blockchains

MDPI

Review

A Review of Approaches for Detecting Vulnerabilities in Smart Contracts within Web 3.0 Applications

Hui Li *, Ranran Dang, Yao Yao and Han Wang *

Shenzhen Graduate School, Peking University, Shenzhen 518055, China
* Correspondence: lih64@pkusz.edu.cn (H.L.); wanghan2017@pku.edu.cn (H.W.)

Abstract: Smart contracts, programs running on a blockchain, play a crucial role in driving Web 3.0 across a variety of domains, such as digital finance and future networks. However, they currently face significant security vulnerabilities that could result in potential risks and losses. This paper outlines the inherent vulnerabilities of smart contracts, both those typical of their applications and those unique to Web 3.0 applications. We then systematically classify the techniques based on their core approach to detecting vulnerabilities in smart contracts. Using these approaches, we conduct a comparative analysis of existing tools in terms of their vulnerability coverage, detection effectiveness, open-source availability, and integration capabilities. Finally, we present the Co-Governed Sovereignty Multi-Identifier Network (CoG-MIN) as a case study to demonstrate the significance of smart contract application security in establishing a community with a shared future in cyberspace during the Web 3.0 era and anticipate future research directions with challenges. To conclude, this study addresses the gap in integrating existing smart contract security research with the advancement of Web 3.0 development, while also providing recommendations for future research directions.

Keywords: smart contract; vulnerability detection; Web 3.0; Co-Governed Sovereignty Multi-Identifier Network; community with a shared future in cyberspace

Citation: Li, H.; Dang, R.; Yao, Y.; Wang, H. A Review of Approaches for Detecting Vulnerabilities in Smart Contracts within Web 3.0 Applications. *Blockchains* **2023**, *1*, 3–18. https://doi.org/10.3390/blockchains1010002

Academic Editors: Keke Gai and Liehuang Zhu

Received: 30 June 2023
Revised: 22 August 2023
Accepted: 22 August 2023
Published: 23 August 2023

1. Introduction

The concept of blockchain technology was initially introduced by Satoshi Nakamoto in Bitcoin [1], a cryptocurrency system. Decentralized transaction records are stored in the blockchain via cryptography to resist tampering. In December 2013, Vitalik Buterin presented the Ethereum white paper, introducing smart contract and enabling the development of a blockchain system capable of handling general value and functioning as a distributed transaction-based state machine [2]. Due to their decentralization and programmability, smart contracts have found extensive applications in various domains, including digital finance and future networks [3,4]. For example, the flash loan uses the execution principle of smart contracts, allowing users to take advantage of arbitrage opportunities in the market to achieve low-cost, high-yield operations [5]. In the context of future networks, the Multi-Identifier System (MIS) [6], operating as the management layer for co-management and Co-Governed Sovereignty Multi-Identifier Network (CoG-MIN) [7,8], utilizes identifier management contracts to enable flexible identifier functions and rule formulation.

Smart contracts come in two main forms: high-level language code and Ethereum Virtual Machine (EVM) bytecode. There are currently many high-level languages that can be used to write smart contracts, the most popular of which is Solidity. The smart contract written by Solidity will first be compiled into EVM bytecode that can be directly accepted by the virtual machine and then sent to Ethereum by the user in the form of a Transaction for smart contract deployment. Additionally, developers have defined mnemonics called opcodes to map the meaning of the bytecode, making it easier to understand. The relationship between these three elements is shown in Figure 1.

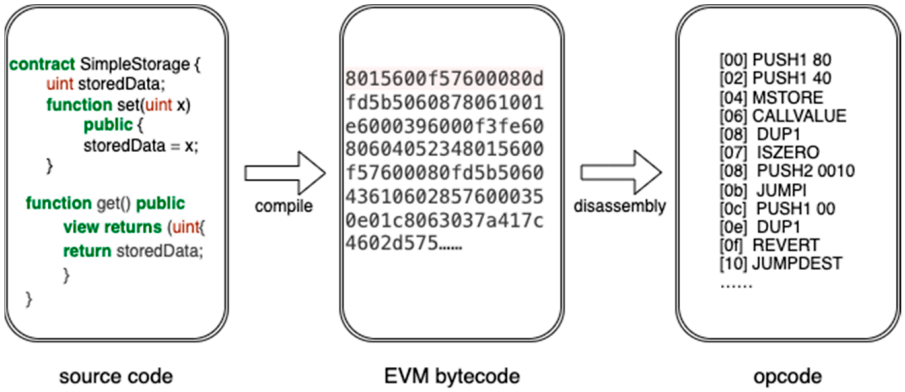

Figure 1. The relationship between the three forms of Ethereum smart contracts.

As the adoption of smart contracts on blockchains has increased, numerous security challenges have emerged. Among them, the most notable incident was "The DAO" [9] in June 2016. In 2022, there were 116 security incidents related to smart contract vulnerabilities, accounting for 27% of all blockchain security incidents, whose losses exceeded $1.7 billion [10]. In fact, due to the transparency of blockchain, the consequences of vulnerabilities in smart contracts are more severe than those in traditional programs. Anyone can access deployed contracts on the chain, allowing attackers to analyze contract bytecode and attempt to exploit any discovered vulnerabilities [11]. Moreover, once a contract is deployed, its owner faces limitations in making repairs unless he implements an upgradeable write mode.

Considering the difficulty in expecting developers to create completely secure contracts, extensive research efforts have been dedicated to vulnerability detection techniques for smart contracts. After the DAO attack in 2016, the research on smart contract security has witnessed significant growth year by year. Yamashita et al. [12] collected and individually classified a variety of common vulnerability patterns that could compromise the security of smart contracts. Praitheeshan et al. [13] investigated the vulnerability detection technology of smart contracts, introduced and compared the various characteristics of some smart contract vulnerability detection technologies. By conducting a systematic review and an analysis of the research progress in smart contract vulnerability detection technology, it was observed that, during 2019 and 2020, there was rapid development in this field. Various vulnerability detection methods were proposed, such as fuzz testing, taint analysis, formal verification, and machine learning, specifically applied to smart contract vulnerability detection. These vulnerability detection technologies have garnered extensive attention and research within the realm of smart contract security.

The aforementioned studies concerning smart contract vulnerability detection techniques have not comprehensively encompassed the diverse range of security vulnerabilities in smart contracts emerging within the Web 3.0 era [14]. Additionally, an issue of the inadequate comprehensive analysis of detection technology information exists [15]. Our investigation encompasses the progress and countermeasures of smart contract vulnerabilities in the current context of Web 3.0 development. We extensively analyze the disparities of existing technologies in terms of vulnerability coverage, detection effectiveness, open-source availability, and integration capabilities. We firmly believe that this endeavor contributes to a more comprehensive understanding, from the Web 3.0 perspective, of the existing smart contract vulnerability detection technologies among researchers.

The rest of this paper is organized as follows. Section 2 introduces traditional smart contract vulnerabilities at the levels of Solidity, EVM, and the block, and further explores vulnerabilities in the context of Web 3.0 advancements. Section 3 discusses common methods for smart contract vulnerability detection. Section 4 presents an overview of mainstream vulnerability detection tools, followed by a comparison. Section 4 takes CoGMIN as an example to demonstrate the necessity of smart contract application security in the Web 3.0 era for building a community with a shared future in cyberspace. Finally, the

paper concludes by summarizing the findings, discussing limitations in existing research, and providing suggestions for future investigation in Section 6.

2. Vulnerability in Smart Contracts

In this section, we present an overview of traditional smart contract vulnerabilities, focusing on three distinct layers: the Solidity layer, the EVM layer, and the block layer, as observed in Ethereum [16]. Additionally, we discuss unique vulnerabilities that arise in the context of Web 3.0 applications, taking into account their respective characteristics.

2.1. Solidity Layer

- Reentrancy

The Reentrancy Attack, a type of security vulnerability targeting smart contracts, exploits the properties of reentrant functions within the contract [17]. This attack manipulates the execution sequence of the contract by repeatedly invoking the function of another contract or external address during its execution. Through improper means, the attacker can gain unauthorized access to additional assets or execute malicious operations.

- Integer Error

Integer errors in smart contracts encompass arithmetic errors, truncation errors, and sign errors [18]. Arithmetic errors encompass situations such as integer overflow, division by zero, and modulus by zero. Similar to other programming languages, Solidity defines fixed-length representations for integers within a specified range. When the result of an integer operation exceeds this range, an integer overflow occurs.

- Exception Handling

Exception handling vulnerabilities in smart contracts pertain to flaws or inadequate design in contract implementations when dealing with abnormal or erroneous conditions, leading to security risks or unexpected outcomes [19]. When a smart contract encounters exceptional circumstances, if the error handling mechanism is inadequate or contains vulnerabilities, attackers can exploit these vulnerabilities to execute malicious activities or disrupt the normal operation of the contract [20].

- Logical Error

The logical error in a smart contract pertains to design or implementation errors that result in the contract exhibiting unexpected or undesired behavior under specific conditions [21]. These bugs arise from flaws in the logical structure or reasoning within the contract, leading to inconsistencies or unexpected outcomes during contract execution.

2.2. EVM Layer

- Short Address

Short Address Attack [22,23] is a vulnerability where attackers exploit the characteristic of address encoding algorithms to ignore the trailing characters of the encoded string. By constructing a specific encoded string that completely matches the prefix portion of a legitimate address, attackers deceive users into using the address controlled by the attacker when sending funds or performing operations. This malicious action results in economic losses or abnormal contract functionalities.

- Tx.origin

Tx.origin [24] is a global variable utilized to retain the address of the external account responsible for triggering the ongoing transaction. It signifies the genuine initiator of the transaction, specifically the account address that initiated the contract call. Exploiting the Tx.origin vulnerability, an attacker can simulate the intended contract caller by leveraging Tx.origin and establishing an intermediary contract. Through this manipulation, the attacker can execute a Tx.origin vulnerability attack.

- Call-Stack Overflow

Call-Stack Overflow occurs when recursive calls or improper utilization of local variables lead to the exhaustion of the available call stack [25]. Attackers can exploit this vulnerability to launch attacks or manipulate contracts.

2.3. Block Layer

- Timestamp Dependency

Timestamps are frequently employed to capture the creation and execution time of transactions or contracts [26]. Attackers take advantage of a contract's reliance on future timestamps to disrupt the execution sequence or alter the outcome by manipulating the system time or exerting control over the generation of future blocks.

- Transaction Order Dependency

Transaction order dependency vulnerabilities, commonly referred to as transaction ordering attacks, pertain to security vulnerabilities within blockchain systems. These vulnerabilities arise from the inherent uncertainty in the processing order of transactions, which attackers exploit to gain undue advantages or execute malicious operations [27].

2.4. Web 3.0 Vulnerabilities

Web 3.0 aims to establish a network ecosystem grounded in blockchain technology, user-centricity, shared data, and decentralization. However, network security risks have also become more pronounced within this context. The subsequent section outlines several vulnerabilities that are distinctive to Web 3.0 applications.

- Identifier Verification

The identifier management [28] contract primarily performs write operations related to identifiers, such as registration, update, renewal, revocation, restoration, and deletion. Each write operation necessitates verification of the source's identity within the multi-identity management system or registration of a username. However, an attacker can exploit this system by utilizing a non-identifying owner's address as the destination address for transfers, thereby facilitating fund theft.

- Rent Tampering

During the registration and renewal process of the identifier, users are required to pay rent, while deleting the identifier results in a return of the remaining rent to the owning address [29]. The rent value is directly proportional to the lease duration. Exploiting the ability to tamper with the lease time, an attacker can extend the token's validity period without paying rent, thereby profiting from a rent amount surpassing the intrinsic value of the token.

- Single Oracle

DeFi (Decentralized Finance) encompasses a collection of financial applications developed on an open, decentralized platform, where the entire business process is conducted through on-chain interactions. Flash loans represent a relatively new form of unsecured lending in the DeFi ecosystem, allowing users to borrow funds from on-chain liquidity pools on the condition that they repay the borrowed amount along with a small transaction fee within the same transaction [30]. However, the vulnerability of flash loan products lies in their dependence on a single oracle, which exposes them to price manipulation risks. Attackers exploit this vulnerability by employing substantial funds to purchase specific tokens and artificially inflate their prices within a short period [31]. By manipulating the token market, attackers secure arbitrage opportunities for their own gain.

- Sandwich Attack

The attacker in a Sandwich Attack exploits price or status fluctuations to interpose their own transaction between two trades, thus obtaining undue economic gains. This

form of attack is commonly observed in decentralized exchanges (DEX) and other smart contract platforms [32]. Sandwich Attacks may lead to traders executing transactions under unfavorable prices or conditions, resulting in financial losses. This attack leverages the delay in transaction execution and the inherent uncertainty in the order of transactions within the transaction pool, enabling attackers to swiftly gain profits [33].

3. Taxonomy of Approaches to Detecting Vulnerabilities

In this section, we conduct a comprehensive review and analysis of the pertinent literature on smart contract vulnerability detection technology with a careful selection of representative detection approaches. Based on their core methodologies, these approaches can be classified into four categories: formal verification, symbolic execution, fuzzing, and taint analysis.

3.1. Formal Verification

Formal Verification is a mathematical and logic-based method employed to rigorously verify the correctness of computing systems, software, or hardware [34]. As shown in Figure 2, the formal verification method is an effective means of deterministic verification for smart contracts. By using formal languages, the concepts, judgments, and reasoning in smart contracts can be transformed into smart contract models, eliminating the ambiguity and lack of generality in natural language. Formal tools are then employed to model, analyze, and verify smart contracts, conduct semantic consistency testing, and ultimately generate verified contract codes. This method offers a comprehensive analysis of all potential states and execution paths within the system.

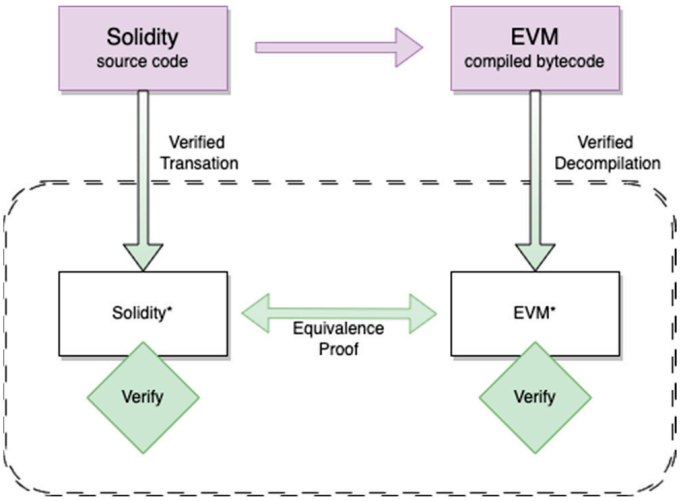

Figure 2. The conversion process of solidity code and EVM bytecode in formal verification.

- F* Framework

Bhargavan et al. made significant contributions by endeavoring to formalize Ethereum instructions using the functional programming language F* [35]. They employed interactive proof functions to conduct program verification within this language.

- EthIR

Albert et al. presented EthIR [36], an Ethereum bytecode analysis framework. They devised a process to decompile the bytecode into a rule-based representation (RBR), enabling the construction of a control flow graph for the smart contract.

3.2. Symbolic Execution

The symbolic execution method executes the bytecode instructions of the smart contract in a symbolic form, constructs a symbolic execution path, and analyzes the symbolic

constraints on the path to find potential vulnerabilities [37]. As shown in Figure 3 below, using symbolic execution technology for vulnerability analysis, the program code is first analyzed to obtain an intermediate representation of the program code. Next, the control flow graph and call graph that describe the program path are constructed. Finally, the vulnerability analysis is carried out. The analysis process mainly includes two parts: symbolic execution and constraint solving, which are executed alternately.

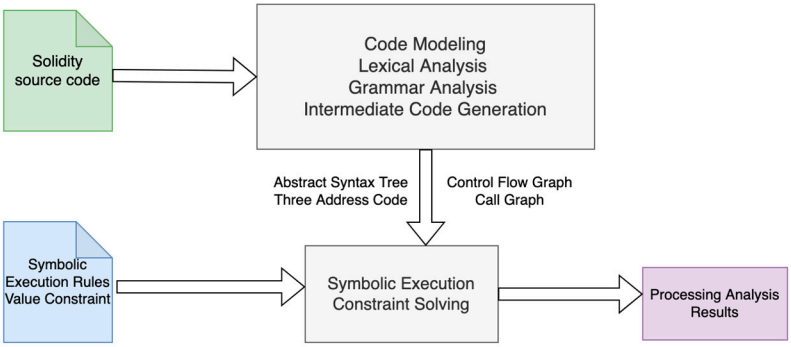

Figure 3. The process of symbolic execution technology for vulnerability analysis.

- Oyente

Oyente [38], proposed by Luu et al., was the first technology for detecting vulnerabilities in smart contracts and the first to utilize symbolic execution for smart contract vulnerability detection. It analyzes the symbolic state and symbolic paths based on a set of predefined attributes for the four types of vulnerabilities. By doing so, it detects security vulnerabilities in smart contracts and conducts reachability inspection using the obtained constraints to reduce false positive rates.

- teEther

Krupp et al. proposed teEther [39], a vulnerability detection and exploitation technology based on symbolic execution. The tool identifies the critical paths leading to four specific sensitive instructions from the control flow graph constructed from bytecode. It then performs symbolic execution starting from the root node of the control flow graph to obtain path constraints for these critical paths. Finally, teEther utilizes the Z3 constraint solver to solve the combined constraints of the critical paths and state change paths, generating exploit samples and detecting vulnerabilities in the contract.

3.3. Fuzzing

Fuzzing is a dynamic analysis technique that explores the contract's response to abnormal or malicious input by performing random or semi-random mutation operations on smart contract inputs, thereby discovering possible vulnerabilities [40]. Figure 4 depicts the main processes of traditional fuzzing tests. The working process is composed of four main stages, the test case generation stage, test case running stage, program execution state monitoring, and analysis of exceptions. Fuzzing does not depend on the source code of the contract, but analyzes the execution results based on the input, so it can be applied to smart contracts without source code or binary files of contracts [41].

- ContractFuzzer

ContractFuzzer [42] proposed by Jiang et al. is the first smart contract vulnerability detection technology using the fuzzing method. It first performs static analysis on the application binary interface (ABI), the bytecode of the smart contract, and then conducts a test and analyzes the information recorded by the EVM during the execution of the contract to detect vulnerabilities.

- sFuzz

Nguyen et al. proposed sFuzz [43], an adaptive fuzzing technology based on feedback guidance, for the problem of generating efficient test cases for smart contracts. The sFuzz first initializes the test input from the existing contract transaction information, then monitors the execution process of the initial test, and combines the adaptive function of AFL with a lightweight multi-objective search strategy according to the feedback information to optimize the test.

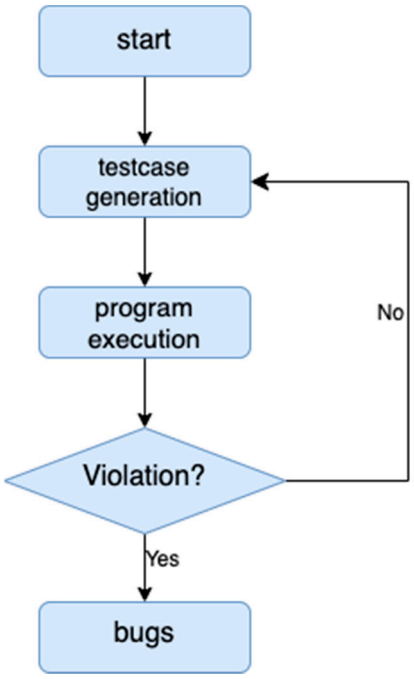

Figure 4. The main processes of traditional fuzzing.

3.4. Taint Analysis

Taint Analysis is a static analysis technique that identifies potential sources of vulnerabilities and code paths that may be affected by tracking and analyzing the propagation of taints in data streams [44]. The process of taint analysis can be divided into three stages, as shown in Figure 5. Firstly, it is necessary to identify the sources of tainted data and the points of taint convergence. Then, the propagation paths of tainted data in the program are analyzed. Finally, the data is sanitized to reduce the number of taint marks in the system. However, the coverage of taint analysis is limited, and it may fail to detect certain specific vulnerabilities [45]. Therefore, its primary role lies in achieving more precise data flow analysis, often requiring integration with other techniques.

- Sereum

The earliest application of taint analysis methods for smart contract vulnerability detection was introduced by Rodler et al., known as Sereum [46]. Sereum is a technology aimed at safeguarding contracts from reentrancy attacks on the extended EVM client. It leverages the K-framework as a symbolic execution engine and deduces vulnerability conditions on the execution path by collecting and constraining symbolic values throughout the execution process.

- Ethainter

Brent et al. proposed Ethainter [47], a harmless handling technique for capturing compound vulnerability data using taint analysis. Ethainter leverages taint analysis to abstract the transfer, loading, and storing of variables between operations in an abstract language onto persistent storage. It then captures the compound vulnerability data by applying predefined information flow rules.

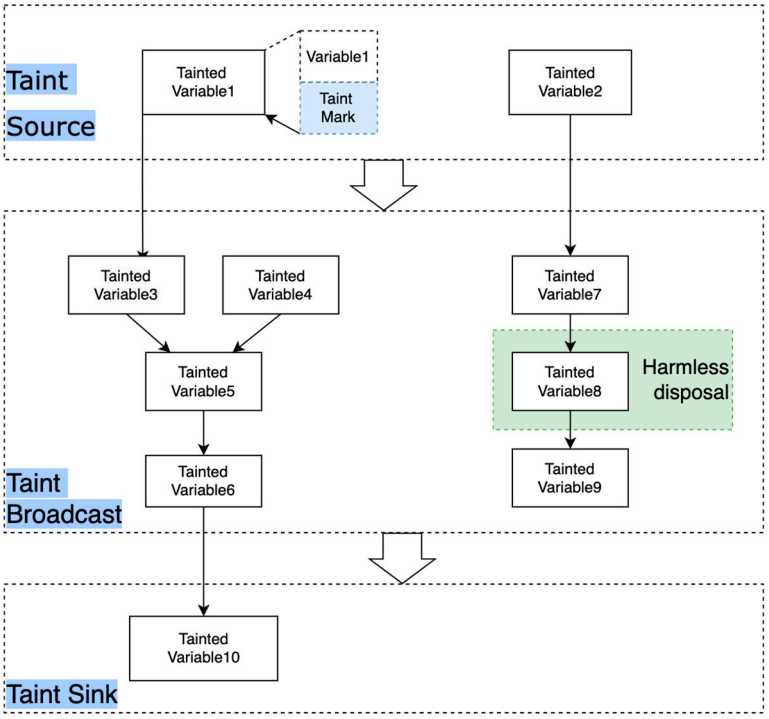

Figure 5. Schematic diagram of three steps in the taint analysis process.

4. Smart Contract Vulnerability Detection Tools and Comparison

Drawing upon the classification of existing smart contract vulnerability detection methods outlined in Section 3, this chapter introduces several relevant tools. Moreover, a comprehensive comparison is conducted among these tools, taking into consideration factors such as vulnerability coverage, detection accuracy, availability of open-source information, and integration capabilities. Furthermore, an analysis is performed to examine the variations in efficiency observed in smart contract vulnerability detection across these tools.

4.1. Enumeration of Smart Contract Vulnerability Detection Tools

We conducted an extensive survey of the current smart contract vulnerability detection tools and selected the following five most representative tools for a detailed introduction.

- Vaas

Vulnerability as a Service (Vaas) [48] is a cloud-based service model for smart contract vulnerability scanning and analysis. Users can submit their own developed smart contracts to the Vaas platform, which performs static code analysis to examine the presence of known vulnerability patterns, coding errors, or potential security issues within the contracts. Additionally, dynamic execution is conducted on the contracts, simulating different execution paths and inputs to observe their behavior and state changes, aiming to detect any vulnerabilities or abnormal behavior that may exist.

- Mythril

Mythril [49] is an intelligent contract security tool based on EVM bytecode developed by ConSensys. It is designed to analyze smart contracts on EVM-compatible blockchains, such as Ethereum, Hedera, Quorum, Vechain, Roostock, and Tron. Mythril employs a combination of taint analysis, SMT solving, and symbolic execution techniques to identify vulnerabilities in smart contract code. Over time, Mythril has emerged as one of the most popular Ethereum smart contract security analysis tools.

- Securify

Securify [50] is a tool used for the secure analysis of Ethereum contracts, capable of verifying the security of contracts for given properties. It examines the compliance and

security vulnerabilities of contracts by analyzing the contract's dependency graph and extracting precise semantic information from the code. The security analysis process of Securify involves two main steps. Firstly, it performs a symbolic analysis of the contract's dependency graph and extracts semantic information from the code. Secondly, it checks for compliance and violation patterns to obtain sufficient conditions, thereby proving the validity of the given properties. Securify offers advantages such as scalability, full automation, and high accuracy.

- Manticore

Manticore [51] is an open-source framework for dynamic symbolic execution, specifically designed for analyzing binary files and Ethereum smart contracts. Its core engine component makes certain assumptions about the underlying execution model. The native binary symbolic execution module implements the high-level execution interface expected by the core engine.

- Slither

Slither [52] is a static analysis framework for smart contracts that encompasses over 30 vulnerability detection models. It is capable of detecting code optimization issues that might have been overlooked by compilers and provides optimization recommendations. Additionally, Slither has the ability to generate visual representations such as inheritance topology diagrams and method invocation graphs, which help developers comprehend the code structure and relationships.

4.2. Comparison of Existing Tools

The following provides a comparative analysis of the aforementioned smart contract detection tools. Table 1 presents 16 popular tools and compares their properties of vulnerability coverage, detection effectiveness, open-source availability, and integration capabilities.

Table 1. Comparison of smart contract vulnerability detection tools.

Detection Tool	Detection Approach	Supported Type of Vulnerabilities									Open-Source Language	Detection Accuracy
		Solidity Layer			EVM Layer			Block Layer				
		Reentrancy	Integer Error	Exception Handling	Logical Error	Short Address Attack	Tx.origin	Call-Stack Overflow	Timestamp Dependency	Transaction Order Dependency		
F* framework	Formal Verification	√	√	√	-[1]	-	-	-	√	√	-	Medium
EthIR		√	√	√	-	-	-	-	√	√	Python	Medium
EthBMC		√	-	-	-	-	-	-	√	√	Python	High
Oyente	Symbolic Execution	√	√	√	-	√	-	√	√	√	Python	Low
teEther		√	√	√	√	√	-	-	-	-	Python	Medium
Slither		√	-	√	-	-	√	√	√	-	Python	High
Manticore		√	√	-	-	√	-	-	-	-	Python	Medium
ContractFuzzer	Fuzzing	√	√	-	-	√	-	-	√	-	Go	Medium
sFuzz		√	√	-	√	√	-	-	√	-	C++	Medium
Harvey		√	√	-	√	√	-	-	√	-	Solidity	Medium
Sereum	Taint Analysis	√	√	√	-	-	-	-	√	√	Java	High
Ethainter		√	√	-	-	-	-	-	√	-	Solidity	High
Vaas	Integrated	√	√	√	-	-	√	-	√	-	Solidity	High
Mythril		√	√	-	-	-	√	-	√	-	Python	High
Securify		√	-	√	√	√	√	√	√	√	Solidity	Medium
Mythx		√	√	-	-	-	√	-	√	-	python	High

[1] "-" indicates that the method cannot detect the corresponding vulnerability, or is not open source.

The detection status presented in Table 1 is derived from the detection experimental results provided by the respective original authors of each tool in their respective papers, and these findings have been collated and synthesized. Additionally, certain papers include comparisons among various tools. For instance, the paper discusses comparisons of vulnerability coverage and accuracy between "oyente" and other tools that build upon its improvements. This article utilizes these comparative analyses as references to supplement the table.

4.2.1. Vulnerability Coverage

Vulnerability coverage pertains to the tool's ability to detect and identify various types of smart contract vulnerabilities, including, but not limited to, integer overflows, uninitialized variables, permission control issues, and reentrancy attacks. An excellent vulnerability detection tool should offer a broad coverage of vulnerabilities, thereby comprehensively identifying potential security issues and enhancing the overall security of smart contracts.

From the perspective of vulnerability types, most detection tools support the detection of vulnerabilities that have caused significant contract attack incidents, including reentrancy vulnerabilities, integer error vulnerabilities, Ethereum freeze vulnerabilities, and others. However, for less frequent and easily preventable vulnerabilities such as permission control, denial of service, and short address vulnerabilities, there are relatively fewer tools available for their detection. Among the commonly detected and easily detectable vulnerabilities, most detection tools support the detection of short address vulnerabilities.

4.2.2. Detection Effectiveness

Detection effectiveness refers to the tool's accuracy and precision in identifying vulnerabilities. An effective vulnerability detection tool should minimize false positives and false negatives, providing specific and accurate vulnerability reports. Such tools enable developers to swiftly identify and address potential security issues, thereby bolstering the security of smart contracts.

Oyente was the first tool to utilize symbolic execution for identifying potential security vulnerabilities. Among 19,366 Ethereum contracts analyzed, it classified 8833 contracts as vulnerable. However, the tool exhibited a relatively high rate of false positives in its detection results. MAIAN [53], a dynamic symbolic executor, was specifically designed to detect self-destructing contracts. It employed inter-procedural symbolic analysis and concrete validation to uncover real vulnerabilities. Analyzing nearly one million contracts, MAIAN successfully reproduced real vulnerabilities with an 89% true positive rate on a subset of 3759 contracts, resulting in the identification of vulnerabilities in 3686 contracts.

The teEther tool combined binary slicing and symbolic execution to examine execution paths containing vulnerable instructions. It generated exploit samples and successfully analyzed 85.65% of the 784,344 accounts, reporting 1532 vulnerable accounts. On the other hand, ETHBMC [54] served as a symbolic execution-based automatic analysis framework for smart contracts. In comparison to teEther, ETHBMC identified an additional 10.3% of vulnerable accounts and 22.8% more vulnerabilities within a shorter time frame. Furthermore, ETHBMC was capable of identifying false positives in teEther and revealed additional vulnerabilities when compared to MAIAN.

4.2.3. Open-Source Availability

Open-source availability concerns whether the tool is open source and widely used by smart contract developers and auditors. Open-source tools offer higher transparency and credibility, allowing more individuals to contribute to their improvement and maintenance.

In the discussed section regarding smart contract vulnerability detection techniques, some of the detection tools provide the technical source code, while others do not. However, they offer web interfaces for utilizing the respective techniques. Smart contract developers can assess the security performance of their smart contracts on these web pages.

In the selection of development languages, the main application is Python, while some tools utilize Go, C++, and Solidity. The primary reasons for choosing a development language include language compatibility, ease of use, performance requirements, and developers' familiarity. Different languages can provide different functionalities and features. For instance, Python offers a rich library and tool ecosystem, facilitating tasks such as formal verification, symbolic execution, and vulnerability detection in smart contracts. Solidity, designed specifically for writing Ethereum smart contracts, possesses the capability to directly analyze the code structure and logic of smart contracts.

4.2.4. Integration Capabilities

Based on the above discussions, it is evident that integrated tools exhibit more prominent performance in terms of vulnerability coverage and detection accuracy. Detection tools that rely on a single detection method have certain limitations, such as the low path coverage in fuzzing [55], the path explosion in symbolic execution [56], and the challenges of over-tainting and under-tainting in taint analysis [57]. Integrated tools effectively integrate each method's strengths, address their deficiencies, and enhance the overall detection performance. For instance, EthPloit [58] integrates taint analysis and fuzz testing techniques. By establishing the dependency relationship between variable data and variable control flow in the source code through taint analysis, EthPloit further enhances the fuzzing test cases based on this dependency relationship. As a result, the path coverage of fuzzing and the efficiency of vulnerability discovery are improved.

5. Future Directions and Challenges in Web 3.0

Although blockchain technology possesses tremendous potential, security remains an unavoidable concern for such an automated, decentralized, and constantly evolving system [59]. Furthermore, as the exploration of the Web 3.0 ecosystem deepens, the developmental trend of the Internet will be a future network centered around individuals, supporting diverse identities, and featuring multi-party governance [60]. In the following, we use the future network domain as an example to elucidate the application of vulnerability detection in Web 3.0. Lastly, we delve into the challenges of vulnerability detection as one of the primary protective measures for constructing the underlying blockchain technology of the Web 3.0 era, focusing on aspects such as accuracy, efficiency, and adaptability to emerging vulnerabilities.

5.1. Community with a Shared Future in Cyberspace

With the development of blockchain technology in the future internet domain, the establishment of a co-governed network space community has become an inevitable trend in the era of Web 3.0. The CoG-MIN is proposed as a novel future network that centers on identity and supports the coexistence of multiple identifiers, including content, service, geographic location, and IP address, etc. The MIS is responsible for generating and managing various identifiers, storing the operation logs of users, issuing translation tables to MIR, and managing blockchain nodes [61]. The management functionality of multiple identifiers, such as identity, content, IP, and domain name in CoG-MIN, is implemented in the EMIS contract and various identifier space contracts. These functionalities include binding user identifiers to their real identities, verifying the publication of user identifiers, managing user public keys and certificates, as well as registering, modifying, revoking, resolving, and translating various identifiers.

Currently, a smart contract vulnerability detection system has been deployed within the MIS. It supports the simultaneous detection of specific vulnerabilities in the identifier management contracts as well as common vulnerabilities in general contracts, thus providing a reliable security guarantee for cyberspace constructed in CoG-MIN.

Considering the diverse characteristics of future network scenarios, CoG-MIN is proposed to achieve flexible and unified management of multiple identifiers, and its smart contract security is ensured through a smart contract vulnerability detection system, which supports simultaneous detection of special vulnerabilities in identity management contracts and six common vulnerabilities in general contracts, so as to provide reliable and efficient security for identity management contracts before the chaining review. Combining the advantages of symbolic execution and machine learning methods, not only reduces the contract detection time but also improves the accuracy and interpretability of vulnerability detection results.

Smart contracts, as the infrastructure of cyberspace, directly influence the security and trustworthiness of the network environment. By ensuring the security of smart contracts, malicious attacks, data breaches, and contract vulnerabilities can be prevented, thus

maintaining the stability and reliability of cyberspace [62]. The security of smart contracts also involves protecting user rights, ensuring the fairness and traceability of transactions, and contract execution [63]. Only by establishing a secure and trusted environment for smart contracts can all parties be encouraged to participate and collaborate, achieving interconnectedness and common development in cyberspace, and building a community with a shared future in cyberspace.

5.2. Challenges and Discussions

The application of smart contract vulnerability detection technologies reveals that their development is still in the early exploration stage, with certain limitations and challenges to address.

1. **Evolution of vulnerabilities:** With the development of Web 3.0, the number and complexity of smart contract platforms and protocols will continue to increase, leading to more potential vulnerabilities and security risks. Therefore, smart contract security detection techniques need to constantly evolve and adapt to the characteristics and functionalities of emerging platforms and protocols. For instance, Ethereum introduced a new token standard, ERC777, which allows fallback functions to be invoked during token transfers. Due to developers' misunderstandings regarding the new features of ERC777, a new form of reentrancy vulnerability emerged, resulting in substantial financial losses for smart contracts.
2. **Dynamic nature of smart contracts:** Smart contracts often involve interactions and data flows among multiple contracts, including receiving external data and invoking external contracts. This dynamic nature adds complexity to the analysis and increases the number and types of potential vulnerabilities, since the behavior of external interactions is unknown and can lead to security loopholes. Therefore, vulnerability detection techniques need to be able to analyze and understand complex relationships among contracts and accurately identify potential security issues.
3. **Interoperability of smart contracts:** Integration and interoperability of smart contracts with other technologies will also pose challenges. The Web 3.0 ecosystem will include multiple smart contract platforms and blockchain protocols, which may have incompatibilities and security vulnerabilities. Therefore, smart contract security vulnerability detection techniques need to have the capability to work across platforms and protocols to ensure comprehensive security.
4. **Limitations of detection methods:** Most of the current techniques rely on vulnerability detection methods such as fuzz testing, symbolic execution, and formal verification, which themselves have limitations. For example, formal verification methods have advantages in verifying the correctness of smart contracts but are limited by contract size and complexity. Symbolic execution methods can explore different execution paths of contracts but may suffer from path explosion issues, leading to insufficient computational resources for complex contracts. Fuzz testing methods can uncover some implicit vulnerabilities but may have limited effectiveness in complex contract logic and data flow dependencies. Taint analysis methods can trace and analyze potential vulnerability sources in data flows but may not accurately identify and locate all vulnerabilities in complex data flows and interaction patterns.
5. **Resource constraints:** The rapid development of the Web 3.0 field has led to the emergence of numerous small projects and start-ups. However, these entities may face challenges in securing sufficient funds and professionals to conduct comprehensive smart contract security audits.

Secure smart contracts are of utmost importance for establishing trust and reliability in decentralized systems. Within such systems, smart contracts serve as core components responsible for executing various functions and business logic. The presence of loopholes or unsafe code in smart contracts can lead to severe consequences, including fund losses, user information leaks, and service interruptions. Consequently, users' trust in the system may be severely undermined, potentially resulting in user churn and project failure. In the

face of these challenges, the development of smart contract vulnerability detection requires several approaches.

For advancing smart contract vulnerability detection research, the following steps should be taken. Firstly, establishing a unified and comprehensive experimental dataset that covers vulnerability types and smart contract platforms is essential for providing reference data for security testing tools and machine learning model training. Secondly, regularly updating the vulnerability database is necessary, including collecting and organizing known contract vulnerabilities and attack techniques, so that smart contract vulnerability detection tools can identify and detect newly emerging vulnerabilities in a timely manner. Finally, from a systematic perspective, the development and improvement of new smart contract vulnerability detection techniques need to consider factors such as vulnerability detection rate, false positive rate, the exploitability of vulnerabilities, detection time, coverage of vulnerability types, and platform support.

For smart contract developers, the following recommendations can enhance the security posture of smart contracts and effectively mitigate potential vulnerabilities. Smart contract developers and teams should prioritize their training in smart contract security to recognize and prevent common vulnerabilities. Regularly auditing and reviewing smart contract codes is essential to detect any security issues. Additionally, utilizing reputable vulnerability detection tools can expedite issue identification. Collaborating to construct diverse and large-scale smart contract datasets enables more robust vulnerability detection. Lastly, continuous learning and staying updated on the latest developments in smart contract security are crucial for maintaining a secure environment.

6. Conclusions

Smart contracts are one of the most promising technologies, providing a rich, secure, and trusted decentralized application landscape. They align with the practical significance of digital finance and future network. However, the accompanying security issues have severely hindered their development. Smart contract vulnerability detection technology has emerged as a new research hotspot. This paper examines a series of smart contract vulnerability detection techniques proposed by researchers. These techniques are categorized as follows: formal verification, symbolic execution, fuzzing, and taint analysis. The paper also introduces smart contract vulnerability detection tools within each of these five categories. Furthermore, it presents a statistical analysis of the existing tools, covering vulnerability types, open-source information, and integration methods. Finally, taking the network community of the Web 3.0 era as an example, the limitations and potential improvements of existing smart contract vulnerability detection methods are discussed and analyzed.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

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

References

1. Nakamoto, S.; Bitcoin, A. A Peer-to-Peer Electronic Cash System. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 9 June 2023).
2. Buterin, V. A next-generation smart contract and decentralized application platform. *White Pap.* **2014**, *3*, 1–2.
3. Wang, S.; Huang, C.; Li, J.; Yuan, Y.; Wang, F.-Y. Decentralized construction of knowledge graphs for deep recommender systems based on blockchain-powered smart contracts. *IEEE Access* **2019**, *7*, 136951–136961. [CrossRef]
4. Gupta, B.B.; Li, K.-C.; Leung, V.C.; Psannis, K.E.; Yamaguchi, S. Blockchain-assisted secure fine-grained searchable encryption for a cloud-based healthcare cyber-physical system. *IEEE CAA J. Autom. Sin.* **2021**, *8*, 1877–1890.
5. Wang, D.; Wu, S.; Lin, Z.; Wu, L.; Yuan, X.; Zhou, Y.; Wang, H.; Ren, K. Towards a first step to understand flash loan and its applications in defi ecosystem. In Proceedings of the Ninth International Workshop on Security in Blockchain and Cloud Computing, Matsue, Japan, 23–26 November 2021; pp. 23–28.

6. Li, H.; Wu, J.; Xing, K.; Yi, P.; Lan, J.; Ji, X.; Liu, Q.; Chen, S.; Liang, W.; Wei, J. The Prototype of Decentralized Multilateral Co-Governing Post-IP Internet Architecture and Its Testing on Operator Networks. *arXiv* **2019**, arXiv:1906.06901.
7. Li, H.; Wu, J.; Yang, X.; Wang, H.; Lan, J.; Xu, K.; Tan, H.; Wei, J.; Liang, W.; Zhu, F. MIN: Co-governing multi-identifier network architecture and its prototype on operator's network. *IEEE Access* **2020**, *8*, 36569–36581. [CrossRef]
8. Li, H.; Yang, X. *Co-Governed Sovereignty Network: Legal Basis and Its Prototype & Applications with MIN Architecture*; Springer Nature: Berlin/Heidelberg, Germany, 2021.
9. Mehar, M.I.; Shier, C.L.; Giambattista, A.; Gong, E.; Fletcher, G.; Sanayhie, R.; Kim, H.M.; Laskowski, M. Understanding a revolutionary and flawed grand experiment in blockchain: The DAO attack. *J. Cases Inf. Technol.* **2019**, *21*, 19–32. [CrossRef]
10. Cao, X.; Zhang, J.; Wu, X.; Liu, B. A survey on security in consensus and smart contracts. *Peer Peer Netw. Appl.* **2022**, *15*, 1008–1028. [CrossRef]
11. Kushwaha, S.S.; Joshi, S.; Singh, D.; Kaur, M.; Lee, H.-N. Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access* **2022**, *10*, 6605–6621. [CrossRef]
12. Yamashita, K.; Nomura, Y.; Zhou, E.; Pi, B.; Jun, S. Potential risks of hyperledger fabric smart contracts. In Proceedings of the 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Hangzhou, China, 24 February 2019; pp. 1–10.
13. Praitheeshan, P.; Pan, L.; Yu, J.; Liu, J.; Doss, R. Security analysis methods on ethereum smart contract vulnerabilities: A survey. *arXiv* **2019**, arXiv:1908.08605.
14. Zuo, Z. Development, Application, And Regulation of Web3.0. *Front. Bus. Econ. Manag.* **2023**, *9*, 22–27. [CrossRef]
15. Gupta, N.A.; Bansal, M.; Sharma, S.; Mehrotra, D.; Kakkar, M. Detection of Vulnerabilities in Blockchain Smart Contracts: A Review. In Proceedings of the 2023 International Conference on Computational Intelligence, Communication Technology and Networking (CICTN), Ghaziabad, India, 20–21 April 2023; pp. 558–562.
16. Atzei, N.; Bartoletti, M.; Cimoli, T. A survey of attacks on ethereum smart contracts (sok). In Proceedings of the Principles of Security and Trust: 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, 22–29 April 2017; pp. 164–186.
17. Grossman, S.; Abraham, I.; Golan-Gueta, G.; Michalevsky, Y.; Rinetzky, N.; Sagiv, M.; Zohar, Y. Online detection of effectively callback free objects with applications to smart contracts. *Proc. ACM Program. Lang.* **2017**, *2*, 1–28. [CrossRef]
18. Lai, E.; Luo, W. Static analysis of integer overflow of smart contracts in ethereum. In Proceedings of the 2020 4th International Conference on Cryptography, Security and Privacy, Nanjing, China, 10–12 January 2020; pp. 110–115.
19. Dwivedi, V.; Pattanaik, V.; Deval, V.; Dixit, A.; Norta, A.; Draheim, D. Legally enforceable smart-contract languages: A systematic literature review. *ACM Comput. Surv.* **2021**, *54*, 1–34. [CrossRef]
20. Modi, R. *Solidity Programming Essentials: A Beginner's Guide to Build Smart Contracts for Ethereum and Blockchain*; Packt Publishing Ltd.: Birmingham, UK, 2018.
21. Zupan, N.; Kasinathan, P.; Cuellar, J.; Sauer, M. Secure smart contract generation based on petri nets. In *Blockchain Technology for Industry 4.0: Secure, Decentralized, Distributed and Trusted Industry Environment*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 73–98.
22. Chen, W.; Zheng, Z.; Ngai, E.C.-H.; Zheng, P.; Zhou, Y. Exploiting blockchain data to detect smart ponzi schemes on ethereum. *IEEE Access* **2019**, *7*, 37575–37586. [CrossRef]
23. Ji, M.; Liang, G.; Li, M.; Zhang, H.; He, J. Security Analysis of Blockchain Smart Contract: Taking Reentrancy Vulnerability as an Example. In Proceedings of the Advances in Artificial Intelligence and Security: 7th International Conference, ICAIS 2021, Proceedings, Part III 7, Dublin, Ireland, 19–23 July 2021; pp. 492–501.
24. Samreen, N.F.; Alalfi, M.H. A survey of security vulnerabilities in ethereum smart contracts. *arXiv* **2021**, arXiv:2105.06974.
25. Wang, C.; Jiang, H.; Wang, Y.; Huang, Q.; Zuo, Z. Research on smart contract vulnerability detection method based on domain features of solidity contracts and attention mechanism. *J. Intell. Fuzzy Syst.* **2023**, *45*, 1513–1525. [CrossRef]
26. Tantikul, P.; Ngamsuriyaroj, S. Exploring Vulnerabilities in Solidity Smart Contract. In Proceedings of the ICISSP, Valletta, Malta, 25–27 February 2020; pp. 317–324.
27. Fu, M.; Wu, L.; Hong, Z.; Feng, W. Research on vulnerability mining technique for smart contracts. *J. Comput. Appl.* **2019**, *39*, 1959.
28. Wei, G.; Li, H.; Bai, Y.; Yang, X.; Zhang, H.; Que, J.; Li, W. Co-governed Space-Terrestrial Integrated Network Architecture and Prototype Based on MIN. In Proceedings of the 2021 International Conference on Computer Communications and Networks (ICCCN), Athens, Greece, 19–22 July 2021; pp. 1–6.
29. Wang, H.; Li, H.; Smahi, A.; Zhao, F.; Yao, Y.; Chan, C.C.; Wang, S.; Yang, W.; Li, S.-Y.R. MIS: A Multi-Identifier Management and Resolution System in the Metaverse. *ACM Trans. Multimedia Comput. Commun. Appl.* **2023**. [CrossRef]
30. Qin, K.; Zhou, L.; Livshits, B.; Gervais, A. Attacking the defi ecosystem with flash loans for fun and profit. In Proceedings of the Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, 1–5 March 2021; pp. 3–32.
31. Cao, Y.; Zou, C.; Cheng, X. Flashot: A snapshot of flash loan attack on DeFi ecosystem. *arXiv* **2021**, arXiv:2102.00626.
32. Wu, J.; Lin, K.; Lin, D.; Zheng, Z.; Huang, H.; Zheng, Z. Financial Crimes in Web3-empowered Metaverse: Taxonomy, Countermeasures, and Opportunities. *IEEE Open J. Comput. Soc.* **2023**, *4*, 37–49. [CrossRef]
33. Chen, C.; Zhang, L.; Li, Y.; Liao, T.; Zhao, S.; Zheng, Z.; Huang, H.; Wu, J. When digital economy meets web 3.0: Applications and challenges. *IEEE Open J. Comput. Soc.* **2022**, *3*, 233–245. [CrossRef]

34. O'Regan, G. Overview of Formal Methods. In *Concise Guide to Formal Methods: Theory, Fundamentals and Industry Applications*; Springer: Newy York, NY, USA, 2017; pp. 41–63.
35. Vivar, A.L.; Orozco, A.L.S.; Villalba, L.J.G. A security framework for Ethereum smart contracts. *Comput. Commun.* **2021**, *172*, 119–129. [CrossRef]
36. Albert, E.; Gordillo, P.; Livshits, B.; Rubio, A.; Sergey, I. Ethir: A framework for high-level analysis of ethereum bytecode. In Proceedings of the Automated Technology for Verification and Analysis: 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, 7–10 October 2018; pp. 513–520.
37. Coward, P.D. Symbolic execution systems—A review. *Softw. Eng. J.* **1988**, *3*, 229–239. [CrossRef]
38. Luu, L.; Chu, D.-H.; Olickel, H.; Saxena, P.; Hobor, A. Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24 October 2016; pp. 254–269.
39. Krupp, J.; Rossow, C. teether: Gnawing at ethereum to automatically exploit smart contracts. In Proceedings of the 27th {USENIX} Security Symposium ({USENIX} Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1317–1333.
40. He, J.; Balunović, M.; Ambroladze, N.; Tsankov, P.; Vechev, M. Learning to fuzz from symbolic execution with application to smart contracts. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 531–548.
41. Li, J.; Zhao, B.; Zhang, C. Fuzzing: A survey. *Cybersecurity* **2018**, *1*, 1–13. [CrossRef]
42. Jiang, B.; Liu, Y.; Chan, W.K. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Montpellier, France, 3–7 September 2018; pp. 259–269.
43. Nguyen, T.D.; Pham, L.H.; Sun, J.; Lin, Y.; Minh, Q.T. sfuzz: An efficient adaptive fuzzer for solidity smart contracts. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, Seoul, Republic of Korea, 27 June–19 July 2020; pp. 778–788.
44. Medeiros, I.; Neves, N.; Correia, M. Detecting and removing web application vulnerabilities with static analysis and data mining. *IEEE Trans. Reliab.* **2015**, *65*, 54–69. [CrossRef]
45. Ji, S.; Dong, J.; Qiu, J.; Gu, B.; Wang, Y.; Wang, T. Increasing fuzz testing coverage for smart contracts with dynamic taint analysis. In Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan Island, China, 6–10 December 2021; pp. 243–247.
46. Rodler, M.; Li, W.; Karame, G.O.; Davi, L. Sereum: Protecting existing smart contracts against re-entrancy attacks. *arXiv* **2018**, arXiv:1812.05934.
47. Brent, L.; Grech, N.; Lagouvardos, S.; Scholz, B.; Smaragdakis, Y. Ethainter: A smart contract security analyzer for composite vulnerabilities. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, London, UK, 15–20 June 2020; pp. 454–469.
48. Beosin. Automated Formal Verification Platform for Smart Contract. Available online: https://beosin.com/ (accessed on 9 June 2023).
49. Mythril. A Framework for Bug Hunting on the Ethereum Blockchain. Available online: https://mythx.io/ (accessed on 9 June 2023).
50. Tsankov, P.; Dan, A.; Drachsler-Cohen, D.; Gervais, A.; Buenzli, F.; Vechev, M. Securify: Practical security analysis of smart contracts. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 67–82.
51. Mossberg, M.; Manzano, F.; Hennenfent, E.; Groce, A.; Grieco, G.; Feist, J.; Brunson, T.; Dinaburg, A. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 10–15 November 2019; pp. 1186–1189.
52. Feist, J.; Grieco, G.; Groce, A. Slither: A static analysis framework for smart contracts. In Proceedings of the 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), Montreal, QC, Canada, 27 May 2019; pp. 8–15.
53. Nikolić, I.; Kolluri, A.; Sergey, I.; Saxena, P.; Hobor, A. Finding the greedy, prodigal, and suicidal contracts at scale. In Proceedings of the 34th Annual Computer Security Applications Conference, San Juan, PR, USA, 3–7 December 2018; pp. 653–663.
54. Frank, J.; Aschermann, C.; Holz, T. ETHBMC: A bounded model checker for smart contracts. In Proceedings of the 29th USENIX Conference on Security Symposium, Boston, MA, USA, 12–14 August 2020; pp. 2757–2774.
55. Godefroid, P. Fuzzing: Hack, art, and science. *Commun. ACM* **2020**, *63*, 70–76. [CrossRef]
56. Cadar, C.; Godefroid, P.; Khurshid, S.; Păsăreanu, C.S.; Sen, K.; Tillmann, N.; Visser, W. Symbolic execution for software testing in practice: Preliminary assessment. In Proceedings of the 33rd International Conference on Software Engineering, Honolulu, HI, USA, 21–28 May 2011; pp. 1066–1071.
57. Dai, P.; Pan, Z.; Li, Y. A Review of Researching on Dynamic Taint Analysis Technique. In Proceedings of the 2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018), Chongqing, China, 15–16 December 2018; pp. 118–123.
58. Zhang, Q.; Wang, Y.; Li, J.; Ma, S. Ethploit: From fuzzing to efficient exploit generation against smart contracts. In Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), London, ON, Canada, 21–28 February 2020; pp. 116–126.
59. Atzori, M. Blockchain Technology and Decentralized Governance: Is the State Still Necessary? Available online: https://ssrn.com/abstract=2709713 (accessed on 9 June 2023).

60. Wang, Q.; Su, M. Integrating blockchain technology into the energy sector—From theory of blockchain to research and application of energy blockchain. *Comput. Sci. Rev.* **2020**, *37*, 100275. [CrossRef]
61. Bai, H.; Li, H.; Que, J.; Zhang, M.; Chong, P.H.J. DSCCP: A Differentiated Service-based Congestion Control Protocol for Information-Centric Networking. In Proceedings of the 2022 IEEE Wireless Communications and Networking Conference (WCNC), Shanghai, China, 7–10 April 2022; pp. 1641–1646.
62. Litvinenko, V. Digital economy as a factor in the technological development of the mineral sector. *Nat. Resour. Res.* **2020**, *29*, 1521–1541. [CrossRef]
63. Xu, J.J. Are blockchains immune to all malicious attacks? *Financ. Innov.* **2016**, *2*, 25. [CrossRef]