operating model [31].

2.2.2 Decentralized Exchange

Decentralized Exchanges (DEXs) emerged as early DeFi applications, designed to offer cryptocurrency trading services akin to traditional financial systems like Binance, eToro, TradeRepublic, or various bank-provided and third-party trading platforms. In these platforms, trading typically follows an order book model. In simple terms, Trader A initiates a trade, specifying the amount of their asset X and the desired asset Y at a particular trading rate. If Trader B, at that moment, wants to trade their asset Y for asset X at the same rate defined by Trader A, the trade is matched, and both traders obtain the desired assets. This order book mechanism facilitates decentralized trading by connecting users with complementary trading preferences.

However, as illustrated in Figure 6, DEXs aspire to go beyond traditional banking and regulatory entities by leveraging the power of Smart Contracts. In contrast to centralized financial systems, which rely on intermediaries like banks and regulatory bodies to facilitate and oversee transactions, DEXs utilize self-executing and programmable Smart Contracts to automate and secure the exchange of cryptocurrencies.

There are different types of DEXs: Automated Market Makers (AMM), Order Book, and DEX Aggregator. In the scope of this research, we will be mostly focusing on the Automated Market Maker (AMM). For this, we examine the example of Uniswap V2, one of the earliest and most prominent AMMs, which played a crucial role in the early years of DeFi [32].

Unlike traditional trading platforms, Uniswap uses Smart Contracts to solve the liquidity puzzle. The key to it is Liquidity Pools. Liquidity Pool is the term to refer to a Smart Contract that is, from its creation, designated to reserve in its Smart Contract Account only 2 types of tokens A and B.

For a liquidity pool to function, two roles must be fulfilled:

1. **Liquidity Providers(LPs):** deposit at once an equivalent value of each underlying token. Furthermore, LPs are given the option to create a liquidity pool of two arbitrary

3 https://www.ampleforth.org/

tokens [4]. In return, they receive pool tokens, which can be used to withdraw the funds at any time in the future. Some lending protocols even accept this pool token to be collateral for a loan.

2. **Traders:** As long as the necessary liquidity pool exists, they can trade for a desirable asset at the current rate defined by the protocol with *slippage* at all times.

How does Liquidity Pool work: At all times, following each trade or deposit, an invariant must be maintained. This invariant can be articulated as follows:

$$\text{Amount of token A} \times \text{Amount of token B} = k \tag{2.1}$$

In essence, after each trade, regardless of the number of tokens A withdrawn from the pool, an equivalent amount of token B should be redeposited into the pool to ensure that the constant k remains unchanged.

Theoretically, if there are the following tokens in reserve i) 1200 tokens A and ii) 400 tokens B (k = 1200 * 400, a trader, who wants to trade for 1 token B, must put around 3 tokens A back in reserve to maintain the integrity of k. Of course, such a system brings no economic value to the protocol that it cannot justify its existence.

In reality, there are multiple factors involved. Besides 3 tokens A put back in reserve, traders should also pay i) the transaction fee to initiate that trade to the Ethereum Network, and ii) the 0.3% trading fee on SushiSwap. Not to mention, there is also a slippage in prices, which is the difference between the price of the token when the trader initiates the transaction (expected price) and the price of the token when the transaction is executed (actual price). The slippage formula is given by:

$$\text{Slippage} = \frac{\text{Expected Price} - \text{Actual Price}}{\text{Expected Price}} \times 100$$

In summary, Decentralized Exchanges (DEXs), particularly Automated Market Makers (AMMs) like Uniswap V2, utilize Smart Contracts and Liquidity Pools. Liquidity Providers (LPs) deposit tokens to maintain a constant product (k) formula, enabling traders to execute transactions with minimal slippage. This mechanism, despite its theoretical simplicity, involves factors like fees and slippage, contributing to the broader evolution of DeFi

4 https://github.com/Uniswap/v2-periphery/blob/master/contracts/UniswapV2Router01.sol#L30-L57

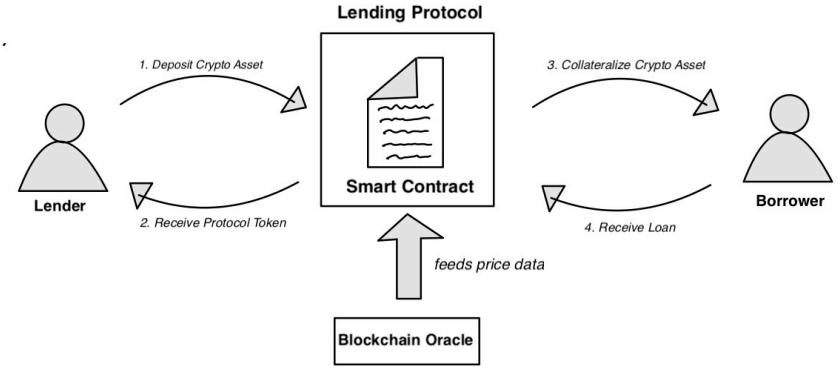

Figure 7: DeFi Lending and Borrowing

2.2.3 Lending and Borrowing

Besides DEX, DeFi Lending [3, 33, 34] is among the most popular DeFI services. Trying to keep the core value proposition of cryptocurrency that everyone's asset should be held in self-custody, DeFi Lending Protocols are like other DeFi services open to public use and open-source for any further third-party development.

Figure 7 describes a standard flow of a lending protocol:

1. A lender deposits his crypto asset to the protocol or a certain money market. If one wants to provide funds to the Ethereum market, he should deposit Ethereum.
2. In return, he receives a protocol token (CETH on Compound, AETH on Aave), the value of which should indicate the interest he wins over time.
3. A borrower collateralizes his asset based on the collateral factor depending on the asset.
4. Borrower receives a loan.

Besides that, there should be an Oracle constantly updating the price of the collateralized asset and the borrowed asset. The loan is bound to no time limit, however, there is one invariant that must hold indefinitely:

$$\text{Value of borrowed amount} < \text{Value of collateral amount} * \text{Collateral factor} \tag{2.2}$$

If the value of the collateral falls below the borrowed amount, the collateral asset will be liquidated by the protocol to keep the lender's fund safe and the borrowed amount always over-collateralized

Flash Loan

Flash loan is a novel construct that allows the borrower to take a loan from a lender and leave the lender with no default risk since the borrower has to repay the loan with service cost in the same transaction. This concept was introduced by Aave and implemented in Aave v3 [35]. This opened up new possibilities for complex financial strategies within the DeFi ecosystem. However, Flash Loan is not friendly to all DeFi users, as it requires good technical knowledge to perform.

2.2.4 Oracles

Very often, protocols depend on external sources or even outside of the blockchain realm to feed them the necessary data. Compound Protocol needs to constantly check the price of Ethereum to make sure loans are not undercollateralized, in which case, they must be liquified in defense of the protocol's fund.

However, inherently, blockchain is unable to access external data. Moreover, API calls in between transactions can break the deterministic trait of a blockchain. In the world of Blockchain, this is defined as an "Oracle Problem" [36]

According to Marriam-Webster dictionary, the term Oracle stems from ancient Greek and refers to a person (as a priestess of ancient Greece) through whom a god is believed to speak [37]. A blockchain oracle is any device or entity that connects off-chain data with a deterministic blockchain. These entities feed the blockchain external data through a transaction, which preserves the blockchain's deterministic trait.
For example, Compound Protocol [3] has its data feed taking data from highly liquid exchange, while Aave [33] depends on an Oracle network, Chainlink [38] to get external data and falls back to its internal data-feed when necessary.

However centralized Oracle can unintentionally create a single point of failure. Oracle hacks are always among the most damaging hacks in DeFi [39, 11]. To address this, decentralized protocols [40, 41, 38], have been developed with a primary emphasis on validating data and establishing consensus on individual off-chain values. This approach aims to enhance the reliability of these values, ensuring their accuracy and trustworthiness when triggering smart contracts.

2.2.5 Decentralized Autonomous Organization (DAO)

Decentralized Autonomous Organization leverages blockchain to apply a governance system on all DAO participants. Most DAOs are built on the Ethereum network and operate through rules defined in their Smart Contract. Participants in DAO should "vote" using their DAO

token to make collective decisions and govern decentralized applications or protocols.

The type of governance described here is known as token-based governance. Participants, who hold the DAO token, can perform their voting right. The more tokens a user holds, the more weight he has in the vote. For instance one of the recent proposals on MakerDao [42], passed on 19th November 2023. There have been 45 Token-holders supporting the proposal that the contract on this address 0xda6960 should be executed. The majority token holders voted in favor of the proposal, and the contract was executed.

2.3 DeFi Security and Common Attack Vectors on Smart contracts

While numerous doors for innovation are opening, developers also encountered plenty of new challenges unique to the development of smart contracts [43, 44, 45]. Indeed, from a technical standpoint, once a smart contract is deployed on the main network, it becomes impossible to apply updates or patches to that specific contract. The perfection of a smart contract is crucial from the outset, as updates or modifications become impractical once the contract is deployed on the main network.

In practice, however, there are workarounds for this technical challenge like data migration to another contract with bug-fix updates. This requires a lot of community management efforts, as teams should inform every user to instead use the new smart contracts. Another option, which is also widely implemented are proxy contracts, in which users only directly communicate with the proxy contracts [46], which are responsible for forwarding those command to the core contracts. This way, developers can simply replace the core contracts that need an update with new ones without sacrificing user experience. This is usually referred to as the upgradable smart contracts approach.

In certain scenarios, especially when substantial sums, often amounting to millions of USD, are secured within these smart contracts, the inability to easily modify or rectify issues poses a significant challenge for both developers and users. As the biggest DeFi protocols like Aave [5] or Uniswap[6] publish their smart contract code on open-source platforms, adversarial parties have the space and time to study the protocol's vulnerability months before the actual exploits. Therefore, it is extremely important to understand the common smart contract vulnerability and DeFi attack vectors to apply best practices when developing a smart contract which can reduce the risk of fund loss due to technical exploitations. In the next part, we will explore the most common smart contract vulnerabilities and DeFi attack vectors that are related to

5 https://github.com/aave
6 https://github.com/Uniswap

the scope of our research.

2.3.1 Smart Contract Vulnerabilities

Reentrancy

Reentrancy is one of the most common vulnerabilities known to smart contract development. As in its name, the attacker or attack contract will try to re-enter the control flow of the vulnerable contract by making a recursive call back to the original function before the first innovation of that function is finished. This usually happens when the vulnerable contract performs an external call to an untrusted smart contract. Hereby, the attack contract can attempt to drain all the funds stored in the vulnerable contract.

To help avoid this, it is recommended to follow the best practices [47]. Besides, there are several static analysis tools [48, 49, 50] and libraries [51].

Integer Overflow, under flow

It is common in computer systems that values are only granted a fixed limit amount of memory. If this limit is crossed, an overflow or underflow error will occur, causing the value to wrap around to the opposite extreme [52]. In solidity, the main language used to write smart contracts, there are two types of integers provided i) signed integers, with values ranging from -2^{255} to $2^{255}-1$, and ii) unsigned integers, from 0 to $2^{256}-1$ [53]. This is considered an inherent vulnerability of the Solidity language.

To mitigate this risk, Solidity versions from 0.8.0 onward incorporate built-in safeguards against such vulnerabilities. Additionally, developers commonly employ tools like SafeMath or equivalent libraries to ensure secure arithmetic operations in their code.

Logic Error

Poor programmed smart contracts can lead to easy to comprise contract integrity. Such logical flaws can facilitate unintended transactions or faulty mathematical operations. Given the case of Level Finance. In the contract *LevelReferralControllerV2*, it allows users to claim rewards during an epoch. However, there is no condition to check whether the *claimable()* function on line 6, has already been called. This means, that as long as the call is made in the same epoch, the attacker can exploit to mounted sum of *totalReward* in the end. The exploit was executed in transaction 0xe1f25 resulted in a 1.1 million USD lost.

Listing 2.1: Solidity code for the 'claimMultiple' function

```
1 function claimMultiple(uint256[] calldata _epochs, address _to) external {
2     uint256 totalReward;
3     for (uint256 i = 0; i < _epochs.length; ++i) {
4         uint256 epoch = _epochs[i];
```

```
        if (epoch < currentEpoch) {
            uint256 reward = claimable(epoch, msg.sender);
            if (reward > 0) {
                users[epoch][msg.sender].claimed = reward;
                totalReward += reward;
                emit Claimed(epoch, _to, reward);
            }
        }
    }

    LVL.safeTransfer(_to, totalReward);
}
```

Access control

If a smart contract fails to implement effective access control or employs a rudimentary control mechanism that is susceptible to easy circumvention, it may expose crucial functions. This vulnerability could allow unauthorized users to execute actions that ought to be restricted, such as modifying the contract's state, withdrawing funds, or claiming rewards multiple times.

2.3.2 Oracle Manipulation Attack

Although, normally, a protocol depends on another external source to fetch the necessary data, problems arise when that source becomes compromised and starts feeding manipulated data. This type of attack is more dangerous when multiple protocols depend on one compromised oracle. The consequences of manipulated data feeds are numerous, from unwarranted liquidations to malicious arbitrage trades. Mitigation can be that protocols should get data from multiple oracles to reduce to risk of their only oracle being compromised.

2.3.3 Flash Loan Attack

As mentioned above, flash loans are a novel construct that is only possible on the blockchain thanks to its atomicity in execution. Adversarial parties are not required to deposit any collateral, which in this case allows them to borrow a large amount of funds that they do not possess. With that fund, they can execute multiple price manipulation attacks to create a massive artificial arbitrage.

The difference between a flash loan attack and a flash loan arbitrage lies in the intention of the adversary. In a flash loan attack, the adversary tries to intentionally disrupt the normal functionality of the platform to manipulate prices or simply seeks chaos. While in a flash loan arbitrage, the arbitrageur takes a straight-forward path to gain financially through a price difference opportunity.

2.3.4 Governance Attack

This type of attack usually targets protocol with governance systems or DAOs in an attempt to manipulate or control the decision-making processes. The adversaries would try to gain the majority of governance tokens of the protocol to influence the decision-making. Then they inject malicious governance commands in an attempt to make beneficial changes for the parties, these can include changes to protocol parameters, upgrades, or other modifications.

To defend against governance attacks, protocols often implement mechanisms to prevent the concentration of voting power, discourage Sybil attacks, and ensure that proposals are in the best interest of the broader community. Active community engagement, transparent governance processes, and security audits are essential components of a robust decentralized governance system.

3

Related Work

3.1 Overview of existing research

As of the current moment, as reported by Defillama[1], the cumulative value locked in DeFi Protocols has surpassed 61 billion USD across various blockchain networks. It is imperative to highlight that these substantial funds are consistently exposed to potential risks within the dynamic and rapidly evolving landscape of DeFi. This makes the field of DeFi security have to evolve and adapt dynamically to its broader developments. This parallel growth has led to notable advancements in fortifying the security landscape of DeFi protocols. There have been great efforts to demonstrate a collective knowledge base of DeFi incidents [11, 45, 54] and classify the incidents [55, 11] for seemless reference and to lay the groundwork for other work to build upon.

Static analysis has always been an important approach during smart contract development and audit [56, 57]. Formal Verification is also an often used method to analyze the safety of smart contracts [58] And there are more automated tools using the fuzzing approach to automatically generate inputs to test behaviors of smart contracts under different circumstances [59, 60, 61, 62], in which the smart contract fuzzer ItyFuzz also makes use of symbolic execution that helps to increase coverage of the generated input.

With the advancement of Machine Learning and Large Language Models (LLM), many doors have been opened for researchers and tools to investigate smart contract vulnerabilities with very high precision [63, 64, 65]. Beyond research, in practice, there are very efficient tools based on LMM that can red-flag malicious contracts or actors based on past patterns [66].

1 https://defillama.com/

3.2 Challenges in this field of research

In the dynamic landscape of blockchain and smart contracts, security challenges persistently evolve. With hacks occurring frequently, keeping up with the constantly changing threat landscape has proven to be challenging. The rapid pace of innovation, coupled with the sophisticated tactics employed by malicious actors, underscores the ongoing challenges faced by developers, auditors, and the broader blockchain community in maintaining the robust security of smart contracts.

In a stateful system like the EVM, the execution of smart contracts results in changes to the global state of the blockchain. This includes modifications to account balances, storage values, and contract addresses. Auditing tools must carefully examine the state changes caused by each contract function, considering how they impact the overall system. This poses a challenge for automated tools to handle state changes efficiently without having high overhead, which slows down the auditing process [62].

Moreover, DeFi is a space where the protocols are heavily intertwined with each other. Smart contracts often interact with other contracts, tokens, oracles, and external data sources. Auditors need to consider the entire ecosystem and understand how different components interact, as vulnerabilities might arise from these interactions.

4

DeFi Incident Dataset

In this chapter, we will present the extension of an already existing dataset [11]. This extension will allow us to further analyze the data and gain deeper insights into the underlying patterns. To be more precise, the existing database is not up-to-date since it only contains incidents on the Ethereum Network and BNB Smart Chain (BNB) until April 2022. In this work, we collected another 147 incidents from 2nd May 2022 to 1st October 2022. 1st of November 2023. This new data has been added to our database, providing a comprehensive set of incidents that can be used to further investigate the problem. This extension is also serving as data for further contributions made in this paper.

4.1 What are DeFi Incidents?

Similar to any traditional cyber security incidents, DeFi incidents involve adversarial parties or accidents that result in the loss of funds from users, and platforms. Incidents can occur on different levels of the DeFi reference frame presented by Zhou et. al [11]: Network Layer, Consensus Layer, Smart Contract Layer, Protocol Layer, or Auxiliary Service Layer.

In the scope of this research, our primary focus lies on technical incidents within the DeFi protocols rather than network components, consensus components, or external services of those protocols. Hence the main focus will be on the Smart Contract Layer and Protocol Layer. These incidents are distinguished by their association with technical vulnerabilities present in smart contracts, protocols, or unsafe dependencies of the protocols on each other. We specifically concentrate on investigating and understanding the intricacies of security breaches, exploits, or other technical shortcomings that may lead to unexpected outcomes in the DeFi ecosystem.

In our research, we do not consider rug pulls or scams. While rug pulls and scams are indeed

prevalent within the DeFi landscape, they fall outside the purview of our investigation. Our emphasis remains on the examination of incidents arising from identifiable technical vulnerabilities, providing a more nuanced understanding of the challenges and risks associated with the technical aspects of DeFi.

4.2 Data Availability on the Blockchain

To gather and analyze the dataset accurately, a foundational understanding of the EVM, transaction execution processes, and the ability to identify victim contracts, adversaries, and attack contracts is essential. Fortunately, public blockchains are transparent by design, as mentioned in the previous chapter 2. Every one of the transactions has its unique identifier, the transaction Hash (usually abbreviated: TRXHash). The transaction hash serves as a cryptographic fingerprint for a specific transaction, and it is generated using a one-way hashing algorithm. This hash uniquely represents the entire set of data associated with a transaction, including sender and recipient addresses, amounts, and any other relevant details. As a result, any change, no matter how small, in the transaction's information would lead to a completely different transaction hash. What this means is that the history of completed transactions cannot be tampered with, is immutable, and is available at all times for the public. We can always find out what happened in a specific time-stamp given the transaction hash and even track back to previous transactions that lead to the examined situation.

In addition to displaying the transactions themselves, the blockchain also reveals the addresses responsible for initiating those transactions. An address on the blockchain acts like a digital identifier for participants in the network, providing transparency about who is involved in specific transactions. This visibility of addresses enhances accountability and traceability, allowing users to examine the transaction history associated with each address. It's a key aspect of the blockchain's transparency, as users can openly inspect the activities of participants while maintaining the pseudonymous nature of addresses.

In addition, there are other things available on the blockchain, such as the Bytecode, ABI, or sometimes, the deployed smart contract of a DeFi Protocol. Bytecode is a set of instructions that tell the computer how to run the smart contract. Once a smart contract is written, it gets turned into this bytecode (refer to figure 3), and everyone can take that bytecode to decompile it and see what is defined in that smart contract. This makes the rules of the contract unchangeable, so you can trust that they won't be tampered with. Bytecode also helps the computer network work together by letting each part independently follow the same set of instructions. Think of it as a recipe that everyone follows exactly, making sure that smart contracts operate fairly and predictably for everyone involved. This allows us to examine how any smart contract interacts with the EVM, which is crucial for most security tools since they need to study those interactions in-depth and find vulnerabilities. And that does not

exclude the tool we used for this research, ItyFuzz [67].

4.2.1 How to extract on-chain data?

Nodes

Ethereum or any other blockchain is hosted on different Nodes across the globe. Essentially, these nodes are computers that participate in the process of validating transactions. They also store transaction history and all other information on the blockchain. In Ethereum, there are three types of Nodes: Full-Node, Archive-Node, and Light Node. Only two of them, Full-Node and Archive-Node, participate in the consensus process to validate transactions emitted in the network.

Full-Node maintains a complete copy of the current state of the Ethereum blockchain. It stores recent transactions, contract states, and the latest block information. However, storage requirement is lower, since it can discard historical data and only store the most recent transactions. Full nodes are suitable for most applications where access to recent blockchain data is sufficient, such as transaction verification, smart contract interactions, and monitoring recent network activity.

An **Archive-Node** possesses most of the functionalities a Full-Node has. However, they consume significantly more storage space than full nodes because they keep a record of all historical states. The storage requirement grows over time as more blocks are added to the blockchain. Archive-Nodes are most suitable for research or audit purposes

Remote Procedure Call (RPC) Providers

Running a full or archive node for blockchain networks such as Ethereum and Binance Smart Chain (BSC) involves considerable effort and resources. For a full node, which stores the entire transaction history and validates transactions, a system with at least 16GB of RAM and 1TB of storage is typically required. An archive node, which maintains the complete historical state of the blockchain, demands even more substantial resources, ranging from 32GB to 64GB of RAM and 3TB to 12TB of storage [68]. Due to that fact, individual researchers or developers often turn to RPC providers such as Alchemy, Infura, QuickNode, or MetaMask, who take care of operating Nodes and offer users public RPC endpoints that can be used to interact with the blockchain or fetch data from it. Usually, there is a programming library being used to make HTTP calls to those RPC endpoints, refer to figure 8.

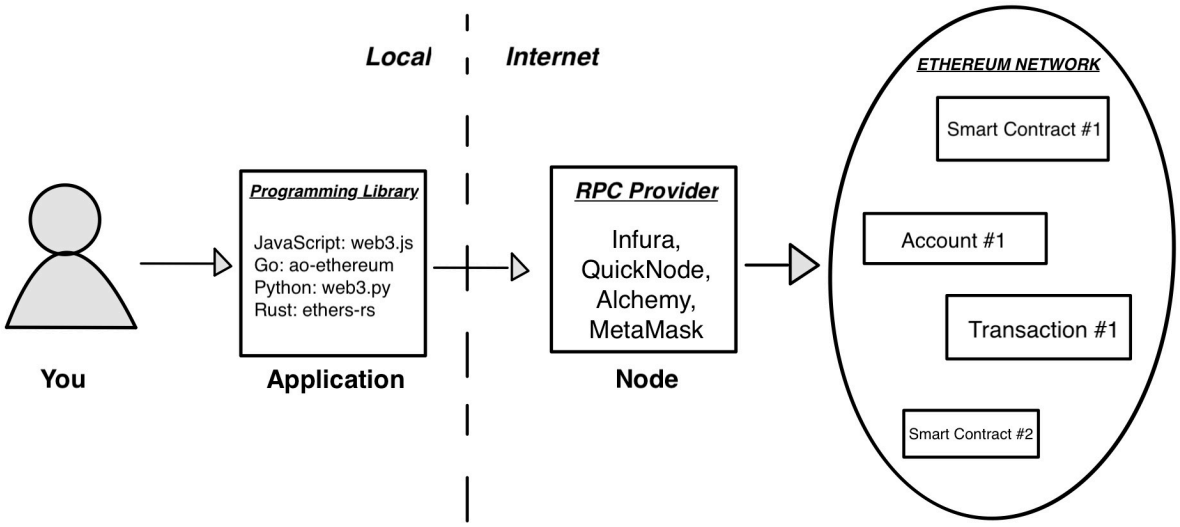

Figure 8: Interaction with Ethereum

Blockchain Explorer's Public API

In addition to obtaining data directly from Ethereum and Binance Smart Chain (BSC) nodes, researchers often leverage blockchain explorers that provide public APIs. These APIs offer convenient access to crucial blockchain information without the need to run a local node. In the scope of this research, three widely utilized blockchain explorers, Bscscan [1] for Binance Smart Chain, Etherscan[2] for Ethereum, and more universal tool, Phalcon [3], have been chosen for their straightforward interfaces and reliability.

4.3 Data Collection

The original dataset [11] has been used and extended by several researchers [45, 69] over the years. Despite this, due to the unnecessary complexity of those studies, we have decided to extend only from the original dataset to simplify the process. The collection process is mostly manual. However, every transaction of the related incidents is available on the Blockchain. They are mostly not flagged or there is little to no further information regarding the incidents except for the information mentioned above. Therefore, we had to heavily rely on these sources: chainsec [70], slowmist [71], rektnew [72]

In line with the original dataset, we focus on collecting incidents that occur on Ethereum as well as BSC. We also collected data from Twitter accounts of blockchain audit firms using

1 https://bscscan.com/
2 https://etherscan.io/
3 https://explorer.phalcon.xyz/

specific keywords related to the incidents such as the protocol name. There will be an exclusion from the dataset for all incidents that are scams, rug pulls or incidents lacking an obvious method of exploiting the vulnerability since they are not technically relevant to this dataset. These incidents do not contain the technical details necessary to understand the severity of a vulnerability and provide useful insights for our research. By excluding them from this dataset, we were better able to focus on the technical aspects of the vulnerability.

For each incident, we aim to address the following inquiries whenever the relevant information is accessible:

- Identification of the victim.
- Timestamp of the incident occurrence.
- Blockchain on which the incident transpired (Ethereum/BSC).
- Identification and addresses of the targeted vulnerable contracts.
- Sources that reported these incidents.
- The total USD amount exploited.
- The malicious contracts and their addresses used to target the victim

As part of our decision to use SQLite [73] to be our main database management system, we chose to use this engine due to its reliability, speed, scalability, and ease of use. SQLite is also free and open source, making it an ideal choice for our project. SQLite operates as a serverless library, storing databases in a single file and it supports SQL as it is a specific implementation of a relational database management system.

We have created a visual representation, Figure 9, depicted in the form of a figure, to illustrate the interrelationships among the various tables within the database. This graphical depiction serves to elucidate the connections and associations between different elements, providing a comprehensive overview of the relational structure within the database.

Similar to the original dataset, the extended version possesses six tables, the primary key of which is bolded and underlined in the figure. To align with the research's temporal scope of the incidents, an extra table has been added to the dataset structure. The additional table specifically focuses on gathering details about adversarial contracts involved in each incident. This inclusion enriches the database by providing insights into the characteristics and attributes of these contracts within the context of different occurrences. Furthermore, it allows us to automate the process of calculating the rescue time frame of the incidents.

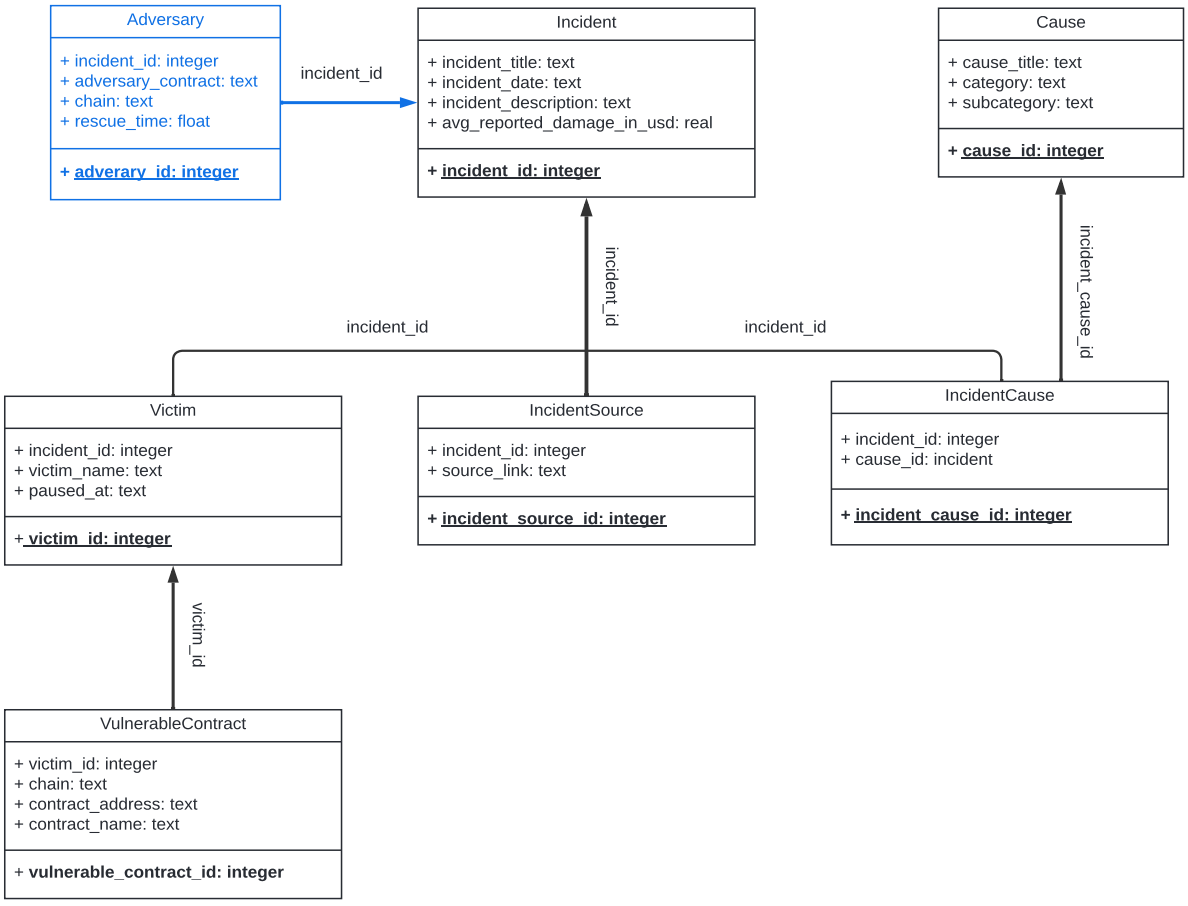

Figure 9: Database strucutre

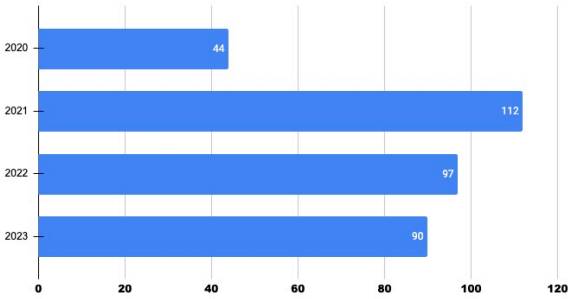

Figure 10: Number of attacks over the years

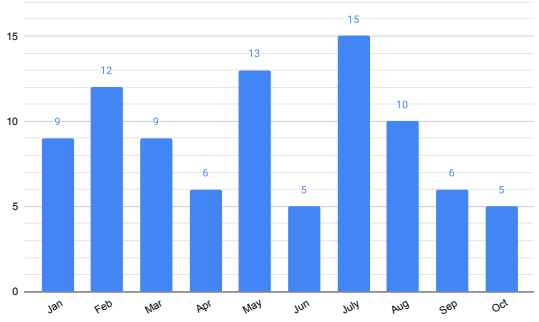

Figure 11: Attack Frequency from Jan to Oct 2023

Following are the short descriptions of the dataset's structure:

1. **Incident:** describe general information about the incident and reported damage in USD
2. **IncidentSource:** linked to Incident by **incident_id** to cite the sources that reported the incidents
3. **IncidentCause:** linking table between Incident and Cause by **incident_id** and **cause_id**
4. **Cause:** linked to IncidentCause by **cause_id** to map incidents by defined categories
5. **Victim:** linked to Incident by **incident_id** to map victims to each incidents
6. **VulnerableContract:** linked to Victim by **victim_id** to map Victim to its exploited contract (if existed)
7. **Adversary:** linked to Incident by **incident_id** to map incident to its adversarial smart contract (if existed)

After the process, we managed to add 147 new entries within the time frame of 1st May to 30th October to the dataset, which resulted in over one billion USD lost. At the same time, 59 incidents were excluded, since they were simply rug-pulls, scams, or incidents with unknown methods of exploitation. 62/147 (42.17%) of the accepted incidents are on BSC, while 85/147 (57.83%) others occurred on the Ethereum blockchain.

4.4 Analysis

This section dives into an analysis of DeFi incidents spanning from May 2022 to October 2023. Figures 11 and 10 have been included to provide visual representations of the numerical data presented in the dataset. By examining total losses and Total Value Locked (TVL), we aim to glean insights into vulnerabilities, risks, and consequences, offering a nuanced perspective on the challenges and opportunities within the DeFi space. It is important to notice that there can be multiple causes for each incident.

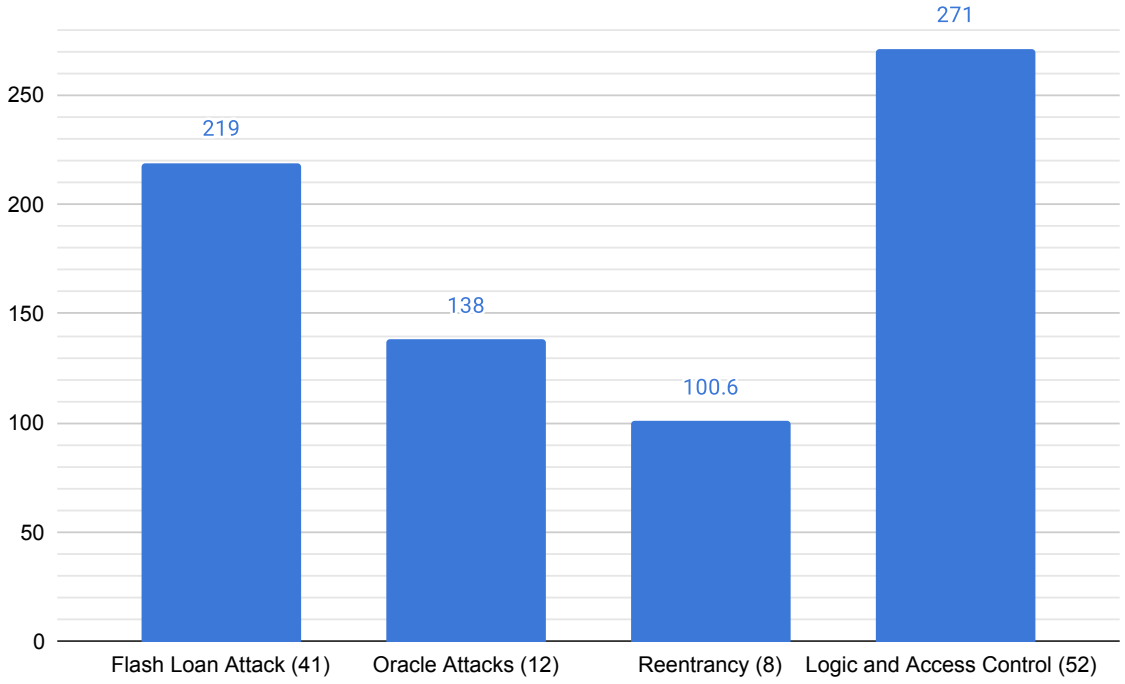

Figure 12: Total USD Loss of Common Attack Vectors

While flash loans 2.2.3 initially seemed like a groundbreaking concept, they also brought hidden dangers. Flash loans allow anyone willing to pay the gas fee and service fee to borrow money without having to put up any assets as collateral. This opens up big opportunities for quick and smart trading moves, but it also leads to potential problems 2.3.3. As depicted in Figure 12, we have collected 41 incidents using Flash Loan. These attacks resulted in a total loss of 219 million USD. Attacks on Oracles also resulted in great losses, 138 million USD after just 12 Incidents. Besides that, exploits targeted common smart contract vulnerabilities (Reentrancy 2.3.1, integer overflow 2.3.1, logic error 2.3.1, and access control 2.3.1) are amongst the most common, as they were responsible for 65 out of 147 (44.2%) incidents from the extended entries of the dataset and exploited 271 million USD. Due to the intertwinement of DeFi protocols, it is challenging to spot possible malicious flash loan combinations. However, with the available auditing tool on the market, it should be more straightforward to detect logical flaws or code that introduces reentrancy vulnerability to the smart contract. Therefore, greater emphasis should be placed on auditing processes to uncover these logical inadequacies before deployment.

As we can observe from Figure 13, while certain months, like September 2023 with losses amounting to 195,000 USD, experience minimal impact, other months witness significant spikes in losses, August 2022 (200,7 million USD loss) and March 2023 (208,8 million USD loss). These are the result of only a few high-damage incidents targeted protocol with the highest TVL in the market. For instance, the incident of Nomad Bridge on August 2022 [74]

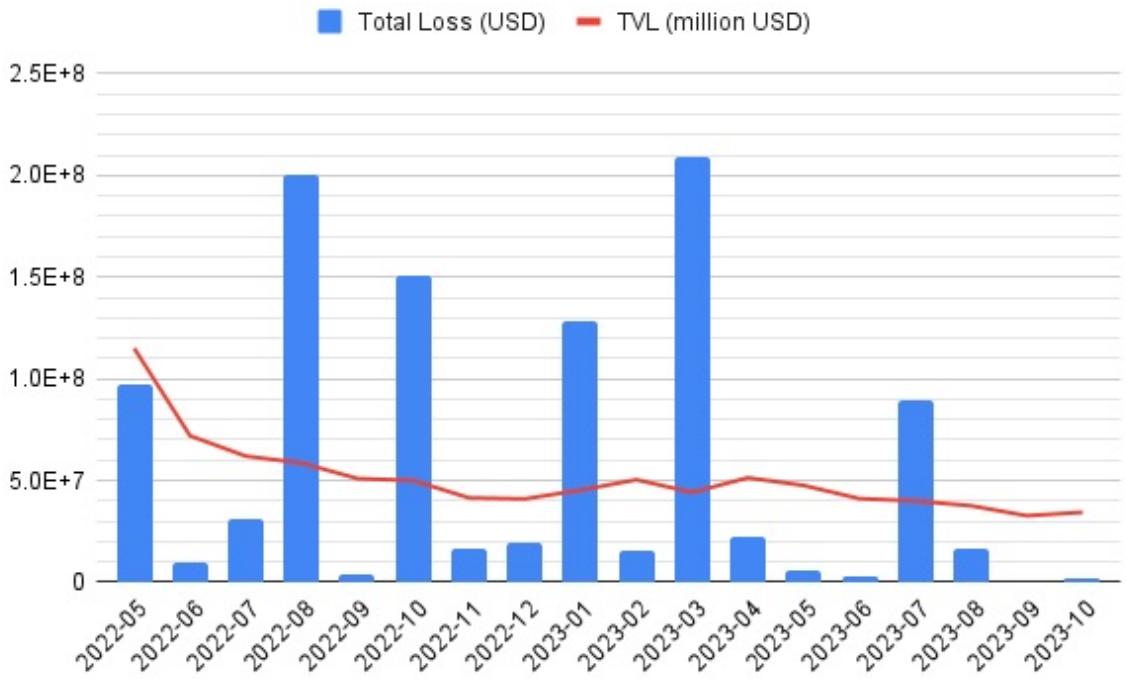

Figure 13: Total Loss in USD and TVL (May 2022 - Oct 2023)

took 200 million USD due to a logic flaw that happened only during a routine upgrade. The most substantial loss rooted in very common attack vectors, the flash loan attack 2.3.3 on Euler Finance [75] took 197 million USD, the double attack on Rari Capital and Fei Protocol [76] based on reentrancy 2.3.1 resulted in an 80 million USD loss, and 29 million USD hack on TransitSwap was due to a lack of logic 2.3.1 in smart contract code.

4.5 Limitations

Our data collection method is human error-prone because we had to manually check, and assess every single entry. During the process of preparing the dataset, typos, misinterpretations, and inaccuracies may be introduced, resulting in the overall accuracy of the dataset being compromised. We try to mitigate this problem by organizing several rounds of data review. In these rounds, we skim through the dataset, check for typographical errors, and ensure that the data is aligned with external sources to detect errors.

Furthermore, for some cases, we also double-check the incident transaction on block explorers to make sure the incidents on the same protocol are not mixed up. Besides that, manual data collection is a tedious task. It requires significant effort and resources to gather, verify, and organize the information from diverse sources. As a result, scalability challenges arise because

there are always new incidents every week or month, and it requires additional effort to keep up with the most recent developments.

5

Are Smart Contract Attack Strategies optimized?

5.1 Navigating Security Challenges in DeFi Protocols

Decentralized Finance is a space where the protocols are interconnected in a complex web, creating opportunities and challenges within the ecosystem. This intertwining can lead to vulnerabilities or exploits in one protocol having ripple effects on others. Even when the adversary targets only one vulnerable protocol, it requires serious effort and technical knowledge to perform these types of exploits.

The flash loan attack 2.3.3 through this transaction 0x897c2d against protocol PancakeBunny is one of the examples. Flash Loan is not a feature, which is widely used by common users as it involves conducting advanced financial transactions on the blockchain by writing code, interacting with smart contracts, and understanding the intricacies of DeFi protocols. The exploit unfolded as follows: 1) the adversary performed a flash loan to borrow more than 700 million USD BNB from 8 and 3 million USDT. 2) used the borrowed funds to deposit into PancakeBunny's BNB-USDT Liquidity pool, and manipulated the price of USDT/BNB and Bunny/BNB, which allowed the minting of 7 million LP tokens. 3) dumped all LP tokens in the market to plummet LP token's price. 4) repaid the flash loan and earned the remaining BNB.

As difficult as it is for malicious actors to execute these exploits, detecting such vulnerabilities during the auditing process also poses a considerable challenge. The EVM is a stateful machine, and there are parts of Smart Contract code only be triggered in a certain state, emphasizing the importance of fetching real-time on-chain Ethereum state for accurate security assessments

with tools. While there are tools available [60, 77, 78] to target this problem, questions persist regarding their practicality and effectiveness in identifying and addressing these vulnerabilities. For this research, we decided to leverage a new SoTA fuzzing tool, ItyFuzz [62], to answer a question: "If an attack could have been more damaging financially?" and according to chapter 6, we need to answer that question fast.

5.2 Smart Contract Fuzz Testing

Fuzz testing is a well-known and common testing technique in software development[79, 80]. Fuzz testing, particularly in the context of smart contracts, is an advanced testing technique aimed at identifying vulnerabilities and weaknesses in software code. In the realm of blockchain and decentralized finance (DeFi), smart contract fuzzing becomes crucial for ensuring the robustness and security of decentralized applications (DApps). This method involves subjecting a smart contract to a barrage of random or semi-random inputs to uncover potential bugs, vulnerabilities, or unforeseen edge cases. By simulating various scenarios, fuzzing helps developers and security professionals detect and address security issues before deploying smart contracts to production environments.

5.3 ItyFuzz

ItyFuzz is an open-source [1] Bytecode-Level Hybrid Fuzzer for Smart Contracts. It was built with efficiency and speed in mind, and according to Shou et al., [62], existing sequence-based smart contract fuzzing tools face challenges exploiting certain vulnerabilities due to the exponential re-execution time. ItyFuzz employs an innovative snapshot-based approach to address this issue, eliminating re-execution time by memoizing noteworthy states. This not only reduces fuzzing time but also enhances overall efficiency.

ItyFuzz starts with a corpus of seed inputs, users can specify the on-chain block from which, they want to start the fuzz, from which ItyFuzz would pull state information. Leverage snapshot-based fuzzing and based on the bytecode of the contracts, addresses of which one gives to fuzz argument, it mutates a transaction and state pair from the corpus and executes the mutated input pair on the Ethereum Virtual Machine (EVM). ItyFuzz uses the execution waypoints, explored by Rohan et. al [81] to decide if the input lead to the trace collected during the execution is "interesting". The input is deemed interesting when it covers a new area of the uncovered code in the fuzzing session. Once a violation/vulnerability is found, the fuzzer stops and gives a trace back to how it reached this vulnerability.

Following are the violations, we have been able to observe on ItyFuzz during this research:

1 https://github.com/fuzzland/ityfuzz

- Imbalanced Pair: a violation that breaks the token pair invariant pair 2.1 when swapping on UniSwap
- Arbitrary Call: the vulnerability that allows arbitrary calls to burn/mint tokens
- Profited: Most interesting for this research, this feature detects possible monetary exploits.

5.4 Research Design

For our research, we employed a machine equipped with a 2.6 GHz 6-Core Intel Core i7 processor and 32 GB RAM. From our database, we randomly selected 35 incidents, distributed between 20 on Binance Smart Chain and 15 on Ethereum. To enhance the efficiency of ItyFuzz, we've established an archive node on Infura [2], enabling it to fetch on-chain state data before initiating the fuzzing process. To determine the interesting addresses, we used blockchain the explorers [3] to refer to the attack's transaction hash and the open-source repository [4]. Interesting contracts are usually, vulnerable contracts, involved protocol contracts, and involved token contracts.

Using the attack transaction hash, we pinpointed the precise block number where the adversary initiated the hack and inputted it into ItyFuzz for analysis (Exact Block exploit). Moreover, to support the research, we present a proposition centered on monetary exploits: If we utilize ItyFuzz on the blocks before and after the block where the actual hack occurred and find no positive results concerning monetary value, we will deduce that the initial incident was optimized. We decided to fuzz the smart contracts 12 hours before, and 12 hours after the actual incident. Additionally, we implemented a one-hour timeout for the automated script.

Below is the script we implemented to automate the fuzzing process, monitor errors, and organize the fuzzing results. We aggregated all fuzzing commands into a JSON file to input into the script:

Listing 5.1: Automation script for ItyFuzz

```
const fs = require("fs");
const util = require("util");
const exec = util.promisify(require("child_process").exec);
const fsExtra = require("fs-extra");

//extract data from Incident JSON
const jsonString = fs.readFileSync("incident.json", "utf8");
const tasks = JSON.parse(jsonString);

const errorLogFile = "error_log.txt";

async function executeItyfuzzCommandWithTimeout(
```

2 https://www.infura.io/
3 https://explorer.phalcon.xyz/
4 https://github.com/SunWeb3Sec/DeFiHackLabs

```
  command,
  timeout,
  executionName,
  id
) {
  // Set limit for buffer size
  const maxBuffer = 1024 * 1024 * 50;

  const timeoutPromise = new Promise((_, reject) => {
    setTimeout(() => {
      reject(new Error(`Command execution for ${executionName} timed out`));
      fs.promises.appendFile(
        errorLogFile,
        `${id} : ${executionName} : TIMEOUT\n`
      );
    }, timeout * 1000);
  });

  const executionPromise = new Promise(async (resolve, reject) => {

    try {
      const {stdout, stderr} = await exec(command, {maxBuffer});

      if (fs.existsSync("work_dir")) {
        const destinationFolder = `result_test/${executionName}`;
        fsExtra.copySync("work_dir", destinationFolder);
        fsExtra.removeSync("work_dir");
      } else {
        console.log('"work_dir" folder does not exist.');
      }

      resolve();
    } catch (error) {
      await fs.promises.appendFile(
        errorLogFile,
        `${executionName} : ${error.message}\n`
      );
      if (fs.existsSync("work_dir")) {
        fsExtra.removeSync("work_dir");
      } else {
        console.log('"work_dir" folder does not exist.');
      }
    }
  });

  return Promise.race([executionPromise, timeoutPromise]);
}

async function runTasksWithTimeout(timeout) {
// run through all entries
  for (const task of tasks) {
    try {
      await executeItyfuzzCommandWithTimeout(
        task.fuzzing_command,
        timeout,
        task.name,
        task.id
      );
    } catch (error) {
      console.error("Error executing task:", error.message);
    }
    console.log("\n");
  }
}

// Set time-out for one hour
const timeout = 3600;
runTasksWithTimeout(timeout);
```

5.5 Results

As the result, see 5.2 and 5.1, on earlier block fuzzing, we received 2/15 on Ethereum and 0/20 on BNB Smart Chain positive results that point out the monetary exploits. These are OlympusDao attack on 21st Oct. 2022 and DFXFinance attack on 11th Nov. 2022. Exact block fuzzing on EVM returned the same violation with DFXFinance. In addition, the fuzz execution on the involved contracts in the RevertFinance incident also returned monetary exploits on UniSwap V3: Position NFT 0xc3644 contract which could have caused users to transfer all of their balance to the caller. This exploit is very similar to the actual attack which exploited the new "atomic swap into v3 position" feature, where all allowance to the exploited contract could be drained [82]. Despite the introduction of a new feature on RevertFinance several weeks before the incident, ItyFuzz failed to identify the vulnerability when executed on both earlier and later blocks. This indicates that there may be factors related to the state of the Ethereum Virtual Machine (EVM) affecting ItyFuzz's coverage and concealing the vulnerability. Our inference is that the RevertFinance incident requires execution on the specific block number 16653390 to successfully exploit the vulnerable contract and the amount of exploited value given by ItyFuzz suggests that the original attack is optimized.

About the Axioma incidents, ItyFuzz indicates that in the contract with address 0x2c25a (AxiomaPresale), there is a vulnerability from which more than 33000 USD could have been exploited instead of just 7000 USD like the original incident. The fuzz result suggests, the adversary could have drained the balance of AxiomaPresale contract and used the ***swapExactTokensForETHSupportingFeeOnTransferTokens()*** function to swap for around 18.14 WETH (slippage already calculated). The vulnerability being exploited by ItyFuzz is very similar to the original incident [83], however a change in the parameters (amount of Presale Token exploited) results in a much bigger exploit. This can be achieved by borrowing a larger amount of funds from a flash loan than the adversary originally has done (32.5 WBNB).

Finally, in cases where the fuzzing attempts did not result in monetary exploits during both earlier and later block fuzzing, we interpret this as an indication that the original attack for each incident had already been optimized. Including the incidents described above, there are in total 6/35 suboptimal strategies from both BSC and Ethereum.

Incident	Earlier Block	Exact Block	Later Block
IndexedFinance-20211014	timed out	timed out	timed out
RevertFinance-20220327	timed out	profited	timed out
Xave-20221009	timed out	timed out	timed out
TempleDao-20221011	timed out	timed out	timed out
OlympusDao-20221021	profited	timed out	timed out
TeamFinance-20221027	timed out	timed out	timed out
BrahTOPG-20221110	timed out	timed out	timed out
DFXFinance-20221111	profited	profited	arbitrary call
RoeFinance-20230112	timed out	timed out	timed out
CowSwap-20230207	arbitrary call	arbitrary call	arbitrary call
Dexible-20230217	arbitrary call	arbitrary call	timed out
Euler-20230313	timed out	timed out	timed out
SushiSwap-20230409	timed out	timed out	timed out
Arcadia-20230710	timed out	timed out	timed out
EarningFarm-20230809	timed out	timed out	timed out

Table 5.1: Ethereum Incidents Fuzz Result

Incident	Earlier Block	Exact Block	Later Block
ValueDefi-20210507	timed out	timed out	timed out
PancakeHunny-20210603	timed out	timed out	timed out
CFTToken-20220411	timed out	imbalanced pair	timed out
Novo-20220530	imbalanced pair	arbitrary call	imbalanced pair
LPC-20220725	timed out	timed out	timed out
EgdFinance-20220808	timed out	profited	profited
ShadowFinance-20220902	imbalanced pair	imbalanced pair	timed out
PLTD-20221017	imbalanced pair	imbalanced pair	imbalanced pair
SheepFarm-20221116	imbalanced pair	imbalanced pair	imbalanced pair
LaunchZone-20230227	arbitrary call	arbitrary call	arbitrary call
BigFinance-20230422	timed out	timed out	timed out
MetaPoint-20230412	timed out	profited	timed out
Axioma-20230424	timed out	profited	timed out
CSToken-20230524	timed out	imbalanced pair	timed out
ARA-20230619	arbitrary call	arbitrary call	arbitrary call
ApeDAO-20230718	timed out	arbitrary call	arbitrary call
WGPT-20230712	timed out	timed out	profited
Palmswap-20230725	imbalanced pair	imbalanced pair	imbalanced pair
Linear-20230921	timed out	timed out	timed out
MicDAO-20231019	timed out	timed out	timed out

Table 5.2: BNB Smart Chain Incidents Fuzz Result

6

Is Rescue Time Frame shrinked?

6.1 Rescue Time Frame of DeFi Incidents

According to our newly collected dataset, many of the attacks on Smart Contracts (around 90%) were not executed in one transaction due to attack complexity or gas limit constraints which disallow too computational-heavy strategies to be executed in the same transactions [11]. This grants the defender a reaction time which was mentioned by Zhou et. al as **rescue time frame**. Rescue time is defined as the time frame between the successful malicious smart contract deployment and its first completed malicious transaction. This critical metric plays a pivotal role in comprehending the diverse spectrum of strategies available for safeguarding DeFi Protocols against the impacts of malicious contracts and transactions.

6.2 Data Collection

As mentioned in chapter 4, we have added one new table "Adversary" to the original data structure to help store information about the adversarial contracts responsible for each incident. The process of collecting contract information is also a manual job. In the end, we were able to collect 76 malicious contracts whose attack vectors are the common attack vectors mentioned in chapter 4, Flash loan 2.3.3, Reentrancy 2.3.1, Oracle 2.3.2, and Logic/Access Control vulnerability 2.3.1.

We leverage both the BSC Scan API and the Etherscan API for our analysis, as utilizing an archive node directly proves to be less straightforward for our specific requirements. Two API endpoints from BSC Scan were used to conduct this analysis: **GetContractCreatorAndCreationTxHash** [1] and **GetInternalTransactionsbyTransactionHash** [2]. Etherscan API's

1 https://docs.bscscan.com/api-endpoints/contracts
2 https://docs.bscscan.com/api-endpoints/accounts

endpoints are almost identical to BSC Scan.

We calculate the rescue timeframe against each malicious contract based on the following simple formula:

$$(\#\text{blockOfFirstTRX} - \#\text{blockOfContractCreation}) \times \text{blockTime} \tag{6.3}$$

in which, the block time of Ethereum is on average 15 seconds, while that of BNB is 3 seconds. To retrieve and calculate the necessary data for the analysis, we implemented the **saveRescueTime** 6.2 which calls another external function **getRescueTimeBlocks** that retrieves the contract creation block and first transaction block by calling BSC Scan endpoints mentioned above. In the end, we save all results to a JSON file and populate the Adversary table with rescue times.

Listing 6.1: Rescue Time on BSC Calculation

```
const fs = require("fs");
// external function to call APIs
const {
  getFirstTransactionAfterCreation,
} = require("./getTransationsBlockWithAddress");

// get contract faddress from the NoSQL Database
const getAddressByChainType = require("./getAddressFromDB");

async function saveRescueTime() {
  try {
  // select addresses on BNB Smart Chain
    const addressArray = await getAddressByChainType("BSC");

    let rescueTimeArray = [];

    // calculate rescue time on each address
    for (let i = 0; i < addressArray.length; i++) {
      try {
        const result = await
    ↪ getFirstTransactionAfterCreation(addressArray[i]);

        rescueTimeArray.push({
          address: addressArray[i],
          rescueTime:
            (result.firstTransactionAfterCreation.blockNumber -
              result.creationTransaction.blockNumber) *
            3,
        });
      } catch (error) {
        console.error(`Error for ${addressArray[i]}: ${error.message}`);
      }
    }

    // Write the rescueTimeArray to a JSON file
    const jsonContent = JSON.stringify(rescueTimeArray, null, 2);
    fs.writeFileSync("./rescueTimeArray.json", jsonContent);
      } catch (error) {
    console.error("Error:", error);
  }
}

saveRescueTime();
```

6.3 Analysis

Based on our dataset, we have calculated the mean rescue time is approximately 2.4 hours. As demonstrated in Figure 14, we can see that there are spikes in rescue time in April 2023 and July 2023 suggesting that the attack took significantly longer to execute than the rest. This raises a question: which incidents caused that spike? And what is the vector the adversary used?

To answer that, we conducted a further investigation on the most common attack vectors mentioned in the Background 2. We used our dataset to calculate 90% confidence intervals for each type of attack vector and received the following rescue time frame: i) Flash Loan Attacks: 1.13±1.3 hours with the highest rescue time frame of 2.75 hours, ii) Reentrancy: 0.006±0.004 hours with the highest rescue time frame of: 0.024 hours, iii) Logic/Access Control: 0.9 ± 1.29 hours with the highest rescue time frame of: 12.6 hours, and iv) Oracle Attacks: 5.7 ± 9.4 hours with the highest rescue time frame of 51.8 hours, refer to Figure 15.
These findings underscore that Oracle Attacks tend to take longer to resolve compared to other attack vectors, emphasizing the need for robust defenses against such prolonged threats.

Before May 2022, the rescue time frame for incidents on the Protocol Layer (1±4.1 hours) was significantly shorter than the rescue time frame for incidents on the Smart Contract Layer (14.4±65 hours) [11]. In our research on the extended dataset, from May 2022 to October 2023, the combined rescue time frame (2.4±2.3 hours) is narrower than both individual rescue time frames from the earlier period. This suggests that, during this time frame, the rescue times for incidents on both layers became more consistent and less variable. Given these observations, it is reasonable to conclude that the rescue time frame from May 2022 to October 2023 is, on average, shorter and more consistent than it was before May 2022.

6.3.1 Existing defending strategy

First of all, malicious smart contracts should be detected. Addressing specific concerns, Siwei Wu et al. introduced an oracle solution with defiRanger to detect price manipulation [84]. Additionally, Forta Network [66] employs machine learning models to identify and flag malicious smart contracts, enhancing security measures in the DeFi space.

Leveraging the rescue time frame as a key metric, an effective strategy to thwart potential attacks involves front-running. Researchers including Qui et al. introduced APE, a tool to imitate and counteract potential attacks [85]. Similarly, Xun Deng et al. proposed FrontDef [86], a technique designed for front-running and proactive defense in blockchain environments. These approaches contribute to the arsenal of tools used for safeguarding against emerging threats in the blockchain space.

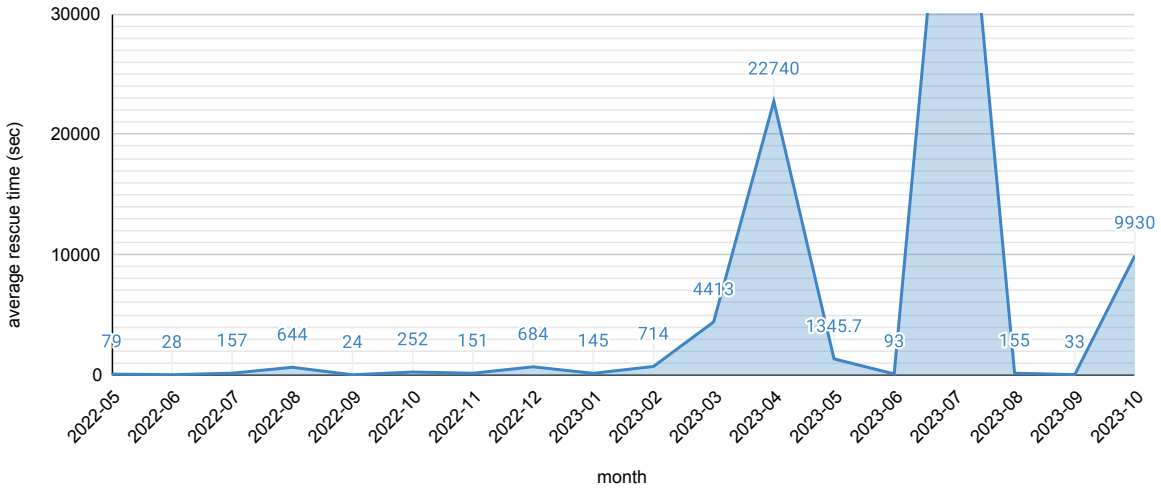

Figure 14: Average rescue-time by month

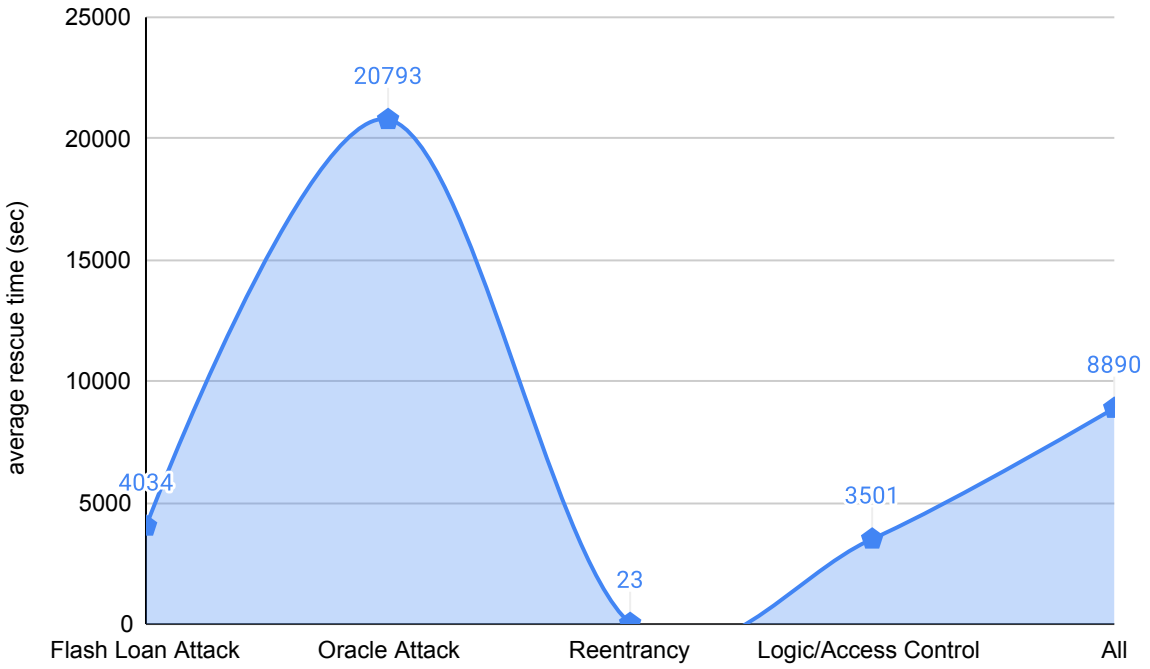

Figure 15: Rescue-time by Attack Vector

7

Conclusion

7.1 Summary of Contributions

Our research mainly focused on exploring the more recent DeFi Incidents, in which we have collected more than 140 incident data and expanded the original dataset from zhou et. al [11]. Moreover, we also add to the original dataset a new collection of malicious smart contracts involved in the incidents. This new collection allows us to shed light on the rescue time frame for the defending party. Our research shows that the rescue time frame has been shrinking recently to around 2.4±2.3 hours, instead of 1±4.1 hours for Protocol Layer incidents and 14.4±65 hours for Smart Contract Layer incidents. This grants the defending party less time to stop or interfere with the attack. For that, we also suggest a few possibilities for the defending party to handle the short rescue time frame. Finally, we leverage the SoTA fuzzing tool, ItyFuzz, to determine the optimality of recent incidents regarding monetary exploitation. We found out, out of the incidents we picked from our dataset, 29/35 attack strategies were already optimized.

7.2 Limitation

The data collection process was performed manually, introducing the potential for errors, biases, and inaccuracies. Human involvement may lead to subjective interpretations and the unintentional inclusion of incorrect information, impacting the overall reliability of the dataset. the accuracy of the new data is pivotal as it forms the cornerstone of our research. Inaccuracies in data entries have a cascading effect, potentially magnifying errors and undermining the reliability of subsequent analyses. Ensuring precision in data collection is paramount to maintaining the integrity and validity of our research findings.

Furthermore, The fuzzing process employed a relatively short timeout, and the automated

script was implemented using JavaScript. This may have affected the thoroughness and performance of the fuzzing, as a longer timeout could have allowed for more in-depth exploration of potential vulnerabilities.

Additionally, the choice of JavaScript for automation may have introduced limitations in terms of speed and efficiency compared to more efficient languages. Eventually, the research was conducted on a machine with specific hardware specifications, including a 2.6 GHz 6-Core Intel Core i7 Processor and 32 GB RAM. While this setup was suitable for the research, using a more powerful machine could potentially enhance the speed of the fuzzing process, allowing for broader coverage of the corpus within the same time frame.

7.3 Future Direction

In our future efforts, we aim to enhance ItyFuzz's capabilities beyond merely identifying bugs by extending its functionality to potentially generate monetary exploits for every type of bug it detects. This expansion aims to provide a more comprehensive assessment of vulnerabilities, going beyond mere identification to understand their potential financial implications.

Additionally, we plan to broaden the scope of our dataset by further extending and refining it. This involves incorporating data from additional sources to create a more diverse and comprehensive dataset. By expanding the dataset, we seek to improve the robustness and relevance of our research, enabling a more comprehensive understanding of potential vulnerabilities in the DeFi landscape.

Finally, our research underscores the need for an effective front-run tool in DeFi. Collaboration between academia and industry is essential to bridge the gap between theoretical advancements and practical implementation. This synergy can accelerate the adoption of innovative tools, enhancing the security of DeFi ecosystems.

List of Figures

Listings

List of Tables

Bibliography

[1] "Sushi." https://www.sushi.com/. (Accessed on 11/09/2023). cited on p. 1

[2] "Home — uniswap protocol." https://uniswap.org/. (Accessed on 11/09/2023). cited on p. 1

[3] "Compound." https://compound.finance/. (Accessed on 11/09/2023). cited on p. 1, 13, 14

[4] "Defi flash loans made easy." https://equalizer.finance/. (Accessed on 11/09/2023). cited on p. 1

[5] "Lido - liquid staking for digital tokens." https://lido.fi/. (Accessed on 11/10/2023). cited on p. 1

[6] "Ankr staking." https://www.ankr.com/staking/stake/. (Accessed on 11/10/2023). cited on p. 1

[7] "dydx - trade perpetuals on the most powerful trading platform." https://dydx.exchange/. (Accessed on 11/09/2023). cited on p. 1

[8] "Synthetix." https://synthetix.io/. (Accessed on 11/09/2023). cited on p. 1

[9] "Home — ethereum.org." https://ethereum.org/en/. (Accessed on 11/10/2023). cited on p. 1

[10] "Bnb smart chain: A parallel bnb chain to enable smart contracts." https://www.bnbchain.org/en/smartChain. (Accessed on 11/10/2023). cited on p. 1

[11] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, "Sok: Decentralized finance (defi) attacks," 2023. cited on p. 2, 14, 19, 21, 24, 37, 39, 41

[12] S. Nakamoto, "bitcoin.pdf." https://bitcoin.org/bitcoin.pdf. (Accessed on 11/13/2023). cited on p. 3

[13] T. T., "ethereum-evm-illustrated." https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf. (Accessed on 11/13/2023). cited on p. 3, 8, 43

[14] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform.." https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf. (Accessed on 11/13/2023). cited on p. 4

[15] "Ethereum virtual machine (evm) exceptions – ethereum specification documentation." https://ethereum.github.io/execution-specs/diffs/constantinople_istanbul/vm/exceptions/index.html. (Accessed on 11/13/2023). cited on p. 4

[16] "Smart contracts." https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html. (Accessed on 11/13/2023). cited on p. 5

[17] "Using the compiler – solidity 0.8.17 documentation." https://docs.soliditylang.org/en/v0.8.17/using-the-compiler.html. (Accessed on 11/14/2023). cited on p. 6

[18] "Evm codes - an ethereum virtual machine opcodes interactive reference." https://www.evm.codes/?fork=shanghai. (Accessed on 11/14/2023). cited on p. 6

[19] S. Panda, "An investigation into smart contract deployment on ethereum platform using web3.js and solidity using blockchain," 05 2021. cited on p. 6, 43

[20] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F.-Y. Wang, "An overview of smart contract: Architecture, applications, and future trends," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 108–113, 2018. cited on p. 6

[21] "Erc-20: Token standard." https://eips.ethereum.org/EIPS/eip-20. (Accessed on 12/02/2023). cited on p. 6

[22] "Lido: Curve liquidity farming pool contract — address 0xdc24316b9ae028f1497c275eb9192a3ea0f67022 — etherscan." https://etherscan.io/address/0xDC24316b9AE028F1497c275EB9192a3Ea0f67022. (Accessed on 11/14/2023). cited on p. 7

[23] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," 2019. cited on p. 8

[24] "Flashbots." https://www.flashbots.net/. (Accessed on 11/14/2023). cited on p. 8

[25] A. Berentsen and F. Schär, "Stablecoins: The quest for a low-volatility cryptocurrency," *The economics of Fintech and digital currencies*, pp. 65–75, 2019. cited on p. 10

[26] D. Bullmann, J. Klemm, and A. Pinna, "In search for stability in crypto-assets: are stablecoins the solution?," *Available at SSRN 3444847*, 2019. cited on p. 10

[27] J. Smith and M. Johnson, "The maker protocol: Makerdao's multi-collateral dai (mcd) system," *Decentralized Finance Journal*, vol. 42, no. 3, pp. 123–145, 2020. cited on p. 10

[28] S. Kazemian, J. Huan, J. Shomroni, and K. Iyer, "Frax: A fractional-algorithmic stablecoin protocol," in *2022 IEEE International Conference on Blockchain (Blockchain)*, pp. 406–411, 2022. cited on p. 11

[29] "The fall of terra: A timeline of the meteoric rise and crash of ust and luna." https://www.coindesk.com/learn/the-fall-of-terra-a-timeline-of-the-meteoric-rise-and-crash-of-ust-and-luna/. (Accessed on 11/25/2023). cited on p. 11

[30] R. Clements, "Built to fail: The inherent fragility of algorithmic stablecoins," *Wake Forest L. Rev. Online*, vol. 11, p. 131, 2021. cited on p. 11

[31] S. Fu, Q. Wang, J. Yu, and S. Chen, "Rational ponzi game in algorithmic stablecoin," in *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 1–6, 2023. cited on p. 11

[32] Y. C. Lo, "Uniswap and the emergence of the decentralized exchange," $https://papers.ssrn.com/sol3/papers.cfm?abstract_id = 3715398paper - citations - widget, 2021. citedonp.11$

[33] "Aave - open source liquidity protocol." https://aave.com/. (Accessed on 11/09/2023). cited on p. 13, 14

[34] "Home — internet computer." https://internetcomputer.org/. (Accessed on 11/27/2023). cited on p. 13

[35] "V3 overview - developers." https://docs.aave.com/developers/getting-started/readme. (Accessed on 11/27/2023). cited on p. 14

[36] "What is the blockchain oracle problem? l chainlink." https://chain.link/education-hub/oracle-problem. (Accessed on 11/26/2023). cited on p. 14

[37] "Merriam-webster: America's most trusted dictionary." https://www.merriam-webster.com/. (Accessed on 12/07/2023). cited on p. 14

[38] "Chainlink: The industry-standard web3 services platform." https://chain.link/. (Accessed on 11/26/2023). cited on p. 14

[39] S. Eskandari, M. Salehi, W. C. Gu, and J. Clark, "Sok: Oracles from the ground truth to market manipulation," in *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pp. 127–141, 2021. cited on p. 14

[40] "Band protocol." https://www.bandprotocol.com/. (Accessed on 11/26/2023). cited on p. 14

[41] "Switchboard." https://switchboard.xyz/. (Accessed on 11/26/2023). cited on p. 14

[42] "Makerdao — an unbiased global financial system." https://makerdao.com/en/. (Accessed on 11/27/2023). cited on p. 15

[43] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2084–2106, 2021. cited on p. 15

[44] R. Sujeetha and C. A. S. Deiva Preetha, "A literature survey on smart contract testing and analysis for smart contract based blockchain application development," in *2021 2nd International Conference on Smart Electronics and Communication (ICOSEC)*, pp. 378–385, 2021. cited on p. 15

[45] S. Chaliasos, M. A. Charalambous, L. Zhou, R. Galanopoulou, A. Gervais, D. Mitropoulos, and B. Livshits, "Smart contract and defi security: Insights from tool evaluations and practitioner surveys," *arXiv preprint arXiv:2304.02981*, 2023. cited on p. 15, 19, 24

[46] V. C. Bui, S. Wen, J. Yu, X. Xia, M. S. Haghighi, and Y. Xiang, "Evaluating upgradable smart contract," in *2021 IEEE International Conference on Blockchain (Blockchain)*, pp. 252–256, 2021. cited on p. 15

[47] "Reentrancy - ethereum smart contract best practices." https://consensys.github.io/smart-contract-best-practices/attacks/reentrancy/. (Accessed on 12/02/2023). cited on p. 16

[48] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "Reguard: Finding reentrancy bugs in smart contracts," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pp. 65–68, 2018. cited on p. 16

[49] B. Li, Z. Pan, and T. Hu, "Redefender: Detecting reentrancy vulnerabilities in smart contracts automatically," *IEEE Transactions on Reliability*, vol. 71, no. 2, pp. 984–999, 2022. cited on p. 16

[50] M. Mamoon, M. Saim, I. Shah, and A. Samad, "A decentralized byod authentication system secure against reentrancy attacks," in *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–7, 2023. cited on p. 16

[51] "openzeppelin-contracts/contracts/utils/reentrancyguard.sol at master · openzeppelin/openzeppelin-contracts." https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol. (Accessed on 12/02/2023). cited on p. 16

[52] P. Tantikul and S. Ngamsuriyaroj, "Exploring vulnerabilities in solidity smart contract.," in *ICISSP*, pp. 317–324, 2020. cited on p. 16

[53] "Types – solidity 0.8.24 documentation." https://docs.soliditylang.org/en/latest/types.html. (Accessed on 11/25/2023). cited on p. 16

[54] W. Li, J. Bu, X. Li, and X. Chen, "Security analysis of defi: Vulnerabilities, attacks and advances," in *2022 IEEE International Conference on Blockchain (Blockchain)*, pp. 488–493, IEEE, 2022. cited on p. 19

[55] W. Li, J. Bu, X. Li, H. Peng, Y. Niu, and Y. Zhang, "A survey of defi security: Challenges and opportunities," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 10, pp. 10378–10404, 2022. cited on p. 19

[56] J. Stephens, K. Ferles, B. Mariano, S. Lahiri, and I. Dillig, "Smartpulse: automated checking of temporal properties in smart contracts," in *2021 IEEE Symposium on Security and Privacy (SP)*, pp. 555–571, IEEE, 2021. cited on p. 19

[57] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, IEEE, May 2019. cited on p. 19

[58] A. Permenev, D. Dimitrov, P. Tsankov, D. Drachsler-Cohen, and M. Vechev, "Verx: Safety verification of smart contracts," in *2020 IEEE symposium on security and privacy (SP)*, pp. 1661–1677, IEEE, 2020. cited on p. 19

[59] G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, "Echidna: effective, usable, and fast fuzzing for smart contracts," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 557–560, 2020. cited on p. 19

[60] B. Jiang, Y. Liu, and W. K. Chan, "Contractfuzzer: Fuzzing smart contracts for vulnerability detection," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 259–269, 2018. cited on p. 19, 32

[61] V. Wüstholz and M. Christakis, "Harvey: A greybox fuzzer for smart contracts," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1398–1409, 2020. cited on p. 19

[62] C. Shou, S. Tan, and K. Sen, "Ityfuzz: Snapshot-based fuzzer for smart contract," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 322–333, 2023. cited on p. 19, 20, 32

[63] I. David, L. Zhou, K. Qin, D. Song, L. Cavallaro, and A. Gervais, "Do you still need a manual smart contract audit?," 2023. cited on p. 19

[64] Y. Gai, L. Zhou, K. Qin, D. Song, and A. Gervais, "Blockchain large language models," 2023. cited on p. 19

[65] Z. Liu, P. Qian, X. Wang, Y. Zhuang, L. Qiu, and X. Wang, "Combining graph neural networks with expert knowledge for smart contract vulnerability detection," *IEEE Transactions on Knowledge and Data Engineering*, 2021. cited on p. 19

[66] "Forta network." https://forta.org/. (Accessed on 11/27/2023). cited on p. 19, 39

[67] "fuzzland/ityfuzz: Blazing fast bytecode-level hybrid fuzzer for smart contracts." https://github.com/fuzzland/ityfuzz. (Accessed on 11/19/2023). cited on p. 23

[68] "Run a node — ethereum.org." https://ethereum.org/en/run-a-node/. (Accessed on 12/03/2023). cited on p. 23

[69] K. Qin, Z. Ye, Z. Wang, W. Li, L. Zhou, C. Zhang, D. Song, and A. Gervais, "Towards automated security analysis of smart contracts based on execution property graph," *arXiv preprint arXiv:2305.14046*, 2023. cited on p. 24

[70] "Comprehensive list of defi hacks & exploits - chainsec." https://chainsec.io/defi-hacks/. (Accessed on 11/19/2023). cited on p. 24

[71] "Slowmist hacked - slowmist zone." https://hacked.slowmist.io/. (Accessed on 11/19/2023). cited on p. 24

[72] "Rekt - home." https://rekt.news/. (Accessed on 11/19/2023). cited on p. 24

[73] "Sqlite home page." https://www.sqlite.org/index.html. (Accessed on 11/20/2023). cited on p. 25

[74] "Nomad crypto bridge loses $200 million in 'chaotic' hack - the verge." https://www.theverge.com/2022/8/2/23288785/nomad-bridge-200-million-chaotic-hack-smart-contract-cryptocurrency. (Accessed on 11/28/2023). cited on p. 28

[75] "Euler finance attack: How it happened, and what can be learned." https://cointelegraph.com/news/euler-finance-attack-how-it-happened-and-what-can-be-learned. (Accessed on 11/28/2023). cited on p. 29

[76] "Defi lender rari capital/fei loses $80m in hack." https://www.coindesk.com/business/2022/04/30/defi-lender-rari-capitalfei-loses-80m-in-hack/. (Accessed on 11/28/2023). cited on p. 29

[77] B. Wang, X. Yuan, L. Duan, H. Ma, C. Su, and W. Wang, "Defiscanner: Spotting defi attacks exploiting logic vulnerabilities on blockchain," *IEEE Transactions on Computational Social Systems*, 2022. cited on p. 32

[78] K. Qin, L. Zhou, B. Livshits, and A. Gervais, "Attacking the defi ecosystem with flash loans for fun and profit," 2021. cited on p. 32

[79] I. Andrianto, M. I. Liem, and Y. D. W. Asnar, "Web application fuzz testing," in *2017 International Conference on Data and Software Engineering (ICoDSE)*, pp. 1–6, IEEE, 2017. cited on p. 32

[80] J. Liang, M. Wang, Y. Chen, Y. Jiang, and R. Zhang, "Fuzz testing in practice: Obstacles and solutions," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 562–566, IEEE, 2018. cited on p. 32

[81] R. Padhye, C. Lemieux, K. Sen, L. Simon, and H. Vijayakumar, "Fuzzfactory: Domain-specific fuzzing with waypoints," *Proc. ACM Program. Lang.*, vol. 3, oct 2019. cited on p. 32

[82] "An attack on v3utils – revert." https://mirror.xyz/revertfinance.eth/3sdpQ3v9vEKiOjaHXUi3TdEfhleAXXlAEWeODrRHJtU. (Accessed on 12/02/2023). cited on p. 35

[83] "0x05eabbb665a5b99490 — phalcon explorer." https://explorer.phalcon.xyz/tx/bsc/0x05eabbb665a5b99490510d0b3f93565f394914294ab4d609895e525b43ff16f2. (Accessed on 12/02/2023). cited on p. 35

[84] S. Wu, D. Wang, J. He, Y. Zhou, L. Wu, X. Yuan, Q. He, and K. Ren, "Defiranger: Detecting price manipulation attacks on defi applications," 2021. cited on p. 39

[85] K. Qin, S. Chaliasos, L. Zhou, B. Livshits, D. Song, and A. Gervais, "The blockchain imitation game," pp. 3961–3978, 2023. cited on p. 39

[86] X. Deng, Z. Zhao, S. M. Beillahi, H. Du, C. Minwalla, K. Nelaturu, A. Veneris, and F. Long, "A robust front-running methodology for malicious flash- loan defi attacks," in *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pp. 38–47, 2023. cited on p. 39