INSTITUTO POLITÉCNICO
DE VIANA DO CASTELO

STATE OF WEB3 SECURITY

Analysis of vulnerabilities in bug bounty reports

Ana Rita Amorim Melo

Escola Superior de Tecnologia e Gestão

Instituto Politécnico
de Viana do Castelo

State of Web3 security

Analysis of vulnerabilities in bug bounty reports

Autor(a)

Ana Rita Amorim Melo

Trabalho efetuado sob a supervisão de

Professor Pedro Filipe Cruz Pinto

Professor António Alberto dos Santos Pinto

Mestrado em Cibersegurança

18 de Dezembro de 2023

Mestrado em
Cibersegurança
Master in
Cybersecurity

State of Web3 security

Analysis of vulnerabilities in bug bounty reports.

a master's thesis authored by

Ana Rita Amorim Melo

and supervised by

Pedro Filipe Cruz Pinto
Professor Adjunto, Instituto Politécnico de Viana do Castelo

António Alberto dos Santos Pinto
Professor Coordenador, Instituto Politécnico do Porto

This thesis was submitted in partial fulfillment of the requirements for the
Master's degree in Cybersecurity at the Instituto Politécnico de Viana do Castelo

ipvc

18 of December, 2023

Abstract

Web3 has its basis in blockchain and smart contract technologies, supporting secure, distributed, and decentralized applications. Nonetheless, Web3 is still in the process of evolution, and, as with any other software-based product, software bugs, security flaws, and other vulnerabilities are expected to appear.

This thesis analyzes the severity of security vulnerabilities in Web3 based on publicly available bug reports. Furthermore, an evaluation of several vulnerability detection tools in smart contracts is carried out. Finally, a plugin is developed that allows integration with a smart contract testing tool.

Through this analysis, it is possible to obtain a comprehensive view of the evolution and trends related to the number of reports presented, growth by platform, severity classification, and amounts paid for discovering and reporting vulnerabilities. The plugin developed as part of this study provides an additional tool to improve the security and reliability of Web3-based applications.

Keywords: Web3. Blockchain. Smart Contract. Ethereum.

Resumo

A Web3 tem como base a tecnologias blockchain e os contratos inteligentes, suportando aplicações seguras, distribuídas e descentralizadas. No entanto, o Web3 ainda está em processo de evolução e, tal como acontece com qualquer outro produto baseado em software, é expectável bugs de software, falhas de segurança e outras vulnerabilidades.

Esta tese analisa a severidade das vulnerabilidades de segurança na Web3 com base em relatórios de bugs disponíveis publicamente. Para além disso, é realizada uma avaliação de diversas ferramentas de detecção de vulnerabilidades em contratos inteligentes. Por fim, é desenvolvido um plugin que permite a integração com uma ferramenta de teste de contratos inteligentes.

Através desta análise é possível obter uma visão abrangente da evolução e tendências relacionadas ao número de relatórios apresentados, crescimento por plataforma, classificação de severidade e valores pagos por descoberta e submissão de vulnerabilidades. O plugin desenvolvido como parte deste estudo fornece uma ferramenta adicional para melhorar a segurança e confiabilidade de aplicações baseadas em Web3.

Palavras-chave: Web3. Blockchain. Smart Contract. Ethereum.

Aknowledgements

I want to thank my teachers and advisors, Pedro Pinto and António Pinto. Throughout my academic journey, they were like guides, leading me through the complexities of this challenging research and writing process. In addition to being experienced in their field, they demonstrated a tireless dedication to sharing valuable knowledge and providing personalized guidance. They were more than just academic mentors; They were proper drivers of my intellectual and personal growth. Their constant availability to brainstorm ideas, answer questions, and offer specific guidance was invaluable support.

I want to thank IPVC-ESTG for all the support and resources provided throughout my academic journey. The institution was essential in completing this thesis, providing the environment and tools necessary for my academic growth.

I want to thank Tectank, who generously allowed me to use working hours to conduct the meetings and research necessary for this thesis. The company not only provided practical conditions for the development of this project but also provided a stimulating and collaborative work environment. I want to express my sincere gratitude to the entire team that was by my side. The support I received was unbelievable, with co-workers understanding my academic responsibilities and constantly encouraging me to achieve my goals. The gestures of support and understanding from the entire company have made a substantial difference in my academic and professional journey. I am deeply grateful for this opportunity and the incredible team I have worked with.

Finally, I want to thank my family and my boyfriend sincerely. The love, support, and patience they showed me throughout this journey were the foundation that allowed me to complete this stage. My family, especially my parents, brother, and grandmother, have always given me unwavering support and constant encouragement and always believed in me and my potential. To my boyfriend João for his understanding and unconditional support that made this path a more peaceful and motivating journey. My dedication to this project and its successful completion are, in large part, a result of the love and support I received from them.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

CSV Comma-separated Values

CVE Common Vulnerabilities and Exposures

DApps Develop Decentralized Applications

DeFi Distributed Finance

ETH Ether

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTP GET Hypertext Transfer Protocol GET

HTTPS Hypertext Transfer Protocol Secure

MCyber Master in Cybersecurity

OWASP Open Web Application Security Project

SSL Secure Sockets Layer

URL Uniform Resource Locator

Chapter 1

Introduction

Web3 is envisioned as a backend revolution that replaces central data storage with a widely distributed environment [35]. As a decentralized web, Web3 can bring advantages such as mitigating problems concerning network vulnerabilities, misinformation, and information disorganization. Web3 allows for creating, identifying, contracting, exchanging, commercializing, and managing public or private content, products, and services [6]. Also, it can provide more security when creating scalable and accessible applications for private information exchange, sharing, and transfer.

Web3 builds on technologies such as Blockchain and smart contracts, among others. A Blockchain is a group of transactions that are part of a shared ledger that has been entered into data storage and has been previously verified by multiple sources. In addition, Blockchain is expected to make data secure and tamper-proof [46]. A smart contract is purely electronic, written as code, supported, and enforced by the entire Blockchain system [46]. Web3's purpose is to bring about network decentralization and allow users to have control over their online activities. Ethereum is a decentralized network that can run applications, which can be seen as a set of smart contracts in a distributed environment. These smart contracts can be implemented in Solidity, a high-level programming language designed to create Develop Decentralized Applications (DApps) on the Ethereum platform. The idea is to avoid dependency on a single entity, which must also store and manage users' personal and business data [14].

1.1 Problem Statement and Motivation

With its increased use, Web3 has become a more interesting and valuable target for attacks. These attacks are usually targeted against its components, such as the underlying Blockchain or the smart contracts themselves [26]. The problem is aggravated if is considered the Distributed Finance (DeFi) part of Web3, where financial services and solutions move funds and money. Such funds and money can be stolen by hackers [23]. As with any system and software, users are expected to encounter bugs and flaws that malicious people might use to exploit it. An example is the Qubit Finance attack [25], which stemmed from exploiting a security flaw in a smart contract, resulting in the unauthorized transfer of a substantial sum of money, totaling $80 million that had never been deposited.

White hackers, upon discovering a software vulnerability or bug, aim to report them so that the involved entities can correct them. This is widely known as bug-bounties, and some online platforms enable this. The key idea behind bug bounties is to enable better overall security of the online services and platforms that adopt it and still enable white hackers to be awarded for their efforts. These programs offer monetary rewards for discovering and reporting a vulnerability. Reports produced in this context can then be used to improve the security of new or similar existing services.

1.2 Objectives

The main objective of this project is to collect, analyze, and categorize the reports of vulnerabilities present in Web3. This research analyzes and perceives the level of growth of Web3 over the last few years. In addition, a plugin is aimed to be created based on an existing tool that enables the analysis of the vulnerabilities of an already published smart contract.

1.3 Contributions

The work presented herein resulted in the following contributions:

1. Rita Melo. "Web3 Cybersecurity". In: *CyberSec 23.* 24th January, Viana do Castelo, Portugal, 2023 - In this work, an analysis of the severity of vulnerabilities

related to Web3 was carried out, which was conducted based on publicly available bug reports. The results examining trends and patterns made it possible to obtain valuable conclusions about the security of Web3 and its potential future growth.

2. Rita Melo, Pedro Pinto, and António Pinto. "Severity Analysis of Web3 Security Vulnerabilities based on Publicly Bug Reports". In: *Blockchain and Applications, 5th International Congress.* (to appear). 12th-14th July, Guimarães, Portugal, 2023 - This study analyzed the severity of vulnerabilities in the context of Web3 based on public bug reports. It took a more comprehensive approach, exploring a more significant number of bug bounty platforms. This resulted in collecting a substantially more considerable amount of bug reports and expanding the analyzed data set. This expansion in data collection allowed for a more complete and comprehensive analysis of Web3-related vulnerabilities. Various public reporting sources were examined, enabling more robust trends and patterns to be identified in the Web3 security landscape.

1.4 Organization

This document is organized in the following chapters. Chapter 2 aims to further introduce the relevant technologies and address contributions related to this topic that others have made. Chapter 3 presents the explored platforms and discusses bug bounties platforms. Chapter 4 details the methodology used for the analysis and collection of vulnerabilities in Web3, also analyzing the results obtained. Chapter 5 describes the entire process of developing a plugin for the Remix tool. Finally, in Chapter 6, the conclusions are presented.

Chapter 2

Blockchain and Web3

At the beginning of this chapter, the world of blockchains is explored, explaining both their functioning and their application, covering both blockchains and smart contracts. Ethereum, one of the leading blockchain platforms, is also discussed as to how it relates to smart contracts and the general functioning of the blockchain. Next, Web3 addresses a new version of the internet that is more decentralized, gives more power to users, guarantees more privacy and security, and is more efficient.

Through this analysis, it is possible to understand how these technologies are related and work together to shape the current technological scenario.

2.1 Blockchain

Blockchain is a digital data structure that stores information securely, transparently, and decentralized [47]. It comprehends a series of data blocks, each of which contains a set of transactions or information. These transactions range from cryptocurrency transfers to property registrations or smart contracts [38].

Fig. 2.1 represents the logic of the blocks in the blockchain system. Each block on the blockchain contains a secure hash, a unique string of characters generated from the data collected in the previous block, along with new data from the current block. This process creates a cryptographic link between blocks, ensuring the chain's integrity [18].

This secure hash on each block is critical to ensuring the integrity and security of the blockchain. When a new block is created, it contains information not only about the

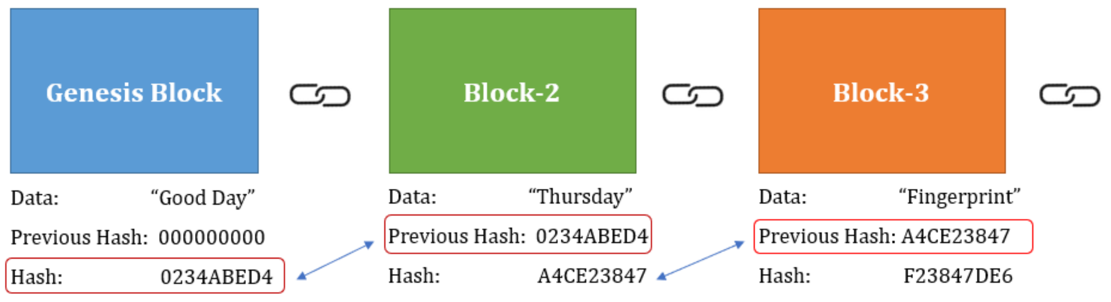


Figure 2.1: Blockchain process [20]

transactions within it but also a hash of the previous block. This process begins a cryptographic link that connects each block to its predecessor, thus forming the distinctive feature of a blockchain, the continuous sequence of blocks. These hashes have significant implications. First, they ensure that any attempt to change a particular block, such as tampering with a past transaction, would require modifying all subsequent blocks, which is infeasible, complex, and time-consuming due to the proof-of-work process or consensus mechanism. Furthermore, this link between blocks creates a public report that allows anyone to verify the integrity of the blockchain. Any discrepancy would be immediately detected, making blockchain manipulation a significant challenge. Combining these elements - the formation of blocks, the cryptographic link between them, and consensual validation - establishes the security and immutability of the blockchain.

Unlike traditional centralized systems, such as banks or government institutions, a single entity is responsible for recording and validating transactions. In the case of blockchain, it operates decentralized, meaning no central authority controls the system [29]. On the contrary, blockchain copies are distributed by network users, and these users validate and record transactions consensually. Validation of transactions on the blockchain is carried out through a "mining" process (in the context of cryptocurrencies). Network participants compete to solve complex mathematical puzzles, with the first person to solve the puzzle being responsible for validating and adding a block to the chain. This is known as "proof of work" in the case of Bitcoin. Other consensus mechanisms are adopted on different blockchains [32].

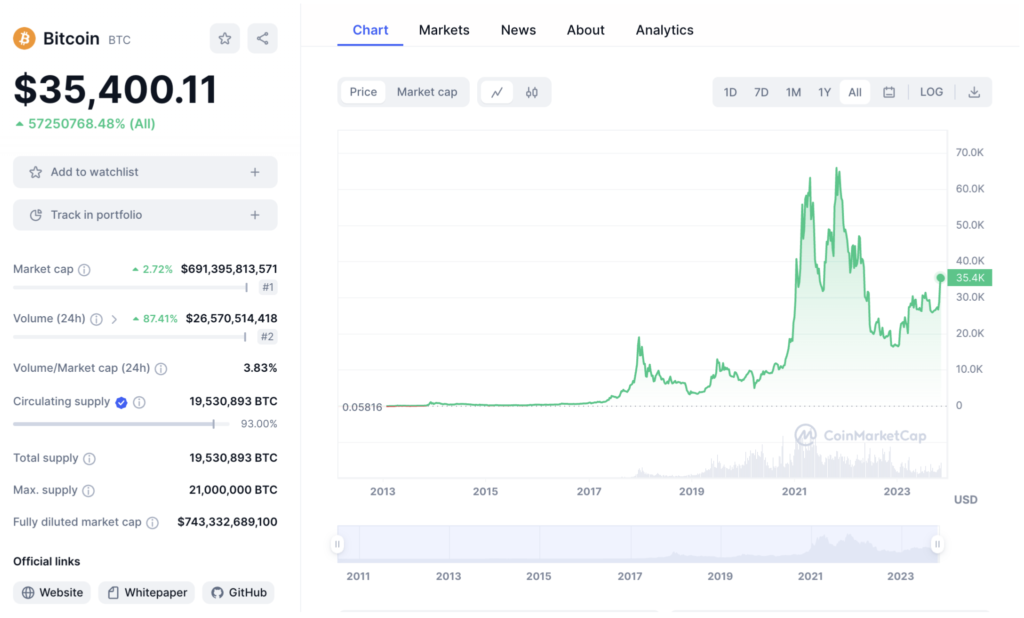
The main features of blockchain are:

- **Immutability** - Once a transaction is recorded in a block and added to the blockchain, it becomes practically impossible to change. This is because any attempted change would require modifying all subsequent blocks, which is extremely difficult and would require massive computing power [2].
- **Transparency** - All transactions recorded on a public blockchain are visible to all network participants, creating transparency that is essential for transaction trust.
- **Decentralization** - Blockchain distributes copies of the entire blockchain to every node in the network, making it highly resistant to failures and attacks since there is no single point of failure. This technology allows for a wide range of applications in different sectors: finance, logistics chains, digital voting, and document management, due to its ability to eliminate intermediaries, ensure security, and promote transparency [44].
- **Security** - Is maintained through cryptography, whereby cryptographic keys digitally sign and protect each transaction.

In addition to Bitcoin, other digital currencies, such as Ethereum, have gained significant space. Blockchain enables secure and fast financial transactions, eliminating intermediaries and reducing transaction costs. Currently, there are already companies that accept cryptocurrencies as a form of payment [4], and banks are exploring the potential of Blockchain to improve the efficiency of financial services. Another promising application of Blockchain is the management of property and asset records. By creating smart contracts on the Blockchain, it is possible to automate the transfer of ownership of properties, vehicles, and other assets. This process reduces bureaucracy and costs associated with real estate and property transactions, making the process faster and safer. Regarding healthcare, patients can securely control access to their health data through blockchain-based digital medical records [3]. Furthermore, technology allows for the rapid and secure sharing of medical information between healthcare providers, improving the quality of services and coordination of treatments. As blockchain adoption continues to grow, the world must continue to explore the potential of Blockchain and develop appropriate regulations to promote an enabling environment for innovation. With a strategic and collaborative approach, it is possible to reap the benefits of this technological innovation and lead the adoption of Blockchain in Europe and the world.

The evolution of Bitcoin and Ethereum from the beginning of their journeys to their peaks in value in 2021 is fascinating and reveals the remarkable growth of the cryptocurrency market. From the data presented in the Bitcoin evolution graph in Fig. 2.2, the cryptocurrency has experienced remarkable growth since its initial emergence. This growth culminated in the historic peak in value in 2021, when Bitcoin reached the impressive mark of $64.44k. However, after this peak, the cryptocurrency entered a period of decline, experiencing a devaluation trend.

Recently, Bitcoin has experienced ups and downs, but it is currently on an upward trajectory [10]. This indicates that cryptocurrency continues to be a volatile asset and is subject to significant fluctuations. Bitcoin's evolution is influenced by several factors, such as market demand, global adoption, and relevant news, contributing to its constantly changing price dynamics.


Bitcoin BTC

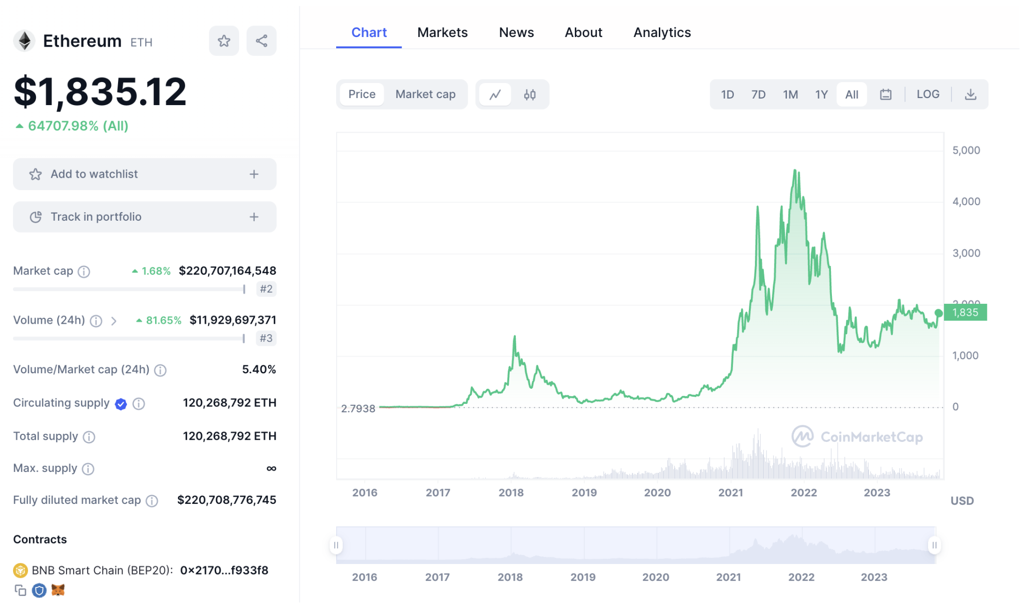
$35,400.11

▲ 57250768.48% (All)

Market cap	▲ 2.72%	$691,395,813,571 (#1)
Volume (24h)	▲ 87.41%	$26,570,514,418 (#2)
Volume/Market cap (24h)		3.83%
Circulating supply		19,530,893 BTC (93.00%)
Total supply		19,530,893 BTC
Max. supply		21,000,000 BTC
Fully diluted market cap		$743,332,689,100

Figure 2.2: Bitcoin evolution [10]

Fig. 2.3 represents the trajectory of Ethereum, and it can be checked that the evolution of this cryptocurrency has been remarkably upward since its emergence. This growth culminated in its value peak in 2021 when Ethereum reached an impressive level of $4,724.31. However, after this culmination, the cryptocurrency began to experience a devaluation trend.


Ethereum ETH

$1,835.12

▲ 64707.98% (All)

Add to watchlist

Track in portfolio

Market cap	▲ 1.68%	$220,707,164,548 (#2)
Volume (24h)	▲ 81.65%	$11,929,697,371 (#3)
Volume/Market cap (24h)		5.40%
Circulating supply		120,268,792 ETH
Total supply		120,268,792 ETH
Max. supply		∞
Fully diluted market cap		$220,708,776,745

Contracts

BNB Smart Chain (BEP20): 0x2170...f933f8

Figure 2.3: Ethereum evolution [10]

Both Bitcoin and Ethereum reached their peaks in value in 2021, a year that marked the rise of public and institutional interest in cryptocurrencies. This simultaneity of peaks reflects the interlinkage of the leading cryptocurrencies on the market, as many investors keep a close eye on both assets. However, the subsequent trajectories of both currencies show that, although they share points of contact, they also have distinct characteristics and fundamentals that affect their evolutions.

2.2 Smart Contracts

A smart contract is a self-executing computer program that operates on top of a blockchain, the decentralized digital infrastructure that underpins cryptocurrencies such as Bitcoin and Ethereum [48]. These contracts are designed to automate and facilitate the execution of agreements and transactions between parties without the need for intermediaries such as banks, notaries, or lawyers.

The main functionality of a smart contract is the ability to define and comply with predefined rules automatically and transparently. When two parties agree to establish a smart contract, the terms of the contract are codified. This code is executed on the

Blockchain when specified conditions are met [31]. For example, an insurance contract could be programmed to automatically pay compensation to an insured when specific criteria, such as a road accident recorded on the Blockchain, are met. Smart contracts offer several advantages. They automate contract execution, speeding up processes and reducing errors. Furthermore, they guarantee complete transparency, as all transactions and terms are immutably recorded on the Blockchain [19]. Security is undeniable, thanks to the encryption and decentralization of Blockchains, making them highly resistant to fraud. They also reduce costs, eliminating intermediaries and making contracts more accessible to different businesses. These characteristics make smart contracts a promising innovation in commercial transactions. Although smart contracts offer several advantages, they face challenges, such as legal issues surrounding their application and the risk of code failures [39].

The relationship between blockchain and smart contracts is a synergistic union transforming how society conducts business and transactions. Blockchain provides the secure and reliable foundation necessary for the immutable execution of smart contracts, while smart contracts bring automation and authenticity to these transactions. As these technologies continue to evolve, they are likely to witness an exponential increase in the applications of smart contracts across industries, redefining how individuals interact and do business in the digital world [33].

Ethereum is a decentralized global network allowing users to conduct transactions, create smart contracts, and DApps [45]. Its cryptocurrency unit, Ether (ETH), is used as a medium of exchange within the network and as an incentive for miners who ensure its security and operation [9].

It allows the development of dApps, decentralized applications that work on the blockchain [43]. These dApps can be used in various industries, from finance to gaming. Furthermore, it allows the creation of complex smart contracts, which makes it a versatile platform for a wide range of use cases, such as electronic voting systems, DeFi, and even blockchain-based games. DeFi refers to protocols, applications, and smart contracts built on blockchain technology to make various financial services available without intermediaries, such as banks or conventional financial institutions [21]. These services include lending, interest-bearing, asset swapping, staking, yield farming, and more.

The relationship between Ethereum, Blockchain, and smart contracts is complex and is profoundly reshaping the digital landscape. Exploring the core technological innovation that has driven substantial transformation in financial systems and other sectors is essential to understanding this interconnection. Ethereum, a blockchain platform, is the epicenter of this relationship. It allowed smart contracts to become a functional reality. These smart contracts are self-executing computer programs based on pre-defined rules and are housed on the Ethereum blockchain. Blockchain, in turn, serves as the foundation of this infrastructure. It is a decentralized, immutable, and transparent registration technology that records all transactions carried out on the Ethereum network. This transparency and security are crucial for trust in digital operations. Smart contracts are essentially the point of intersection between Ethereum and blockchain. They are written in code and stored on the Ethereum blockchain. When the conditions specified in these contracts are met, they are automatically executed, eliminating the need for intermediaries or trust in the parties involved. In short, the relationship between Ethereum, Blockchain, and smart contracts is symbiotic. Ethereum provides the environment for the creation and execution of smart contracts, while Blockchain ensures the security and immutability necessary to sustain these smart contracts. This technological triad is challenging traditional conventions, opening up new possibilities for process automation, transaction security, and the creation of innovative decentralized applications.

2.3 Web3

Web3 is a revolutionary vision that creates a decentralized and autonomous online environment where users have greater control over their data, digital identities, and interactions on the network [5]. It represents a fundamental change compared to Web2, characterized by the centralization of power in the hands of large technology companies [13].

In Web3, decentralization is the keyword. It is built on technologies such as blockchain and cryptocurrencies, which allow for the creation of peer-to-peer networks where transactions and interactions occur directly between users without centralized intermediaries. Users can exchange information, carry out financial transactions, and participate in smart contracts without depending on companies or government institutions [37]. Furthermore, on Web3, users have greater control over their data. This implies that personal information is not stored on centralized servers but on distributed and encrypted storage systems controlled by the user. Privacy concerns persist and can be mitigated as Web3 applications and services are built around the need to adhere to certain privacy principles. This becomes challenging, especially since many blockchain technologies are inherently transparent, recording all transactions in plain text on the ledger [30].

Web3 is the result of a gradual evolution from Web2. Web2 is the Internet as it is known today, where large technology companies like Google, Facebook, and Amazon dominate the digital landscape [34]. These companies collect and control massive user data, becoming essential intermediaries in online lives. The transition to Web3 began with the popularization of blockchain and cryptocurrencies, notably Bitcoin, which was the first practical application of this revolutionary technology. Blockchain introduced the idea of a distributed, immutable ledger where transactions are recorded transparently and securely without a central authority. Later, Ethereum, a programmable blockchain platform, brought the ability to create smart contracts, which are autonomous programs that automatically perform actions when certain conditions are met. This paved the way for various decentralized applications (dApps) and digital tokens, shaping Web3. Web3 responds to growing concerns about privacy, security, and centralization in Web2.

Decentralization is the foundation of Web3 and one of its fundamental principles. In Web3, decentralization means that power is distributed more evenly among users and is not monopolized by intermediaries or centralized authorities [6]. Blockchain, the core technology of Web3, plays a crucial role in decentralization. It is a distributed ledger maintained by a global network of users rather than a single central entity. This means there is no single point of failure, making the system more resistant to attacks.

Web3 enhances autonomy, privacy, security, scalability, and efficiency, providing individuals with greater control over their digital identities and interactions on the Internet [12].

Regarding autonomy, Web3 presents itself through various means [7]:

- **Data Ownership:** Users have complete control over their data. Instead of trusting companies to store and protect this information, users can store their data securely in distributed, encrypted systems. This avoids concerns about privacy breaches and data leakage.

- **Self-sovereign identity:** On Web3, users can create self-sustainable digital identities. In other words, a single digital identity can be used across multiple online platforms, eliminating the need to create separate accounts. These user controls have access to information about their identity.

To ensure privacy and security, decentralization and cryptography are important [8], as follows.

- **Data Privacy:** Decentralization prevents user data from being centralized on company servers, minimizing the risk of privacy violations. Users control how their data is shared and who has access to it.
- **Secure Communications:** Communication on Web3 is often end-to-end encrypted, ensuring that only parties can read messages. This protects against interception of communications by third parties.
- **Secure Authentication:** Self-sustaining digital identities are protected by encryption, ensuring that only the authorized user can access the identity.

However, despite these privacy and security improvements, Web3 faces challenges, such as the need for user education on protecting their private keys and sensitive information.

Scalability [37] and efficiency [49] are significant challenges Web3 faces as it grows. While decentralization and security are key priorities, the ability to handle a large volume of transactions and interactions is essential to making Web3 viable on a global scale. One of the main concerns regarding scalability is the ability of blockchain networks to process transactions quickly and efficiently. Many blockchains, such as Bitcoin and Ethereum, have faced congestion and high transaction fees during high demand. This could be an obstacle to mass adoption and widespread use of Web3. To face this challenge, many projects on Web3 are working on developing scalability solutions, such as sharding, i.e., a technique that divides the blockchain into small parts, into fragments; each shard operates independently and processes only a subset of the network's transactions, and more efficient consensus protocols. These solutions aim to increase the processing capacity of blockchains, allowing a more significant number of transactions per second.

In summary, Web3 is shaping the future of the Internet and digital society, introducing a revolutionary vision of decentralization, user autonomy, privacy, and security. It represents a direct response to the challenges of excessive centralization and lack of control over user data on Web2. However, Web3 also faces significant challenges, from scalability issues to the delicate balance between privacy and transparency. As this new digital era evolves, Web3 and its community of users and developers must work together to overcome these challenges and build a fairer, more transparent, and secure Internet.

Chapter 3

Web3 Vulnerability Status and Reporting

This chapter covers related work regarding vulnerability status and reporting, highlighting the significant contributions of researchers and practitioners concerning security while using bug bounty platforms and scanning vulnerabilities in smart contracts. Then, the concept of bug bounty platforms is explored in more detail. Later, the approach differentiates and complements current research is then outlined.

3.1 Web3 vulnerabilities

The usage of Web3 contributed to the growth of related cyber-attacks, with Blockchain and smart contracts being significant targets [40]. These attacks are primarily based on vulnerability exploitation [26]. Snegireva [41] analyzed different vulnerabilities related to Blockchain and smart contracts. A list of recently found vulnerabilities, publicly disclosed using a Common Vulnerabilities and Exposures (CVE) identifier, is also presented. This list includes vulnerabilities in software such as the Bitcoin Core (CVE-2020-14198), the Ethereum virtual machine (CVE-2021-29511), or the Hyperledger product family (CVE-2020-11093), for instance. The analysis concludes that the existing vulnerabilities were either previously in the source code or have recently emerged due to the development of new system features, such as the addition of smart contracts.

Sapna et al. [40] examined various types of vulnerabilities and attacks to the Blockchain.

They presented real attacks, which occurred between 2016 and 2019, in their explanation of the different vulnerabilities they identified. These real-life attacks had impacts that ranged from tens of thousands of dollars to 60 million dollars. They also compile a list of security solutions and methods that could be used to mitigate security vulnerabilities. They conclude that the analyzed attacks demonstrate the insecurity of smart contracts and that it is necessary to continue researching this area to discover new security options and appropriate testing methods.

Pise et al. [36] also analyzed blockchain-based security vulnerabilities that are specific to smart contracts. They classify the different smart contract vulnerabilities into three main blocks: platform-related, application-related, and code-related. The first is related to security flaws that exist in the adopted platform, such as Ethereum or Hyperleger. The second relates to the web applications that usually accompany smart contracts. The third focuses specifically on security flaws in the smart contract code itself. The authors specify each one and present a set of security measures. However, they mention that with the growth of these systems, it is necessary to develop best practices to combat bugs and future security risks.

Marchesi et al. [22] proposed a security assessment checklist for developing smart contracts. Similarly to Pise et al., these authors classify vulnerabilities into the same three (platform, application, and code). They start their work by identifying and analyzing security patterns for DApps. Next, the authors propose that security checklists be applied in different phases of the smart contract development lifecycle. In particular, they propose security assurance checklists for the design, coding, and testing phases. Although this list is being updated over time, this work is based on public sources, focusing on scientific and forum articles. Their work is heavily based on the work of Open Web Application Security Project (OWASP) and can be summed up to a collection of 48 patterns and best practices for the secure development of smart contracts in Ethereum and Solidity.

Connelly in [11] addressed smart contract security, particularly focusing on the Ethereum platform and providing a status of its current security state. In his work, Connelly proposes a classification system for smart contracts. Through his analysis, he concluded that smart contracts have immense vulnerabilities, many of which are unknown in their consequences. A key distinction between vulnerabilities and exploits is made, stating that the presence of a level of insecurity in the code of a smart contract (vulnerability) does not mean that it can be actively exploited, or that its exploitation has a relevant impact. The symbolic execution through Mythril, a code security analysis tool, was adopted to create a digital registry of flawed smart contracts, accompanied by a rating system.

Matulevicius et al. [26] analyzed the adequacy of the existing static code analysis tools to Web3 and their smart contracts. To do so, they developed purposely flawed smart contracts and used tools to analyze them. The vulnerabilities in the flawed smart contracts followed a confidentiality, integrity, and availability classification. They concluded that Slither was the best tool in terms of accuracy and one of the best in terms of performance. Nonetheless, the maximum obtained accuracy was 75%, with tools reaching only 25% accuracy. In the best case, 25% of the vulnerabilities remain undetected by such static analysis tools.

In this case, a critical vulnerability in the DeFi protocol called Yield Protocol was identified and reported in April 2023 through the bug bounty platform Immunefi [17]. This vulnerability could have allowed an attacker to empty token reserves in a pool by manipulating the balance of tokens in a specific contract in the protocol. Thanks to the immediate detection and reports of a white hat via the Immunefi platform, the team behind Yield Protocol acted quickly to patch the vulnerability. This action prevented a possible loss of around $950,000 across several pools operating on the Arbitrum and Ethereum blockchains. Yield Protocol is a DeFi protocol that offers loans with fixed interest rates and fixed terms between borrowers and lenders. It operates with tokens known as fyTokens, which can be exchanged directly for an underlying asset on a predefined expiration date. The vulnerability was related to the Yield Protocol strategy contract, which allowed liquidity providers to pool their funds into YieldSpace. By depositing funds into this contract, liquidity providers generated strategic tokens whose quantity was related to the amount deposited. These tokens could be burned to redeem the liquidity provider's tokens, along with any subsequent fee or interest earnings. The vulnerability allowed manipulation of this process, which could have resulted in significant losses for protocol users, but was avoided due to the team's quick action and white hat reporting.

```
// Original calculation of pool tokens to be acquired
poolTokensObtained = pool.balanceOf(address(this)) * burnt / totalSupply_;
```

Listing 3.1: Vulnerability code - Yield Protocol

The mentioned vulnerability involved the use of the original formula that used the function *pool.balanceOf(address(this))* to calculate the number of tokens from a pool that should be transferred to a user. This calculation method was considered insecure as it allowed an attacker to manipulate the balance of this pool, making it susceptible to attack. To address this vulnerability, the team behind Yield Protocol corrected the strategy contract:

```
poolTokensObtained = poolCached_ * burnt / totalSupply_;
```

Listing 3.2: Corrected calculation of pool tokens to be acquired

This fix replaced *pool.balanceOf(address(this))* with *poolCached_* to calculate the pool tokens transferred to the caller. This change eliminated vulnerability to attack, making the system more secure.

In November 2022, a white hat identified and reported a critical vulnerability in the Beanstalk protocol through Immunefi [16] that could result in the potential theft of assets worth up to $3.1 million, including $537,000 in BEAN tokens and $2.5 million in non-BEAN assets. Fortunately, due to the quick action of this white hat and the Beanstalk rewards system in Immunefi, the Beanstalk team was able to remedy the issue, preventing the loss of users' funds. Beanstalk is a protocol built on Ethereum, which aims to create a stablecoin called Bean and establish a monetary base for a decentralized economy on the Ethereum network, keeping the value of the Bean *at par* with the dollar. The vulnerability was identified in one of the Token Facet libraries used by the Beanstalk diamond proxy contract, which allows modular updates and extensions after deployment. Token Facet is responsible for the token transfer logic and the vulnerability was found in the *transferTokenFrom()* function, which did not correctly check the provision for external transfers. It allowed attackers to receive funds from accounts that had approved the Beanstalk contract to manipulate their tokens via the ERC20 *approve()* function.

```
// Snippet 1: TokenFacet: transferTokenFrom()
function transferTokenFrom(address from, address to, uint256 amount,
    TransferFromMode fromMode, TransferFromMode toMode) public override {
    if (fromMode == TransferFromMode.INTERNAL) {
        require(balanceOf(from) >= amount, "Bean: INSUFFICIENT_BALANCE");
    }

    if (fromMode == TransferFromMode.EXTERNAL) {
        // Vulnerability: No balance check for external transfers
        LibTransfer.transferToken(from, to, amount, TransferFromMode.
            EXTERNAL);
    }
}
```

Listing 3.3: Vulnerability code - Beanstalk

The Beanstalk team quickly resolved the issue, removing the vulnerable function and introducing a new function called *transferInternalTokenFrom()*, which always operates in *INTERNAL fromMode* mode. These changes were implemented by the Beanstalk Improvement Proposal (BIP).

```
// Introduction of a new function to fix the vulnerability
function transferInternalTokenFrom(address from, address to, uint256 amount
    ) public override {
    require(balanceOf(from) >= amount, "Bean: INSUFFICIENT_BALANCE");

    LibTransfer.transferToken(from, to, amount, TransferFromMode.INTERNAL)
        ;
}
```

Listing 3.4: Correction Code - Beanstalk

These code snippets illustrate the vulnerability found in the original *transferTokenFrom()* function and how it was fixed by introducing the *transferInternalTokenFrom()* function to ensure the security of transfers in the Beanstalk protocol. of the necessary corrections.

To combat the vulnerabilities in Web3, specifically in smart contracts, exists on the SolidityScan platform. This online platform allows users to verify the security of smart contracts that have been published, see the general security of smart contracts, and allow programmers to reduce the number of vulnerabilities that may arise.

3.2 Bug bounty platforms

Bug bounty platforms are online environments where companies and organizations invite ethical hackers and cybersecurity experts to identify and report vulnerabilities in their systems, applications, or websites [15]. These vulnerabilities, often called bugs, can range from authentication flaws to complex security issues compromising data integrity or user privacy. What makes bug bounty platforms unique is their collaborative, community-driven approach. Instead of waiting for a malicious hacker to discover and leverage a vulnerability, organizations hire ethical hackers to look for and report these vulnerabilities responsibly. In return, these hackers receive monetary rewards or other incentives such as public recognition.

The operation of a bug bounties platform follows the steps below:

- **Invitation and Registration**: The organization interested in strengthening its cybersecurity invites ethical hackers to participate in its bug bounties platform. These hackers register on the platform, agree to the established rules and conditions, and are ready to start.
- **Testing and Reporting**: Ethical hackers exploit an organization's systems for vulnerabilities. When they find one, they report it to the organization through the platform, accompanying it with detailed information about how the vulnerability was found and how it could be exploited.
- **Assessment and Reward**: The organization assesses the severity of the vulnerability and takes action to correct it. If the vulnerability is validated, the ethical hacker is rewarded with money, prizes, or public recognition, depending on the platform's policies.
- **Continuous Improvements**: This process is not a one-time event. It is a continuous activity, as systems and applications are constantly evolving. Organizations encourage ethical hackers to continue to look for vulnerabilities and contribute to constant security improvements.

Bug bounty platforms represent an innovative approach to cybersecurity, leveraging the collective intelligence of ethical hackers to protect digital systems. These platforms may be considered to offer the following set of benefits:

- **Early Discovery of Vulnerabilities**: By engaging the ethical hacking community, organizations can detect and fix vulnerabilities [24] before someone with bad intentions exploits them.
- **Cost Savings**: Paying vulnerability bounties is often more cost-effective [42] than dealing with a data breach's financial and reputational consequences.
- **Community Engagement**: These platforms promote collaboration between ethical hackers and organizations, building a more robust security environment.
- **Continuous Improvement**: Cybersecurity is a constant effort. Bug bounty platforms encourage continuous security improvement.

However, despite the clear advantages of bug bounty platforms, they come with significant challenges. One of the most pressing challenges lies in screening and managing vulnerability reports. As these platforms gain popularity, the volume of reports can increase considerably, which can overwhelm organizations' internal resources, delaying the assessment and remediation of vulnerabilities. Furthermore, the quality of reports can vary, with few needing to be more specific or adequately documented, making the assessment task more complex. Another challenge is the appropriate delimitation of the scope of the bug bounty programs, as the inadequate inclusion of critical systems or assets can expose the organization to unplanned risks. Additionally, managing communication with ethical hackers, including ensuring that reported vulnerabilities are kept confidential until resolved, is critical to preventing premature and potentially harmful disclosure of information. Ultimately, the fine line between ethical hackers and cybercriminals and concerns about legal and ethical issues are essential considerations organizations must face when implementing bug bounty programs. Therefore, while these platforms offer notable benefits, their implementation requires a careful and well-thought-out approach to mitigate these challenges and limitations.

In conclusion, bug bounty platforms represent an innovative and practical approach to bolstering cybersecurity in an increasingly digitized world. Organizations can identify and report vulnerabilities in systems, applications, and websites by inviting ethical hackers to identify security holes before someone with bad intentions exploits them. These platforms promote collaboration, continuous learning, and improved security while rewarding bounty hunters. As technology evolves and cyber threats become more sophisticated, bug bounties are poised to play an even more essential role in protecting data and ensuring digital integrity. For companies and organizations committed to security, adopting and investing in bug bounty programs is a proactive measure demonstrating responsibility and commitment to protecting their assets and users' privacy. With the support of the ethical hacking community and the constant evolution of best practices, bug bounty platforms continue to shape the future of cybersecurity.

3.3 Summary

This analysis differs from the others mainly because of the used sources that consist of publicly available reports of bug-bounties platforms or company audits. This approach made it possible to collect detailed and up-to-date information on security vulnerabilities relating to Web3 that were identified and corrected, thus identifying the main trend. In addition, data collection and analysis were automated, which enabled the rapid and efficient extraction of accurate data from hundreds of collected reports.

Chapter 4

Severity Analysis

In this chapter, the methodology is detailed step by step, to ensure a comprehensive understanding of the used procedures and techniques. The collected data then be analyzed, exploring the information and identifying relevant patterns and trends. Finally, in the results and discussion phase, the meaning of the information previously obtained and its connection with the research is discussed.

4.1 Methodology

The study adopted a specific methodology to systematically collect and analyze publicly available reports. Fig. 4.1 depicts all the steps performed, from the starting point of identifying and choosing the platforms, up to the moment of the final analysis of the reports.

First, a stage of identification and careful selection of the platforms that would be the target of this research was carried out. The sample included two main categories: platforms that offer rewards for identifying security flaws (bug bounty) and provide reports or audits related to security vulnerabilities in Web3 technology. In the platform selection process, factors such as the number of reports available on each platform and the structure of the reports were considered. Emphasis was placed on finding reports with a fixed and predefined format. This has become a key criterion as it makes it possible to automate the analysis of these reports. With a common structure, it is more feasible to develop code for automated analysis that is capable of efficiently and effectively collecting and interpreting

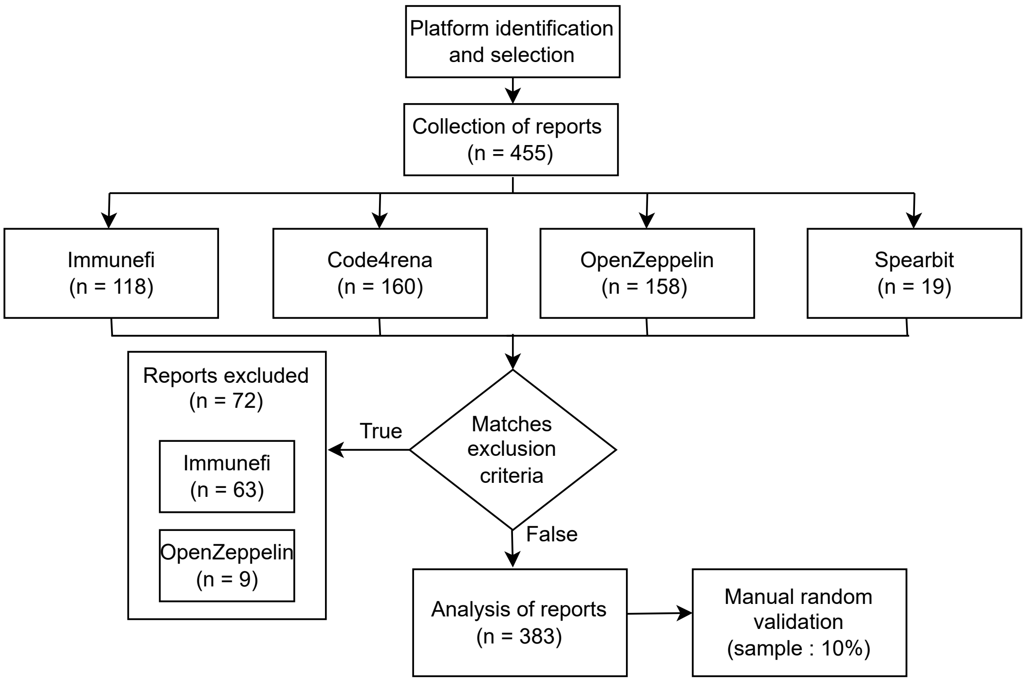

Platform identification and selection
Collection of reports (n = 455)
Immunefi (n = 118)
Code4rena (n = 160)
OpenZeppelin (n = 158)
Spearbit (n = 19)
Reports excluded (n = 72)
Immunefi (n = 63)
OpenZeppelin (n = 9)
True
Matches exclusion criteria
False
Analysis of reports (n = 383)
Manual random validation (sample : 10%)

Figure 4.1: Adopted methodology

the data in the reports. As a result of this selection, four specific platforms were chosen to serve as reporting sources: Immunefi[1], Code4rena[2], Spearbit[3], and OpenZeppelin[4]. Each platform has a substantial number of reports and meets the established selection criteria. Therefore, the initial step of identifying and judiciously selecting the platforms was crucial in ensuring that the subsequent process of collecting and analyzing reports was organized and result-oriented.

In the subsequent phase, the collection of reports from the mentioned platforms was carried out. This collection covered the period until the end of 2022 since, at the time of collection, there was a reduced number of results referring to the year 2023. To carry out this analysis, two different approaches to data collection were adopted: one manual and the other automated.

The automated approach involves running custom code to analyze and extract crucial data from the reports. This approach has been successful on the Code4rena and Spearbit platforms. For each platform, scripts adapted to the context and particularities of each platform were developed, allowing the automated collection of links to all available reports.

[1]`https://immunefi.medium.com/`
[2]`https://code4rena.com/reports/`
[3]`https://github.com/spearbit/portfolio/tree/master/pdfs`
[4]`https://blog.openzeppelin.com/`

In contrast, a manual approach was employed to collect data from the Immunefi and OpenZeppelin platforms. This was due to the less standardized structure of reports on these platforms, which made automated collection difficult. The report links were carefully and manually assembled in this manual process. Once the links to all the reports were collected, specific information of interest was extracted, such as the report's title, the date of publication, and, when applicable, the amount paid for the discovery and notification of each vulnerability. Due to variations in report formats on each platform, specific codes were created and adapted to each source, enabling the organization and management of the relevant data coherently. Details on the amounts paid were only available on the Immunefi platform, as it is dedicated to vulnerability detection rewards. These reports provide information on the amount paid for each identified and reported vulnerability. In summary, at the end of this collection phase, 455 reports were obtained from the following sources: 118 from Immunefi, 160 from Code4rena, 158 from OpenZeppelin, and 19 from Spearbit. Each platform had specific codes adapted to the context for automated or manual collection of relevant data.

The listings 4.1, 4.2, and 4.3 show an example code that analyzes the Code4rena platform, more precisely, each report through the developed code. The full code is presented in Appendix A.

```python
import requests
from bs4 import BeautifulSoup
import datetime
import csv

# Part 1: Collect the links
url = "https://code4rena.com/reports/"
response = requests.get(url)
page_content = response.content
soup = BeautifulSoup(page_content, "html.parser")
links = []

wrapper_report = soup.find("div", class_="wrapper-report")
if wrapper_report:
    wrapper_contests = wrapper_report.find_all("div", class_="wrapper-
        contest undefined")
    for wrapper_contest in wrapper_contests:
        wrapper_contest_content = wrapper_contest.find("div", class_="
            wrapper-contest-content")
        if wrapper_contest_content:
            a = wrapper_contest_content.find("a")
            if a:
                links.append("https://code4rena.com/" + a["href"])
```

Listing 4.1: Code used to analyze the Code4rena platform - Collect links

First, on Listing 4.1 the Uniform Resource Locator (URL) of the web page to be analyzed is defined. An Hypertext Transfer Protocol GET (HTTP GET) request is made to the specified URL, thus obtaining the page content. This content is stored in the page_content variable. Next, a BeautifulSoup object called soup is created to parse the HyperText Markup Language (HTML) content of the page. Furthermore, an empty list called links is initialized to store the links collected later. The page is scanned for the div with the class "wrapper-report." If this div is found, all divs with the "wrapper-contest undefined" class inside it are collected. For each of these divs, the anchor element (<a>) is located, and the "href" attribute of that element is extracted and added to the links list after being prefixed with "https://code4rena.com/".

```python
# Part 2: Analyze each collected link
results = []

for link in links:
    response = requests.get(link)
    soup = BeautifulSoup(response.content, 'html.parser')
    report_header = soup.find("div", class_="report-header")

    title = None
    date = None

    if report_header:
        h1 = report_header.find("h1")
        if h1:
            title = h1.text
        h4 = report_header.find("h4")
        if h4:
            date = datetime.datetime.strptime(h4.text, "%Y-%m-%d").strftime
                ("%d/%m/%Y")

    div = soup.find('div', class_='report-container')
    if div:
        div_text = div.text.lower()
        category = ""
        if "high risk findings" in div_text:
            category += " High "
        if "there were no high risk findings identified in this contest" in
             div_text:
            category = category.replace(" High", "")
        if "medium risk findings" in div_text:
            category += " Medium "
        if "low risk and non-critical issues" in div_text:
            category += " Low "
            category += " Non-Critical "
        if "low risk findings" in div_text:
            category += " Low "
        if "non-critical findings" in div_text:
            category += " Non-Critical "

        category_list = []

        if "Low" in category:
            category_list.append("Low")
        if "Medium" in category:
            category_list.append("Medium")
        if "High" in category:
            category_list.append("High")
        if "Non-Critical" in category:
            category_list.append("Non-Critical")

        category_str = ", ".join(category_list)
```

Listing 4.2: Code used to analyze the Code4rena platform - Analyze each collected link

In the next phase, on Listing 4.2, the code iterates over the list of collected links. For each connection, a new HTTP GET request is made to access the associated page. BeautifulSoup [5] is used again to analyze the page content. The div with the class "report-header" is searched on the page for information such as the title and date of the report. If the div is found, the <h1>and <h4>elements are examined to extract the title and date, respectively. The date is formatted in "DD/MM/YYYY" style. Subsequently, the div with the class "report-container" is analyzed for keywords that allow the report to be categorized. Based on the identified keywords, a category string is constructed, which may include values such as "Low," "Medium," "High," and "Critical," depending on the characteristics of the report. The collected data, including title, date, category, and URL, is added to the results list.

```
# Export the results to a CSV file
with open('results.csv', 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(['Title', 'Date', 'Category', 'URL'])
    csv_writer.writerows(results)
```

Listing 4.3: Code used to analyze the Code4rena platform - Export the results to a CSV file

[5] https://pypi.org/project/beautifulsoup4/

Finally, on Listing 4.3 the results are exported to a Comma-separated Values (CSV) file called "results.csv". This code was initially developed for the Code4rena platform. For other platforms, the code was adapted as necessary to handle the different reporting formats on each platform.

Then, all reports were subjected to careful analysis to determine their relevance within the identified scope, resulting in the exclusion of a set of these reports. Code4rena and Spearbit platforms focus exclusively on Web3 technology. Therefore, all reports from these platforms were considered relevant for the analysis. Concerning the Immunefi and OpenZeppelin platforms, it was found that these included reports that fell outside the scope of the research. Therefore, these reports were excluded from the study. Both the collection of data from these sources and the exclusion of inappropriate reports were carried out manually. Reports configured as informative articles, monthly reports, and essays that did not explicitly explain or analyze a vulnerability related to Web3 technology were excluded from the analysis. In total, 72 reports were removed from the dataset, 63 from Immunefi, and 9 from OpenZeppelin.

Next, the remaining 383 reports were analyzed. In this stage, the focus was on collecting the severity rating assigned to each report. For this, a code was developed which examined the text of the reports and collected information related to the severity of each platform. During the development, execution, and subsequent testing of these codes, it became evident that the classification obtained needed to be more accurate due to the diversity of reporting formats and the use of different terms to indicate severity, even within the same platform. This situation led to an iterative code improvement process to improve classification accuracy. To evaluate the effectiveness of the codes on all platforms and each one individually, a random manual check was performed on a sample corresponding to 10% of the results. This manual verification verified that 95.65% of the analyzed reports were classified correctly, while 4.35% presented incorrect classifications. When analyzing each source in detail, it was found that the reports from the Code4rena, Spearbit, and Immunefi sources reached an accuracy of 100%. All reports sampled and analyzed from these sources were classified correctly. However, concerning the OpenZeppelin platform, the accuracy was 87%, indicating that fewer reports presented incorrect classifications. This accuracy discrepancy is likely due to the variety of reporting formats and the choice of different terms to refer to the severity on this platform.

Once the data collection phase of the reports was completed, the severity was classified. Each platform uses its severity rating scale, with most platforms having four levels, except Code4rena only has three. Table 4.1 shows the correspondence between the severity rating of each platform and the used severity rating. To allow for a better comparison, the severity ratings were normalized and the data was adjusted accordingly to make it comparable across different platforms.

Table 4.1 presents the correspondence of the different severity classifications of each platform to the classification used in the study. Terms used on each platform have been normalized to common terminology within the scope of this study. This made it possible to establish a basis for comparing the severity ratings of the different platforms, facilitating the analysis of the collected data. The term ”Low” across all platforms has defaulted to ”Low” in the table. Likewise, ”Medium” has been retained as ”Medium” for all platforms. For the highest severity level, which may be referred to as ”High” on some platforms and ”Critical” on others, the table adopts the term ”Critical” uniformly.

Table 4.1: Normalization of severity classifications

Code4rena	**Immunefi**	**OpenZeppelin**	**Spearbit**	**Adopted Classification**
Low	Low	Low	Low	Low
Medium	Medium	Medium	Medium	Medium
	High	High	High	High
High	Critical	Critical	Critical	Critical

After completing the normalization process, a set of instructions was created and executed to establish the correspondence between the severity classifications of each platform and the classification used in the study. Then the data was reviewed again, keeping only the highest severity rating for each report. For example, if a report originally had four rating levels (critical, high, medium, and low), only the critical rating would be considered for that report after this further analysis. This adaptation reduced the number of ratings displayed, giving greater prominence to the highest severity level. This is especially relevant as this level is associated with more severe and urgent vulnerabilities. This approach of highlighting the highest severity rating provides a clearer view of the most severe vulnerabilities that require immediate attention. In addition, it simplifies data analysis, making it easier to understand and identify the main threats.

In conclusion, the study adopted a specific methodology to systematically collect and analyze publicly available reports. The steps involved the careful identification of platforms, the collection of reports, the analysis of relevance, the classification of severity, and the normalization of the classifications. The cautious selection of platforms ensured that the data was relevant and consistent with the scope of the research. Manual and automated collection of reports allowed for a comprehensive analysis, while severity analysis revealed the importance of the most severe vulnerabilities. Normalizing the ratings made it easier to compare across platforms. The study demonstrated the complexity of Web3 vulnerability analysis and the need for adaptive approaches. In short, the applied methodology provided valuable insights into the security of Web3 technology, contributing to the understanding and mitigating potential risks.

4.2 Results

After collection and normalization, the data was analyzed to perceive the growth, trends, and patterns of the identified vulnerabilities. Despite collecting reports that refer to situations dated until the beginning of 2023, the records from 2023 were not considered in the analysis because these were too few to be statistically significant.

The number of reports per source and severity is shown in Fig. 4.2. Considering the Code4rena platform, there are only reports of critical, medium, and low severity, the critical being predominant with 134 reports. Considering the Immunefi platform, there are only reports of high and critical severity, the critical being predominant with 37 reports. Considering the OpenZeppelin platform, there are reports of all levels of severity, the critical being predominant with 35 reports. Finally, considering the Spearbit platform, there are reports of all levels of severity, the critical and high being the predominant ones with 8 reports each. Code4rena has the most significant number of reports, with 158 reports in total, and Spearbit has the smallest number of reports, with 19 reports in total. Immunefi has a total of 39 reports, and OpenZeppelin has 79 reports. It is also noticeable that in all platforms, the predominant severity level is the critical one, so it is possible to conclude that most problems encountered on the analyzed platforms have critical levels of severity.

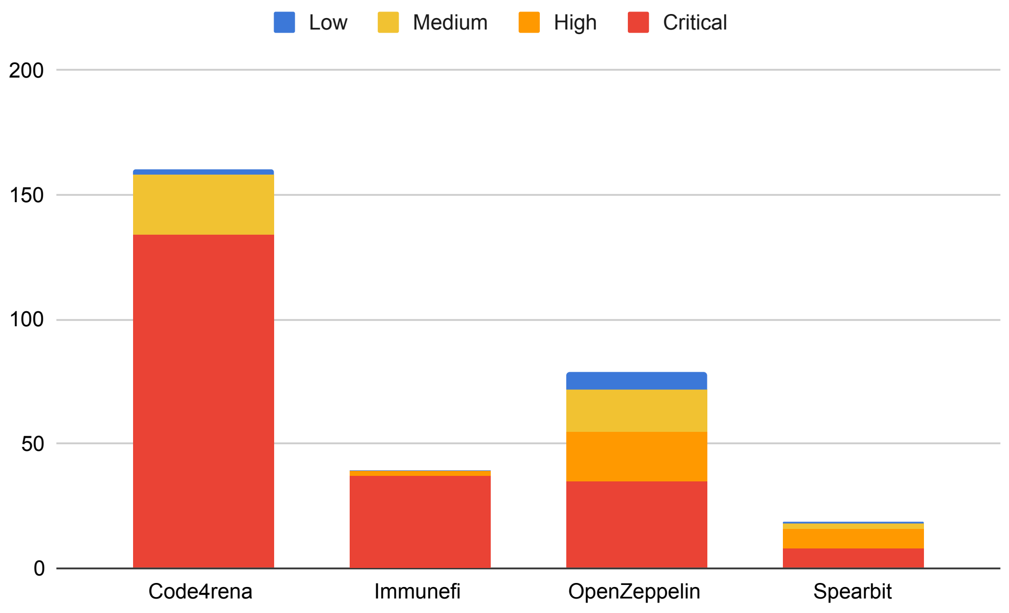


Figure 4.2: Number of reports per source and severity

The number of reports by date and severity is shown in Fig. 4.3. The dates are organized in semesters, and this figure depicts its evolution and trend over time. The period analyzed comprises reports dated from the second half of 2016 to the second half of 2022.

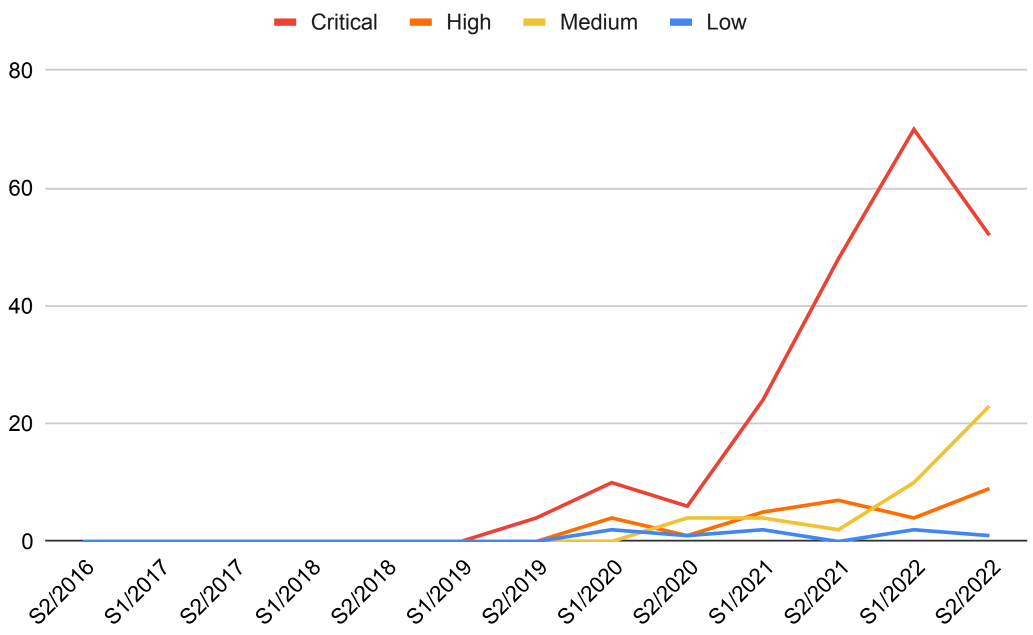


Figure 4.3: Number of reports by date and severity

Analyzing the low severity level, it has never had a very high number over the years, with its peak being in the first half of 2020, 2021, and 2022 when it reached 8 reports. The reasoning is that low-level vulnerabilities might not be appealing to be reported from the security researchers' point of view. Analyzing the medium level, it has been increasing in recent years, and its peak is in the second half of 2022 when it reached 23 reports. Analyzing the high level, it shows an increasing trend over the years. Particularly in 2022, it is possible to observe a steady increase, peaking in the second half of 2022 when it reached 9 reports. Analyzing the critical level, it is noticeable that, over the last few years, there has been significant growth in the number of reports, peaking in the first half of 2022 with 70 reports. Overall, from 2020 onwards, there was a significant increase, which may have been triggered by the forced digitalization brought about by the COVID-19 pandemic. In summary, a significant growth trend in the number of critical reports can be identified.

The stacked number of reports by date and severity is shown in Fig. 4.4, showing the total number of reports per semester over the years.

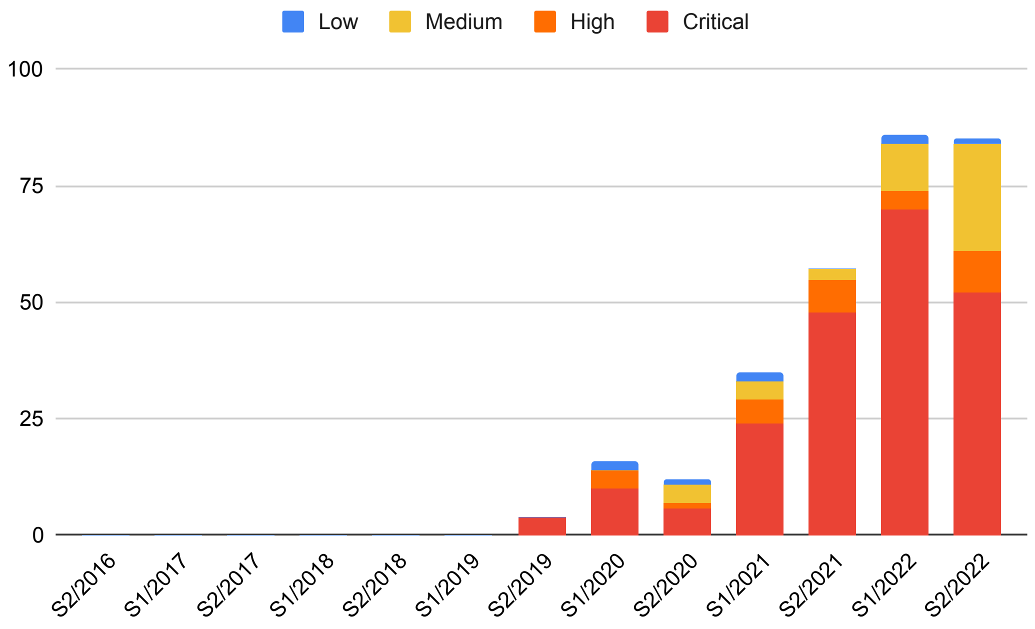


Figure 4.4: Number of reports by date and severity, stacked

There was a peak in the first semester of 2022, in which there was a more significant number of critical reports, precisely 70 out of a total of 86. The total number of reports made in the two semesters of 2022 is similar and it can be concluded that there is a steady increase in the total number of reports per year, which is also true if only considering the critical reports. Since OpenZeppelin was founded in 2015, Spearbit in 2021, Code4rena in 2021, and Immunefi in 2020, only the oldest reports refer to OpenZeppelin, given that the other platforms were founded more recently.

The total number of reports by date and platform is presented in Fig. 4.5.

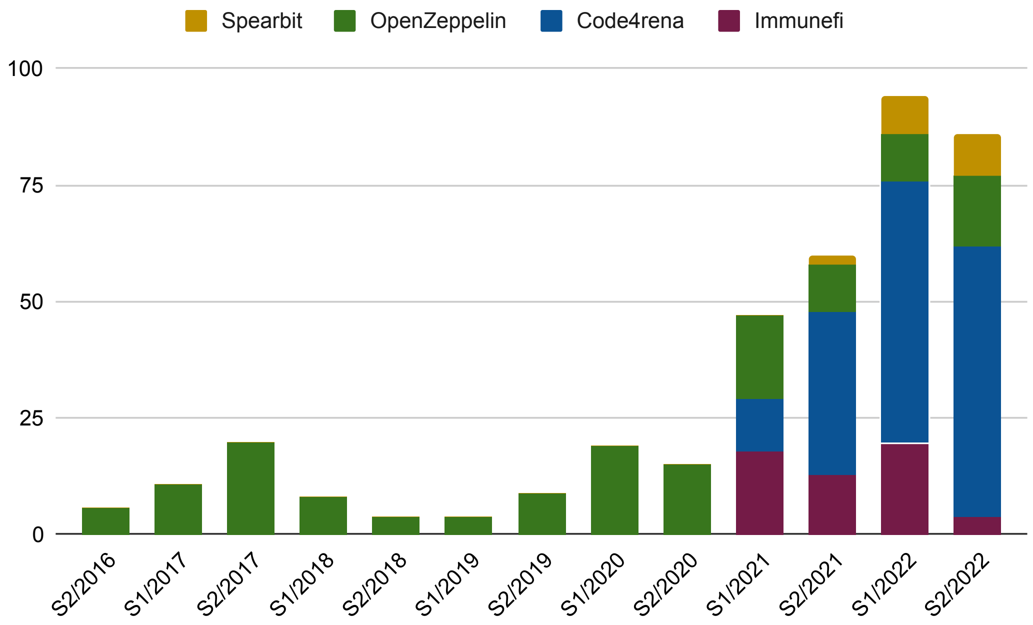


Figure 4.5: Number of reports by date and platform, stacked

The results show that the OpenZeppelin platform has the longest database of records, starting in the first semester of 2016 and up to the end of 2022. Worthy of note is the fact that OpenZeppelin is not exclusively dedicated to Web3 security vulnerabilities. Code4rena, Spearbit, and Immunefi sources have only recently begun to receive reports. Comparing this Fig. 4.5 with Fig. 4.4, it is possible to identify a peak in the same period, the first semester of 2022. Note that Fig. 4.4 does not include reports between 2016 and the first semester of 2019, while Fig. 4.5 does. In particular, the reports collected from the OpenZeppelin platform in such a period, do not include a severity classification. The OpenZeppelin platform only more recently started to include the classification in their reports. In summary, there is a clear growing trend over the years and the Code4rena platform has amassed more reports in recent years when compared to the other platforms.

Security researchers receive bounties when they report real security vulnerabilities. Fig 4.6 displays the total amount paid over the years, in thousands of dollars, for the Immunefi platform.

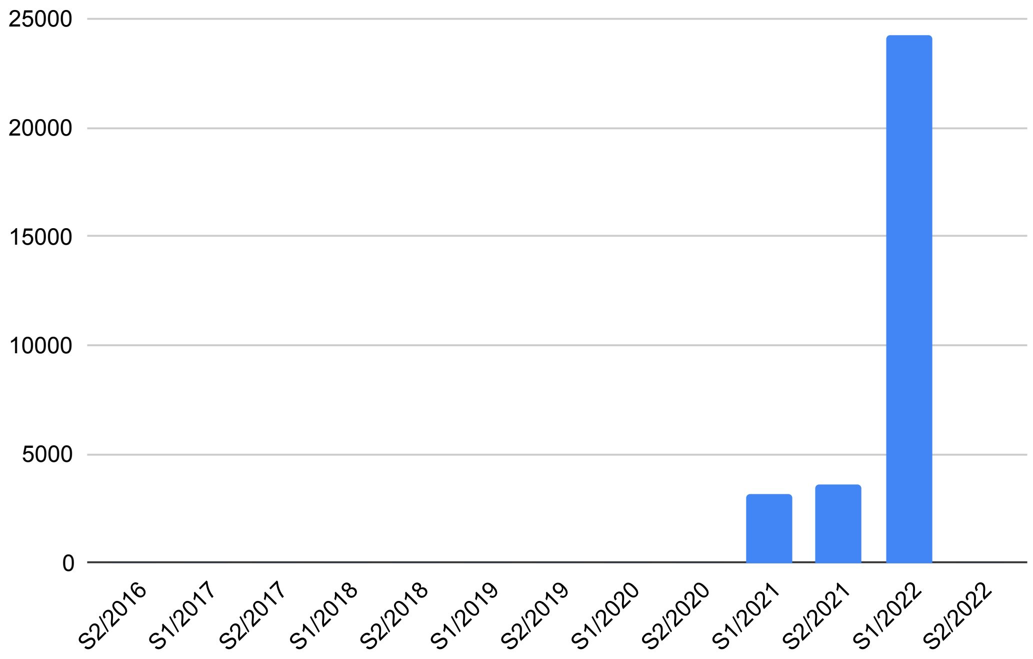


Figure 4.6: Total amount paid over the years, in thousands of dollars

From the collected data, the total amount paid for the reports analyzed was only available in the Immunefi platform and the values obtained were in dollars, 3,162,347$ in S1/2021, 3,608,515$ in S2/2021, and 24,215,042$ in S1/2022. These values show a steep increase in the revenues paid, more precisely, an increase of more than 20,000,000$ in the first half of 2022. Concerning the second half of 2022, there is a significant decrease in the total amount paid, contrasting with a similar total number of reports. Nonetheless, the number of reports on the Immunefi platform also decreased significantly, justifying the results. Over the years, it is possible to verify that by 2022 the amount paid in reports increased significantly, so by analyzing the previous graphs, in conclusion, there was an increase in the number of reports, mainly of those who have a level of critical severity, so it led to the rise in the amount paid.

4.3 Analysis and Discussion

Considering the results presented, reports have significantly increased over the last few years, with a peak in the first half of 2022, when all platforms received reports. In addition to the high number of reports, the growth of critical reports is also visible, demonstrating that more and more vulnerabilities emerge that pose a risk to the security of smart contracts. Concerning the amounts paid for discovering and reporting vulnerabilities, these peaked in the same period as the remaining amounts, with a notable rise.

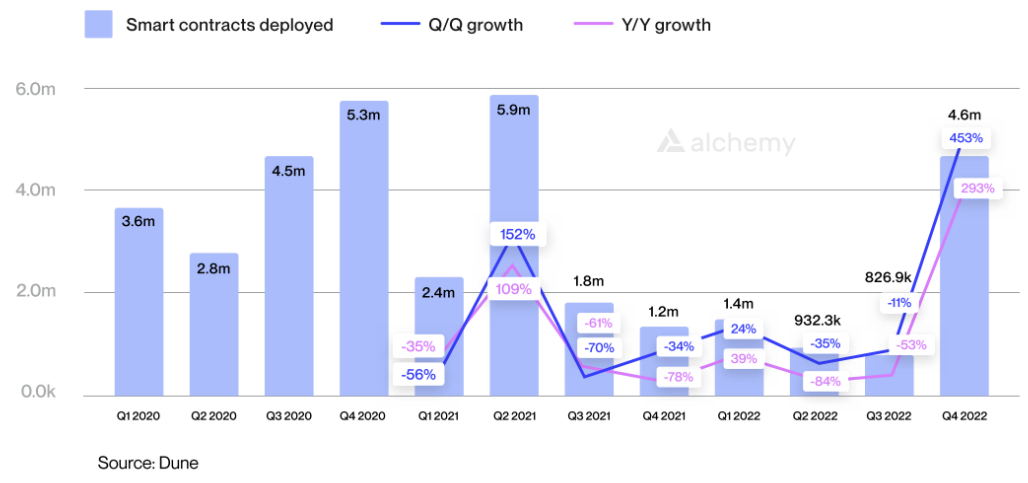


Figure 4.7: Smart contracts deployed in the Ethereum [1]

The Alchemy's report [1] includes the number of recently deployed smart contracts, per quartile, as shown in Fig. 4.7. In this graph, it is possible to analyze that the y-axis represents the number of smart contracts deployed, and the x-axis represents quartiles between 2020 and 2022. The minimum value was obtained in the third quartile of 2022, with 826.9 thousand smart contracts deployed, and on the other hand, the maximum value was in the second quartile of 2021, with 5.9 million smart contracts deployed.

Considering only the last quartile of 2022, it shows an increase of more than 453%, corresponding to a total of 4.6 million smart contracts deployed. Still, the numbers show a significant increase in smart contract deployment in 2021 and a significant overall decline in 2022, when compared to 2021. On the other hand, there was a substantial increase in the detection of vulnerabilities in 2022. This leads to the conclusion that, proportionately, there are more smart contracts with security concerns.

In conclusion, the comprehensive analysis of data regarding vulnerabilities in smart contract platforms provided an in-depth insight into the evolution of these vulnerabilities over time and their implications. Some crucial findings emerged by studying reporting sources, vulnerability severities, and time trends.

First, the analysis revealed a steady increase in the total number of reports, mainly from 2020 onwards. The peak observed in the first half of 2022 suggests an intensification of security concerns, possibly related to the digital acceleration induced by the COVID-19 pandemic. In addition, the predominance of vulnerabilities classified as critical in all analyzed platforms highlights the seriousness of the situation and the need for more effective preventive measures.

The analysis of the distribution of vulnerabilities by severity revealed specific trends. While the low-severity vulnerabilities showed limited variation over the years, the medium-severity vulnerabilities gradually increased, with a notable peak in the second half of 2022. The high severity category also showed a steady increase, reaching its peak in the same year. However, critical vulnerabilities were the ones that drew the most attention, showing significant growth, especially in the first half of 2022.

Analysis of bounties paid to security investigators uncovered a direct relationship between increasing detection of critical vulnerabilities and amounts paid. The exponential increase in rewards in the first half of 2022 reflects the appreciation of early identification and notification of these vulnerabilities, ensuring the soundness of smart contracts and the security of ecosystems.

Graphs comparing the evolution of vulnerabilities with the number of smart contracts deployed show a worrying correlation. The notable increase in critical vulnerabilities coincides with a dramatic increase in the deployment of smart contracts, suggesting that adopting these technologies is also accompanied by a proportional increase in the risk of security breaches.

The detailed analysis of this data demonstrated the imperative need to strengthen security measures in smart contracts. The current scenario, marked by the constant increase in critical vulnerabilities, requires continuous collaboration between security researchers, developers, and communities, to mitigate risks and strengthen the integrity of these systems.

Chapter 5

Remix plugin development

In this chapter, a detailed comparison of the characteristics of vulnerability detection tools is carried out, and the plugin development is discussed. Analyzing the functionalities of these tools is crucial to understanding how they can be improved to meet the specific needs of checking vulnerabilities in smart contracts. Furthermore, the plugin development process is explored, highlighting how plugins can be created and, in this case, integrated into an existing tool.

5.1 Features Comparison of Vulnerability Scanners

In the first phase, an analysis was carried out on various tools/platforms that allowed testing smart contracts. This analysis was made with the tools being open-source and free and, each platform was analyzed individually, focusing on the characteristics addressed in this study. The tools chosen for analysis and comparison were the following:

- **Oyente** [1] scans contracts for common vulnerabilities such as suspicious function calls and incorrect value handling.
- **Mythril** [2] performs static analysis to identify vulnerabilities and security issues such as access violations, indentation, and overflow/underflow issues.
- **Securify** [3] highlights potential issues such as access control failures and improper

[1]`https://github.com/enzymefinance/oyente`
[2]`https://github.com/Consensys/mythril`
[3]`https://github.com/eth-sri/securify`

handling of values.

- **Remix** [4] provides features for writing, testing, deploying, and debugging smart contracts. Additionally, it has static analysis capabilities to help identify potential security issues.
- **SmartCheck**[5] scans contracts for known vulnerabilities, such as malicious function calls and suspicious code patterns.
- **Vandal**[6] looks for common security vulnerabilities such as suspicious function calls, incorrect value handling, and access control issues.
- **Slither**[7] identifies known security issues and suspicious code patterns in contracts, providing helpful information to improve contract security.

Table 5.1 summarizes the findings regarding support for blockchains and smart contract programming languages. Most platforms support the Ethereum blockchain and the Solidity smart contract language, however, the Mythril platform is here highlighted since it runs in several versions, namely on Ethereum, EOS, Quorum, etc. Furthermore, it is the only one that presents different languages for smart contracts, such as Solidity, Vyper, Serpent, etc.

Table 5.1: Comparison of vulnerability detection tools for smart contracts

Tool	**Blockchain**	**Programming language**
Oyente	Ethereum	Solidity
SMythril	Ethereum, EOS, Quorum, etc.	Solidity, Vyper, Serpent, etc.
Securify	Ethereum	Solidity
Remix	Ethereum, ETC, Quorum, etc.	Solidity
SmartCheck	Ethereum	Solidity
Vandal	Ethereum	Solidity
Slither	Ethereum	Solidity

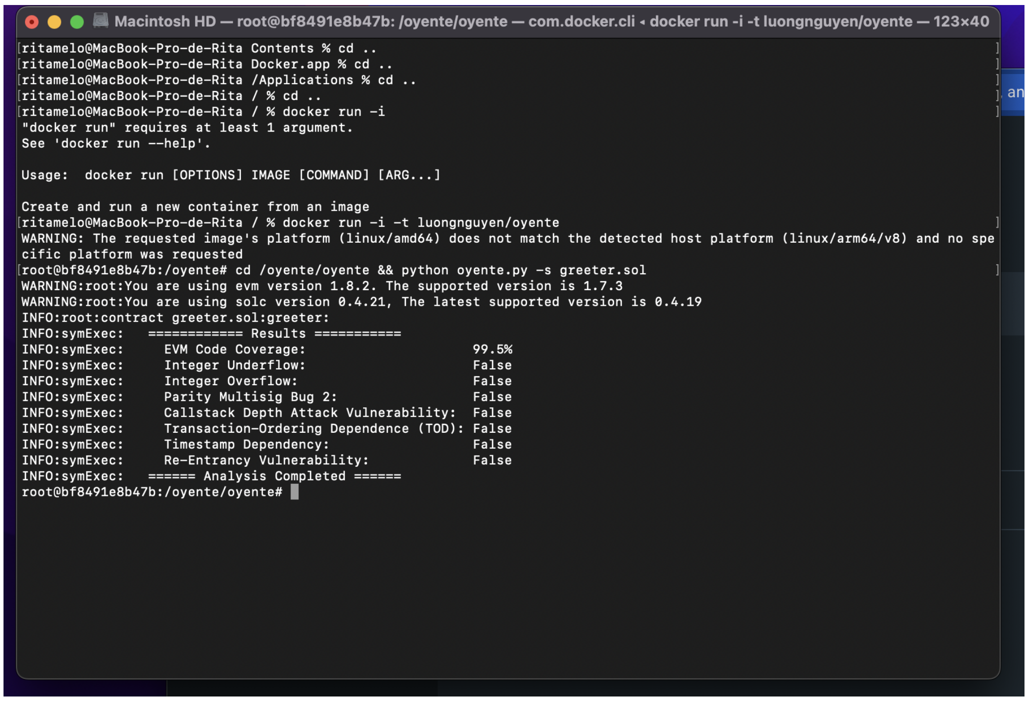
The testing phase of all the platforms selected for analyzing vulnerabilities in smart contracts has begun, but challenges were encountered during this process. One of the main complications discovered was that many platforms appear limited to only running

[4]`https://remix.ethereum.org/`
[5]`https://github.com/smartdec/smartcheck`
[6]`https://github.com/usyd-blockchain/vandal`
[7]`https://github.com/crytic/slither`

older versions of smart contracts. This presented a challenge, as many current projects use newer versions of programming languages and, consequently, smart contract structures.

For instance, during the tests conducted with the Oyente tool, it was identified that it can run smart contracts only up to Solidity version 0.4.19. Unfortunately, this limitation prevents the analysis of contracts written in newer Solidity versions. This is particularly problematic since frequent updates to the language can bring significant improvements in security and efficiency.

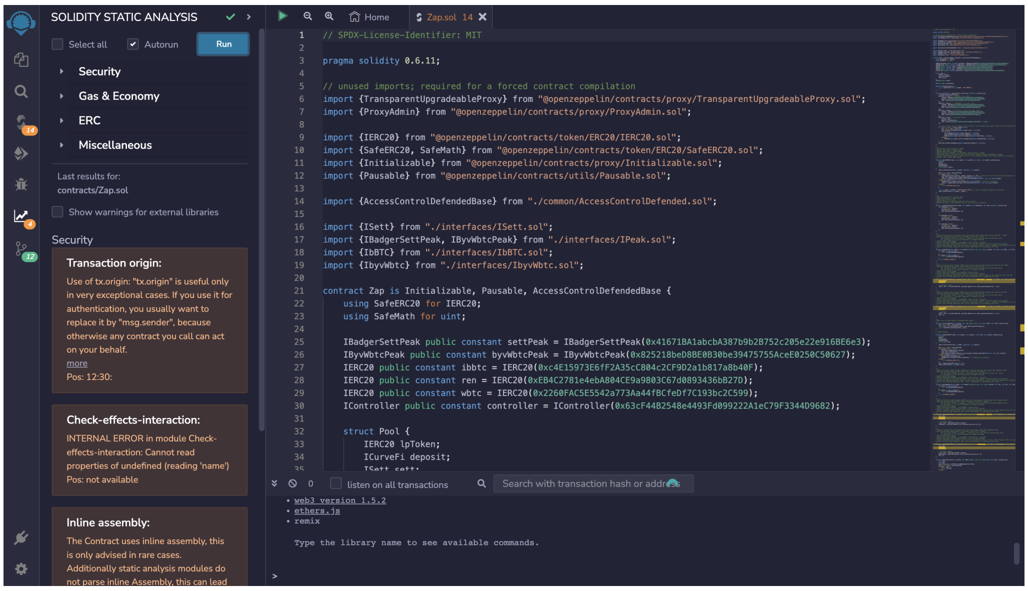
Fig. 5.1 shows that a smart contract was tested with the Oyente tool to detect its vulnerabilities. It is possible to observe that most of the vulnerabilities are false; they are not present. Additionally, it encountered significant difficulties when trying to run the other tools. Among them, Remix was the only one that proved more accessible and, consequently, the only one that could execute the contracts without significant problems.


Macintosh HD — root@bf8491e8b47b: /oyente/oyente — com.docker.cli ◂ docker run -i -t luongnguyen/oyente — 123×40

```
ritamelo@MacBook-Pro-de-Rita Contents % cd ..
ritamelo@MacBook-Pro-de-Rita Docker.app % cd ..
ritamelo@MacBook-Pro-de-Rita /Applications % cd ..
ritamelo@MacBook-Pro-de-Rita / % cd ..
ritamelo@MacBook-Pro-de-Rita / % docker run -i
"docker run" requires at least 1 argument.
See 'docker run --help'.

Usage:  docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

Create and run a new container from an image
ritamelo@MacBook-Pro-de-Rita / % docker run -i -t luongnguyen/oyente
WARNING: The requested image's platform (linux/amd64) does not match the detected host platform (linux/arm64/v8) and no spe
cific platform was requested
root@bf8491e8b47b:/oyente# cd /oyente/oyente && python oyente.py -s greeter.sol
WARNING:root:You are using evm version 1.8.2. The supported version is 1.7.3
WARNING:root:You are using solc version 0.4.21, The latest supported version is 0.4.19
INFO:root:contract greeter.sol:greeter:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                     99.5%
INFO:symExec:     Integer Underflow:                     False
INFO:symExec:     Integer Overflow:                      False
INFO:symExec:     Parity Multisig Bug 2:                 False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:                  False
INFO:symExec:     Re-Entrancy Vulnerability:             False
INFO:symExec:   ====== Analysis Completed ======
root@bf8491e8b47b:/oyente/oyente#
```

Figure 5.1: Oyente - Analyzing vulnerabilities on a smart contract

Fig. 5.2 shows that a smart contract was tested with the Remix tool to detect its vulnerabilities. It is possible to see that the test has been completed successfully, and the results are present on the left side of the panel. However, the other tools still need to be improved. On the contrary, the compatibility limitation may reflect the ever-evolving complexities of smart contract languages and structures.



Last results for:
contracts/Zap.sol

Show warnings for external libraries

Security

Transaction origin:
Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
more
Pos: 12:30:

Check-effects-interaction:
INTERNAL ERROR in module Check-effects-interaction: Cannot read properties of undefined (reading 'name')
Pos: not available

Inline assembly:
The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead

```
// SPDX-License-Identifier: MIT

pragma solidity 0.6.11;

// unused imports; required for a forced contract compilation
import {TransparentUpgradeableProxy} from "@openzeppelin/contracts/proxy/TransparentUpgradeableProxy.sol";
import {ProxyAdmin} from "@openzeppelin/contracts/proxy/ProxyAdmin.sol";

import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import {SafeERC20, SafeMath} from "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";
import {Initializable} from "@openzeppelin/contracts/proxy/Initializable.sol";
import {Pausable} from "@openzeppelin/contracts/utils/Pausable.sol";

import {AccessControlDefendedBase} from "./common/AccessControlDefended.sol";

import {ISett} from "./interfaces/ISett.sol";
import {IBadgerSettPeak, IByvWbtcPeak} from "./interfaces/IPeak.sol";
import {IbBTC} from "./interfaces/IbBTC.sol";
import {IbyvWbtc} from "./interfaces/IbyvWbtc.sol";

contract Zap is Initializable, Pausable, AccessControlDefendedBase {
    using SafeERC20 for IERC20;
    using SafeMath for uint;

    IBadgerSettPeak public constant settPeak = IBadgerSettPeak(0x41671BA1abcbA387b9b2B752c205e22e916BE6e3);
    IByvWbtcPeak public constant byvWbtcPeak = IByvWbtcPeak(0x825218beD8BE0B30be39475755AceE0250C50627);
    IERC20 public constant ibbtc = IERC20(0xc4E15973E6fF2A35cC804c2CF9D2a1b817a8b40F);
    IERC20 public constant ren = IERC20(0xEB4C2781e4ebA804CE9a9803C67d0893436bB27D);
    IERC20 public constant wbtc = IERC20(0x2260FAC5E5542a773Aa44fBCfeDf7C193bc2C599);
    IController public constant controller = IController(0x63cF44B2548e4493Fd099222A1eC79F3344D9682);

    struct Pool {
        IERC20 lpToken;
        ICurveFi deposit;
```

```
 0
• web3 version 1.5.2
• ethers.js
• remix

Type the library name to see available commands.

>
```

Figure 5.2: Remix - Analyzing vulnerabilities on a smart contract

These challenges highlight the importance of keeping tools up-to-date and compatible with the latest versions of smart contract languages and frameworks. During this research, a promising solution to overcome the imposed limitations when running current versions of smart contracts is the Remix tool. The main advantage of Remix is its support for recent versions of smart contract programming languages. Although it was considered a significant advancement, it detected the opportunity to expand its functionality to detect vulnerabilities in various smart contracts more efficiently.

To achieve this goal, an innovative plugin within a React project was developed, which incorporates the functionality of SolidityScan's Quick Scan tool. This plugin can be integrated with Remix, allowing developers to analyze the vulnerabilities of already published smart contracts. Through localhost, users can easily add this plugin to Remix, expanding its vulnerability scanning capabilities.

5.2 Plugin development

Fig. 5.3 presents a sequence diagram describing the interaction between Remix IDE, the plugin, and the online security scanning service SolidityScan. This diagram offers a clear view of the messages exchanged during the process.

When submitting requests for execution in the plugin, the order in which they are

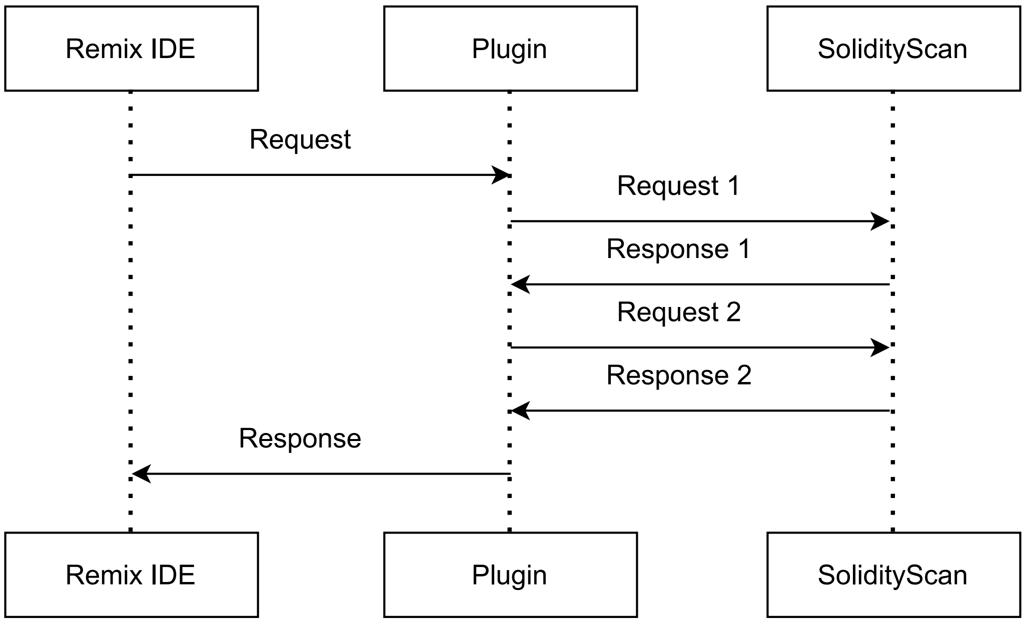


Figure 5.3: Plugin operation sequence diagram

processed is relevant. The system ensures that orders are executed sequentially, one after the other, rather than in parallel. This approach allows the plugin to keep track of requests and wait for responses from SolidityScan before proceeding to the subsequent request. This wait for a response is important to ensuring the security check's accuracy and completeness.

SolidityScan, as an online service, analyzes each security request in detail and returns the results to the plugin. Only when the plugin receives the response from SolidityScan, it will move it on to the subsequent request. This ensures that there are no conflicts or overloads in the system, ensuring the effectiveness of the security check.

The key differentiator of this plugin compared to the original SolidityScan tool is the ability to analyze multiple smart contracts simultaneously. This is critical for complex projects that have numerous interdependent contracts. By allowing developers to analyze and identify vulnerabilities in various parts of their smart contract system at once, the plugin saves time and effort while strengthening the security of the project as a whole.

Furthermore, the developed project can be run in the local environment or integrated into the Remix platform, meaning there are two different ways of using it. When run locally, only the project is run; when integrated into Remix, the project is combined with the Remix platform, transforming the final product into a plugin.

Additionally, direct integration with Remix provides a familiar environment for developers, allowing them to use a platform they already know to perform advanced security analysis on their smart contracts. This may help to reduce the learning curve and facilitate adopting the vulnerability analysis practice, increasing awareness and responsibility for contract security.

When starting a new React project, the first step is to create the basic framework using the "npx create-next-app" command. This hands-on approach gives a solid foundation to start developing quickly. However, building an app goes beyond the initial framework and involves configuration and customization to meet specific needs. In this context, the "App.js" file plays a central role, where can define what is displayed on the frontend. In this case, there was a need to interact with a web service. This motivated the creation of a backend in Node.js to handle calls to the web service. React, and the backend integration has become a fundamental part of the project. After configuring the "App.js" file, securing the development environment was essential. To achieve this goal, steps were taken to run the local server (localhost) with Hypertext Transfer Protocol Secure (HTTPS). This approach ensures encrypted and secure communication between the frontend and back end. The steps for this configuration included generating Secure Sockets Layer (SSL) certificates and implementing settings to enable HTTPS during local development. While this may seem complex, ensuring the security of transmitted data cannot be underestimated.

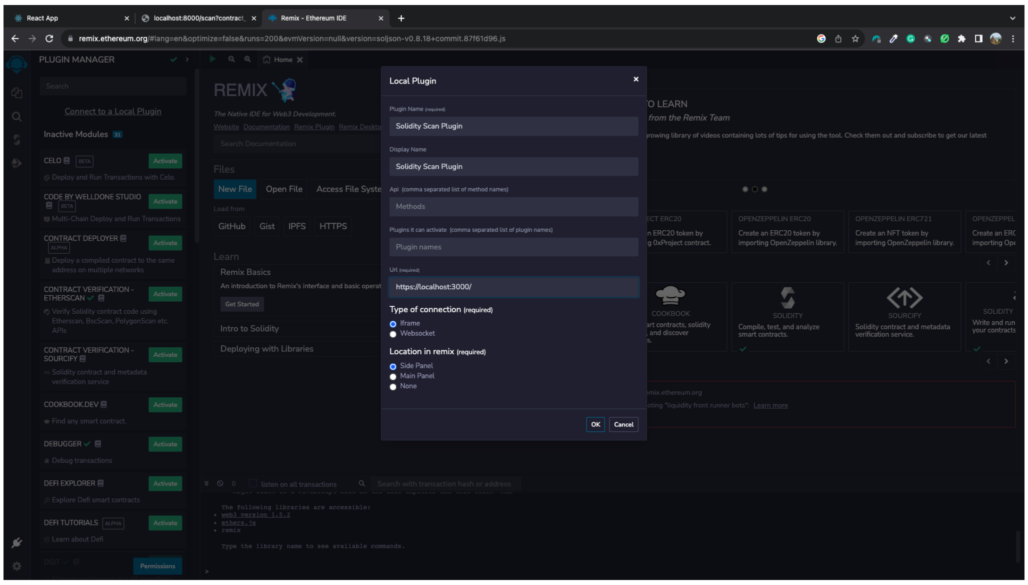
The Listing 5.1 shows that the code sets up a web server using Node.js and Express.js.

```
app.get("/scan", (req, res) => {
  var contractAddress = req.query.contract_address;
  var contractPlatform = req.query.contract_platform;
  var contractChain = req.query.contract_chain;

  var url =
    "https://solidityscan.com/app/api-quick-scan-sse/?" +
    "contract_address=" +
    contractAddress +
    "&contract_platform=" +
    contractPlatform +
    "&contract_chain=" +
    contractChain;

  res.setHeader("Access-Control-Allow-Origin", "https://localhost:3000");

  https
    .get(url, (response) => {
      let data = "";
      response.on("data", (chunk) => {
        data += chunk;
      });
      response.on("end", () => {
        res.send(data);
      });
    })
    .on("error", (error) => {
      console.log("Request error: ", error.message);
      res.status(500).send("Request error");
    });
});

// HTTPS server configuration
const privateKey = fs.readFileSync("certificates/key.pem", "utf8");
const certificate = fs.readFileSync("certificates/cert.pem", "utf8");
const credentials = { key: privateKey, cert: certificate };

// Creating the HTTPS server
const httpsServer = https.createServer(credentials, app);

// Start the HTTPS server
httpsServer.listen(8000, () => {
  console.log("HTTPS server running on port 8000");
});
```

Listing 5.1: Backend Code

This server defines a route accessible via HTTP GET requests on the ”/scan” route. When a client requests this route, the server executes the associated code. The ”/scan” route expects to receive three parameters in the form of query parameters in the URL: *contractAddress* (contract address), *contractPlatform* (contract platform), and *contractChain* (contract chain). These parameters are essential to identify which smart contract should be scanned by SolidityScan. The server extracts these parameters from the Hypertext Transfer Protocol (HTTP) request. The code then constructs a URL to request a specific endpoint in SolidityScan. This URL is assembled by combining the request parameters with a base URL, allowing the server to effectively communicate to SolidityScan which contract to verify, on which platform, and on which chain. After that, the code makes an HTTPS request to SolidityScan using the assembled URL. As the response is received, data is accumulated in the "data" variable as it arrives in chunks. When the answer is ultimately obtained, the data is returned to the client who made the request. If an error occurs during the request, the code captures the error and responds with a status of 500 (Internal Server Error) and an appropriate error message. Additionally, the code configures an HTTPS server that listens on port 8000, ensuring that communication with SolidityScan is secure and encrypted. In short, this code allows the plugin to communicate with SolidityScan efficiently and securely, providing accurate and fast responses for smart contract security verification. It is important to emphasize that for the plugin to work in Remix and, consequently, the project in a local environment, it is necessary to have HTTPS communication to communicate with the SolidityScan platform.

In the following figures, it is possible to observe the final version of the plugin already integrated with the Remix tool.


React App
localhost:8000/scan?contract_
Remix - Ethereum IDE
remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js
PLUGIN MANAGER
Connect to a Local Plugin
Inactive Modules
REMIX
Home
Files
New File
Open File
Learn
Remix Basics
Intro to Solidity
Deploying with Libraries
OK
Cancel

Local Plugin

Plugin Name (required): Solidity Scan Plugin

Display Name: Solidity Scan Plugin

Api (comma separated list of method names): Methods

Plugins it can activate (comma separated list of plugin names): Plugin names

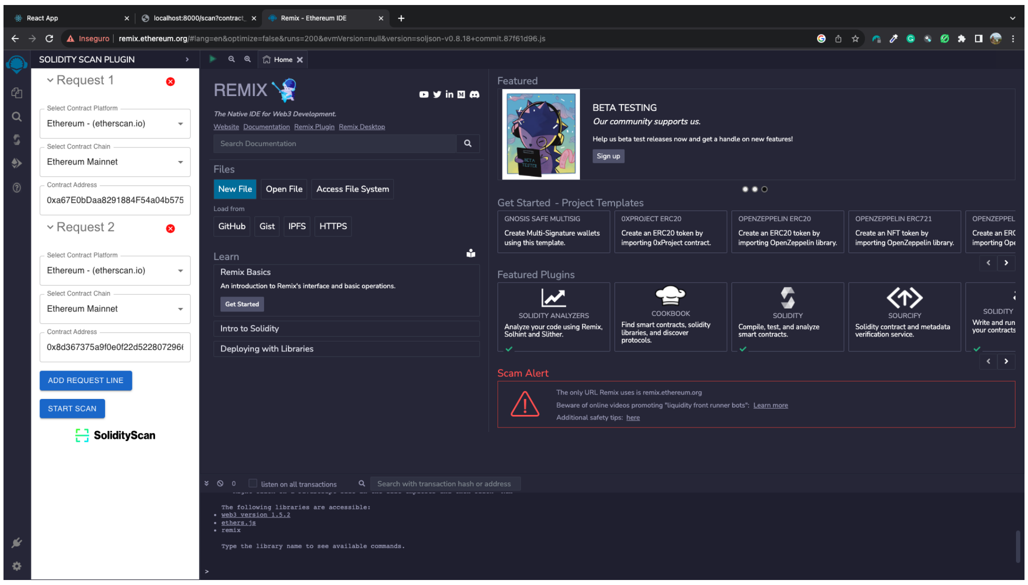
Url (required): https://localhost:3000/

Type of connection (required): [✓ Iframe] Websocket

Location in remix (required): [✓ Side Panel] Main Panel None

Figure 5.4: Plugin - Remix integration

In Fig. 5.4, it is possible to see the Remix platform. In this image, the plugin is being integrated with the platform. Within the remix, the Plugins option was selected, and then the Connect to a Local Plugin option was selected, which opened the window shown in the figure. In this same window, the name of the future plugin and the port on which it is running on localhost was also indicated.



Select Contract Platform: Ethereum - (etherscan.io)
Select Contract Chain: Ethereum Mainnet
Contract Address: 0xa67E0bDaa8291884F54a04b575

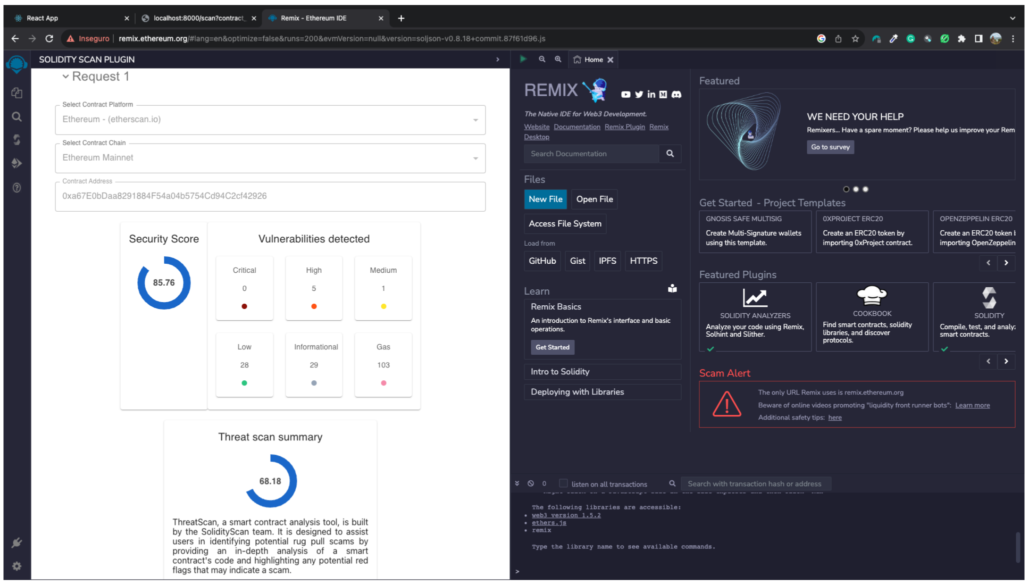
Select Contract Platform: Ethereum - (etherscan.io)
Select Contract Chain: Ethereum Mainnet
Contract Address: 0x8d367375a9f0e0f22d522807296

```
  The following libraries are accessible:
• web3 version 1.5.2
• ethers.js
• remix

  Type the library name to see available commands.

>
```

Figure 5.5: Plugin - Remix request

In Fig. 5.5, it is possible to see the plugin already integrated within the Remix tool. On the left, it is possible to see a white page representing the various order options that aim to be placed. Creating several order lines and deleting them in the image analyzed is possible. Three fields must be filled in for each order: the platform, the contract chain, and the contract address. In other words, it is only possible to analyze published smart contracts.

In Fig. 5.6, it is possible to see the result of the first order executed. Initially, a list of results for all requests is given, in which each request is expandable, allowing the user to have a more detailed view and a deeper analysis of that specific contract. In this image, it is possible to observe that the security score is 85,76. Furthermore, it is possible to observe how many Critical, High, Medium, Low, Informational, and Gas vulnerabilities they have. Then it is possible to analyze a threat scan summary that allows the developer to provide an in-depth analysis of the code of a smart contract and highlight any possible warning signs that could indicate a vulnerability.


SOLIDITY SCAN PLUGIN

Request 1

Select Contract Platform: Ethereum - (etherscan.io)

Select Contract Chain: Ethereum Mainnet

Contract Address: 0xa67E0bDaa8291884F54a04b5754Cd94C2cf42926

Security Score: 85.76

Vulnerabilities detected

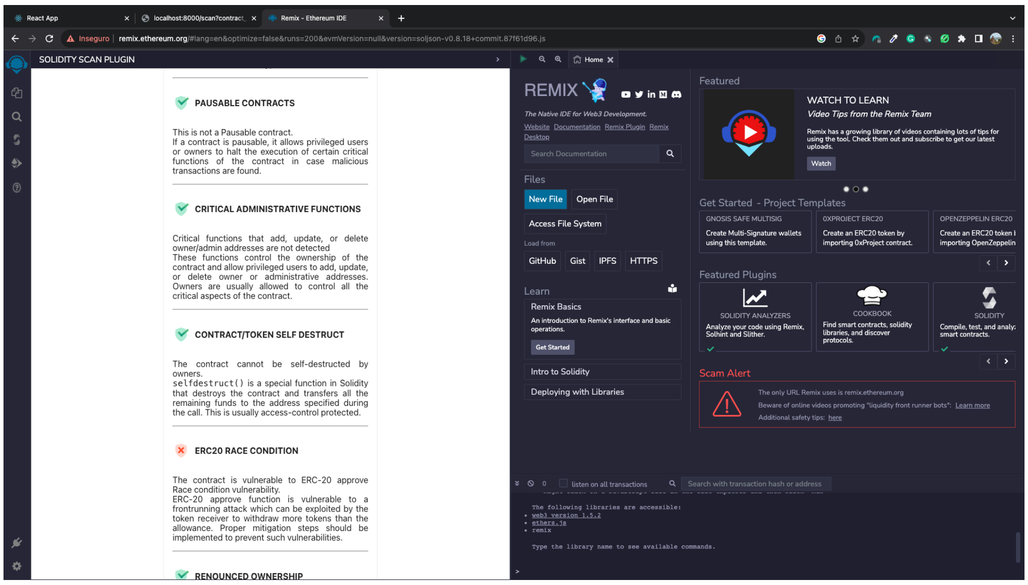
Critical	High	Medium
0	5	1

Low	Informational	Gas
28	29	103

Threat scan summary: 68.18

ThreatScan, a smart contract analysis tool, is built by the SolidityScan team. It is designed to assist users in identifying potential rug pull scams by providing an in-depth analysis of a smart contract's code and highlighting any potential red flags that may indicate a scam.

Figure 5.6: Plugin - Request 1 - Part 1


SOLIDITY SCAN PLUGIN

PAUSABLE CONTRACTS

This is not a Pausable contract.
If a contract is pausable, it allows privileged users or owners to halt the execution of certain critical functions of the contract in case malicious transactions are found.

CRITICAL ADMINISTRATIVE FUNCTIONS

Critical functions that add, update, or delete owner/admin addresses are not detected
These functions control the ownership of the contract and allow privileged users to add, update, or delete owner or administrative addresses. Owners are usually allowed to control all the critical aspects of the contract.

CONTRACT/TOKEN SELF DESTRUCT

The contract cannot be self-destructed by owners.
`selfdestruct()` is a special function in Solidity that destroys the contract and transfers all the remaining funds to the address specified during the call. This is usually access-control protected.

ERC20 RACE CONDITION

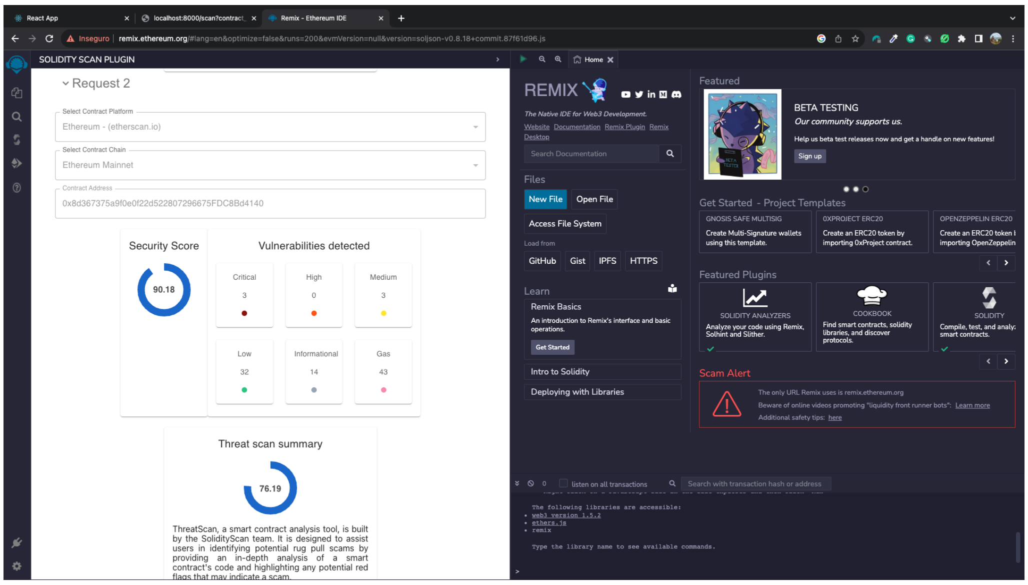
The contract is vulnerable to ERC-20 approve Race condition vulnerability.
ERC-20 approve function is vulnerable to a frontrunning attack which can be exploited by the token receiver to withdraw more tokens than the allowance. Proper mitigation steps should be implemented to prevent such vulnerabilities.

RENOUNCED OWNERSHIP

Figure 5.7: Plugin - Request 1 - Part 2

In Fig. 5.7, it is possible to observe the remaining result of the first request, the continuation of Fig. 5.6. Here, it is possible to notice that the contract is vulnerable to ERC-20 approval Race condition vulnerability. The ERC-20 approve function is vulnerable to a frontrunning attack, which the token receiver can exploit to withdraw more tokens than the allowance. Proper mitigation steps should be implemented to prevent such vulnerabilities.

In Fig. 5.8, it is possible to see the result of the second order performed. The severity score is 90,18.



React App | localhost:8000/scan?contract_ | Remix - Ethereum IDE

Inseguro | remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js

SOLIDITY SCAN PLUGIN

Request 2

Select Contract Platform: Ethereum - (etherscan.io)

Select Contract Chain: Ethereum Mainnet

Contract Address: 0x8d367375a9f0e0f22d522807296675FDC8Bd4140

Security Score: 90.18

Vulnerabilities detected

Critical	High	Medium
3	0	3

Low	Informational	Gas
32	14	43

Threat scan summary

76.19

ThreatScan, a smart contract analysis tool, is built by the SolidityScan team. It is designed to assist users in identifying potential rug pull scams by providing an in-depth analysis of a smart contract's code and highlighting any potential red flags that may indicate a scam.

REMIX

The Native IDE for Web3 Development.

Website Documentation Remix Plugin Remix Desktop

Featured

BETA TESTING

Our community supports us.

Help us beta test releases now and get a handle on new features!

Get Started - Project Templates

GNOSIS SAFE MULTISIG: Create Multi-Signature wallets using this template.

0XPROJECT ERC20: Create an ERC20 token by importing 0xProject contract.

OPENZEPPELIN ERC20: Create an ERC20 token by importing OpenZeppelin

Featured Plugins

SOLIDITY ANALYZERS: Analyze your code using Remix, Solhint and Slither.

COOKBOOK: Find smart contracts, solidity libraries, and discover protocols.

SOLIDITY: Compile, test, and analyze smart contracts.

Learn

Remix Basics: An introduction to Remix's interface and basic operations.

Intro to Solidity

Deploying with Libraries

Scam Alert

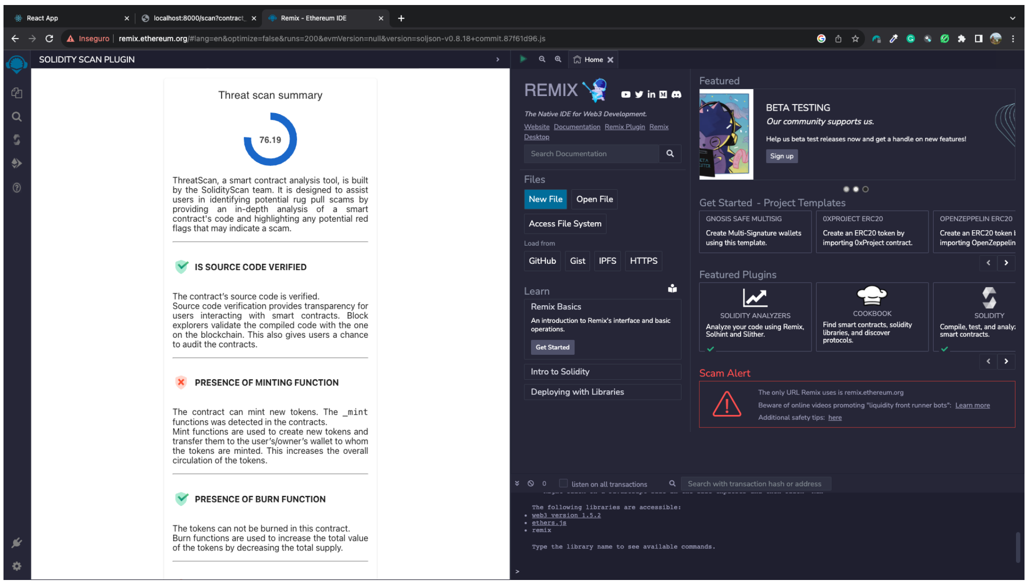
The only URL Remix uses is remix.ethereum.org

Beware of online videos promoting "liquidity front runner bots": Learn more

Additional safety tips: here

```
The following libraries are accessible:
• web3 version 1.5.2
• ethers.js
• remix

Type the library name to see available commands.
```

Figure 5.8: Plugin - Request 2 - Part 1



SOLIDITY SCAN PLUGIN

Threat scan summary

76.19

ThreatScan, a smart contract analysis tool, is built by the SolidityScan team. It is designed to assist users in identifying potential rug pull scams by providing an in-depth analysis of a smart contract's code and highlighting any potential red flags that may indicate a scam.

IS SOURCE CODE VERIFIED

The contract's source code is verified.
Source code verification provides transparency for users interacting with smart contracts. Block explorers validate the compiled code with the one on the blockchain. This also gives users a chance to audit the contracts.

PRESENCE OF MINTING FUNCTION

The contract can mint new tokens. The `_mint` functions was detected in the contracts.
Mint functions are used to create new tokens and transfer them to the user's/owner's wallet to whom the tokens are minted. This increases the overall circulation of the tokens.

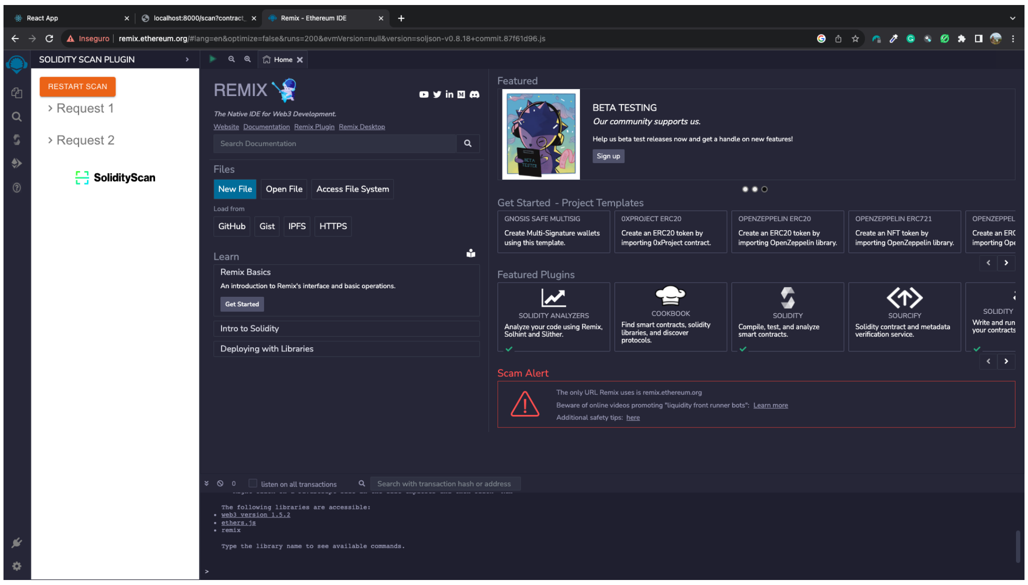
PRESENCE OF BURN FUNCTION

The tokens can not be burned in this contract.
Burn functions are used to increase the total value of the tokens by decreasing the total supply.

Figure 5.9: Plugin - Request 2 - Part 2

In Fig. 5.9, it is possible to check the remaining result of the second request. Here, it is possible to see that the contract is vulnerable to the presence of minting function condition. The contract analyzed can mint new tokens. The *_mint* functions were detected in the contracts. Mint functions are used to create new tokens and transfer them to the user's/owner's wallet to whom the tokens are minted. This increases the overall circulation of the tokens.

In Fig. 5.10, it is possible to see the compressed list of request results. The developer



```
The following libraries are accessible:
• web3 version 1.5.2
• ethers.js
• remix

Type the library name to see available commands.
```

Figure 5.10: Plugin - Review

can then expand each request to review the analysis performed and must press the Restart scan button to start the process again and clear all data.

Chapter 6

Conclusions

In the context of Web3, whose objective is to decentralize the network and give users control over their online activities, it is crucial to recognize the existence of vulnerabilities that can have severe consequences and negatively affect the entire system. This investigation allowed to analyze the number of vulnerabilities reported in smart contracts, presenting total values and trends related to the number of reports, severity, and financial compensation. This analysis shows that over the past few years, there has been a significant increase in the total number of reports and critical reports. There has also been an increase in the amounts paid to security researchers, directly related to the increased availability of smart contracts on the market. This study reveals that security vulnerabilities in Web3 are constantly being discovered and reported. This demonstrates the continued evolution of Web3 and the need to explore solutions to combat these ever-evolving vulnerabilities. The developed plugin significantly contributes to identifying and mitigating vulnerabilities in already published smart contracts. Through its implementation, developers can monitor the status of their already published contracts, understanding their vulnerabilities, if any. This tool not only helps improve security but also promotes trust in Web3.

For future work, a better categorization of the collected reports is recommended, considering the codes' vulnerabilities. In this way, it may be possible to create a classification of the main flaws and, subsequently, an audit checklist for the development of smart contracts. Regarding the plugin, its expansion and improvement can further contribute to security on Web3, improving vulnerability detection and providing an additional layer of protection for decentralized applications.

References

[1] Alchemy. *Web3 Developer Report Q4 2022*. 2022. URL: `https://www.alchemy.com/blog/web3-developer-report-q4-2022` (visited on 01/17/2023).

[2] Omar Ali et al. "A Comparative Study: Blockchain Technology Utilization Benefits, Challenges and Functionalities". In: *IEEE Access* 9 (2021), pp. 12730–12749. DOI: `10.1109/ACCESS.2021.3050241`.

[3] Asaph Azaria et al. "MedRec: Using Blockchain for Medical Data Access and Permission Management". In: *2016 2nd International Conference on Open and Big Data (OBD)*. 2016, pp. 25–30. DOI: `10.1109/OBD.2016.11`.

[4] Connor Brooke. *Who Accepts Bitcoin as Payment? 13 Companies and Websites That Accept Cryptocurrency*. 2023. URL: `https://www.techopedia.com/cryptocurrency/who-accepts-bitcoin`.

[5] Bin Cao, Zheng Yan, and Xu Xia. "Web3". In: *IEEE Communications Magazine* 61.8 (2023), pp. 18–19. DOI: `10.1109/MCOM.2023.10230032`.

[6] Longbing Cao. "Decentralized AI: Edge Intelligence and Smart Blockchain, Metaverse, Web3, and DeSci". In: *IEEE Intelligent Systems* 37.3 (2022), pp. 6–19. DOI: `10.1109/MIS.2022.3181504`.

[7] Tomer Chaffer and Justin Goldston. "On the Existential Basis of Self-Sovereign Identity and Soulbound Tokens: An Examination of the "Self" in the Age of Web3". In: (Dec. 2022), p. 2022.

[8] Chuan Chen et al. "When Digital Economy Meets Web3.0: Applications and Challenges". In: *IEEE Open Journal of the Computer Society* 3 (2022), pp. 233–245. DOI: `10.1109/OJCS.2022.3217565`.

[9] Huashan Chen et al. "A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses". In: *ACM Comput. Surv.* 53.3 (June 2020). ISSN: 0360-0300. DOI: 10.1145/3391195. URL: https://doi.org/10.1145/3391195.

[10] *CoinMarketCap*. Accessed November 2, 2023. URL: https://coinmarketcap.com/.

[11] Daniel Steven Connelly. "Smart Contract Vulnerabilities on the Ethereum Blockchain: A Current Perspective". PhD thesis. Portland State University, 2020.

[12] Linus Gasser. "WEB3". In: Apr. 2023. URL: https://link.springer.com/chapter/10.1007/978-3-031-33386-6_34.

[13] Chong Guan, Ding Ding, and Jiancang Guo. "Web3.0: A Review And Research Agenda". In: *2022 RIVF International Conference on Computing and Communication Technologies (RIVF)*. 2022, pp. 653–658. DOI: 10.1109/RIVF55975.2022.10013794.

[14] R. Gupta. *Hands-On Cybersecurity with Blockchain: Implement DDoS protection, PKI-based identity, 2FA, and DNS security using Blockchain*. Packt Publishing, 2018. ISBN: 9781788991858. URL: https://books.google.pt/books?id=upBiDwAAQBAJ.

[15] Hideaki Hata, Mingyu Guo, and M. Ali Babar. "Understanding the Heterogeneity of Contributors in Bug Bounty Programs". In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2017, pp. 223–228. DOI: 10.1109/ESEM.2017.34.

[16] Immunefi. "Beanstalk Logic Error Bugfix Review". In: (Feb. 2023). URL: https://medium.com/immunefi/beanstalk-logic-error-bugfix-review-4fea17478716.

[17] Immunefi. "Yield Protocol Logic Error Bugfix Review". In: (July 2023). URL: https://medium.com/immunefi/yield-protocol-logic-error-bugfix-review-7b86741e6f50.

[18] Leila Ismail and Huned Materwala. "A Review of Blockchain Architecture and Consensus Protocols: Use Cases, Challenges, and Solutions". In: *Symmetry* 11.10 (2019). ISSN: 2073-8994. DOI: 10.3390/sym11101198. URL: https://www.mdpi.com/2073-8994/11/10/1198.

[19] Niclas Kannengießer et al. “Challenges and Common Solutions in Smart Contract Development”. In: *IEEE Transactions on Software Engineering* 48.11 (2022), pp. 4291–4318. DOI: 10.1109/TSE.2021.3116808.

[20] Venkat Kasthala. “Blockchain key characteristics and the conditions to use it as a solution”. In: (Aug. 2019). URL: https://medium.com/swlh/blockchain-characteristics-and-its-suitability-as-a-technical-solution-bd65fc2c1ad1.

[21] Robert Leonhard. “Decentralized Finance on the Ethereum Blockchain”. In: *SSRN Electronic Journal* (Mar. 2019). URL: http://dx.doi.org/10.2139/ssrn.3359732.

[22] Livio Pompianu Lodovica Marchesi Michele Marchesi and Roberto Tonelli. “Security checklists for Ethereum smart contract development: patterns and best practices”. In: (Aug. 2020). URL: https://arxiv.org/pdf/2008.04761.pdf.

[23] Gavin Lucas. “Poly Network attack: Here’s what happened to biggest DeFi hack in history”. In: *CoinGeek* (Aug. 2021). URL: https://coingeek.com/poly-network-attack-heres-what-happened-to-biggest-defi-hack-in-history/.

[24] Suresh S. Malladi and Hemang C. Subramanian. “Bug Bounty Programs for Cybersecurity: Practices, Issues, and Recommendations”. In: *IEEE Software* 37.1 (2020), pp. 31–39. DOI: 10.1109/MS.2018.2880508.

[25] Shaurya Malwa. “DeFi Protocol Qubit Finance Exploited for $80M”. In: *CoinDesk* (Jan. 2022). URL: https://www.coindesk.com/markets/2022/01/28/defi-protocol-qubit-finance-exploited-for-80m/.

[26] Nedas Matulevicius and Lucas C. Cordeiro. “Verifying Security Vulnerabilities for Blockchain-based Smart Contracts”. In: *2021 XI Brazilian Symposium on Computing Systems Engineering (SBESC)*. 2021, pp. 1–8. DOI: 10.1109/SBESC53686.2021.9628229.

[27] Rita Melo. “Web3 Cybersecurity”. In: *CyberSec 23*. 24th January, Viana do Castelo, Portugal, 2023.

[28] Rita Melo, Pedro Pinto, and António Pinto. “Severity Analysis of Web3 Security Vulnerabilities based on Publicly Bug Reports”. In: *Blockchain and Applications, 5th International Congress*. (to appear). 12th-14th July, Guimarães, Portugal, 2023.

[29] Oliver Hinz Michael Nofer Peter Gomber and Dirk Schiereck. “Blockchain”. In: 2017.

[30] Marcus Bremseth Moe. “Web3 and its impact on Privacy and Personal Data Management”. In: (2022). URL: https://www.theseus.fi/bitstream/handle/10024/751597/Moe_Marcus_Thesis.pdf?sequence=2&isAllowed=y.

[31] Bhabendu Kumar Mohanta, Soumyashree S Panda, and Debasish Jena. “An Overview of Smart Contract and Use Cases in Blockchain Technology”. In: *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 2018, pp. 1–4. DOI: 10.1109/ICCCNT.2018.8494045.

[32] Ch. V. N. U. Bharathi Murthy et al. “Blockchain Based Cloud Computing: Architecture and Research Challenges”. In: *IEEE Access* 8 (2020), pp. 205190–205205. DOI: 10.1109/ACCESS.2020.3036812.

[33] Ahmed S. Musleh, Gang Yao, and S. M. Muyeen. “Blockchain Applications in Smart Grid–Review and Frameworks”. In: *IEEE Access* 7 (2019), pp. 86746–86757. DOI: 10.1109/ACCESS.2019.2920682.

[34] Shimaa Ouf, Mona Nasr, and Yehia Helmy. “An enhanced e-learning ecosystem based on an integration between cloud computing and Web2.0”. In: *The 10th IEEE International Symposium on Signal Processing and Information Technology*. 2010, pp. 48–55. DOI: 10.1109/ISSPIT.2010.5711721.

[35] Andrew Park et al. “Interoperability: Our exciting and terrifying Web3 future”. In: *Business Horizons* (2022). ISSN: 0007-6813. DOI: https://doi.org/10.1016/j.bushor.2022.10.005. URL: https://www.sciencedirect.com/science/article/pii/S0007681322001318.

[36] Rohini Pise and Sonali Patil. “A Deep Dive into Blockchain-based Smart Contract-specific Security Vulnerabilities”. In: *2022 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*. 2022, pp. 1–6. DOI: 10.1109/ICBDS53701.2022.9935949.

[37] Partha Pratim Ray. “Web3: A comprehensive review on background, technologies, applications, zero-trust architectures, challenges and future directions”. In: *Internet of Things and Cyber-Physical Systems* 3 (2023), pp. 213–248. ISSN: 2667-3452. DOI:

https://doi.org/10.1016/j.iotcps.2023.05.003. URL: https://www.sciencedirect.com/science/article/pii/S2667345223000305.

[38] Muhammad Habib ur Rehman et al. “Trust in Blockchain Cryptocurrency Ecosystem”. In: *IEEE Transactions on Engineering Management* 67.4 (2020), pp. 1196–1212. DOI: 10.1109/TEM.2019.2948861.

[39] Sara Rouhani and Ralph Deters. “Security, Performance, and Applications of Smart Contracts: A Systematic Survey”. In: *IEEE Access* 7 (2019), pp. 50759–50779. DOI: 10.1109/ACCESS.2019.2911031.

[40] Sapna and Deepak Prashar. “Analysis on Blockchain Vulnerabilities & Attacks on Wallet”. In: *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*. 2021, pp. 1515–1521. DOI: 10.1109/ICAC3N53548.2021.9725403.

[41] Daria A. Snegireva. “Review of Modern Vulnerabilities in Blockchain Systems”. In: *2021 International Conference on Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS)*. 2021, pp. 117–121. DOI: 10.1109/ITQMIS53292.2021.9642862.

[42] Kiran Sridhar and Ming Ng. “Hacking for good: Leveraging HackerOne data to develop an economic model of Bug Bounties”. In: *Journal of Cybersecurity* 7.1 (Mar. 2021), tyab007. ISSN: 2057-2085. DOI: 10.1093/cybsec/tyab007. eprint: https://academic.oup.com/cybersecurity/article-pdf/7/1/tyab007/36578302/tyab007.pdf. URL: https://doi.org/10.1093/cybsec/tyab007.

[43] Ruhi Taş and Ömer Özgür Tanrıöver. “Building A Decentralized Application on the Ethereum Blockchain”. In: *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. 2019, pp. 1–4. DOI: 10.1109/ISMSIT.2019.8932806.

[44] Pinyaphat Tasatanattakool and Chian Techapanupreeda. “Blockchain: Challenges and applications”. In: *2018 International Conference on Information Networking (ICOIN)*. 2018, pp. 473–475. DOI: 10.1109/ICOIN.2018.8343163.

[45] Dejan Vujičić, Dijana Jagodić, and Siniša Ranđić. “Blockchain technology, bitcoin, and Ethereum: A brief overview”. In: *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. 2018, pp. 1–6. DOI: 10.1109/INFOTEH.2018.8345547.

[46] J. William. *Blockchain: The Simple Guide to Everything You Need to Know*. CreateSpace Independent Publishing Platform, 2016. ISBN: 9781533161574. URL: https://books.google.pt/books?id=xYauDAEACAAJ.

[47] Kou G. Xu M. Chen X. “A systematic review of blockchain”. In: 2019. DOI: https://doi.org/10.1186/s40854-019-0147-z.

[48] Weiqin Zou et al. “Smart Contract Development: Challenges and Opportunities”. In: *IEEE Transactions on Software Engineering* 47.10 (2021), pp. 2084–2106. DOI: 10.1109/TSE.2019.2942301.

[49] Ziqiang Zuo. “Development, Application, And Regulation of Web3.0”. In: *Frontiers in Business, Economics and Management* 9.3 (June 2023), pp. 22–27. DOI: 10.54097/fbem.v9i3.9431. URL: https://drpress.org/ojs/index.php/fbem/article/view/9431.

Appendices

Appendix A

Code for analysis of the Code4rena platform

```python
import requests
from bs4 import BeautifulSoup
import datetime
import csv

# Part 1: Collect the links
url = "https://code4rena.com/reports/"
response = requests.get(url)
page_content = response.content
soup = BeautifulSoup(page_content, "html.parser")
links = []

wrapper_report = soup.find("div", class_="wrapper-report")
if wrapper_report:
    wrapper_contests = wrapper_report.find_all("div", class_="wrapper-contest undefined")
    for wrapper_contest in wrapper_contests:
        wrapper_contest_content = wrapper_contest.find("div", class_="wrapper-contest-content")
        if wrapper_contest_content:
            a = wrapper_contest_content.find("a")
            if a:
                links.append("https://code4rena.com/" + a["href"])
```

```python

# Part 2: Analyze each collected link and export to CSV
results = []

for link in links:
    response = requests.get(link)
    soup = BeautifulSoup(response.content, 'html.parser')
    report_header = soup.find("div", class_="report-header")

    title = None
    date = None

    if report_header:
        h1 = report_header.find("h1")
        if h1:
            title = h1.text
        h4 = report_header.find("h4")
        if h4:
            date = datetime.datetime.strptime(h4.text, "%Y-%m-%d").strftime
                ("%d/%m/%Y")

    div = soup.find('div', class_='report-container')
    if div:
        div_text = div.text.lower()
        category = ""
        if "high risk findings" in div_text:
            category += " High "
        if "there were no high risk findings identified in this contest" in
             div_text:
            category = category.replace(" High", "")
        if "medium risk findings" in div_text:
            category += " Medium "
        if "low risk and non-critical issues" in div_text:
            category += " Low "
            category += " Non-Critical "
        if "low risk findings" in div_text:
            category += " Low "
        if "non-critical findings" in div_text:
            category += " Non-Critical "

        category_list = []

        if "Low" in category:
            category_list.append("Low")
        if "Medium" in category:
            category_list.append("Medium")
        if "High" in category:
            category_list.append("High")
        if "Non-Critical" in category:
            category_list.append("Non-Critical")

        category_str = ", ".join(category_list)


# Export the results to a CSV file
with open('results.csv', 'w', newline='') as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(['Title', 'Date', 'Category', 'URL'])
    csv_writer.writerows(results)
```