Blockchain: Research and Applications 7 (2026) 100347

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Review Article

Welcome back: A systematic literature review of smart contract reentrancy and countermeasures

Fatemeh Ghiyami Pour [a,b,*], Gabriele Costa [b], Letterio Galletta [b]

[a] International School of Advanced Studies, University of Camerino, Camerino 62032, Italy
[b] IMT School for Advanced Studies Lucca, Lucca 55100, Italy

ARTICLE INFO

Keywords:
Reentrancy
Blockchain security
Smart contract vulnerability
Systematic literature review

ABSTRACT

Background: Smart contracts are revolutionizing the way two or more parties subscribe to and apply an agreement. The main reason is that they promise to increase efficiency, transparency, and security. However, their inherent vulnerabilities can lead to automated exploits, resulting in significant resource losses. Among these, *reentrancy* is one of the most impactful security flaws. Over the past few years, reentrancy vulnerabilities have caused substantial financial damage and threatened the viability of entire blockchain ecosystems. Therefore, investigating whether reentrancy can be effectively mitigated and exploring the methodologies designed to address it is crucial.

Methodology: This paper aims to provide a comprehensive understanding of the impact of reentrancy vulnerabilities in Ethereum smart contracts and to assess the theoretical and practical approaches proposed to counter them. By following the PRISMA framework, we conduct a literature review of academic publications to identify, screen, and analyze relevant studies on detection and mitigation techniques.

Results: Our findings indicate that the estimated financial loss due to reentrancy attacks amounted to around 350 million USD by 2023, with increasing frequency and profitability of incidents. Despite advancements in mitigation tools, particularly machine learning, they only partially address vulnerabilities and remain ineffective against zero-day attacks.

Contribution: This paper identifies critical challenges in reentrancy detection, including the lack of standardized benchmarks and data on zero-day vulnerabilities. It emphasizes the need for unified datasets and evaluation frameworks to facilitate fair comparisons, improving detection effectiveness. Additionally, it highlights the need for tools addressing all four reentrancy vulnerability types and reporting performance results.

1. Introduction

Bitcoin, the foremost application of blockchain technology, employs a constrained set of scripts for transaction execution. The advent of Ethereum smart contracts represented a further advancement, propelling blockchain technology into the era of programmable finance.

Briefly, smart contracts allow controlling transactions with a high degree of customization through code that is public and immutable. The first mention of smart contracts was due to Szabo [1], who introduced the concept as an alternative to traditional paper contracts. Nowadays, smart contracts manage digital currencies valued at billions of dollars, making them appealing targets for potential attackers. The main attack vector is vulnerability exploitation, where attackers take advantage of poorly implemented smart contracts to drain resources from the blockchain.

For instance, SlowMist Hacked [2] reported that blockchain platforms including Ethereum [3], EOS [4], and TRON [5] have suffered from losses surpassing 3.3 billion USD due to the security breaches in smart contracts.

One of the most impactful vulnerabilities is *reentrancy*, which represents a formidable threat within blockchain systems [6]. The reentrancy vulnerability occurs when an external callee contract invokes a function in the victim contract in the middle of its execution. This permits the attacker to circumvent sanity checks that are not yet executed by the victim. Consequences of this attack include balance depletion and gas exhaustion [7].

* Corresponding author at: International School of Advanced Studies, University of Camerino, Camerino 62032, Italy.
E-mail addresses: fatemeh.ghiyamipour@imtlucca.it (F. Ghiyami Pour), gabriele.costa@imtlucca.it (G. Costa), letterio.galletta@imtlucca.it (L. Galletta).

https://doi.org/10.1016/j.bcra.2025.100347
Received 14 October 2024; Received in revised form 11 April 2025; Accepted 14 April 2025

Available online 25 July 2025

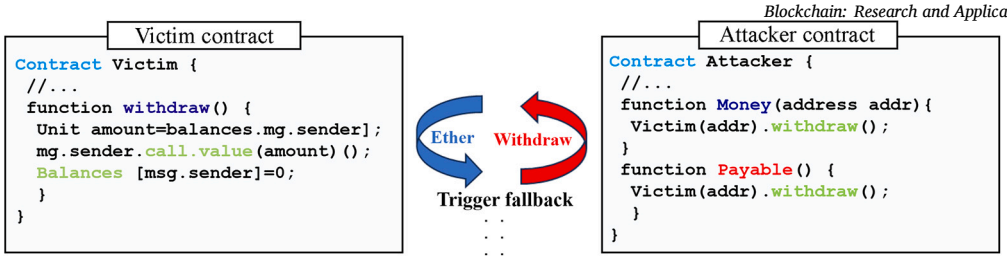

Fig. 1. A real-world instance of smart contract reentrancy attack [13].

A major reentrancy-related episode was the attack on The DAO [8], one of the first decentralized autonomous organizations and a crowdfunded investment project [9], in 2016. In that case, an attacker exploited a reentrancy vulnerability to steal about $60M worth of Ether (the native cryptocurrency of Ethereum). The attack had heavy effects, causing Ethereum's market cap to drop by over $600M in a day and leading to a community split that resulted in a hard fork and the creation of the Ethereum Classic blockchain [10]. In general, reentrancy vulnerabilities can significantly impact decentralized finance by enabling attackers to manipulate token prices, drain liquidity pools, or exploit flash loan mechanisms, leading to substantial financial losses [11]. Furthermore, reentrancy vulnerabilities can affect other Web3 applications (e.g., the metaverse), leading to financial losses and undermining trust, as demonstrated by the 2022 attack against Project Paraluni, based on Binance Smart Chain, where $1.7M were stolen due to a smart contract flaw [12]. Such vulnerabilities are not isolated cases, but rather persistent issues in practice.

A crucial aspect of understanding reentrancy vulnerabilities lies in the behavior of the fallback function. Briefly, the fallback function takes no input, returns no output, and is triggered in two scenarios:

1. When a contract is called but does not contain any matched function, fallback is called by default.
2. In the event of an Ether transfer to the contract, the fallback function will also be executed.

Fig. 1 depicts a real-world example of a reentrancy vulnerability [13]. This attack belongs to the second scenario described above. In the "Attacker" contract, the `Money` function endeavors to perform a withdrawal operation by invoking the `withdraw` function of the "Victim" contract, incorporating the Ether transfer method. As a result, this triggers the fallback function in the "Attacker" contract. Subsequently, the triggered fallback function in the "Attacker" contract reinvokes the `withdraw` function, leading to the repetitive execution of the withdrawal operation until the Ether within the "Victim" contract is depleted. Since contracts are immutable, victims cannot block the continuous flow of Ether into the intruder's possession.

The security community has devoted considerable effort to designing and implementing countermeasures for detecting and mitigating reentrancy in smart contracts. These efforts explored several directions, and they produced a myriad of tools. Nevertheless, the main question remains: Can we get rid of reentrancy in practice? Existing studies in the literature on reentrancy vulnerabilities in smart contracts do not fully investigate all types of reentrancy vulnerabilities, and they mainly consider conventional reentrancy detection tools, especially those that focus on the type of reentrancy exploited by the DAO attack in 2016. In addition, no study has compared the performance of different reentrancy detection tools in smart contracts.

To address this gap, we present a systematic review of papers on smart contract reentrancy issues, focusing on its impact, as well as its theoretical and practical addressability. Our research is guided by the following research questions:

RQ1. What is the actual impact/diffusion of reentrancy attacks?
RQ2. Which methods can theoretically address reentrancy?
RQ3. Which tools exist that counter reentrancy in practice?

For each selected paper, we critically revised the proposed methodology and, when present, the implementation and experimental results. All the techniques were categorized and compared in order to provide the reader with a clear understanding and a comparison of their performance. Our findings indicate that reentrancy attacks have caused approximately $350 million in losses so far and that their impact is still growing (RQ1). Most authors propose methodologies based on a few techniques and their combinations, such as machine learning (ML), model checking (MC), symbolic execution (SE), fuzz testing (FT), runtime monitoring (RM), and code patching (CP). Among these, ML is the most frequently used technique (RQ2).

Although various detection tools exist, they generally lack effectiveness against zero-day vulnerabilities, and there is no standardized benchmark for evaluating these tools (RQ3). These insights offer readers a clear understanding of the state of affairs of reentrancy and point out some critical issues and novel directions for the security community.

In summary, the main contributions of this paper are as follows.

1. Analysis of reentrancy in the wild. We carry out a thorough examination of reentrancy issues, encompassing the economic impact of attacks and challenges related to detecting and preventing reentrancy vulnerabilities.
2. Survey of reentrancy detection and defense mechanisms. We present a systematic review of existing literature focusing on methods to counter reentrancy in smart contracts.
3. Critical assessment of existing methodology. We perform a structured comparison of the results achieved by existing methodologies and their implementations, with a specific focus on their capability to prevent reentrancy-based attacks.

The remainder of this paper is structured as follows. Section 2 compares the current paper with the relevant related work and highlights its main novelties. In Section 3, we introduce the relevant information about smart contracts, reentrancy vulnerability, and defense techniques. Section 4 presents the methodology we followed to search and select the relevant papers from online databases and discuss possible threats to the validity of our methodology. Section 5 provides a comprehensive analysis and evaluation of the selected papers, answers to our research questions, and synthesizes our findings. Section 6 presents some actionable recommendations to guide researchers and practitioners in overcoming reentrancy vulnerability. Lastly, in Section 7, we draw some conclusions, briefly discussing future research directions.

2. Related work

This section summarizes the relevant literature and compares it with the current paper, emphasizing its main novelties. Since the appearance of smart contracts, their security has emerged as a vast research area, and many authors have proposed surveys on this field. For instance, Li et al. [14] systematically examined the security risks in popular blockchain systems, surveying real attacks from 2009 to 2017 and analyzing the exploited vulnerabilities.

Table 1
Recent surveys on the security of smart contracts.

Title	Year	SLR	Method	Attack	Detection
Survey on the security of blockchain systems [14]	2020	✗	✓	✓	✗
A Survey on Ethereum Systems Security: Vulnerabilities, Attacks, and Defenses [7]	2020	✗	✓	✓	✓
Empirical review of automated analysis tools on 47,587 Ethereum smart contracts [20]	2020	✗	✓	✗	✓
A Survey of Smart Contract Formal Specification and Verification [19]	2021	✗	✗	✗	✓
Ethereum Smart Contract Analysis Tools: A Systematic Review [21]	2022	✗	✓	✗	✓
Systematic Review of Security Vulnerabilities in Ethereum Blockchain Smart Contract [22]	2022	✗	✓	✗	✓
Review of Automated Vulnerability Analysis of Smart Contracts on Ethereum [23]	2022	✗	✓	✗	✓
A systematic literature review of undiscovered vulnerabilities and tools in smart contract technology [17]	2023	✓	✓	✗	✓
A Survey of Vulnerability Detection Techniques by Smart Contract Tools [24]	2024	✗	✓	✗	✓
Vulnerabilities of smart contracts and mitigation schemes: A Comprehensive Survey [25]	2024	✗	✓	✗	✓
A Systematic Literature Review on Smart Contract Vulnerability Detection by Symbolic Execution [16]	2024	✓	✗	✗	✓
Ethereum Smart Contract Vulnerability Detection and Machine Learning-Driven Solutions: A Systematic Literature Review [15]	2024	✓	✗	✗	✓
This survey	2025	✓	✓	✓	✓

Table 1 lists recent and relevant studies focusing on the security of smart contracts, also including reentrancy. We compare these studies based on their main contributions. The “Year” column indicates the publication year of each study. In column “SLR”, we report whether the paper presents a systematic literature review (SLR) following the definition given in Refs. [15–17]. According to their definition, an SLR identifies, evaluates, and synthesizes all available research papers relevant to a specific research question, topic area, or phenomenon of interest. Its goal is to provide clear, reasonable, and unbiased information on a research topic [18].

Regarding column “Method”, we indicate reviews that consider works using more than a single, specific methodology. For instance, Kiani and Sheng [15] only revise papers employing ML, and similarly in Refs. [16,19], the scope is on SE and formal verification, respectively.

Column “Attack” denotes papers that include details about real attacks due to the exploitation of vulnerabilities in smart contracts. For example, Ref. [7] provides a list of attacks on the Ethereum blockchain and puts them in relation to the vulnerabilities they used.

Column “Detection” lists papers that consider detection tools able to spot vulnerabilities in smart contracts. As one might expect, most papers report on vulnerability detection mechanisms with the exception of Li et al. [14], which systematically explores security threats to blockchain and surveys real attacks on popular blockchain systems.

Compared to other works, the present survey is the only one that, at the same time, follows a systematic approach to literature review, includes a wide spectrum of methodologies, discusses real attacks, and reviews the existing detection tools. This level of detail is due to our aimed scope of only dealing with reentrancy. As a matter of fact, most of the surveys listed above consider smart contract vulnerabilities in general. We claim that our in-depth analysis of a single, yet extremely impactful, vulnerability is crucial for better understanding how to counter such a security flaw in smart contracts effectively.

3. Background

This section briefly recalls the key concepts necessary to correctly understand our work. First, we present the basic notions of smart contracts, then we introduce the reentrancy vulnerability, and finally, we discuss the main proposed defense mechanisms.

3.1. Smart contracts

Smart contracts [26] are a major use case for blockchain technology. Briefly, a smart contract is a self-executing digital contract with the terms of the agreement between two or more parties written directly into code [27], deployed on the blockchain and executed by a distributed virtual machine. Smart contracts are transparent and immutable, and they do not rely on a trusted third party [21]. This makes them suitable for creating decentralized applications [28], and for adopting them in many fields such as supply chain management [29], Internet of Things [30], finance [31], and healthcare [32].

The idea of smart contracts was introduced in 1997 [33], but it is only with the advent of Ethereum in 2015 that they became a reality. In 2022, smart contracts controlled approximately 1.75 million USD, and projections estimate this to rise to 9.85 million USD by 2030 [34]. Needless to say, smart contracts have attracted significant attention from cyber criminals because of their economic value, making their security crucial.

Ethereum is possibly the most famous smart contracts platform. As represented in Fig. 2, the development process of smart contracts into the Ethereum blockchain involves three phases: coding, compiling, and deploying. During the coding phase, developers write the source code using Solidity, a high-level, statically typed programming language. Subsequently, the source code is compiled into Ethereum virtual machine (EVM) bytecode using a language-specific compiler called `Solc`.

Finally, in the last phase, the smart contract is deployed on the blockchain. During the deployment, the contract is assigned a unique address. By accessing its address, users and other contracts can invoke the contract’s application binary interface [35], i.e., the members, either functions or data fields, publicly exposed by the contract, and interact with it.

3.2. Reentrancy vulnerability

Reentrancy is one of the most severe vulnerabilities of smart contracts. Intuitively, this vulnerability arises when an external callee contract invokes a function exposed by a caller contract before the completion of the latter. This can lead an attacker to circumvent the sanity checks and deplete the contract’s balance.

Four types of reentrancy exist that we briefly describe below. Fig. 3 illustrates them, where the color red denotes the fallback function, which is central to the attack; the color green indicates the invocation of a function, while blue represents the definition of a function.

Transfer-based reentrancy Transfer-based reentrancy manifests when an attacker can withdraw an amount from a victim contract multiple times before the contract’s balance is updated. In practice, this may happen when a vulnerable `withdraw` function transfers money and returns control to the caller through the fallback function before the balance of the victim contract is updated. For instance, in Fig. 3, the attacker contract has a function `money` that (1) invokes the victim’s `withdraw` function and (2) gets control back via the `fallback` function. By iteratively calling `money`, the attacker prevents the victim from running `update`, thus depleting the balance of the target contract.

Single-function reentrancy, exemplified by the DAO attack, is a subset of transfer-based reentrancy, where the attacker exploits a specific function call to manipulate contract states or funds.

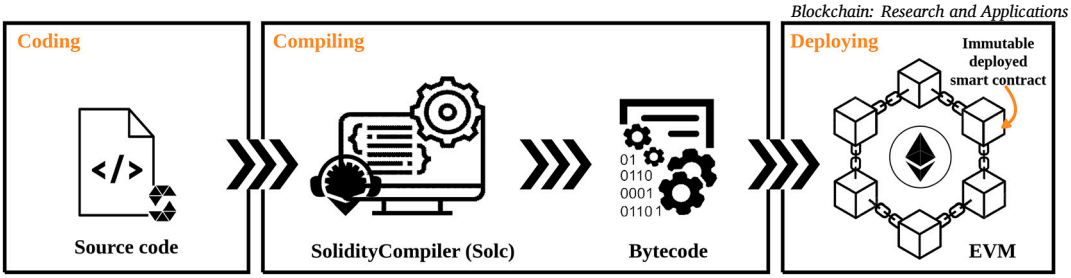

Fig. 2. The compilation process for Ethereum smart contracts.

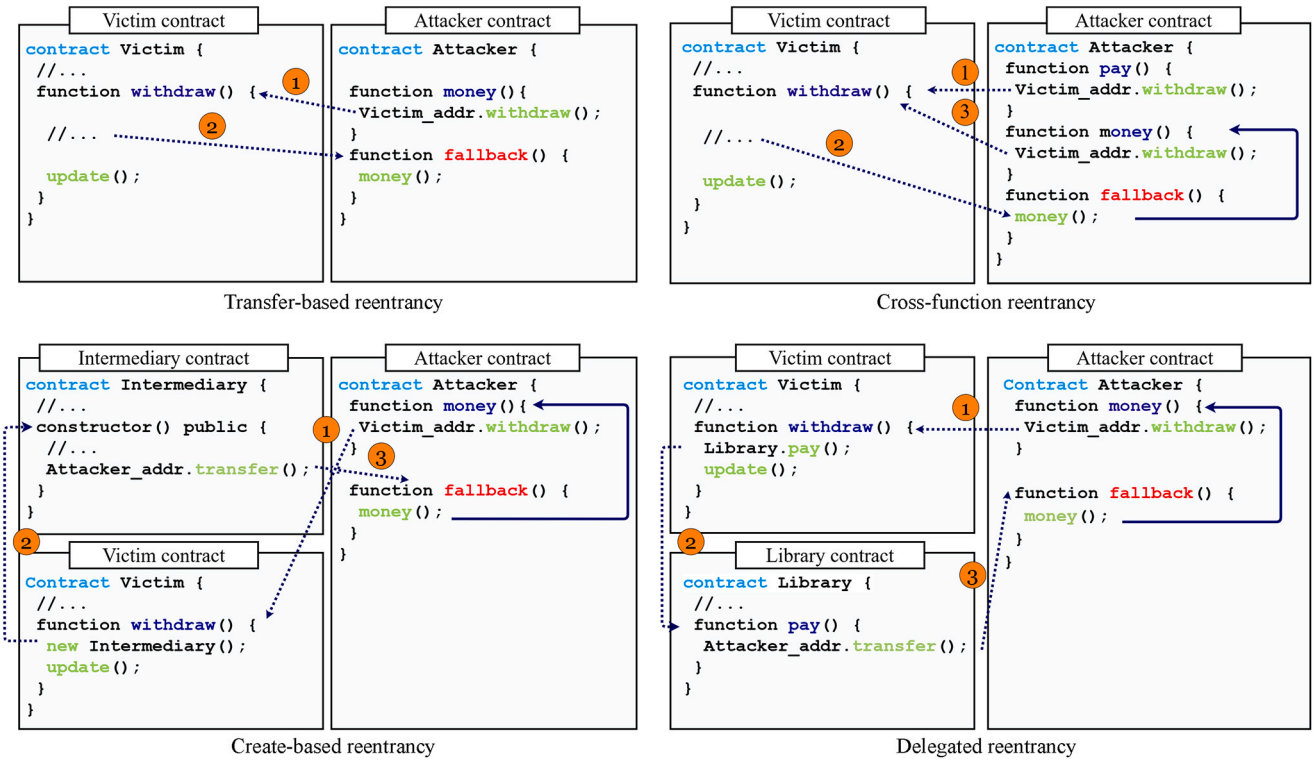

Fig. 3. Four types of reentrancy vulnerabilities.

Cross-function reentrancy Cross-function reentrancy occurs when an attacker exploits the interaction between multiple functions within a contract to repeatedly withdraw funds before the contract's state is updated. This differs from single-function reentrancy, where the exploit is contained within a single function call. For example, in Fig. 3, the attacker contract has a function `pay` that (1) calls the victim's `withdraw` function. Next, the victim's `withdraw` function makes an external call (2), transferring some money and the control back to the attacker's `fallback` function. Then, the `fallback` function calls another function `money` in the attacker contract, which (3) calls `withdraw` again on the victim's contract. By continually invoking `money` function, the attacker perpetuates a loop through these functions, before the victim contract updates its state via the `update` function. This cycle enables the attacker to deliberately and progressively drain the balance of the victim's contract.

Create-based reentrancy Create-based reentrancy occurs when the constructor of a newly created contract makes external calls to other contracts. This type of attack leverages the `new` keyword in Solidity to run reentrant attacks deriving from the execution of the constructor. Fig. 3 shows an example of this attack that occurs when (1) the attacker calls the `withdraw` function of the victim contract that creates a new intermediary contract. When the intermediary contract is initialized, (2) its `constructor` executes immediately, and (3) the `constructor` calls back the attacker contract, so triggering its `fallback` function. The `fallback` function is implemented to invoke the `money` function, which calls `withdraw` in the victim contract again. This process repeats, allowing the attacker to continuously drain the victim contract's balance before running `update`, exploiting the creation process by the victim contract to deplete its funds.

Delegated reentrancy Delegated reentrancy arises from a combination of contracts and library contracts, utilizing `DELEGATECALL` or `CALLCODE` EVM instructions.

These instructions enable a contract to execute instructions of a target contract within its own execution context, thus allowing the library to exert control over the calling contract's funds and internal state. While individual contracts may not exhibit reentrancy vulnerabilities when analyzed independently, combining a state update and an external call in separate contracts may enable this vulnerability. That is, delegate reentrancy does not emerge when each contract is analyzed in isolation. For example, in Fig. 3, the attacks occur in smart contracts when an attacker exploits the delegation calls to an external library contract to repeatedly withdraw funds before the victim can update its state.

The attack takes place as follows: (1) the attacker initiates the attack by calling the `withdraw` function of the victim contract; next, (2) the `withdraw` function calls the `pay` function from the Library contract, to transfer the amount to the attacker's address; (3) the `pay` function

in the Library contract transfers the amount to the attacker's address. During this transfer, the `fallback` function of the attacker contract is triggered. The fallback function in the attacker contract, then invokes the `money` function within the attacker contract, which calls the `withdraw` function of the victim contract once more, starting the process over again.

3.3. Defense mechanisms

Security mechanisms can be applied during the various phases of smart contract development to enhance the security of smart contracts against reentrancy issues. Such defense mechanisms for reentrancy vulnerabilities are typically distinguished between two categories: proactive and reactive defenses [7]. Following this classification, we briefly recall defense mechanisms for smart contracts. As shown in Fig. 2, the development of smart contracts on the Ethereum blockchain involves three phases. In the coding phase, for instance, the developers can use software engineering mechanisms such as code inspection [36], which consists of a systematic review of the source code to check for defects. During the compiling phase, the developers can use different analysis tools to mitigate reentrancy at the application binary interface (ABI) and bytecode levels.

In the last phase, reactive defense mechanisms can be used to defend against reentrancy attacks. Below, we briefly survey these defense mechanisms.

3.3.1. Proactive defenses

Proactive defenses are security mechanisms that are leveraged to mitigate or avoid reentrancy vulnerabilities before deploying smart contracts. Typically, these mechanisms are applied during the coding and compiling phases of the development process.

Language-based security These defenses are based on enhancing programming languages with constructs that help developers write vulnerability-free code. In this context, two families of languages are relevant: high-level languages, which are used to develop smart contracts, and intermediate-level languages, which are used to deploy smart contracts on the blockchain virtual machine. Language-based security mechanisms may be applied at both these levels.

For example, both Obsidian [37] and Flint [38] introduce enhanced type systems to detect and prevent reentrancy.

Additionally, Nomos [39] is a domain-specific language that enhances security by utilizing resource-aware session types. Recognizing that the linearity of session types alone is inadequate to prevent reentrancy, Nomos leverages the resource-tracking capabilities of session types. This approach ensures that attackers cannot gain permission to call a currently used contract, thereby eliminating all forms of (object) reentrancy [40].

Coding best practice As smart contracts are a novel programming paradigm, best practices, which can help developers prevent or mitigate common vulnerabilities, are still under development. The call control principle is one of the best practices in Solidity for preventing reentrancy vulnerabilities during the development of smart contracts. This principle underscores the importance of establishing secure interactions between smart contracts and externally owned accounts. It focuses on preventing unexpected behaviors from the callee contract to mitigate reentrancy attacks. A fundamental aspect of this principle involves following a pattern known as "check, update, then interact". This approach mandates verifying conditions first, updating state variables second, and only then engaging in interactions with other contracts.

Smart contract analysis Examining smart contracts and improving their security through various methods is another technique for defense. We completely discuss these methods in Section 5.2.

Table 2
Lists of primary, secondary, and tertiary keywords.

	Primary (1)	Secondary (2)	Tertiary (3)
A	Reentrancy	Vulnerabilit*	Symbolic execution
B	Smart contract	Detection	Formal verification
C	–	Analys*	Fuzzing
D	–	Security	Machine learning
E	–	–	Deep learning

3.3.2. Reactive defenses

Reactive defenses are leveraged to address the potential exploitation of unknown vulnerabilities during contract runtime, aiming to mitigate damage. Runtime verification methods monitor execution traces to detect and respond to suspicious activities that may violate certain properties. To detect and prevent reentrancy vulnerabilities, Sereum [41], for instance, uses taint analysis to monitor runtime data flows during the execution of smart contracts.

4. Methodology

In this study, we utilized the Preferred Reporting Items for Systematic reviews and Meta-Analysis (PRISMA) model [42] for reporting and analyzing research within the domain of interest, namely reentrancy issues in smart contracts. Thus, we start our analysis with the formulation of a targeted search strategy. We also define exclusion criteria for refining the collected body of knowledge. Then, we move on to study selection, data extraction, and synthesis to address our research questions.

The potential threats to validity that might affect our methodology are then discussed in Section 4.3.

4.1. Search strategy

We commenced with an analysis of existing literature reviews on smart contracts vulnerabilities, in general, and reentrancy, in particular.

To extend the scope of our search, we considered three widely used online repositories, i.e., the ACM Digital Library,[1] IEEEXplore Digital Library,[2] and Science Direct.[3]

The search strings utilized to query these repositories were designed by considering three categories of keywords: *primary*, *secondary*, and *tertiary*. Primary keywords precisely represent the vertical scope of our survey, and they are "reentrancy" and "smart contract". Secondary keywords are terms that capture the security-related topics, and they are "vulnerabilit*", "detection", "analys*" and "security".[4] Finally, tertiary keywords are meant to refine the granularity of our queries by restricting the search to the relevant methodologies. These keywords are "symbolic execution", "formal verification", "fuzzing", "machine learning" and "deep learning". The lists of keywords are reported in Table 2.

Search queries are obtained by combining three keywords from these categories using the conjunction operator. For instance, the query A1 A2 B3 (or AAB for brevity) corresponds to searching all the papers that contain the three terms "reentancy", "vulnerabilit*" and "formal verification". Finally, we also submitted the query for "reentrancy" and "smart contract", which correspond to two terms in the primary category, due to their high relevance. This query is denoted by AB.

We conducted a review of technical studies published between 2018 and 2023. The search items include an "AND" combination of the three types of keywords. The distribution of search results across primary sources for each search term is detailed in Table 3.

[1] https://dl.acm.org/.
[2] https://ieeexplore.ieee.org/Xplore/home.jsp.
[3] https://www.sciencedirect.com/.
[4] Terms ending with the wildcard *, e.g., "vulnerabilit*", are meant to match both singular and plural, e.g., "vulnerability" and "vulnerabilities".

Table 3
Number of papers retrieved for each search string.

	AAA	AAB	AAC	AAD	AAE	ABA	ABB	ABC	ABD	ABE	ACA	ACB	ACC	ACD	ACE	ADA	ADB	ADC	ADD	ADE	BAA	BAB
ACM	3	1	1	3	3	3	0	0	2	3	2	2	1	0	0	2	2	1	2	1	15	9
IEEE	4	1	5	5	9	5	0	2	4	9	4	1	2	1	4	4	0	4	5	9	23	14
ScienceDirect	2	0	0	0	0	2	0	0	0	0	2	0	0	0	0	1	0	0	0	0	4	4
Total	9	2	6	8	12	10	0	2	6	12	8	3	3	1	4	7	2	5	7	10	42	27

	BAC	BAD	BAE	BBA	BBB	BBC	BBD	BBE	BCA	BCB	BCC	BCD	BCE	BDA	BDB	BDC	BDD	BDE	AB
ACM	13	11	21	12	2	5	21	23	7	3	9	19	13	9	14	8	24	16	18
IEEE	30	31	42	25	8	20	44	55	21	18	16	72	42	27	51	33	86	68	32
ScienceDirect	2	6	7	3	3	2	13	13	2	5	1	14	3	4	6	1	25	10	3
Total	45	48	70	40	13	27	78	91	30	26	26	105	58	40	71	42	135	94	53

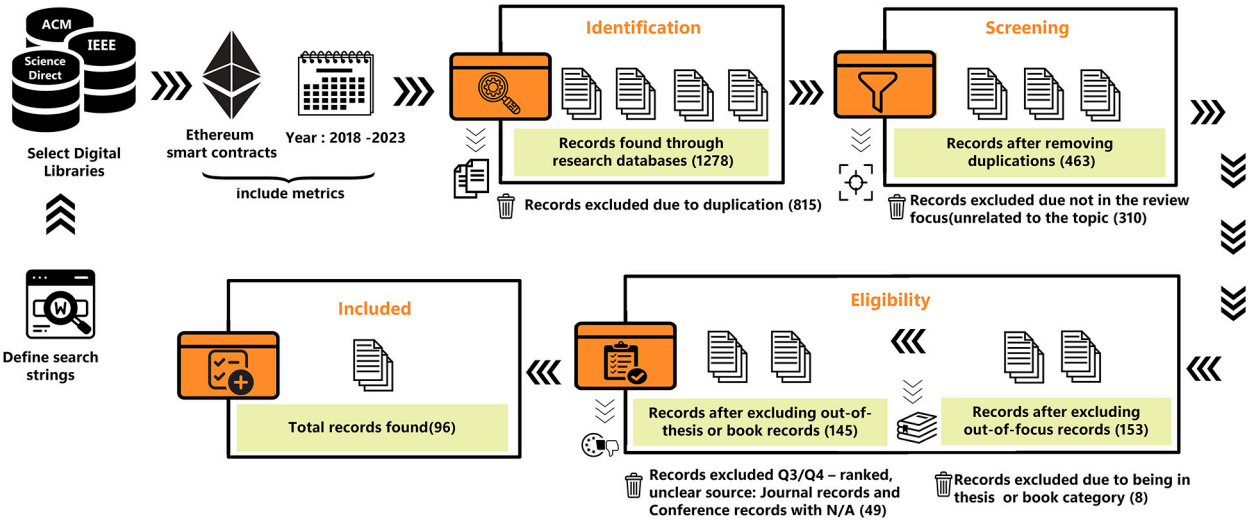

Fig. 4. Overall paper selection process using the Preferred Reporting Items for Systematic reviews and Meta-Analysis (PRISMA) method.

4.2. Study selection, data extraction, and synthesis

This section outlines the methodology for selecting, extracting, and synthesizing data to ensure a thorough review. We specifically detail the inclusion and exclusion criteria used to refine the selection of studies.

4.2.1. Inclusion and exclusion criteria

This study exclusively considered papers published in reputable scientific international journals with Q1 or Q2 ranks and proceedings of international conferences with A, B, and C ranks for inclusion. Exclusion criteria comprised work such as books, technical reports, and Master's theses. Only papers directly relevant to the reentrancy issue in smart contracts were included, while those falling outside this specific scope or originating from journals with Q3, Q4, or N/A rankings or conferences with N/A rankings were excluded. Additionally, articles not written in English and those not indexed were not considered for this comprehensive analysis.

4.2.2. Detailed process of study selection, data extraction, and synthesis

In this part of the process, we conducted an extensive search across top research repositories, resulting in the identification of 1278 research papers. After a meticulous screening process, we excluded 815 papers due to duplication. This left us with 463 studies for further consideration. Also, we excluded another 310 papers as they were not relevant to the review focus. To ensure the relevance of these studies, we performed a manual review of each paper, carefully examining the title and abstract. Subsequently, 8 studies were books and theses that were outside the scope of this review.

Furthermore, only papers that conformed to the predefined inclusion criteria relevant to the ranking of journals and conferences, as detailed in Section 4.2.1, were incorporated. Consequently, 49 additional articles were excluded at this stage because not meeting our quality requirements, resulting in a final selection of 96 studies eligible for inclusion in our paper. Fig. 4 presents an overview of the paper selection process employed in this systematic review.

4.3. Threats to validity

In this section, we discuss the potential threats to the validity of this study.

4.3.1. Internal validity

The internal validity of this study faces the following potential threats.

Known tools exclusion. Some tools may be excluded from our study due to the lack of academic publications, their papers are published at events that do not meet our criteria (e.g., workshops, events for practitioners), or because their publication dates fall outside the specified time frame of 2018–2023. This is the case of some widely used tools such as Mythril [43], SmartCheck [44], Slither [45], and Oyente [46] that were excluded by our selection criteria.

Variation in keywords. Different authors may use varied keywords for the same research subject, or there might be works using techniques not mapped into our keywords.

Selection of databases and time frame. The choice of databases and the specified time frame may limit the scope of the search.

Practical constraints. Practical constraints, e.g., the exclusion of certain databases such as the Web of Science,[5] may lead to missing relevant works. Furthermore, the absence of a unified or a standard dataset, makes a direct comparison between different methodologies and tools potentially inaccurate.

To mitigate the issue of keyword variation, we conducted an extensive review of existing literature to identify common terms and methods related to detecting vulnerabilities in smart contracts, with a particular focus on reentrancy. Although there remains a possibility of having omitted relevant studies, we consider our initial corpus of 1278 research papers adequate for the purpose of this work.

Regarding database selection and time frame, we have strong guarantees that our selection method is adequate. In particular, the number of replicated entries shows that the level of redundancy is quite high. This was also confirmed through a check against Web of Science, which resulted in only replicated entries. Similarly, the recent development of the blockchain ensures that the considered time frame is sound.

Finally, for what concerns practical constraints, as mentioned above, the level of redundancy in online databases is a strong guarantee against potential blind spots in our search methodology. Furthermore, we remark that tools' performances reported in our study (e.g., as in Table 8) are not meant to provide an exact comparison between the proposed techniques, but rather a general indication of the results achieved by them. Indeed, an objective comparison would require, e.g., a unified test dataset whose definition is beyond the scope of the present work.

4.3.2. Construct validity

Construct validity is centered around the construction of research questions and the process of data extraction. We used the PRISMA guidelines [42] as our methodology to conduct the SLR. Despite this, some threats remain. Firstly, the absence of snowballing in our study has potentially hindered the thoroughness of our search process. Snowballing, which involves applying selection criteria to the references of selected papers, was not conducted, potentially resulting in the oversight of additional relevant studies. Secondly, the restriction to English-only publications as inclusion criteria may have inadvertently excluded significant contributions published in other languages. This limitation is common in SLRs, and it may lead to the exclusion of valuable research.

4.3.3. External validity

Our methodology, detailed in Section 4, aims at ensuring that other researchers can replicate this review and obtain consistent results. The primary variable that could affect this consistency is the passage of time, as new research and tools continuously emerge. To address this, future updates to this systematic review will be necessary.

4.3.4. Conclusion validity

Conclusion validity involves the relation between the data extracted and the results inferred from them. In our SLR, we conducted a comparative analysis of tools' performance using data extracted from their respective papers. However, there is a lack of studies directly comparing the effectiveness of different tools, which may hinder the validation of our conclusions. Furthermore, since the evaluation of tools was based on different datasets, the use of such datasets for our comparisons poses a potential threat to the validity of our findings. This was mitigated by avoiding direct comparisons and drawing general conclusions from experimental data generated under diverse settings.

4.3.5. Platform selection

We deliberately limited our review to publications focused on Ethereum smart contracts, the EVM, and the Solidity programming language. This platform was chosen due to its extensive documentation, numerous publications, active community discussions, diverse use cases, significant asset handling, and substantial market value. However, we did not investigate the reentrancy issue in the context of other programming languages and virtual machines (e.g., we do not consider VMs based on WebAssembly – WASM [47]), or other smart contract platforms (e.g., Polkadot, Cardano). We believe that extending our study to these and other platforms may lead to further relevant findings, and we consider this direction as future work.

[5] https://www.webofscience.com.

5. Findings and results

In this section, we explain the findings obtained through a comprehensive analysis of relevant literature, answer the research questions outlined in Section 1, and summarize the literature on Ethereum smart contracts.

5.1. RQ1: What is the impact/diffusion of reentrancy attacks?

To measure the impact and diffusion of reentrancy attacks, we consider two factors: the number of reentrancy attacks and the total loss due to them. Thus, we started collecting data from 2016, when the DAO attack occurred. As stated in Section 1, the reason is that we consider that event a turning point when the community started acknowledging reentrancy as a major threat. Collected data include reports about reentrancy-based attacks that appeared in various sources, such as scientific papers, websites, and newspapers. The full list of these reports is given in Tables 11 and 12 in Appendix A.

The results of our data analysis are in Fig. 5. In terms of the total number of attacks (red plot), after an initial period (2016–2019) of sporadic reentrancy-related incidents, there has been a notable increase in their frequency. By the end of 2023, the cumulative number of reported reentrancy-related attacks had risen to 58.

For the monetary loss, we computed the total amount (blue line) for all the attacks mentioned above. The amount is reported in millions of dollars (on the right scale). As expected, after the initial loss due to the DAO attack ($60M), the total amount follows the same trend as the number of attacks. The total loss due to reentrancy exploitation by 2023 is estimated to be almost $350M, with an average loss per attack of $5.88M. These data confirm that reentrancy is both frequently exploited and impactful. Moreover, the ongoing trend suggests that the scenario might get even worse in the next few years.

To measure the interest of the scientific community in reentrancy attacks, we considered two key indicators: the total number of published papers and the total number of their citations. In particular, we focus on the 96 papers deemed eligible by the methodology process of our study (see Section 4).

The values are shown in Fig. 6. The distribution of papers over time (red plot) is steadily increasing. Similarly, and also consequently, the cumulative number of citations received by these papers (blue plot) has also increased. These data highlight the growing interest of the scientific community towards understanding and addressing reentrancy.

RQ1: What is the actual impact/diffusion of reentrancy attacks?

Since the DAO attack in 2016, reentrancy attacks have notably increased in both frequency and financial impact, with total losses nearing $350M, by 2023. The growing number of publications and citations highlights the increasing academic attention to this critical vulnerability, emphasizing its rising importance in the field.

5.2. RQ2: Is reentrancy theoretically addressable? Which methods exist?

Our analysis of the literature highlights that various methods have been utilized in the 96 papers included in our study to detect and fix reentrancy vulnerabilities in smart contracts. These methods include

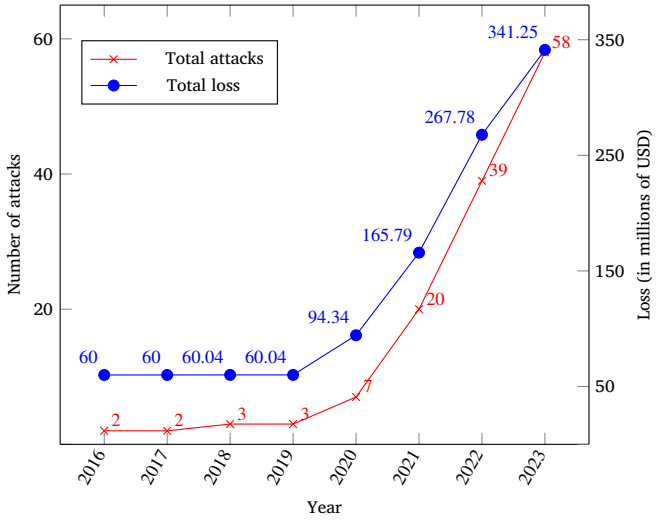

Fig. 5. Total number of attacks and loss.

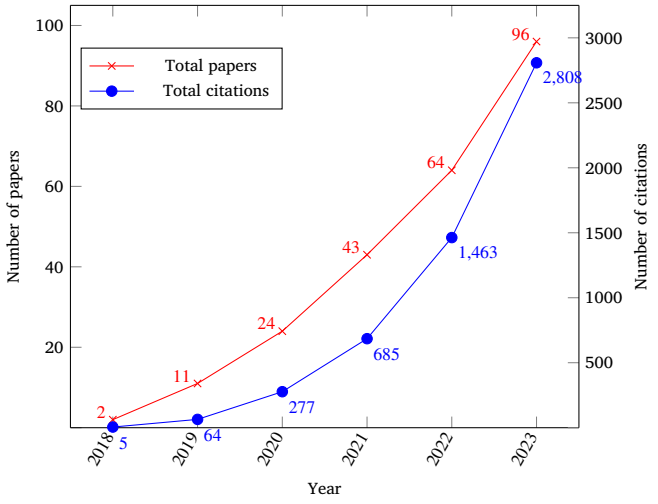

Fig. 6. Total number of papers and citations.

Table 4
Categories of methodologies used for addressing reentrancy vulnerabilities.
ML: machine learning; MC: model checking; SE: symbolic execution; FT: fuzz testing; RM: runtime monitoring; CP: code patching.

Type of analysis	ML	MC	SE	FT	RM	CP
Static analysis	✓	✓	✓			
Dynamic analysis	✓			✓	✓	
Hybrid analysis	✓		✓	✓		✓

ML, MC, SE, FT, RM, and CP. We divide these methods into three categories: static analysis, i.e., when smart contract code is inspected but not executed, dynamic analysis, i.e., when smart contracts are executed, and hybrid analysis, i.e., when code is both inspected and executed. Table 4 shows the methods used in the 96 papers we considered and their categories. For instance, ML was employed in all three types of analysis, while MC was only used for static analysis. Both SE, mainly a static technique, FT, mainly a dynamic one, were also used for implementing hybrid analyses. Finally, RM and CP were only used for dynamic and hybrid approaches, respectively.

We now delve into the specific methods introduced above, and we discuss how they generally apply to the analysis of smart contracts.

Symbolic execution SE [48] is a formal verification technique based on the idea that the concrete values of variables can be replaced with symbolic expressions, thus obtaining a symbolic semantics of programs. SE can explore multiple execution paths in a single run, identify reachable program statements and provide logical invariants that must hold at runtime.

Dynamic SE, also known as concolic execution, is a hybrid verification technique combining static SE with traditional, concrete testing [49]. In this method, the program initially runs with concrete values, while simultaneously gathering symbolic constraints from conditional statements encountered during execution. Subsequently, by employing a constraint solver, concolic execution deduces input modifications needed to steer the program towards alternate execution paths.

Since these techniques can be ported to any programming language and execution environment, they also apply to Solidity and the EVM. For instance, Wana [50] and DefectChecker [51] use SE to statically analyze smart contracts, while ACVDT Tool [52] relies on dynamic SE to detect vulnerabilities in smart contracts.

Fuzz testing Fuzzing is an automated testing method that involves generating inputs randomly [53]. The primary objective of fuzzing is to create diverse inputs to repeatedly run a computer program, capturing execution data to analyze and identify potential vulnerabilities [54].

The primary objective of this process lies in uncovering critical contract states, specifically those appearing as crashes or memory leaks, which potentially indicate vulnerabilities. In the event of encountering such a critical state, the fuzzer reports the exact sequence of inputs that triggered its occurrence. This recorded sequence, often referred to as the execution trace, is a valuable asset for developers, enabling them to identify the root causes of the malfunction and implement targeted remediation strategies.

Runtime monitoring RM is an approach for ensuring the safe execution of programs, including smart contracts.

In general, a runtime monitor is a program or a device that follows and validates the current execution of a target, e.g., a smart contract. This technique encompasses methods for observing the internal operations of the smart contract and its interactions with external entities to determine whether the execution respects or violates a correctness specification [55].

Model checking MC is an automated technique used to systematically verify whether a specification holds for a given model [56]. The verification algorithm may vary between implementations (e.g., see Ref. [57]), but often MC reduces to visiting the model states until an illegal one is found or, eventually, they are all validated against the specification.

A key advantage of MC is that it either provides formal evidence that the specification holds or it returns a counterexample, i.e., a path in the target model that leads to a violation [58]. Although several model checkers exist, their applicability requires a behavioral model of the smart contract under analysis. Obtaining such a model, e.g., by analyzing the smart contract code, is typically a central aspect for proposals relying on MC.

Machine learning ML focuses on training expert systems from available data to make anomaly detection, pattern recognition, and predictive modeling more efficient. The effectiveness of these approaches is strongly related to the quality and quantity of training samples, as well as the adequacy of the features used for characterizing the relevant behaviors.

The ways to apply ML to smart contract analysis are many. The same goes with the underlying models. For instance, in the reviewed literature, some papers [59–62] employed graph neural networks [63],

while others [64,65] rely on recurrent neural networks [66] and multi-objective-based neural networks [64]. Additionally, random forest [67], Bayesian active learning [68], and large language models [69] have been used [70–72]. Also in Ref. [73], easy ensemble classifier [74] was used for fuzzing guidance.

Code patching Software security patches are "pieces of code developed to address security problems identified in software" [75]. The patches can address the related vulnerabilities while ensuring that no other vulnerabilities are introduced [76]. In general, we classify under CP all the techniques that rely on smart contract code manipulation in order to fix a vulnerability or prevent its exploitation. Unlike traditional software, where upgrades can alter any part of the software, patching a smart contract refers to a process of arbitrarily modifying the smart contract code while preserving its stored data or state [77]. However, patching or upgrading a contract conflicts with the blockchain's immutability feature.

Thus far, researchers have developed two primary methodologies, called data segregation and proxy storage, with six distinct patterns to facilitate the upgradability of deployed smart contracts [78]. These six patterns include basic data segregation, satellite data segregation, and the register pattern, which utilize data segregation techniques [79]. Additionally, inherited storage proxy [80], eternal storage proxy [81], and unstructured storage proxy [82] employ the proxy pattern approach. The inherited storage proxy emphasizes that the storage structures between proxy and implementation contracts remain synchronized. The eternal storage proxy ensures a consistent storage structure across different contract versions. The unstructured storage employs hash values to define storage slots, ensuring compatibility between successive contract versions and their predecessors [83].

Discussion of the results Table 5 presents a comparative analysis of static analysis methods used in various tools. This analysis is based on criteria such as the specific tools that implement these methods and the type of input they use, whether source code or bytecode of smart contracts. The table demonstrates that ML is the predominant method employed by static analysis tools, with a significant 77.55 % (38 out of 49) of tools utilizing this approach. Among these, approximately 73.68 % (28 out of 38) primarily support source code (S) as their input. A subset of these tools 18.42 % (7 out of 38) employs the bytecode (B) of smart contracts as their input. Additionally, a smaller fraction of tools, amounting to 7.89 % (3 out of 38), are versatile enough to handle both bytecode and source code (B/S) as inputs.

Following ML, SE and MC are the next most commonly used methods in tools based on static analysis. SE is employed by approximately ∼ 16.33 % (8 out of 49) of these tools, while MC is utilized by around ∼8.16 % (4 out of 49). A notable example of combining different static analysis methods is IVDS-SCSE [112], which integrates SE with ML to detect reentrancy at the level of bytecode.

Clairvoyance [105] employs the SE method on source code input, while SAFEPAY [116,117] is another tool based on SE that supports both source code and bytecode. The remaining tools that use SE typically take bytecode as their input for analyzing smart contracts. ML methods are predominantly utilized by tools designed for source code analysis, whereas SE methods primarily target bytecode, with approximately 75 % (6 out of 8) of such tools focused on bytecode analysis. In contrast, model-checking methods are predominantly employed by tools tailored for analyzing the source code of smart contracts (3 out 4, ∼75 %).

Notably, RRE [118] stands out as the sole MC tool designed to process bytecode as input. Furthermore, within the studies reviewed, no tool was found that accommodates both source code and bytecode analysis simultaneously in the realm of MC.

Table 6 illustrates the methods used by dynamic analysis tools and compares the types of inputs they analyze based on these methods. FT, RM, and ML are the primary methods used in dynamic analysis. The

Table 5
Static analysis and methods.

Tech.	Tool	Ref.	Input
ML	VulHunter	[84]	B/S
	MANDO-HGT	[62]	B/S
	MODNN	[64]	B/S
	SCVC-DL	[85]	B
	DLVA	[60]	B
	EtherGIS	[59]	B
	VSCL	[86]	B
	FSL-Detect	[87]	B
	DNN-SVD	[88]	B
	ContractWard	[89]	S
	MLAM-SCV	[90]	S
	CodeNet	[91]	S
	DeeSCVHunter	[92]	S
	MLM-SCSA	[93]	S
	SCGRU	[94]	S
	Peculiar	[95]	S
	VDM-AEI	[96]	S
	MRN-GCN	[97]	S
	BwdBAL	[72]	S
	SVScanner	[98]	S
	BFS-EA-RGCN	[99]	S
	ReVulDL	[100]	S
	SCScan	[101]	S
	OVD-MMEE	[102]	S
	SCSVM	[103]	S
	ConvMHSA-SCVD	[104]	S
	Clairvoyance	[105]	S
	SCAD-FOSCF	[106]	S
	GraBit	[65]	S
	SVDM-SSFL	[107]	S
	PSCVFinder	[71]	S
	GNN-SCVD	[108]	S
	CGNNEK-SVD	[109]	S
	MANDO-GURU	[61]	S
	BLSTM-ATT	[13]	S
	TMLVD	[110]	S
	SCVD-CRF	[111]	S
ML/SE	IVDS-SCSE	[112]	B
SE	DETECTCHECKER	[51]	B
	WANA	[50]	B
	Park	[113]	B
	SAILFISH	[114]	B
	EtherSolve	[115]	B
	Clairvoyance	[105]	S
	SAFEPAY	[116,117]	B/S
MC	RRE	[118]	B
	MCV-SCA	[119]	S
	FV-FSM	[120]	S
	VeriSolid	[121] [122]	S

Table 6
Dynamic analysis and methods.

Tech.	Tool	Ref.	Input
FT	ContractFuzzer	[123–125]	B/E
	IR-Fuzz	[126]	B/S
	GasFuzzer	[127]	B/S
	ReDefender	[128]	S
	Smartian	[129]	B
	IcyChecker	[130]	B
	ContraMaster	[131]	E
RM/ML	Dynamit	[70]	E
RM	Jyane	[132]	B

Table 7
Hybrid analysis and methods.

Tech.	Tool	Ref.	Input
FT/ML	xFuzz	[73]	S
ML	DM	[133]	S
CP	Aroc	[134]	S
CP/SE	SGUARD	[135]	B/S
FT/SE	ACVDT	[52]	B/S

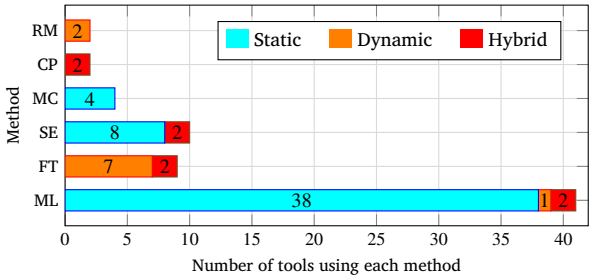

Fig. 7. Data summary from Tables 5–7.

majority of tools employ fuzzing to dynamically analyze smart contracts (7 out of 9, ∼77.78 %).

Regarding two other tools, Jyane [132] utilizes RM on bytecode, whereas Dynamit [70] integrates RM with ML to analyze contracts at the EVM-level (E). It is evident that among the reviewed tools, ContractFuzzer [123–125] and Dynamit [70] conduct EVM-level analysis. Their methodology involves fuzzing, employed by ContractFuzzer, and the combination of RM and ML techniques used by Dynamit.

Table 7 categorizes various hybrid analysis tools based on the methods employed and compares them in terms of their input types. As illustrated, xFuzz [73] utilizes a combination of fuzzing and ML for detection from source code. In contrast, DM [133] employs ML with expert knowledge for its analysis. Aroc [134] uses CP, while SGUARD [135] combines CP with SE. Additionally, ACVDT [52] integrates fuzzing with SE. The tools have varying input requirements: Aroc [134], DM [133], and xFuzz [73] exclusively require the source code of smart contracts (constituting 60 % of tools employing hybrid analysis), whereas SGUARD [135] and ACVDT [52] support both bytecode and source code inputs (making up 40 % of these tools). Fig. 7 summarizes the data from Tables 5–7, providing an overview of the distribution of the different tools across the considered methods.

RQ2. Which methods can theoretically address reentrancy?

The 96 papers reviewed identify six methods to detect and prevent reentrancy in Ethereum smart contracts: code patching, fuzz testing, model checking, runtime monitoring, symbolic execution, and machine learning. Machine learning is the most prevalent approach, applied across static, dynamic, and hybrid analysis techniques.

5.3. RQ3: Is reentrancy countered in practice? Which tools exist?

We acknowledge that some tools are missing from Tables 8 and 9 due to the lack of publicly available data or detailed evaluation metrics in their respective studies. This inconsistency underscores a broader issue in the field: the absence of standardized benchmarks and evaluation criteria. To address this, we compiled the available results for these tools based on the most commonly reported metrics in their original studies. Furthermore, our analysis includes an investigation into zero-day vulnerability detection, which is another critical but underexplored aspect. This review highlights that there is currently no universal metric or benchmark to assess reentrancy detection tools comprehensively, and we have emphasized this as a key challenge and direction for future research in the paper.

This section outlines the effective practices and specialized tools leveraged for detecting and preventing reentrancy and associated exploits. Furthermore, an exploration of the performance and capabilities of these diverse tools is undertaken.

Tables 8 and 9 present a comparative analysis of prominent tools for detecting and preventing reentrancy vulnerabilities. The tools are categorized based on the employed detection/prevention methods, facilitating a structured comparison of their performance. These tables enable a comprehensive understanding of how each tool performs across various metrics.

The comparison considers a multi-faceted evaluation framework encompassing several critical metrics. Firstly, the “Dataset Size” metric (#SC) quantifies the total number of smart contracts comprising each tool’s evaluation dataset. The “Overall Addressed Vulnerabilities” metric (#OAV) represents the total number of vulnerabilities identified/fixed by each tool, encompassing all types of vulnerabilities beyond the scope of reentrancy alone. The metric “REentrancies” (#RE) indicates, for each tool’s dataset, the number of smart contracts that were known to contain at least one instance of a reentrancy vulnerability before the execution of the reported test. Furthermore, according to the tool’s description, we labeled the values under the #RE column to distinguish between the following three cases.

1. The tool cannot distinguish between different types of reentrancy.
2. The tool can distinguish between two or more types of reentrancy.
3. The tool specifically targets a single type of reentrancy.

The “True Positive” metric (#TP) reports the number of correctly addressed reentrancy vulnerabilities, i.e., how many contracts suffer from reentrancy have been correctly identified or fixed. Finally, the “Zero-Day Reentrancies” metric (#0-day) highlights the proactive security analysis capabilities of each tool by enumerating the number of previously unknown (zero-day) reentrancy vulnerabilities discovered/-fixed. The zero values in columns 4 to 7 of the table indicate that no information was available, as the authors of those papers did not provide any data regarding these aspects, or the value is reported as 0. As shown in the table, only a few tools [114,123–125,73] can address zero-day vulnerabilities, and these tools employ fuzzy and SE methods across all categories of dynamic, static, and hybrid analysis. Moreover, the performance of vulnerability detection and prevention tools for smart contracts vary significantly, and the lack of a standardized dataset makes accurate and fair comparisons of these tools challenging.

Table 10 presents the performance assessment results of various analysis types for ML-based tools. Due to the lack of publicly available data or detailed evaluation metrics in the studies of the revised tools, we have only included ML-based tools in Table 10. This underscores a broader issue in the field: the lack of standardized benchmarks and evaluation criteria. Performance measures, including accuracy, precision, recall, F1-score, and Locate are calculated to evaluate the performance of different ML-based reentrancy detection tools within each analysis method. The Locate metric indicates whether the detection tool can identify the specific location of the reentrancy vulnerability, such as the function or line of code. The values of the highest performer are highlighted for each performance measure. The analysis reveals that VDM-AEI and VulHunter achieve superior performance compared to other tools, each attaining a perfect score of 100 % across all evaluated metrics.

Following VDM-AEI and VulHunter, for accuracy, CodeNet demonstrates strong performance with a score of 98.27 %. In terms of precision, CodeNet again leads with 97.65 %, followed closely by PSCVFinder at 97.73 %. For recall, SCGRU stands out with 99.51 %, immediately after the perfect scores. Lastly, in the F1-score metric, CodeNet achieves the highest score of 98.24 % following the tools with perfect scores.

SCGRU and MRN-GCN also exhibit notable performance in recall (99.51 % and 98.18 %, respectively) and precision (91.55 % and

Table 8
Performance and dataset analysis of reentrancy detection and prevention tools.

Method	Tool name	Ref.	#SC	#OAV	#RE	#TP	#0-day
MC	VeriSolid	[122,121]	0	0	0[1]	0	0
	MCV-SCA	[119]	0	0	0[1]	0	0
	FV-FSM	[120]	0	0	0[1]	0	0
CP	Aroc	[134]	598	1428	31[2]	22	0
CP/SE	SGUARD	[135]	5000	0	0[2]	0	0
FT	ContractFuzzer	[123–125]	6991	459	0[1]	14	14
	SMARTIAN	[129]	630	297	0[1]	19	0
	Icycheker	[130]	100	277	0[1]	23	0
	IR-Fuzz	[126]	12,515	485	0[1]	115	0
	ReDefender	[128]	599	22	22[2]	24	4
	GasFuzzer	[127]	3170	610	0[1]	64	0
	ContraMaster	[131]	218	54	14[1]	6	0
FT/ML	xFuzz	[73]	100,139	227	227[1]	0	15
ML	BFS-EA-RGCN	[99]	13,852	0	0[1]	0	0
	EtherGIS	[59]	60,000	0	0[1]	0	0
	DLVA	[60]	24,015	88,188	7,824[1]	3,283	0
	MANDO-HGT	[62]	3235	0	71[1]	0	0
	MODNN	[64]	18,796	58,509	1,528[1]	0	0
	BwdBAL	[72]	4929	0	552[1]	0	0
	PSCVFinder	[71]	200,000	0	0[1]	0	0
	MANDO-GURU	[61]	3235	0	71[1]	0	0
	GraBit	[65]	47,398	0	0[1]	0	0
	VulHunter	[84]	197,123	169,751	0[1]	2,288	0
	GNN-SCVD	[108]	45,102	0	0[1]	0	0
	ASSBert	[136]	20,829	0	0[1]	0	0
	ConvMHSA-SCVD	[104]	40,932	0	0[1]	0	0
	DeeSCVHunter	[92]	40,932	0	0[1]	0	0
	OVD-MMEE	[102]	0	0	0[1]	0	0

Tool considers reentrancy subcategories: [1] = no; [2] = two or more; [3] = one specifically.

Table 9
Performance and dataset analysis of reentrancy detection and prevention tools (continued).

Method	Tool name	Ref.	#SC	#OAV	#RE	#TP	#0-day
ML	Peculiar	[95]	40,932	0	1197[1]	0	0
	ReVulDL	[100]	47,398	0	0[2]	0	0
	SVScanner	[98]	7838	638	0[1]	0	0
	TMLVD	[110]	10,567	0	0[1]	0	0
	MRN-GCN	[97]	12,505	0	3116[1]	0	0
	VSCL	[86]	320,024	0	0[1]	0	0
	FSL-Detect	[87]	19,998	0	110[1]	0	0
	DNN-SVD	[88]	>100K	0	24,161[1]	0	0
	SCVC-DL	[85]	>100K	0	24,161[1]	0	0
	SVDM-SSFL	[107]	5000	0	63[1]	0	0
	CodeNet	[91]	47,518	0	24,367[1]	0	0
	CGNNEK-SVD	[109]	307,396	0	0[1]	0	0
	ContractWard	[89]	297,012	0	100[1]	0	0
	MLM-SCSA	[93]	1013	0	490[1]	48	0
	SCGRU	[94]	9,714	0	1224[1]	0	0
	VDM-AEI	[96]	45,102	0	0[1]	0	0
	BLSTM-ATT	[13]	4000	0	666[1]	0	0
	SCAD-FOSCF	[106]	30,798	0	74[1]	0	0
	MLAM-SCV	[90]	408	0	60[1]	0	0
	SCVD-CRF	[111]	1832	0	207[1]	0	0
	SCSVM	[103]	60	0	10[1]	0	0
	DM	[133]	40,932	795	200[1]	172	0
ML/SE	IVDS-SCSE	[112]	18,912	0	583[1]	0	0
SE	ACVDT	[52]	1000	166	0[1]	22	0
	DEFECTCHECKER	[51]	166,200	29,889	3,892[1]	16	0
	WANA	[50]	89	89	9[1]	9	0
	SAILFISH	[114]	170,494	3391	0[3]	3391	47
	SAFEPAY	[116,117]	46,237	590	0[1]	0	0
	Clairvoyance	[105]	17,770	168	0[2]	168	0
	Park	[113]	1000	5983	0[1]	5983	0
	EtherSolve	[115]	1000	26	0[1]	26	0

Tool considers reentrancy subcategories: [1] = no; [2] = two or more; [3] = one specifically.

Table 10
Comparison of performance of ML-based tools.

Analysis	Tool	Ref.	Accuracy	Precision	Recall	F1	Locate
Static	VSCL	[86]	95.00	99.00	95.00	97.00	✗
	DLVA	[60]	99.70	0	0	0	✗
	GraBit	[65]	0	91.87	97.35	94.44	✗
	SCScan	[101]	0	0	0	0	✓
	SCSVM	[103]	87.55	87.68	87.46	87.51	✗
	SCGRU	[94]	92.64	91.55	99.51	95.36	✗
	TMLVD	[110]	96.14	96.38	95.23	95.80	✗
	CodeNet	[91]	98.27	97.65	98.84	98.24	✗
	MODNN	[64]	98.33	96.51	96.05	96.28	✗
	BwdBAL	[72]	88.80	70.10	81.00	0	✗
	ReVulDL	[100]	84.05	92.00	93.00	93.00	✓
	SCVC-DL	[85]	73.53	0	0	83.81	✗
	ASSBert	[136]	80.2	66.4	74.5	0	✗
	EtherGIS	[59]	95.6	91.00	72.1	80.5	✗
	VulHunter	[84]	**100**	**100**	**100**	**100**	✓
	DNN-SVD	[88]	73.53	0	0	83.81	✗
	MRN-GCN	[97]	96.43	97.07	98.18	97.62	✓
	FSL-Detect	[87]	92.03	0	85.31	90.94	✗
	SVScanner	[98]	94.77	0	90.72	94.64	✗
	VDM-AEI	[96]	**100**	**100**	**100**	**100**	✗
	SCVD-CRF	[111]	90.48	0	90.87	90.56	✗
	IVDS-SCSE	[112]	0	0	0	83.00	✓
	MLM-SCSA	[93]	93.00	69.00	79.00	73.00	✗
	GNN-SCVD	[108]	97.18	92.31	83.33	80.00	✗
	PSCVFinder	[71]	–	97.73	90.53	93.83	✗
	MLAM-SCV	[90]	95.45	95.83	95.45	0	✗
	SVDM-SSFL	[107]	98.64	90.91	78.38	77.33	✗
	OVD-MMEE	[102]	0	89.47	**100**	86.49	✗
	BLSTM-ATT	[13]	89.52	87.38	92.38	89.81	✗
	ContractWard	[89]	0	0	0	94.00	✗
	SCAD-FOSCF	[106]	98.00	99.00	88.00	92.00	✗
	DeeSCVHunter	[92]	93.02	90.70	83.46	86.87	✗
	CGNNEK-SVD	[109]	89.15	85.24	87.62	86.41	✗
	MANDO-HGT	[62]	0	0	0	95.59	✓
	MANDO-GURU	[61]	0	0	0	76.09	✓
	BFS-EA-RGCN	[99]	94.00	94.92	93.96	94.42	✗
	ConvMHSA-SCVD	[104]	94.03	90.49	98.41	94.28	✗
Dynamic	Dynamit	[70]	94.00	0	94.00	93.00	✗
Hybrid	xFuzz	[73]	0	**100**	84.60	0	✓
	DM	[133]	89.74	85.35	86.19	85.76	✗

97.07 %, respectively), further highlighting their effectiveness in reentrancy detection tasks.

In Table 10, seven tools, including MANDO-HGT and MANDO-GURU, are equipped to pinpoint reentrancy vulnerabilities within source code, with varying levels of precision. Some of these tools can locate vulnerabilities at the function level, while others can identify vulnerable statements at the line level, as indicated by checkmarks (✓) in the "Locate" column. The remaining tools, marked with crosses (✗), lack this localization feature.

RQ3. Which tools exist that counter reentrancy in practice?

Existing tools that counter reentrancy vulnerabilities are limited in detecting zero-day vulnerabilities, particularly those relying on machine learning. Furthermore, these tools typically only address reentrancy issues partially, highlighting the need for more comprehensive solutions. There is also a significant need for standardized datasets and benchmarking frameworks to ensure consistent evaluation and enable fair comparisons of detection tools across different types of reentrancy vulnerabilities.

6. Discussion and recommendations

To address the limitations identified in this study, we propose a set of actionable recommendations. These suggestions aim to guide researchers and practitioners in overcoming current challenges and advancing the field of reentrancy vulnerability mitigation in the blockchain.

Unified dataset benchmark. It is essential to develop a standardized dataset that encompasses a range of reentrancy vulnerabilities across various smart contract types. This unified dataset will enable consistent evaluation and comparison of detection tools, thereby improving the effectiveness of detection methodologies.

Unified evaluation framework. It is important to establish a unified evaluation framework where detection tools are assessed under consistent conditions, utilizing standardized datasets, benchmarks, and performance metrics. Such an approach would ensure fair and unbiased comparisons across different tools. Furthermore, developers are encouraged to consider the biases identified in Ref. [137] when designing ML-based detection tools. In this paper, the authors highlight ten common, yet often subtle, pitfalls in applying ML to security. These pitfalls can compromise the validity of results and impede the accurate interpretation of research findings.

Enhancing detection tools for reentrancy types. Future research should focus on developing detection and mitigation tools tailored to the four main types of reentrancy vulnerabilities. Researchers should systematically evaluate and report tool performance for each category to enable accurate benchmarking. This will support the improvement and comparison of detection tools across reentrancy types.

Usage of dynamic and hybrid analysis methods. Based on the categorization of reentrancy vulnerabilities, tools relying solely on static analysis are unable to detect delegate-based and create-based reentrancy types. Therefore, researchers are encouraged to adopt dynamic analysis approaches for comprehensive detection or hybrid analysis methods to ensure full coverage and benefit from the advantages of static analysis.

Exploring beyond Ethereum. Extend investigations of reentrancy vulnerabilities to other blockchain platforms, such as Hyperledger Fabric, Hyperledger Sawtooth, EOS, Tezos, Corda, NEO, and Cardano.

7. Conclusions

Our study represents the first comprehensive investigation into reentrancy vulnerabilities. In this paper, we systematically reviewed the scientific literature from 2018 to 2023 to investigate the issue of reentrancy in Ethereum smart contracts. Our study addresses three primary research questions. To answer RQ1, we found that reentrancy attacks have caused approximately 350 million USD in losses, highlighting their significant financial impact. For RQ2 and RQ3, although various detection and mitigation tools exist, they generally lack effectiveness against zero-day vulnerabilities, and there is no standardized benchmark for evaluating these tools. These insights provide developers and researchers with a clear understanding of the financial impact of reentrancy vulnerabilities and underscore the limitations of current detection methodologies, emphasizing the need for improved evaluation standards.

CRediT authorship contribution statement

Fatemeh Ghiyami Pour: Writing – review & editing, Writing – original draft, Visualization, Resources, Methodology, Data curation, Conceptualization. **Gabriele Costa:** Supervision, Methodology, Data curation, Conceptualization, Writing – review & editing. **Letterio Galletta:** Writing – review & editing, Methodology, Data curation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Fatemeh Ghiyami Pour, Gabriele Costa, and Letterio Galletta report financial support was provided by Next Generation EU-European Union. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding

This work was partially supported by the project SEcurity and RIghts in the CyberSpace SERICS PE0000014 - PNRR M4C2 I.1.3, financed by the European Union Next Generation EU - CUP: D67G22000340001.

Appendix A

See Tables 11 and 12.

References

[1] N. Szabo, Formalizing and securing relationships on public networks, First Monday 2 (1997), https://doi.org/10.5210/fm.v2i9.548.
[2] Slowmist Hacked, https://hacked.slowmist.io/en/, 2018. (Accessed 14 July 2024).
[3] Ethereum, https://github.com/ethereum/goethereum, 2015. (Accessed 12 July 2024).

Table 11
List of attacks exploiting reentrancy bugs and corresponding financial losses.

Case	Name	Ref.	Loss	Date
1	WETH white hat attack	[138]	Unknown	10 Jun 2016
2	The DAO attack	[139]	$60M	17 Jun 2016
3	SpankChain attack	[140]	$40K	9 Oct 2018
4	imBTC Uniswap pool attack	[141]	$300K	18 Apr 2020
5	Lendf.Me attack	[142]	$25M	19 Apr 2020
6	Akropolis attack	[143]	$2M	12 Nov 2020
7	Origin	[144]	$7M	17 Nov 2020
8	ValueDeFi attack	[145]	$1.5M	7 May 2021
9	Rari Capital attack	[146]	$10M	8 May 2021
10	BurgerSwap attack	[147]	$7.2M	27 May 2021
11	Iron Finance attack	[148]	Unknown	16 Jun 2021
12	PolyDEX attack	[149]	$500K	20 Jun 2021
13	DeFiPie attack	[142]	$100K	12 Jul 2021
14	Sanshu Inu attack	[150]	$111K	20 Jul 2021
15	XSurge attack	[151]	$5M	16 Aug 2021
16	C.R.E.A.M. Finance attack	[152]	$5M	31 Aug 2021
17	Siren Protocol attack	[153]	$3.5M	03 Sep 2021
18	CreatureToadz attack	[154]	$340K	21 Oct 2021
19	Grim Finance attack	[155]	$30M	18 Dec 2021
20	Visor Finance attack	[152]	$8.2M	21 Dec 2021
21	HypeBears attack	[156]	Unknown	3 Feb 2022
22	Bacon Protocol attack	[157]	~$1M	5 Mar 2022
23	Paraluni attack	[12]	$1.7M	13 Mar 2022
24	Agave Finance attack	[158]	Unknown	15 Mar 2022
25	Hundred Finance attack	[159]	$6.5M	15 Mar 2022
26	Revest Finance attack	[160]	$2M	27 Mar 2022
27	Voltage Finance attack	[161]	$4M	31 Mar 2022
28	BNB Brokers attack	[162]	Unknown	27 Apr 2022
29	Fei Protocol attack +Rari Capital attack	[163]	$80M	30 Apr 2022
30	Bistroo attack	[164]	$47K	7 May 2022
31	Ownly attack	[165]	$37K	10 May 2022

Table 12
List of attacks exploiting reentrancy bugs and corresponding financial losses (continued).

Case	Name	Ref.	Loss	Date
32	Omni attack	[166]	$1.43M	10 Jul 2022
33	Stader Labs NearX attack	[167]	$830K	16 Aug 2022
34	Thunder Brawl attack	[168]	Unknown	30 Sep 2022
35	QuickSwap Lend attack	[169]	$220K	23 Oct 2022
36	n00dleSwap attack	[170]	$29K	26 Oct 2022
37	DFX Finance attack	[171]	$4M	10 Nov 2022
38	Defrost Finance attack	[172]	$173K	23 Dec 2022
39	Jaypeggers attack	[173]	$18.5K	29 Dec 2022
40	Midas Capital attack	[174]	$660K	15 Jan 2023
41	Orion Protocol attack	[175]	$3M	2 Feb 2023
42	dForce Network attack	[171]	$3.65M	9 Feb 2023
43	Dynamic attack	[176]	$22.4K	22 Feb 2023
44	Sentiment attack	[177]	$1M	4 Apr 2023
45	Paribus attack	[178]	$67.8K	11 Apr 2023
46	Sturdy attack	[170]	$800K	12 Jun 2023
47	Arcadia Finance attack	[179]	$460K	10 Jul 2023
48	Libertify attack	[168]	$452K	11 Jul 2023
49	Conic Finance attack	[180]	$3.35M	21 Jul 2023
50	EraLend attack	[171]	$3.4M	25 Jul 2023
51	Curve attack	[181]	$50M	30 Jul 2023
52	Earning.Farm attack	[182]	$528K	9 Aug 2023
53	Defiway attack	[183]	$400K	3 Oct 2023
54	Stars Arena attack	[184]	$2.88M	7 Oct 2023
55	0X0 attack	[185]	Unknown	27 Oct 2023
56	Peapods Finance attack	[186]	$230K	13 Dec 2023
57	NFT Trader attack	[187]	$3M	16 Dec 2023
58	GoodDollar attack	[188]	$625K	16 Dec 2023

[4] EOS, https://eos.io/, 2018. (Accessed 12 July 2024).
[5] TRON, TRON, https://tron.network/, 2017. (Accessed 14 July 2024).
[6] M. Swapna, et al., Function locking fused with hashing technique to mitigate re-entrancy attacks in smart contract, in: Proceedings of the 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), IEEE, 2023, pp. 1–4, https://doi.org/10.1109/ICCCNT56998.2023.10306463.
[7] H. Chen, M. Pendleton, L. Njilla, et al., A survey on Ethereum systems security: vulnerabilities, attacks and defenses, ACM Comput. Surv. 53 (2020) 1–43, https://doi.org/10.1145/3391195.
[8] P. Daian, Analysis of the DAO exploit, http://hackingdistributed.com/2016/06/18/analysisof-the-dao-exploit/, 2016. (Accessed 14 July 2024).
[9] R. Feichtinger, R. Fritsch, L. Heimbach, et al., Sok: attacks on DAOs, arXiv preprint, arXiv:2406.15071, 2024.
[10] C. Ferreira Torres, M. Baden, R. Norvill, et al., Ægis: shielding vulnerable smart contracts against attacks, in: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, ACM, 2020, pp. 584–597, https://doi.org/10.1145/3320269.3384756.
[11] X. Xiong, Z. Wang, T. Cui, et al., Market misconduct in decentralized finance (defi): analysis, regulatory challenges and policy implications, arXiv preprint, arXiv:2311.17715, 2023.
[12] H. Huang, Q. Zhang, T. Li, et al., Economic systems in the metaverse: basics, state of the art, and challenges, ACM Comput. Surv. 56 (4) (2023) 1–33, https://doi.org/10.1145/3626315.
[13] P. Qian, Z. Liu, Q. He, et al., Towards automated reentrancy detection for smart contracts based on sequential models, IEEE Access 8 (2020) 19685–19695, https://doi.org/10.1109/ACCESS.2020.2969429.
[14] X. Li, P. Jiang, T. Chen, et al., A survey on the security of blockchain systems, Future Gener. Comput. Syst. 107 (2020) 841–853, https://doi.org/10.1016/j.future.2017.08.020.
[15] R. Kiani, V.S. Sheng, Ethereum smart contract vulnerability detection and machine learning-driven solutions: a systematic literature review, Electronics 13 (2024) 2295, https://doi.org/10.3390/electronics13122295.
[16] Y. Wang, S. Sheng, Y. Wang, A systematic literature review on smart contract vulnerability detection by symbolic execution, in: J. Chen, B. Wen, T. Chen (Eds.), Blockchain and Trustworthy Systems, Springer, Singapore, 2024, pp. 226–241, https://doi.org/10.1007/978-981-99-8101-4_16.
[17] O. Zaazaa, H. El Bakkali, A systematic literature review of undiscovered vulnerabilities and tools in smart contract technology, J. Intell. Syst. 32 (2023) 20230038, https://doi.org/10.1515/jisys-2023-0038.
[18] R. Van Dinter, B. Tekinerdogan, C. Catal, Automation of systematic literature reviews: a systematic literature review, Inf. Softw. Technol. 136 (2021) 106589, https://doi.org/10.1016/j.infsof.2021.106589.
[19] P. Tolmach, Y. Li, S.W. Lin, et al., A survey of smart contract formal specification and verification, ACM Comput. Surv. 54 (2021) 1–38, https://doi.org/10.1145/3464421.
[20] T. Durieux, J.F. Ferreira, R. Abreu, et al., Empirical review of automated analysis tools on 47,587 Ethereum smart contracts, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ACM, 2020, pp. 530–541, https://doi.org/10.1145/3377811.3380364.
[21] S.S. Kushwaha, S. Joshi, D. Singh, et al., Ethereum smart contract analysis tools: a systematic review, IEEE Access 10 (2022) 57037–57062, https://doi.org/10.1109/ACCESS.2022.3169902.
[22] S.S. Kushwaha, S. Joshi, D. Singh, et al., Systematic review of security vulnerabilities in Ethereum blockchain smart contract, IEEE Access 10 (2022) 6605–6621, https://doi.org/10.1109/ACCESS.2021.3140091.
[23] H. Rameder, M. Di Angelo, G. Salzer, Review of automated vulnerability analysis of smart contracts on Ethereum, Front. Blockchain 5 (2022) 814977, https://doi.org/10.3389/fbloc.2022.814977.
[24] Z.A. Khan, A.S. Namin, A survey of vulnerability detection techniques by smart contract tools, IEEE Access 12 (2024) 70870–70910, https://doi.org/10.1109/ACCESS.2024.3401623.
[25] W. Haouari, A.S. Hafid, M. Fokaefs, Vulnerabilities of smart contracts and mitigation schemes: a comprehensive survey, arXiv preprint, arXiv:2403.19805, 2024.
[26] D. Macrinici, C. Cartofeanu, S. Gao, Smart contract applications within blockchain technology: a systematic mapping study, Telemat. Inform. 35 (2018) 2337–2354, https://doi.org/10.1016/j.tele.2018.10.004.
[27] A. Vacca, A. Di Sorbo, C.A. Visaggio, et al., A systematic literature review of blockchain and smart contract development: techniques, tools, and open challenges, J. Syst. Softw. 174 (2021) 110891, https://doi.org/10.1016/j.jss.2020.110891.
[28] J. Chen, X. Xia, D. Lo, et al., Maintenance-related concerns for post-deployed Ethereum smart contract development: issues, techniques, and future challenges, Empir. Softw. Eng. 26 (2021) 117, https://doi.org/10.1007/s10664-021-10018-0.
[29] P. De Giovanni, Blockchain and smart contracts in supply chain management: a game theoretic model, Int. J. Prod. Econ. 228 (2020) 107855, https://doi.org/10.1016/j.ijpe.2020.107855.
[30] K. Christidis, M. Devetsikiotis, Blockchains and smart contracts for the Internet of Things, IEEE Access 4 (2016) 2292–2303, https://doi.org/10.1109/ACCESS.2016.2566339.
[31] X. Wang, F. Xu, The value of smart contract in trade finance, Manuf. Serv. Oper. Manag. 25 (2023) 2056–2073, https://doi.org/10.1287/msom.2022.1126.
[32] A. Saini, Q. Zhu, N. Singh, et al., A smart-contract-based access control framework for cloud smart healthcare system, IEEE Internet Things J. 8 (2020) 5914–5925, https://doi.org/10.1109/JIOT.2020.3032997.
[33] K. Petersen, R. Feldt, S. Mujtaba, et al., Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, BCS Learning & Development Ltd., Swindon, 2008, pp. 68–77, https://doi.org/10.14236/ewic/EASE2008.8.
[34] L.S.H. Colin, P.M. Mohan, J. Pan, et al., An integrated smart contract vulnerability detection tool using multi-layer perceptron on real-time solidity smart contracts, IEEE Access (2024), https://doi.org/10.1109/ACCESS.2024.3364351.

[35] A. Pinna, S. Ibba, G. Baralla, et al., A massive analysis of Ethereum smart contracts empirical study and code metrics, IEEE Access 7 (2019) 78194–78213, https://doi.org/10.1109/ACCESS.2019.2921936.

[36] M.E. Fagan, Design and code inspections to reduce errors in program development, IBM Syst. J. 38 (1999) 258–287, https://doi.org/10.1147/sj.382.0258.

[37] M. Coblenz, R. Oei, T. Etzel, et al., Obsidian: typestate and assets for safer blockchain programming, ACM Trans. Program. Lang. Syst. 42 (2020) 1–82, https://doi.org/10.1145/3417516.

[38] F. Schrans, S. Eisenbach, S. Drossopoulou, Writing safe smart contracts in Flint, in: Proceedings of the Companion Proceedings of the 2nd International Conference on the Art, Science, and Engineering of Programming, ACM, 2018, pp. 218–219, https://doi.org/10.1145/3191697.3213790.

[39] A. Das, S. Balzer, J. Hoffmann, et al., Resource-aware session types for digital contracts, in: Proceedings of 2021 IEEE 34th Computer Security Foundations Symposium (CSF), IEEE, 2021, pp. 1–16, https://doi.org/10.1109/CSF51468.2021.00004.

[40] E. Cecchetti, S. Yao, H. Ni, et al., Compositional security for reentrant applications, in: Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 1249–1267, https://doi.org/10.1109/SP40001.2021.00084.

[41] M. Rodler, W. Li, G.O. Karame, et al., Sereum: protecting existing smart contracts against re-entrancy attacks, in: Proceedings of the Network and Distributed System Security Symposium (NDSS'19), Internet Society, 2019, https://doi.org/10.14722/ndss.2019.23413.

[42] M.J. Page, J.E. McKenzie, P.M. Bossuyt, et al., The PRISMA 2020 statement: an updated guideline for reporting systematic reviews, Int. J. Surg. 88 (2021) 105906, https://doi.org/10.1016/j.ijsu.2021.105906.

[43] B. Mueller, Smashing Ethereum smart contracts for fun and real profit, in: Proceedings of the 9th Annual HITB Security Conference (HITBSecConf), HITB, 2018, p. 54.

[44] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, et al., Smartcheck: static analysis of Ethereum smart contracts, in: Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, ACM, 2018, pp. 9–16, https://doi.org/10.1145/3194113.3194115.

[45] J. Feist, G. Grieco, A. Groce, 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), IEEE, 2019, pp. 8–15, https://doi.org/10.1109/WETSEB.2019.00008.

[46] L. Luu, D.H. Chu, H. Olickel, et al., Making smart contracts smarter, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 254–269, https://doi.org/10.1145/2976749.2978309.

[47] WebAssembly, WebAssembly, https://webassembly.org/, 2020. (Accessed 22 August 2024).

[48] J.C. King, Symbolic execution and program testing, Commun. ACM 19 (1976) 385–394, https://doi.org/10.1145/360248.360252.

[49] A. Sabbaghi, M.R. Keyvanpour, A systematic review of search strategies in dynamic symbolic execution, Comput. Stand. Interfaces 72 (2020) 103444, https://doi.org/10.1016/j.csi.2020.103444.

[50] B. Jiang, Y. Chen, D. Wang, et al., WANA: symbolic execution of wasm bytecode for extensible smart contract vulnerability detection, in: Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2021, pp. 926–937, https://doi.org/10.1109/QRS54544.2021.00102.

[51] J. Chen, X. Xia, D. Lo, et al., DefectChecker: automated smart contract defect detection by analyzing EVM bytecode, IEEE Trans. Softw. Eng. 48 (2021) 2189–2207, https://doi.org/10.1109/TSE.2021.3054928.

[52] M. Fu, L. Wu, Z. Hong, et al., A critical-path-coverage-based vulnerability detection method for smart contracts, IEEE Access 7 (2019) 147327–147344, https://doi.org/10.1109/ACCESS.2019.2947146.

[53] C. Shou, J. Liu, D. Lu, et al., LLM4Fuzz: guided fuzzing of smart contracts with large language models, arXiv preprint, arXiv:2401.11108, 2024.

[54] J. Su, H.N. Dai, L. Zhao, et al., Effectively generating vulnerable transaction sequences in smart contracts with reinforcement learning-guided fuzzing, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ACM, 2022, pp. 1–12, https://doi.org/10.1145/3551349.3560429.

[55] I. Cassar, A. Francalanza, L. Aceto, A survey of runtime monitoring instrumentation techniques, arXiv preprint, arXiv:1708.07229, 2017.

[56] C. Baier, J.P. Katoen, Principles of Model Checking, MIT Press, Cambridge, 2008.

[57] M. Almakhour, L. Sliman, A.E. Samhat, et al., Verification of smart contracts: a survey, Pervasive Mob. Comput. 67 (2020) 101227, https://doi.org/10.1016/j.pmcj.2020.101227.

[58] M. Krichen, A survey on formal verification and validation techniques for Internet of Things, Appl. Sci. 13 (2023) 8122, https://doi.org/10.3390/app13148122.

[59] Q. Zeng, EtherGIS: a vulnerability detection framework for Ethereum smart contracts based on graph learning features, in: Proceedings of the 46th IEEE Annual Computers, Software, and Applications Conference (COMPSAC 2022), IEEE, 2022, pp. 1742–1749, https://doi.org/10.1109/COMPSAC54236.2022.00277.

[60] T. Abdelaziz, A. Hobor, Smart learning to find dumb contracts (extended version), arXiv preprint, arXiv:2304.10726, 2023.

[61] H.H. Nguyen, N.M. Nguyen, H.P. Doan, et al., MANDO-GURU: vulnerability detection for smart contract source code by heterogeneous graph embeddings, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, 2022, pp. 1736–1740, https://doi.org/10.1145/3540250.3558927.

[62] H.H. Nguyen, N.M. Nguyen, C. Xie, et al., MANDO-HGT: heterogeneous graph transformers for smart contract vulnerability detection, in: Proceedings of the 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), IEEE, 2023, pp. 334–346, https://doi.org/10.1109/MSR59073.2023.00052.

[63] J. Zhou, G. Cui, S. Hu, et al., Graph neural networks: a review of methods and applications, AI Open 1 (2020) 57–81, https://doi.org/10.1016/j.aiopen.2021.01.001.

[64] L. Zhang, J. Wang, W. Wang, et al., Smart contract vulnerability detection combined with multi-objective detection, Comput. Netw. 217 (2022) 109289, https://doi.org/10.1016/j.comnet.2022.109289.

[65] H. Zhu, K. Yang, L. Wang, et al., GraBit: a sequential model-based framework for smart contract vulnerability detection, in: Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 568–577, https://doi.org/10.1109/ISSRE59848.2023.00024.

[66] H. Salehinejad, S. Sankar, J. Barfett, et al., Recent advances in recurrent neural networks, arXiv preprint, arXiv:1801.01078, 2017.

[67] Y. Liu, Y. Wang, J. Zhang, New machine learning algorithm: Random Forest, in: B. Liu, M. Ma, J. Chang (Eds.), Information Computing and Applications, Springer, Berlin, 2018, pp. 246–252, https://doi.org/10.1007/978-3-642-34062-8_32.

[68] N. Houlsby, Efficient Bayesian active learning and matrix modelling, Ph.D. thesis, University of Cambridge, Cambridge, 2014.

[69] Y. Chang, X. Wang, J. Wang, et al., A survey on evaluation of large language models, ACM Trans. Intell. Syst. Technol. 15 (2024) 1–45, https://doi.org/10.1145/3641289.

[70] M. Eshghie, C. Artho, D. Gurov, Dynamic vulnerability detection on smart contracts using machine learning, in: Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2021, pp. 305–312, https://doi.org/10.1145/3463274.3463348.

[71] L. Yu, J. Lu, X. Liu, et al., PSCVFinder: a prompt-tuning based framework for smart contract vulnerability detection, in: Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 556–567, https://doi.org/10.1109/ISSRE59848.2023.00030.

[72] J. Zhang, L. Tu, J. Cai, et al., Vulnerability detection for smart contract via backward Bayesian active learning, in: J. Zhou, S. Adepu, C. Alcaraz, et al. (Eds.), Applied Cryptography and Network Security Workshops, Springer, Berlin, 2022, pp. 66–83, https://doi.org/10.1007/978-3-031-16815-4_5.

[73] Y. Xue, J. Ye, W. Zhang, et al., xFuzz: machine learning guided cross-contract fuzzing, IEEE Trans. Dependable Secure Comput. 21 (2022) 515–529, https://doi.org/10.1109/TDSC.2022.3182373.

[74] X.Y. Liu, J. Wu, Z.H. Zhou, Exploratory undersampling for class-imbalance learning, IEEE Trans. Syst. Man Cybern., Part B, Cybern. 39 (2008) 539–550, https://doi.org/10.1109/TSMCB.2008.2007853.

[75] N. Dissanayake, A. Jayatilaka, M. Zahedi, et al., Software security patch management-a systematic literature review of challenges, approaches, tools and practices, Inf. Softw. Technol. 144 (2022) 106771, https://doi.org/10.1016/j.infsof.2021.106771.

[76] Q. Chen, T. Zhou, K. Liu, et al., Tips: towards automating patch suggestion for vulnerable smart contracts, Autom. Softw. Eng. 30 (2023) 31, https://doi.org/10.1007/s10515-023-00392-y.

[77] I. Qasse, M. Hamdaqa, B.Þ Jónsson, Immutable in principle, upgradeable by design: exploratory study of smart contract upgradeability, arXiv preprint, arXiv:2407.01493, 2024.

[78] V.C. Bui, S. Wen, J. Yu, et al., Evaluating upgradable smart contract, in: Proceedings of the 2021 IEEE International Conference on Blockchain (Blockchain), IEEE, 2021, pp. 252–256, https://doi.org/10.1109/Blockchain53845.2021.00041.

[79] M. Wöhrer, U. Zdun, Design patterns for smart contracts in the Ethereum ecosystem, in: Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2018, pp. 1513–1520, https://doi.org/10.1109/cybermatics_2018.2018.00255.

[80] OpenZeppelin, Upgradeability using inherited storage, https://github.com/OpenZeppelin/openzeppelin-labs/tree/master/upgradeability_using_inherited_storage, 2024. (Accessed 10 August 2024).

[81] OpenZeppelin, Upgradeability using eternal storage, https://github.com/OpenZeppelin/openzeppelin-labs/tree/master/upgradeability_using_eternal_storage, 2024. (Accessed 10 August 2024).

[82] OpenZeppelin, Upgradeability using unstructured storage, https://github.com/OpenZeppelin/openzeppelin-labs/tree/master/upgradeability_using_unstructured_storage, 2024. (Accessed 10 August 2024).

[83] A.M. Ebrahimi, B. Adams, G.A. Oliva, et al., A large-scale exploratory study on the proxy pattern in Ethereum, Empir. Softw. Eng. 29 (2024) 1–51, https://doi.org/10.1007/s10664-024-10485-1.

[84] Z. Li, S. Lu, R. Zhang, et al., VulHunter: hunting vulnerable smart contracts at EVM bytecode-level via multiple instance learning, IEEE Trans. Softw. Eng. 49 (2023) 4886–4916, https://doi.org/10.1109/TSE.2023.3317209.

[85] M. Rossini, M. Zichichi, S. Ferretti, Smart contracts vulnerability classification through deep learning, in: Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems, ACM, 2022, pp. 1229–1230, https://doi.org/10.1145/3560905.3568175.

[86] F. Mi, Z. Wang, C. Zhao, et al., VSCL: automating vulnerability detection in smart contracts with deep learning, in: Proceedings of the 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2021, pp. 1–9, https://doi.org/10.1109/ICBC51069.2021.9461050.

[87] D. He, K. Ding, S. Chan, et al., Unknown threats detection methods of smart contracts, IEEE Internet Things J. (2023), https://doi.org/10.1109/JIOT.2023.3299492.

[88] M. Rossini, M. Zichichi, S. Ferretti, On the use of deep neural networks for security vulnerabilities detection in smart contracts, in: Proceedings of the 2023 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops), IEEE, 2023, pp. 74–79, https://doi.org/10.1109/PerComWorkshops56833.2023.10150302.

[89] W. Wang, J. Song, G. Xu, et al., ContractWard: automated vulnerability detection models for Ethereum smart contracts, IEEE Trans. Netw. Sci. Eng. 8 (2020) 1133–1144, https://doi.org/10.1109/TNSE.2020.2968505.

[90] Y. Xu, G. Hu, L. You, et al., A novel machine learning-based analysis model for smart contract vulnerability, Secur. Commun. Netw. 2021 (2021) 1–12, https://doi.org/10.1155/2021/5798033.

[91] S.J. Hwang, S.H. Choi, J. Shin, et al., CodeNet: code-targeted convolutional neural network architecture for smart contract vulnerability detection, IEEE Access 10 (2022) 32595–32607, https://doi.org/10.1109/ACCESS.2022.3162065.

[92] X. Yu, H. Zhao, B. Hou, et al., DeeSCVHunter: a deep learning-based framework for smart contract vulnerability detection, in: Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), IEEE, 2021, pp. 1–8, https://doi.org/10.1109/IJCNN52387.2021.9534324.

[93] P. Momeni, Y. Wang, R. Samavi, Machine learning model for smart contracts security analysis, in: Proceedings of the 2019 17th International Conference on Privacy, Security and Trust (PST), IEEE, 2019, pp. 1–6, https://doi.org/10.1109/PST47121.2019.8949045.

[94] J. Liang, Y. Zhai, SCGRU: a model for Ethereum smart contract vulnerability detection combining CNN and BiGRU-attention, in: Proceedings of the 2023 8th International Conference on Signal and Image Processing (ICSIP), IEEE, 2023, pp. 831–837, https://doi.org/10.1109/ICSIP57908.2023.10270857.

[95] H. Wu, Z. Zhang, S. Wang, et al., Peculiar: smart contract vulnerability detection based on crucial data flow graph and pre-training techniques, in: Proceedings of the 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2021, pp. 378–389, https://doi.org/10.1109/ISSRE52982.2021.00047.

[96] L. Li, Y. Liu, G. Sun, et al., Smart contract vulnerability detection based on automated feature extraction and feature interaction, IEEE Trans. Knowl. Data Eng. 36 (2023) 4916–4929, https://doi.org/10.1109/TKDE.2023.3333371.

[97] H. Liu, Y. Fan, L. Feng, et al., Vulnerable smart contract function locating based on multi-relational nested graph convolutional network, J. Syst. Softw. 204 (2023) 111775, https://doi.org/10.1016/j.jss.2023.111775.

[98] H. Zhang, W. Zhang, Y. Feng, et al., SVScanner: detecting smart contract vulnerabilities via deep semantic extraction, J. Inf. Secur. Appl. 75 (2023) 103484, https://doi.org/10.1016/j.jisa.2023.103484.

[99] D. Chen, L. Feng, Y. Fan, et al., Smart contract vulnerability detection based on semantic graph and residual graph convolutional networks with edge attention, J. Syst. Softw. 202 (2023) 111705, https://doi.org/10.1016/j.jss.2023.111705.

[100] Z. Zhang, Y. Lei, M. Yan, et al., Reentrancy vulnerability detection and localization: a deep learning based two-phase approach, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ACM, 2022, pp. 1–13, https://doi.org/10.1145/3551349.3560428.

[101] X. Hao, W. Ren, W. Zheng, et al., SCScan: a SVM-based scanning system for vulnerabilities in blockchain smart contracts, in: Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE, 2020, pp. 1598–1605, https://doi.org/10.1109/TrustCom50675.2020.00221.

[102] D. Yuan, X. Wang, Y. Li, et al., Optimizing smart contract vulnerability detection via multi-modality code and entropy embedding, J. Syst. Softw. 202 (2023) 111699, https://doi.org/10.1016/j.jss.2023.111699.

[103] Z. Yang, W. Zhu, Improvement and optimization of vulnerability detection methods for Ethernet smart contracts, IEEE Access 11 (2023) 78207, https://doi.org/10.1109/ACCESS.2023.3298672.

[104] M. Li, X. Ren, H. Fu, et al., ConvMHSA-SCVD: enhancing smart contract vulnerability detection through a knowledge-driven and data-driven framework, in: Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 578–589, https://doi.org/10.1109/ISSRE59848.2023.00025.

[105] J. Ye, M. Ma, Y. Lin, et al., Clairvoyance: cross-contract static analysis for detecting practical reentrancy vulnerabilities in smart contracts, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, ACM, 2020, pp. 274–275, https://doi.org/10.1145/3377812.3390908.

[106] L. Duan, L. Yang, C. Liu, et al., A new smart contract anomaly detection method by fusing opcode and source code features for blockchain services, IEEE Trans. Netw. Serv. Manag. 20 (2023) 4354–4368, https://doi.org/10.1109/TNSM.2023.3278311.

[107] D. Han, Q. Li, L. Zhang, et al., A smart contract vulnerability detection model based on syntactic and semantic fusion learning, Wirel. Commun. Mob. Comput. 2023 (2023), https://doi.org/10.1155/2023/9212269.

[108] Z. Wei, W. Zheng, X. Su, et al., A graph neural network-based smart contract vulnerability detection method with artificial rule, in: L. Iliadis, A. Papaleonidas, P. Angelov (Eds.), Artificial Neural Networks and Machine Learning - ICANN 2023, Springer, Berlin, 2023, pp. 241–252, https://doi.org/10.1007/978-3-031-44216-2_20.

[109] Z. Liu, P. Qian, X. Wang, et al., Combining graph neural networks with expert knowledge for smart contract vulnerability detection, IEEE Trans. Knowl. Data Eng. 35 (2021) 1296–1310, https://doi.org/10.1109/TKDE.2021.3095196.

[110] Q. Zhou, K. Zheng, K. Zhang, et al., Vulnerability analysis of smart contract for blockchain-based IoT applications: a machine learning approach, IEEE Internet Things J. 9 (2022) 24695–24707, https://doi.org/10.1109/JIOT.2022.3196269.

[111] B. Wang, H. Chu, P. Zhang, et al., Smart contract vulnerability detection using code representation fusion, in: Proceedings of the 2021 28th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2021, pp. 564–565, https://doi.org/10.1109/APSEC53868.2021.00069.

[112] Y. Yao, H. Li, X. Yang, et al., An improved vulnerability detection system of smart contracts based on symbolic execution, in: Proceedings of the 2022 IEEE International Conference on Big Data (Big Data), IEEE, 2022, pp. 3225–3234, https://doi.org/10.1109/BigData55660.2022.10020730.

[113] P. Zheng, Z. Zheng, X. Luo, Park: accelerating smart contract vulnerability detection via parallel-fork symbolic execution, in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, ACM, 2022, pp. 740–751, https://doi.org/10.1145/3533767.3534395.

[114] P. Bose, D. Das, Y. Chen, et al., SAILFISH: vetting smart contract state-inconsistency bugs in seconds, in: Proceedings of the 2022 IEEE Symposium on Security and Privacy (SP), IEEE, 2022, pp. 161–178, https://doi.org/10.2139/ssrn.4106945.

[115] F. Contro, M. Crosara, M. Ceccato, et al., EtherSolve: computing an accurate control-flow graph from Ethereum bytecode, arXiv preprint, arXiv:2103.09113, 2021.

[116] Y. Li, H. Liu, Z. Yang, et al., SafePay on Ethereum: a framework for detecting unfair payments in smart contracts, in: Proceedings of the 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2020, pp. 1219–1222, https://doi.org/10.1109/ICDCS47774.2020.00116.

[117] Y. Li, H. Liu, Z. Yang, et al., Protect your smart contract against unfair payment, in: Proceedings of the 2020 International Symposium on Reliable Distributed Systems (SRDS), IEEE, 2020, pp. 61–70, https://doi.org/10.1109/SRDS51746.2020.00014.

[118] Y. Tang, Z. Li, Y. Bai, Rethinking of reentrancy on the Ethereum, in: Proceedings of the 2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), IEEE, 2021, pp. 68–75, https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech52372.2021.00025.

[119] I. Garfatta, K. Klai, M. Graïet, et al., Model checking of vulnerabilities in smart contracts: a solidity-to-CPN approach, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, ACM, 2022, pp. 316–325, https://doi.org/10.1145/3477314.3507309.

[120] M. Almakhour, L. Sliman, A.E. Samhat, A. Mellouk, A formal verification approach for composite smart contracts security using FSM, J. King Saud Univ., Comput. Inf. Sci. 35 (2023) 70–86, https://doi.org/10.1016/j.jksuci.2022.08.029.

[121] A. Mavridou, A. Laszka, E. Stachtiari, et al., VeriSolid: correct-by-design smart contracts for Ethereum, in: I. Goldberg, T. Moore (Eds.), Financial Cryptography and Data Security, Springer, Berlin, 2019, pp. 446–465, https://doi.org/10.1007/978-3-030-32101-7_27.

[122] K. Nelaturu, A. Mavridoul, A. Veneris, et al., Verified development and deployment of multiple interacting smart contracts with VeriSolid, in: Proceedings of the 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2020, pp. 1–9, https://doi.org/10.1109/ICBC48266.2020.9169428.

[123] B. Jiang, Y. Liu, W.K. Chan, ContractFuzzer: fuzzing smart contracts for vulnerability detection, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ACM, 2018, pp. 259–269, https://doi.org/10.1145/3238147.3238177.

[124] X. Mei, I. Ashraf, B. Jiang, et al., A fuzz testing service for assuring smart contracts, in: Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), IEEE, 2019, pp. 544–545, https://doi.org/10.1109/QRS-C.2019.00116.

[125] W.K. Chan, B. Jiang, Fuse: an architecture for smart contract fuzz testing service, in: Proceedings of the 2018 25th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2018, pp. 707–708, https://doi.org/10.1109/APSEC.2018.00099.

[126] Z. Liu, P. Qian, J. Yang, et al., Rethinking smart contract fuzzing: fuzzing with invocation ordering and important branch revisiting, IEEE Trans. Inf. Forensics Secur. 18 (2023) 1237–1251, https://doi.org/10.1109/TIFS.2023.3237370.

[127] I. Ashraf, X. Ma, B. Jiang, et al., GasFuzzer: fuzzing Ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities, IEEE Access 8 (2020) 99552–99564, https://doi.org/10.1109/ACCESS.2020.2995183.

[128] B. Li, Z. Pan, T. Hu, ReDefender: detecting reentrancy vulnerabilities in smart contracts automatically, IEEE Trans. Reliab. 71 (2022) 984–999, https://doi.org/10.1109/TR.2022.3161634.

[129] J. Choi, D. Kim, S. Kim, et al., SMARTIAN: enhancing smart contract fuzzing with static and dynamic data-flow analyses, in: Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE, 2021, pp. 227–239, https://doi.org/10.1109/ASE51524.2021.9678888.

[130] M. Ye, Y. Nan, Z. Zheng, et al., Detecting state inconsistency bugs in DApps via on-chain transaction replay and fuzzing, in: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ACM, 2023, pp. 298–309, https://doi.org/10.1145/3597926.3598057.

[131] H. Wang, Y. Liu, Y. Li, et al., Oracle-supported dynamic exploit generation for smart contracts, IEEE Trans. Dependable Secure Comput. 19 (2020) 1795–1809, https://doi.org/10.1109/TDSC.2020.3037332.

[132] Y. Fang, C. Wang, Z. Sun, et al., Jyane: detecting reentrancy vulnerabilities based on path profiling method, in: Proceedings of the 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS), IEEE, 2021, pp. 274–282, https://doi.org/10.1109/ICPADS53394.2021.00040.

[133] Z. Liu, M. Jiang, S. Zhang, et al., A smart contract vulnerability detection mechanism based on deep learning and expert rules, IEEE Access 11 (2023) 77990–77999, https://doi.org/10.1109/ACCESS.2023.3298048.

[134] H. Jin, Z. Wang, M. Wen, et al., Aroc: an automatic repair framework for on-chain smart contracts, IEEE Trans. Softw. Eng. 48 (2021) 4611–4629, https://doi.org/10.1109/TSE.2021.3123170.

[135] T.D. Nguyen, L.H. Pham, J. Sun, SGUARD: towards fixing vulnerable smart contracts automatically, in: Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 1215–1229, https://doi.org/10.1109/SP40001.2021.00057.

[136] X. Sun, L. Tu, J. Zhang, et al., ASSBert: active and semi-supervised Bert for smart contract vulnerability detection, J. Inf. Secur. Appl. 73 (2023) 103423, https://doi.org/10.1016/j.jisa.2023.103423.

[137] D. Arp, E. Quiring, F. Pendlebury, et al., Dos and don'ts of machine learning in computer security, arXiv preprint, arXiv:2010.09470, 2020.

[138] R. User, From the maker DAO slack: today we discovered a vulnerability..., https://www.reddit.com/r/ethereum/comments/4nmohu/from_the_maker_dao_slack_today_we_discovered_a/?user_id=360657100019, 2016. (Accessed 16 July 2024).

[139] G. Zhu, D. He, H. An, et al., The governance technology for blockchain systems: a survey, Front. Comput. Sci. 18 (2024) 182813, https://doi.org/10.1007/s11704-023-3113-x.

[140] H. Guo, X. Yu, A survey on blockchain technology and its security, Blockchain Res. Appl. 3 (2022) 100067, https://doi.org/10.1016/j.bcra.2022.100067.

[141] C. Ferreira Torres, A.K. Iannillo, A. Gervais, et al., The eye of Horus: spotting and analyzing attacks on Ethereum smart contracts, in: N. Borisov, C. Diaz (Eds.), Financial Cryptography and Data Security, Springer, Cham, 2021, pp. 33–52, https://doi.org/10.1007/978-3-662-64322-8_2.

[142] Z. Zheng, N. Zhang, J. Su, et al., Turn the rudder: a beacon of reentrancy detection for smart contracts on Ethereum, in: Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, 2023, pp. 295–306, https://doi.org/10.1109/ICSE48619.2023.00036.

[143] W. Li, J. Bu, X. Li, et al., A survey of DeFi security: challenges and opportunities, J. King Saud Univ., Comput. Inf. Sci. 34 (2022) 10378–10404, https://doi.org/10.1016/j.jksuci.2022.10.028.

[144] E. Albert, S. Grossman, N. Rinetzky, et al., Relaxed effective callback freedom: a parametric correctness condition for sequential modules with callbacks, IEEE Trans. Dependable Secure Comput. 20 (2022) 2256–2273, https://doi.org/10.1109/TDSC.2022.3178836.

[145] Inspexco, Value DeFi's invalid share calculation exploit: in-depth analysis, https://inspexco.medium.com/value-defis-invalid-share-calculation-exploit-in-depth-analysis-1c8f97c1416e, 2021. (Accessed 12 July 2024).

[146] Nipunp, 5/8/21 Rari Capital exploit timeline analysis, https://nipunp.medium.com/5-8-21-rari-capital-exploit-timeline-analysis-8beda31cbc1a, 2021. (Accessed 12 July 2024).

[147] I. Ballesteros, C. Benac-Earle, L.E.B. de Barrio, et al., Automatic generation of attacker contracts in solidity, in: Z. Dargaye, C. Schneidewind (Eds.), 4th International Workshop on Formal Methods for Blockchains (FMBC 2022), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, 2022, pp. 3:1–3:14, https://doi.org/10.4230/OASIcs.FMBC.2022.3.

[148] T. Defiant, Capital allocation: the next growth frontier, https://thedefiant.io/news/research-and-opinion/capital-allocation-the-next-growth-frontier, 2023. (Accessed 12 July 2024).

[149] Polydex, PLX locker smart contract incident post-mortem, https://polydex.medium.com/plx-locker-smart-contract-incident-post-mortem-75342124a3e8, 2023. (Accessed 12 July 2024).

[150] SanshuNFT, Woofdate 2.2.0: Keanu compensation, MFUND rebase update, https://sanshunft.medium.com/woofdate-2-2-0-keanu-compensation-mfund-rebase-update-bcac09707e19, 2023. (Accessed 12 July 2024).

[151] K.B. Lab, Comprehensive analysis of XSurge attacks, https://medium.com/@Knownsec_Blockchain_Lab/knownsec-blockchain-lab-comprehensive-analysis-of-xsurge-attacks-c83d238fbc55, 2021. (Accessed 12 July 2024).

[152] S. Yang, J. Chen, M. Huang, et al., Uncover the premeditated attacks: detecting exploitable reentrancy vulnerabilities by identifying attacker contracts, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, ACM, 2024, pp. 1–12, https://doi.org/10.1145/3597503.3639153.

[153] Siren, Siren incident report, https://web.archive.org/web/20221215032356/https://medium.com/siren/siren-incident-report-264e57f16d7, 2022. (Accessed 14 July 2024).

[154] Alan, ExploitedToadz: a technical deepdive, https://medium.com/@ItsCuzzo/exploitedtoadz-a-technical-deepdive-9ceabf46d0ce, 2024. (Accessed 14 July 2024).

[155] G. Ramakrishnan, M. Rehan, G. Sujatha, Solidity vulnerability scanner, in: Proceedings of the 2022 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI), IEEE, 2022, pp. 1–5, https://doi.org/10.1109/ICDSAAI55433.2022.10028877.

[156] B. Team, When SafeMint becomes unsafe: lessons from the HypeBears security incident, https://blocksecteam.medium.com/when-safemint-becomes-unsafe-lessons-from-the-hypebears-security-incident-2965209bda2a, 2024. (Accessed 14 July 2024).

[157] H. Chen, X. Luo, L. Shi, et al., Security challenges and defense approaches for blockchain-based services from a full-stack architecture perspective, Blockchain Res. Appl. 4 (2023) 100135, https://doi.org/10.1016/j.bcra.2023.100135.

[158] R. Xi, K. Pattabiraman, A large-scale empirical study of low-level function use in Ethereum smart contracts and automated replacement, Softw. Pract. Exp. 53 (2023) 631–664, https://doi.org/10.1002/spe.3163.

[159] SlowMist, Another day, another reentrancy attack, https://slowmist.medium.com/another-day-another-reentrancy-attack-5cde10bbb2b4, 2024. (Accessed 14 July 2024).

[160] Y. Gai, L. Zhou, K. Qin, Blockchain large language models, arXiv preprint, arXiv:2304.12749, 2023.

[161] Rekt, Voltage finance REKT, https://rekt.news/voltage-finance-rekt/, 2024. (Accessed 14 July 2024).

[162] BlockSecTeam, Title of the webpage, https://x.com/BlockSecTeam/status/1519249933832171520, 2024. (Accessed 14 July 2024).

[163] I.M. Ali, M.M. Abdallah, On off-chaining smart contract runtime protection: a queuing model approach, IEEE Trans. Parallel Distrib. Syst. 35 (2024) 1345–1359, https://doi.org/10.1109/TPDS.2024.3389153.

[164] Bistroo, Post-incident review: BIST single asset staking binance smart chain security breach. Medium, https://bistroo.medium.com/post-incident-review-bist-single-asset-staking-binancesmartchain-security-breach-5194590605f, 2024. (Accessed 14 July 2024).

[165] OwnChain, Official announcement: exploit in OWN staking contract, https://twitter.com/ownlyio/status/1524362090940895234, 2022. (Accessed 14 July 2024).

[166] K. Zhu, X. Wang, Z. Chen, et al., Evaluating Ethereum reentrancy detection tools via mutation testing, in: Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 545–555, https://doi.org/10.1109/ISSRE59848.2023.00067.

[167] QuillAudits, Decoding a $830,000 exploit, https://medium.com/quillhash/decoding-a-830-000-exploit-quillaudits-c70d1ecfd562, 2022. (Accessed 14 July 2024).

[168] PeckShield, PeckShield tweet, https://x.com/peckshield/status/1575890733373849601, 2023. (Accessed 16 July 2024).

[169] QuillAudits, Decoding $220k read-only reentrancy exploit. Medium, https://quillaudits.medium.com/decoding-220k-read-only-reentrancy-exploit-quillaudits-30871d728ad5, 2024. (Accessed 16 July 2024).

[170] G. Wei, D. Xie, W. Zhang, et al., Consolidating smart contracts with behavioral contracts, Proc. ACM Program. Lang. 8 (2024) 965–989, https://doi.org/10.1145/3656416.

[171] C.C. Tsai, C.C. Lin, S.W. Liao, Unveiling vulnerabilities in DAO: a comprehensive security analysis and protective framework, in: Proceedings of the 2023 IEEE International Conference on Blockchain (Blockchain), IEEE, 2023, pp. 151–158, https://doi.org/10.1109/Blockchain60715.2023.00034.

[172] PeckShieldAlert, PeckShieldAlert on X, https://x.com/PeckShieldAlert/status/1606276020276891650, 2022. (Accessed 17 July 2024).

[173] BlockSecTeam, BlockSecTeam on X, https://x.com/BlockSecTeam/status/1608372475225866240, 2022. (Accessed 17 July 2024).

[174] AnciliaInc, AnciliaInc on X, https://x.com/AnciliaInc/status/1614705804468424704, 2023. (Accessed 17 July 2024).

[175] PeckShield, PeckShield on X, https://x.com/peckshield/status/1621337925228306433, 2023. (Accessed 17 July 2024).

[176] N. Mutual, How was dynamic finance exploited?, https://web.archive.org/web/20240222215530/https://neptunemutual.com/blog/how-was-dynamic-finance-exploited/, 2024. (Accessed 14 July 2024).

[177] QuillAudits, Decoding sentiment protocol's $1 million exploit. Medium, https://quillaudits.medium.com/decoding-sentiment-protocols-1-million-exploit-quillaudits-f36bee77d376, 2024. (Accessed 14 July 2024).

[178] B. Phalcon, Phalcon XYZ tweet, https://x.com/Phalcon_xyz/status/1645742620897955842, 2024. (Accessed 14 July 2024).

[179] A. Finance, Post mortem, https://arcadiafinance.medium.com/post-mortem-72e9d24a79b0, 2024. (Accessed 14 July 2024).

[180] C. Finance, Post-mortem: ETH and crvUSD omnipool exploits, https://medium.com/@ConicFinance/post-mortem-eth-and-crvusd-omnipool-exploits-c9c7fa213a3d, 2024. (Accessed 14 July 2024).

[181] Itez, Curve's $50 million vulnerability, https://itez.com/en/blog/hype/curves-50-million-vulnerability, 2023. (Accessed 14 July 2024).

[182] N. Mutual, How was the earning farm exploited?, https://medium.com/neptune-mutual/how-was-the-earning-farm-exploited-9802135793ab, 2023. (Accessed 14 July 2024).

[183] A. Inc, Tweet by Ancilia Inc., https://x.com/AnciliaInc/status/1709352941541630049, 2023. (Accessed 14 July 2024).

[184] SlowMist, A deep dive into the stars arena attack, https://slowmist.medium.com/a-deep-dive-into-the-stars-arena-attack-765649d7fc6a, 2023. (Accessed 14 July 2024).

[185] MetaSec, Tweet by MetaSec, https://x.com/MetaSec_xyz/status/1717854153324781709, 2023. (Accessed 14 July 2024).

[186] SlowMist, SlowMist hacked - ETH ecosystem page 3, https://hacked.slowmist.io/?c=ETH&page=3, 2024. (Accessed 14 July 2024).

[187] 0xCygaar, Tweet about XYZ topic, https://x.com/0xCygaar/status/1736056050816876626, 2024. (Accessed 13 July 2024).

[188] MetaSec, Tweet on XYZ topic, https://x.com/MetaSec_xyz/status/1736428284756607386, 2024. (Accessed 13 July 2024).