Computer Modeling in
Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2024.046758

REVIEW

A Review of Deep Learning-Based Vulnerability Detection Tools for Ethernet Smart Contracts

Huaiguang Wu, Yibo Peng, Yaqiong He* and Jinlin Fan

School of Computer Science and Technology, Zhengzhou University of Light Industry, Zhengzhou, 450001, China

*Corresponding Author: Yaqiong He. Email: hyqiong9031@163.com

Received: 13 October 2023 Accepted: 08 January 2024 Published: 16 April 2024

ABSTRACT

In recent years, the number of smart contracts deployed on blockchain has exploded. However, the issue of vulnerability has caused incalculable losses. Due to the irreversible and immutability of smart contracts, vulnerability detection has become particularly important. With the popular use of neural network model, there has been a growing utilization of deep learning-based methods and tools for the identification of vulnerabilities within smart contracts. This paper commences by providing a succinct overview of prevalent categories of vulnerabilities found in smart contracts. Subsequently, it categorizes and presents an overview of contemporary deep learning-based tools developed for smart contract detection. These tools are categorized based on their open-source status, the data format and the type of feature extraction they employ. Then we conduct a comprehensive comparative analysis of these tools, selecting representative tools for experimental validation and comparing them with traditional tools in terms of detection coverage and accuracy. Finally, Based on the insights gained from the experimental results and the current state of research in the field of smart contract vulnerability detection tools, we suppose to provide a reference standard for developers of contract vulnerability detection tools. Meanwhile, forward-looking research directions are also proposed for deep learning-based smart contract vulnerability detection.

KEYWORDS

Smart contract; vulnerability detection; deep learning

1 Introduction

In 2008, Nakamoto et al. [1] originally introduced the concept of blockchain in the seminal Bitcoin whitepaper, sparking significant and sustained interest in this technology. Over recent years, virtual currencies and blockchain technologies have emerged as powerful catalysts of social change and financial value creation. Blockchain technology [2–5] has witnessed substantial advancements, finding applications in diverse domains such as healthcare [6–10], Internet of Things (IoT) services [11–15], supply chain management [16–20], copyright protection [21–25], and energy internet [26–30]. Of noteworthy mention is Ether [31], the presently most prevalent blockchain platform. The Ethereum platform boasts Turing completeness [32], allowing developers to craft smart contracts capable of implementing a wide array of intricate logic and business rules. Fig. 1 illustrates the deployment and invocation of Ethereum smart contracts.

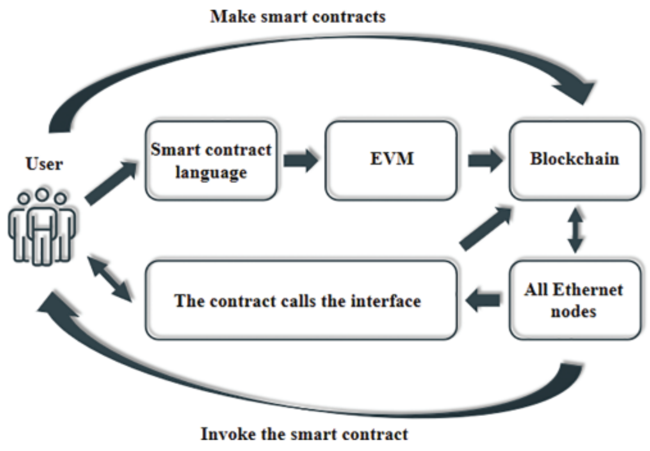


Figure 1: Deployment and invocation of Ethernet smart contracts

In 1994, Szabo [33], an American computer scientist, conceived the notion of smart contracts, drawing inspiration from vending machines. However, due to the constraints of technical development during that era, the concept of smart contracts did not fully realize its inherent value. In recent years, driven by the evolution and maturation of blockchain technology, smart contracts have emerged as a compelling opportunity with promising applications. A smart contract represents a computer program operating on a distributed ledger, characterized by predefined rules, states, and conditional responses [34]. It can be likened to the "legal provisions" or "commercial consensus" governing interactions within a network. Smart contracts possess the capacity to encapsulate, validate, and execute complex behaviors among distributed nodes, enabling them to facilitate functions such as information exchange, value transfer, and asset management [35].

Currently, blockchain platforms such as Ethereum host tens of thousands of deployed smart contracts, with this number continuously witnessing explosive growth. The security of smart contracts has taken on paramount importance, given the inherent characteristics of blockchain technology, including irreversibility and immutability. Recent years have witnessed several high-profile security breaches and attacks targeting blockchain smart contracts, thereby highlighting substantial concerns regarding their security. For instance, the June 2016 incident known as The Decentralized Autonomous Organization (DAO) [36–38] resulted in hackers exploiting a reentrant vulnerability to stealing with $ 60 million worth of Ether. In 2017, the Delegatecall vulnerability within Parity's multi-signature wallet contract [39–41] led to the freezing of nearly $ 300 million in Ether. Moreover, the Beauty Chain (BEC) contract in 2018 [42–44] exhibited an integer overflow vulnerability that attackers exploited to create unlimited copies of BEC tokens, precipitating a collapse in the token's value. Instances like the King of the Ether Throne attack [45–47] have also employed denial-of-service vulnerabilities to perpetrate token scams. In response to the security challenges stemming from smart contract vulnerabilities, a plethora of methods and tools for detecting these vulnerabilities have undergone extensive research.

In the past, smart contract vulnerability detection primarily relied on traditional methods. However, with the increasing speed of smart contract deployments, traditional methods have demonstrated lower efficiency and accuracy in detecting contract vulnerabilities. Consequently, researchers have gradually turned to the application of deep learning methods for smart contract vulnerability detection. Compared to traditional methods, deep learning-based vulnerability detection methods offer numerous advantages. They can simultaneously detect various types of vulnerabilities without relying on expert rules. Notably, since 2020, the utilization of deep learning network models has gained prominence as a focal point in smart contract vulnerability detection research. This paper focuses on elaborating and analyzing deep learning-based smart contract vulnerability detection methods and tools.

1.1 Research Status

In recent years, researchers have conducted comprehensive reviews of smart contract vulnerability analysis and detection tools. Currently, there are two main types of reviews, one is a detailed research on a certain category of smart contract vulnerability detection methods and tools, and the other is a comprehensive and systematic generalization for all categories of smart contract detection methods and tool. In the following paper, we will analyze the research status according to the above two types of reviews with different focuses. Table 1 can directly show the current status of relevant studies and the comparative analysis between them.

Table 1: Comparative analysis of relevant review research status

Literature	Research types	Number of tools	Number of references
[48]	Formal verification	25	90
[49]	Semantic analysis and formal verification	53	71
[50]	Formal verification	34	192
[51]	Comprehensive analysis	27	59
[52]	Comprehensive analysis	39	270
[53]	Comprehensive analysis	12	188
[54]	Comprehensive analysis	3	15
[55]	Comprehensive analysis	86	143

1.1.1 Certain-Category-Methods Analysis

Almakhour et al. [48] examined 25 such tools, classifying them into two categories: validation tools, which aim to ensure correctness, and vulnerability analysis tools, designed for security assurance through vulnerability detection. They focused on formal verification methods, which only verify simple smart contracts and do not provide a comprehensive overview of verification methods for complex smart contracts. Similarly, Liu et al. [49] scrutinized 53 papers related to smart contract vulnerability detection, and divided them into 18 distinct tool categories, including semantic analysis and formal verification. Focusing on the analysis of methods for formal verification of smart contracts, they summarize the current research status of smart contract security and correctness, and point out the future research direction. In another research, Tolmach et al. [50] provided insights into contract vulnerability detection and verification tools for formal modeling and verification techniques. The paper analyzed and classified formal modeling approaches for smart contract security research techniques, revealed the limitations of formal verification of smart contracts at that time and proposes future research directions.

1.1.2 Comprehensive Analysis

Angelo et al. [51] conducted a survey of 27 smart contract vulnerability analysis tools, categorizing them based on attributes such as open-source availability, developmental status, operational methodology, and detection approach. The feasibility tools among them were also compared with experimental analysis to more comprehensively analyze different kinds of detection tools. Hu et al. [52] conducted a comprehensive analysis of 39 contract vulnerability detection tools, with a focus on source code accessibility, detection methodologies, and user-friendliness. Finally, current challenges are identified and future perspectives are provided. Furthermore, Ante [53] conducted a survey of various smart contracts, employing keyword citation statistics and highlighting common smart contract vulnerability analysis tools such as Oyente and SmarkCheck. They also conducted research and analyzed related issues in six areas where smart contracts are used. He et al. [54] delved into the paper related to smart contract vulnerability defense mechanisms and security auditing methods, providing insights into the performance comparisons and vulnerability detection characteristics of tools like Oyente, Mythrill, and Porosity. Kushwaha et al. [55] conducted a comprehensive survey of 155 papers and analyzed 86 security detection tools for Ethereum smart contracts, irrespective of tool type or analysis method. The paper concludes with challenges faced and recommendations for future development. It is important to note that the aforementioned studies primarily offer detailed categorizations of conventional vulnerability detection methods. There remains a significant gap in the systematic examination and research of deep learning-based smart contract vulnerability detection methods.

1.2 Motivation

The emergence of blockchain technology has been accompanied by a concerning frequency of security incidents stemming from smart contract vulnerabilities, resulting in substantial economic losses. Consequently, the security of smart contracts has emerged as a focal point in current research efforts. Addressing these issues is imperative to enhance the security and dependability of smart contracts, ultimately driving the advancement of blockchain technology. Existing review articles predominantly encompass discussions of various categories of smart contract vulnerability detection tools. Furthermore, the majority of articles pertaining to smart contract detection tools tend to concentrate on specific traditional tools, often overlooking the intricate landscape of contract vulnerability detection tools based on deep learning methodologies. Therefore, the primary objectives of this paper are as follows: to systematically present deep learning-based smart contract vulnerability detection tools, to perform a comprehensive analysis of their unique features and advantages, and to suggest recommendations for future research efforts in the domain of deep learning-based smart contract vulnerability detection methods. In order to achieve these goals, we further researched the current research status of review articles in this field and studied the innovation and novelty of their articles. At the same time, we have curated relevant literature from platforms such as Google Scholar [56] and Web of Science [57]. We have organized and presented the developmental trends of both traditional detection tools and deep learning-based detection tools that have been introduced in recent years, as depicted in Fig. 2.

1.3 Research Questions

A multitude of smart contracts have been deployed on various blockchain platforms, each characterized by its unique attributes. Ethereum, as the most widely adopted platform for smart contract development, garners significant attention. Consequently, the primary focus of this paper is confined to the Ethereum platform. We adhere to the systematic review methodology proposed by Kitchenham et al. [58] and Peterson et al. [59] to define the following research questions:

Question 1: What are the deep learning based smart contract vulnerability detection tools?

Question 2: Which open source deep learning based smart contract vulnerability detection tools are available?

Question 3: What are the common vulnerabilities that can be detected by deep learning-based smart contract vulnerability detection tools?

Question 4: How effective are deep learning-based smart contract vulnerability detection tools compared to traditional tools?

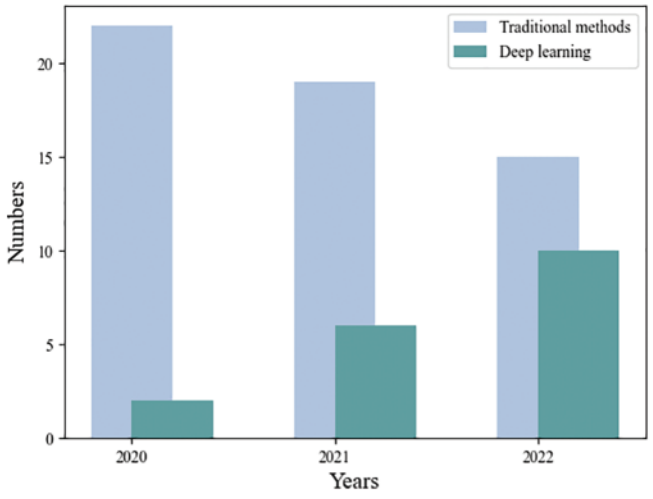


Figure 2: Numbers of traditional and deep learning tools developed in recent years

1.4 Contributions

In light of the research considerations outlined above, our endeavor to systematically survey deep learning-based tools for smart contract vulnerability detection commenced by aggregating pertinent literature from reputable academic platforms, including Google Scholar [56], ACM [60], IEEE [61], Springer [62], Wiley, Web of Science [57], and arXiv [63]. Our emphasis was primarily directed toward articles elucidating the application of deep learning techniques in the domain of smart contract vulnerability detection, with a specific focus on publications beyond the year 2020. Through this process, we identified a total of 20 deep learning-based tools tailored for smart contract vulnerability detection. The contributions of this paper are mainly as follows:

(1) We conducted an investigation into 20 Ethereum smart contract vulnerability detection tools based on deep learning methods in recent years;

(2) We classify and analyze 20 deep learning methods based on their open-source nature, data input format, and feature extraction approach;

(3) We compared the detection performance and disparities between traditional tools and deep learning tools;

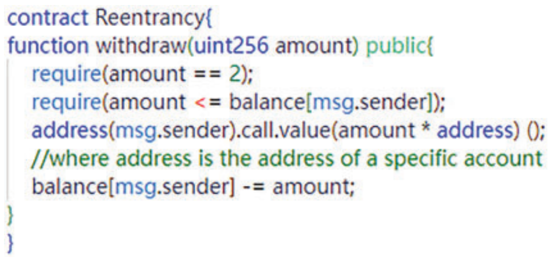
(4) We propose future development recommendations for research on smart contract vulnerability detection methods.

2 Ether Smart Contract Security Vulnerability

As the largest open-source blockchain platform presently available, Ethereum stands out not only as the platform with the highest deployment of smart contracts but also as the platform exhibiting the highest frequency and broadest spectrum of smart contract vulnerabilities. With the rapid proliferation of smart contracts, an increasing number of vulnerabilities have come into the purview of detection experts. In this section, we will enumerate common types of vulnerabilities found in Ethereum smart contracts [64–67].

2.1 Reentrancy

In June 2016, the Ethereum smart contract platform witnessed a severe reentrant vulnerability within The DAO [68–70]. Exploiting a loophole in an external function call within the contract, attackers successfully pilfered approximately $ 60 million worth of digital assets. This incident precipitated the implementation of a hard fork in the Ethereum network [71]. The reentrant vulnerability is a prevalent security flaw found in smart contracts, primarily stemming from issues related to the invocation of other contracts within the smart contract code. When one contract calls another and the called contract can recursively invoke the calling contract, a reentrant vulnerability may ensue, enabling attackers to exploit this mechanism to execute multiple contract calls and inflict substantial losses. Fig. 3 illustrates the specific source code responsible for the reentrant vulnerability.



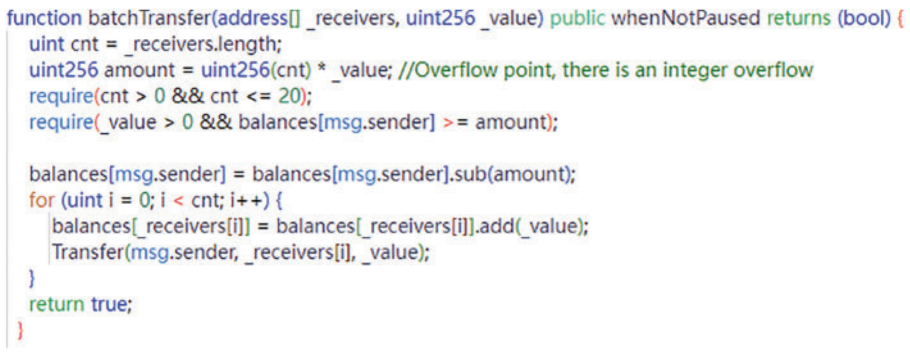
```
contract Reentrancy{
function withdraw(uint256 amount) public{
  require(amount == 2);
  require(amount <= balance[msg.sender]);
  address(msg.sender).call.value(amount * address) ();
  //where address is the address of a specific account
  balance[msg.sender] -= amount;
}
}
```

Figure 3: Reentrancy

2.2 Integer Overflow

An integer overflow vulnerability [72–74] occurs when the value of an integer-type variable in a smart contract exceeds its maximum representable range, resulting in data overflow or underflow. Integer types in smart contracts are typically bounded, for instance, in Solidity, uint256 represents a 256-bit unsigned integer with a maximum value of 2 to the power of 256 minus 1. If a uint256 variable is used in a smart contract without proper range checks, it may lead to data overflow or underflow situations. Unlike other computer programs, the losses caused by integer overflow in smart contracts are irreparable. Fig. 4 below shows the source code diagram of the integer overflow vulnerability. In 2018, the Beauty Chain (BEC) contract [45] suffered from an integer overflow vulnerability, which attackers exploited to duplicate BEC tokens without restrictions, causing the price of BEC tokens to plummet to zero.

To mitigate integer overflow vulnerabilities, auditors can implement range checks on integer-type variables using the require() function. Ethereum developers have also provided the SafeMath library, which includes functions that automatically detect integer overflow errors in Solidity code.



```
function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
  uint cnt = _receivers.length;
  uint256 amount = uint256(cnt) * _value; //Overflow point, there is an integer overflow
  require(cnt > 0 && cnt <= 20);
  require(_value > 0 && balances[msg.sender] >= amount);

  balances[msg.sender] = balances[msg.sender].sub(amount);
  for (uint i = 0; i < cnt; i++) {
    balances[_receivers[i]] = balances[_receivers[i]].add(_value);
    Transfer(msg.sender, _receivers[i], _value);
  }
  return true;
}
```

Figure 4: Integer overflow

2.3 *Access Control*

Access control vulnerabilities [75–77] represent design flaws within the authentication conditions of specific contract functions. These flaws can facilitate an attacker's ability to circumvent access control restrictions, providing unauthorized access to critical resources or functions. Furthermore, they may grant the attacker excessive access privileges, extending beyond their legitimate requirements, and consequently, enabling access to restricted data or functions. Such vulnerabilities typically emerge due to oversights by contract authors who either neglect to implement essential access control measures or rely on insecure default configurations.

2.4 *Denial of Service (DOS)*

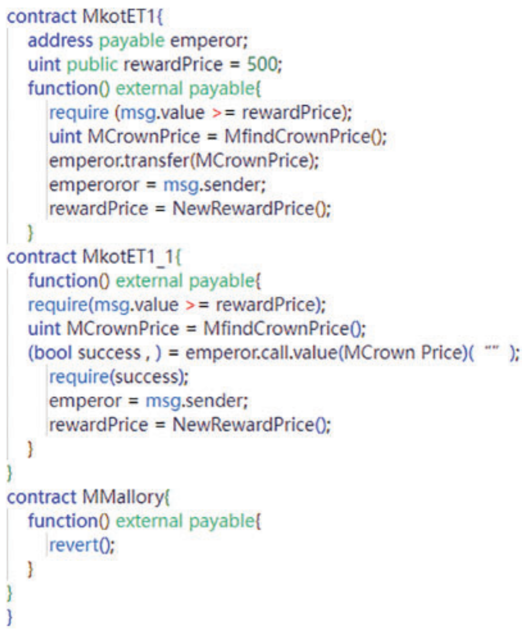
A denial-of-service vulnerability [78–81] occurs when a malicious user transmits specific transaction data that hinders the successful execution of a smart contract's intended function. This not only consumes resources within the Ethereum network but also results in the unavailability of the smart contract service. In some cases, it can even lead to a permanent impairment of the contract's usability. Denial-of-service vulnerabilities represent a prominent security risk for blockchain and smart contract applications. Developers must implement appropriate measures to mitigate this risk, such as designing contract conditions and rules judiciously and imposing restrictions on requests from potentially malicious users. An illustrative example is the King of the Ether Throne [45], where an attacker employed a denial-of-service tactic to prevent unsuccessful participants from recovering the cryptocurrency they had deposited as security. Fig. 5 shows the source code of denial of service.

2.5 *Freezing Ether*

Due to the irreversible nature of the blockchain, if an ethereum smart contract developer sets up the contract development only with the ability to accept ethereum and does not set up any feature that allows for the transfer of ethereum, the ethereum in this contract, as well as the ethereum transferred to this contract from other contracts, is permanently sequestered in this contract, and an asset freezing vulnerability occurs [82–84]. Fig. 6 shows a concrete source code example of the Ether freeze vulnerability.

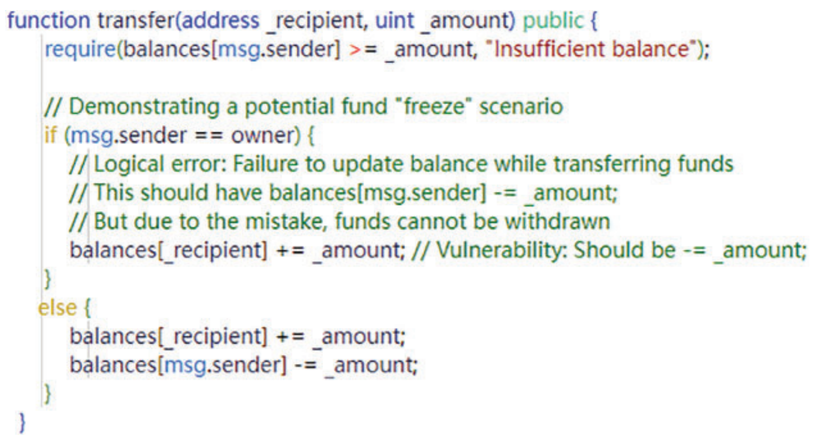
2.6 *Unchecked Call*

Unchecked call vulnerabilities [85–87] are usually caused by irregular code writing and are also related to special exception passing mechanisms in smart contracts. For instance, low-level function calls such as send, call, and delegatecall do not trigger a transaction rollback operation in case of failure; instead, they merely signal an exception through the return value. Therefore, during contract code development, developers must meticulously examine the return values of these low-level function calls to ascertain their accuracy. Given that a low-level call essentially constitutes a "message to send" to another contract account's address, it is justifiable for the Virtual Machine (VM) to incorporate this exception-handling mechanism. This aspect is not explicitly communicated in high-level language design, making it susceptible to inadvertent introduction of unchecked call vulnerabilities by developers lacking this crucial context. Fig. 7 illustrates an example of unchecked call code.



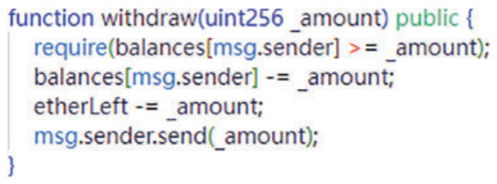
```
contract MkotET1{
  address payable emperor;
  uint public rewardPrice = 500;
  function() external payable{
    require (msg.value >= rewardPrice);
    uint MCrownPrice = MfindCrownPrice();
    emperor.transfer(MCrownPrice);
    emperoror = msg.sender;
    rewardPrice = NewRewardPrice();
  }
contract MkotET1_1{
  function() external payable{
  require(msg.value >= rewardPrice);
  uint MCrownPrice = MfindCrownPrice();
  (bool success , ) = emperor.call.value(MCrown Price)( "" );
    require(success);
    emperor = msg.sender;
    rewardPrice = NewRewardPrice();
  }
}
contract MMallory{
  function() external payable{
    revert();
  }
}
}
```

Figure 5: Denial of service (DOS)



```
function transfer(address _recipient, uint _amount) public {
    require(balances[msg.sender] >= _amount, "Insufficient balance");

    // Demonstrating a potential fund "freeze" scenario
    if (msg.sender == owner) {
      // Logical error: Failure to update balance while transferring funds
      // This should have balances[msg.sender] -= _amount;
      // But due to the mistake, funds cannot be withdrawn
      balances[_recipient] += _amount; // Vulnerability: Should be -= _amount;
    }
    else {
      balances[_recipient] += _amount;
      balances[msg.sender] -= _amount;
    }
 }
```

Figure 6: Freezing ether

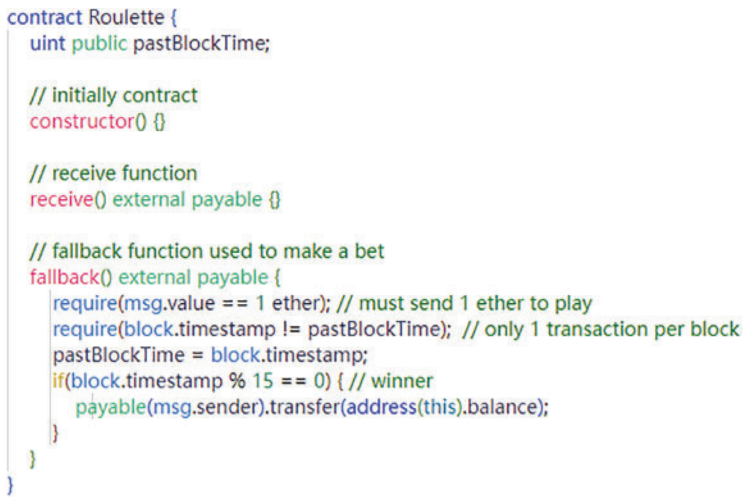


```
function withdraw(uint256 _amount) public {
    require(balances[msg.sender] >= _amount);
    balances[msg.sender] -= _amount;
    etherLeft -= _amount;
    msg.sender.send(_amount);
}
```

Figure 7: Unchecked call

2.7 Timestamp Dependency

In smart contracts, timestamps confirmed by miners are utilized as conditional criteria or to implement time constraints. Timestamps may initially appear to exhibit randomness; however, miners can manipulate them within a specified range of values, thereby introducing a concept known as timestamp dependence [88–90]. Although timestamps are considered a critical decision-making factor within contracts, ordinary attackers are typically unable to subvert the influence of timestamps. Nevertheless, miners themselves possess the capability to readily bypass the constraints imposed by timestamp-based designs in contracts by crafting malicious timestamps that fall within the permissible value range. Fig. 8 illustrates a case where a timestamp dependency vulnerability could arise in a smart contract, in this way, they could win the ethereum block rewards locked in this contract. Since only one person is allowed to place a bet per block, this leaves the contract vulnerable to attack.

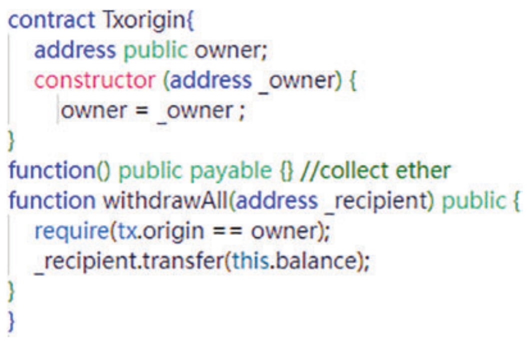


```
contract Roulette {
    uint public pastBlockTime;

    // initially contract
    constructor() {}

    // receive function
    receive() external payable {}

    // fallback function used to make a bet
    fallback() external payable {
        require(msg.value == 1 ether); // must send 1 ether to play
        require(block.timestamp != pastBlockTime);  // only 1 transaction per block
        pastBlockTime = block.timestamp;
        if(block.timestamp % 15 == 0) { // winner
            payable(msg.sender).transfer(address(this).balance);
        }
    }
}
```

Figure 8: Timestamp dependency

2.8 Tx.Origin

"Tx.Origin" [91–93] is a global variable in Ethereum smart contract programming used to store the sender's address of a transaction. It can trace the entire call stack and return the address of the initial contract that initiated the call. However, using the "Tx.Origin" variable can lead to a security vulnerability known as "Tx.Origin attack." For instance, an attacker can exploit this vulnerability by invoking the victim's withdrawal function through a fallback function, enticing the victim contract to transfer Ether to the attacker's contract. Due to the condition "tx.origin==owner," any exceptions may go undetected, leading to the transfer of all Ether from the victim contract to the attacker's contract account. An example of code demonstrating the Tx.Origin vulnerability is shown in Fig. 9.



```
contract Txorigin{
    address public owner;
    constructor (address _owner) {
        owner = _owner ;
}
function() public payable {} //collect ether
function withdrawAll(address _recipient) public {
    require(tx.origin == owner);
    _recipient.transfer(this.balance);
}
}
```

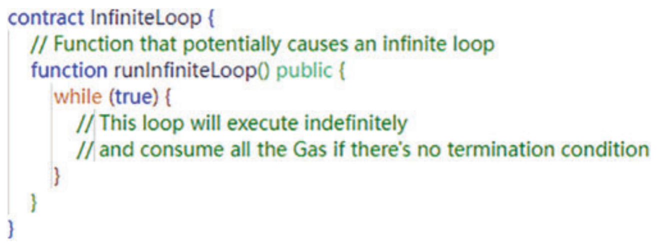
Figure 9: Tx.Origin

2.9 Delegate Call

The vulnerability under discussion [94–96] typically arises in smart contracts employing Delegate Call functions. Delegate Call is a distinctive form of function call that enables the execution of an external contract's code within the current contract's context. This approach is often adopted to economize on Gas fees and decrease the storage space occupied by the contract. Nonetheless, if the called contract maliciously employs EVM opcodes to modify the state variables of the calling contract, it may result in a security vulnerability. A Delegate call vulnerability [42] in Parity's multi-signature wallet contract in 2017 led to the freezing of nearly $300 million in Ether.

2.10 Infinite Loop

Infinite loops [97–99] represent one of the prevailing vulnerabilities within smart contracts. This vulnerability materializes when the code for a contract function includes iterations or loops lacking a termination condition or featuring an unreachable termination condition. Specifically, the contract's 'for' and 'while' loops may perpetually iterate, unless they have been meticulously developed with precision to ensure accurate termination. Fig. 10 shows an example of the infinite loop vulnerability where the function 'runInfiniteLoop' contains a while loop that never ends. If a contract is called to execute this function, it will loop indefinitely and consume all the Gas, causing the transaction or operation to fail. This situation may prevent other contracts from calling or cause the contract to fail permanently.



```
contract InfiniteLoop {
    // Function that potentially causes an infinite loop
    function runInfiniteLoop() public {
        while (true) {
            // This loop will execute indefinitely
            // and consume all the Gas if there's no termination condition
        }
    }
}
```

Figure 10: Infinite loop

2.11 Transaction Order Dependency (TOD)

In a blockchain network, each transaction undergoes a certain processing time before miners acknowledge it and incorporate it into a block, often resulting in a different transaction order within the block compared to the order in which they were initially generated. Malicious actors exploit the reliance on transaction order in smart contracts on the blockchain, enabling them to monitor the transaction progress of such contracts and preemptively deploy their own contracts prior to the targeted contract's execution, thereby manipulating the transaction flow for personal gain. This practice is commonly referred to as Transaction Order Dependence (TOD) [49,100]. For instance, consider an attacker who initiates a bounty contract that permits other users to submit solutions to challenging questions and receive substantial rewards from the contract. After launching the bounty contract, the attacker vigilantly observes the blockchain network. As soon as they notice that a correct answer has been submitted but not yet confirmed, the attacker initiates a transaction that significantly reduces the prize amount while offering a substantial miner's fee. There exists a high likelihood that their transaction will be prioritized by miners, due to the enticing fees, thus enabling the attacker to obtain the correct answer at minimal cost and reap the substantial rewards originally designated for others.

2.12 Call-Stack Depth

Whenever a contract calls an external contract or after its own call, the contract will increase the call stack depth once. Within the Ethereum Virtual Machine (EVM), there exists a maximum call stack limit of 1024. If an attacker devises a sequence of nested call stack operations, there is a significant likelihood of triggering a call stack overflow. This condition poses a considerable risk to the contract, ultimately leading to a stack overflow scenario and giving rise to a call-stack depth vulnerability [101–103].

3 Traditional Tools

Initially, individuals heavily relied on their experience and knowledge of programming languages to manually examine contract code for vulnerabilities and defects. However, this approach not only proved to be inefficient but also suffered from issues such as a lack of expertise and a high degree of subjectivity. In light of these challenges, individuals have begun actively researching detection tools tailored for identifying contract vulnerabilities. As a result, traditional types of tools have emerged. Before deep learning is widely used, the detection of smart contract security problems mainly relies on traditional tools. Detection methods based on traditional tools primarily encompass symbolic execution [104–106], formal verification [107–109], fuzz testing [110,111], intermediate representations, and taint analysis, among others. Oyente [94], as one of the earliest smart contract detection tools, utilizes symbolic execution for contract vulnerability assessment. Tools such as Securify [112], Mythril [113], and TeEther [82] also rely on symbolic execution for vulnerability detection. On the other hand, the F$*$ framework [114], ZEUS [115], and Isabelle/HOL [116] fall under the category of formal verification-based contract analysis tools. ContractFuzzer [117] and Regurad [71] are classified as fuzz testing-based detection tools. Static analysis tools like Slither [118] and SmartCheck [119] are grounded in intermediate representations. Lastly, EasyFlow [120] is a dynamic taint analysis-based detection tool.

Many of the detection tools based on traditional methods mentioned above suffer from incomplete coverage of detection types and heavily rely on expert rules, resulting in relatively low detection efficiency and accuracy. Currently, Ethereum has witnessed the deployment of tens of thousands of smart contracts, with many more expected to be accelerated onto the platform in the future. As research into contract vulnerability detection deepens, triggering contract vulnerabilities has become increasingly complex. Attackers often leverage multiple types of vulnerabilities in tandem to craft attacks against smart contracts, thereby augmenting the difficulty of smart contract vulnerability detection.

4 Deep Learning Tools

As the difficulty of detecting smart contract vulnerabilities continues to escalate, researchers have begun to gradually utilize deep learning for smart contract vulnerability detection. In this section, we categorize deep learning-based smart contract vulnerability detection tools into different subclasses, subdividing them according to whether they are open source or not, the type of inputs to the data, and the way the features are extracted. We surveyed and summarized the relevant literature up to February 2023. Fig. 11 illustrates the annual evolution of deep learning-based smart contract detection tools, while Fig. 12 outlines the distribution of different tools within each subclass.

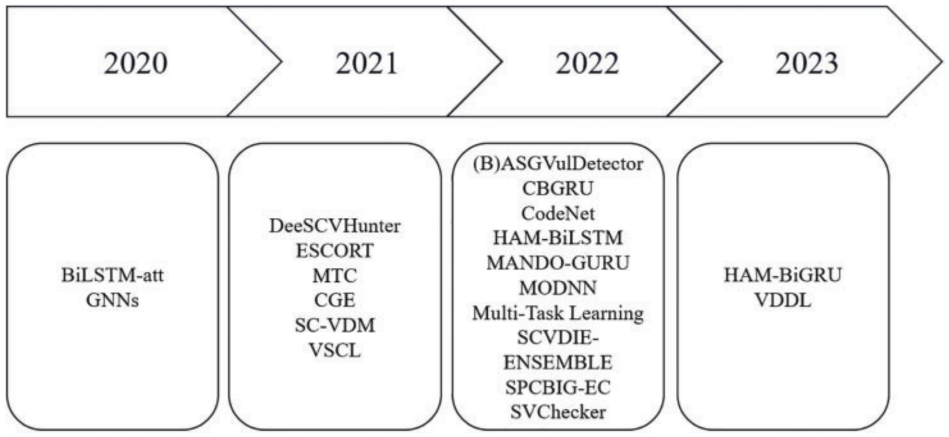


Figure 11: Annual evolution of deep learning based smart contract detection tools

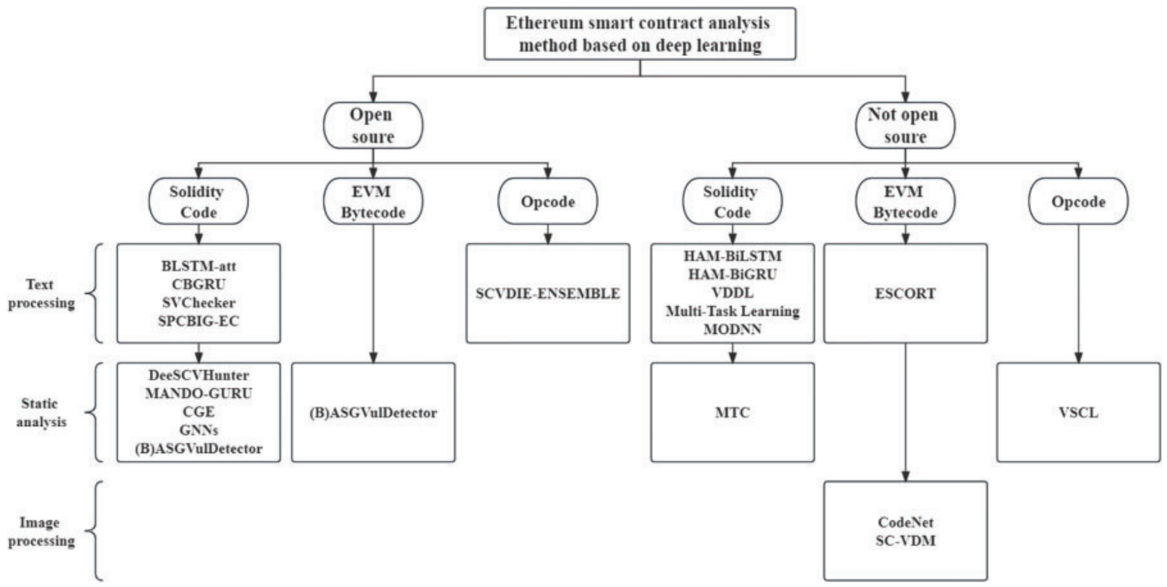


Figure 12: Classification of deep learning based smart contract detection tools

This paper describes such tools in detail from three perspectives: open source, data input type, and feature extraction form. Whether a tool is open source or not has a direct impact on whether such a tool can be quickly installed and used, and through the open source categorization criteria, researchers can easily find out fast and usable deep learning method tools for vulnerability detection. Typically, researchers compose smart contracts using Solidity source code. Prior to deployment, assessing these contracts necessitates scrutiny in the form of source code input. Smart contracts deployed on the Ethereum network exist in bytecode format, necessitating bytecode-based input for contract detection. Opcode serves as the fundamental operational unit within bytecode, comprising a set of Ethereum Virtual Machine (EVM) instructions. Detecting data input anomalies via opcode analysis significantly enhances the identification of functional and logical errors within EtherNetwork contracts. The different forms of feature extraction are mainly manifested in the form of Differences in deep learning methods. The neural network model within deep learning epitomizes a black-box model, wherein the model's underlying basis and rationale for detection remain obscured. Differential feature extraction techniques redirect the focus of model detection. Based on the specific vulnerability type within the targeted contract, adept selection of feature extraction methods becomes paramount. In the following, the deep learning-based smart contract vulnerability detection tools under different categories will be introduced in detail one by one.

4.1 Open Source

4.1.1 Solidity Source Code

BLSTM-att [121]: BLSTM-att is a vulnerability detection tool that takes the Solidity source code of smart contracts as input. This tool was released in 2020 and is among the earliest tools to propose the use of deep-learning neural network models for smart contract vulnerability detection. Its primary focus lies in the detection of reentrancy vulnerabilities, and its detection process can be outlined as follows: first, collect a large number of smart contract source codes on the official website of EtherNet; then process the source codes into contract fragments, and convert the contract fragments into word vectors matching the inputs of the neural network by using Word2vec; then input the word vectors into a sequence model for training and learning; finally, conduct experimental validation of the test set to conclude.

CBGRU [122]: CBGRU is a smart contract vulnerability detection tool based on a novel hybrid deep learning model, which was released in 2022. This tool effectively combines various word embedding methods (Word2Vec and Fasttext) with different deep learning techniques (LSTM, GRU, BiLSTM, BiGRU, CNN) to strategically extract features using distinct deep learning models, enabling comprehensive vulnerability detection. CBGRU is capable of detecting five different types of vulnerabilities.

SVChecker [123]: SVChecker, introduced in 2022, is a smart contract vulnerability detection system that comprises three pivotal modules. The initial module focuses on the transformation of source code into code snippet representations while eliminating code segments unrelated to vulnerabilities. The subsequent step involves utilizing Word2Vec for embedding word vectors within the extracted code snippets, with the subsequent completion of a classification training task facilitated by the deep learning model known as Transformer-Encoder. Following the training phase, an unknown source code detector is constructed to assess the dataset, culminating in the generation of detection results. Notably, SVChecker demonstrates proficiency in identifying six distinct types of vulnerabilities.

SPCBIG-EC [124]: SPCBIG-EC is a Sequential-Parallel Convolutional Bidirectional Gated Recurrent Unit (BiGRU) Ensemble Classifier model designed for smart contract vulnerability detection, introduced in 2022. SPCBIG-EC also introduces a Sequential-Parallel Convolution (SPCNN) method tailored for hybrid models. This approach allows for the extraction of features representing multivariate combinations from input sequences while preserving temporal and positional information. The detection process of this tool can be outlined as follows: Initially, contract datasets are collected from the Ethereum blockchain. Subsequently, feature vectors are extracted using SPCNN. These vectors are then input into the neural network model for training and detection. Following this, feature data is weighted using base classifiers and input into the ensemble classifier. Ultimately, the detection results are produced. SPCBIG-EC is capable of identifying five distinct types of contract vulnerabilities.

DeeSCVHunter [125]: DeeSCVHunter is an intelligent smart contract vulnerability detection framework that integrates multiple deep learning network models, introduced in 2021. Within DeeSCVHunter, various convolutional neural networks and recurrent neural networks are incorporated, enabling active learning for smart contract vulnerability detection without the need for domain experts or external knowledge sources. The framework also introduces the concept of Vulnerability Candidate Segments (VCS), which leverages data dependencies or control dependencies within the source code to enhance the efficiency of network models. This tool primarily focuses on detecting two types of vulnerabilities: reentrancy vulnerabilities and timestamp dependencies.

MANDO-GURU [126]: MANDO-GURU, introduced in 2022, is an intelligent smart contract vulnerability detection tool based on a heterogeneous graph attention neural network. It is designed to detect vulnerabilities in both coarse-grained contract-level and fine-grained line-level smart contracts. MANDO-GURU combines Control Flow Graphs (CFGs) and Solidity code Call Graphs (CGs). It employs a newly designed heterogeneous graph neural network to encode various types of nodes and their structural and potential semantic relationships within these graphs. Finally, it utilizes the encoded graphs and nodes to detect vulnerabilities within the contract.

CGE [127]: The tool is an approach to smart contract vulnerability detection that combines graph neural networks with expert knowledge and was released in 2021. CGE achieves this by transforming the source code's control flow and data flow semantics into a contract graph. Simultaneously, it includes a node elimination stage to normalize the graph, emphasizing key nodes within the contracted graph. Additionally, it introduces a Time-Information Propagation network (TMP) to extract graph features from the normalized graph. These graph features are then combined with expert knowledge to form the final detection system. CGE primarily focuses on detecting three types of vulnerabilities: reentrancy vulnerabilities, timestamp dependencies, and infinite loop vulnerabilities.

GNNs [97]: GNNs (Graph Neural Networks) were introduced in 2020 with a primary focus on representing smart contract source code as contract graphs, based on the data and control dependencies among program source code statements. In these graphs, nodes represent crucial function calls or variables, and edges denote their temporal execution paths. To automatically detect smart contract vulnerabilities, GNNs incorporate a novel Time-Message Propagation network (TMP) and a Degree-Revised Graph Convolutional Network (DR-GCN). Compared to existing methods, this approach models the fallback mechanisms of smart contracts, takes into account rich dependencies among program elements, and explores the potential of using novel graph neural networks for vulnerability detection. GNNs primarily detect three types of vulnerabilities: reentrancy, timestamp dependencies, and infinite loop vulnerabilities.

4.1.2 EVM Bytecode

(B)ASGVulDetector [128]: (B)ASGVulDetector comprises two statically analyzed methods for detecting smart contract vulnerabilities, targeting both source code and bytecode data formats. This tool was introduced in 2022. The approach introduces a novel intermediate representation termed the Abstract Semantic Graph (ASG), designed to facilitate the capture of both syntactic and semantic features within contracts. The (B)ASGVulDetector framework is structured around three key detection steps: firstly, the conversion of smart contracts into Abstract Semantic Graph (ASG) representations; secondly, the computation of vector representations for the contracts; and finally, the assessment of code pair similarity through GMN measurements. Notably, (B)ASGVulDetector demonstrates proficiency in detecting four distinct types of vulnerabilities.

4.1.3 Opcode

SCVDIE-ENSEMBLE [129]: The SCVDIE-ENSEMBLE is an intelligent contract vulnerability detection method based on ensemble learning, which was published in 2022. SCVDIE-ENSEMBLE leverages seven distinct neural networks for contract-level vulnerability screening using a dataset of contract vulnerabilities. The detection process can be summarized as follows: firstly, pre-training of the seven neural networks is conducted using an information graph (IG) composed of the source dataset; subsequently, these networks are integrated into the SCVDIE model; finally, the effectiveness of SCVDIE-ENSEMBLE is validated using a target dataset composed of information graphs. SCVDIE-ENSEMBLE can detect six types of vulnerabilities, including integer overflow, transaction order dependency, call stack overflow vulnerabilities, timestamp dependencies, reentrancy vulnerabilities, and delegate call vulnerabilities.

4.2 Not Open Source

4.2.1 Solidity Source Code

HAM-BiLSTM [130]: The HAM-BiLSTM is a model of a bidirectional Long Short-Term Memory (BiLSTM) network with a hierarchical attention mechanism, which was introduced in 2022. This approach operates at three levels during the input stage: word, sentence, and document levels, and incorporates attention mechanisms at each of these levels to enhance the accuracy of reentrancy vulnerability detection while minimizing false positives as much as possible. HAM-BiLSTM learns feature information from training samples and then produces classification detection results through a softmax function. Its primary focus is on the detection of reentrancy vulnerabilities.

HAM-BiGRU [131]: HAM-BiGRU is an approach to contract vulnerability detection utilizing deep learning network models and incorporating single and multi-head attention mechanisms, released in 2023. Vulnerability detection using HAM-BiGRU is divided into 4 main stages: fragment extraction of contract source code; word vector embedding with Word2Vec; feature learning through network modeling; and vulnerability detection. The tool can detect 5 types of contract vulnerabilities.

VDDL [132]: The VDDL model was released in 2023 and utilizes a multi-layer bidirectional Transformer architecture as its model structure. This architecture includes multi-head attention mechanisms and masking mechanisms. Specifically, the encoder-decoder layer employs a multi-head attention mechanism, while the masking mechanism is applied to randomly mask input labels. Additionally, the model combines contextual information to predict masked labels, thereby achieving bidirectional representation during training. In the data preprocessing stage of this approach, CodeBERT is introduced to enhance training performance.

Multi-Task Learning [133]: Multi-Task Learning is a smart contract vulnerability detection model that utilizes multi-task learning techniques, and it was introduced in 2022. This model primarily adopts a hard-shared design and is composed of two main components: Initially, text data is transformed into new vectors through word embedding and positional embedding. Subsequently, the network model is employed to learn and extract contracted feature vectors; Following the feature extraction stage, a convolutional neural network is employed to construct a classification model for each task. Features are learned and extracted for training to fulfill the specific objectives of each task. Multi-Task Learning has the capability to detect three distinct types of vulnerabilities.

MODNN [134]: MODNN is a contract vulnerability detection method that employs a multi-object detection neural network model and was published in 2022. MODNN is capable of detecting 12 different types of vulnerabilities. It achieves this by utilizing implicit features and a multi-object detection (MOD) algorithm to identify a broader range of unknown vulnerabilities without the need for expert knowledge or predefined rule sets. MODNN supports parallel detection of multiple vulnerabilities, demonstrating high scalability as it does not require training separate models for each vulnerability type. This effectively reduces both manual and time-related costs associated with the detection process.

MTC [135]: MTC (Markovian Tree Convolution) is a specialized tool designed for the detection and analysis of Ponzi contracts, and it was released in 2021. In the detection process, MTC follows a series of steps: Initially, it uses "solidity-parser-antlr" to convert the source code of smart contracts into an abstract syntax tree (AST); Next, MTCformer is employed to transform the abstract syntax tree of the smart contract source code into a special format of word vector sequences through structural traversal; Subsequently, MTCformer utilizes a multi-head TextCNN (Convolutional Neural Network) to learn local structures and semantic features from the word vector sequences and MTCformer captures long-term dependencies between code and word vectors; Finally, a fully connected neural network with a cost-sensitive loss function is used for classifying the detection results. This approach enables MTC to effectively analyze Ponzi contracts by extracting meaningful features from the contract source code and leveraging neural network models for classification.

4.2.2 EVM Bytecode

ESCORT [136]: ESCORT is a static smart contract vulnerability detection framework based on deep neural networks (DNN) that was introduced in 2021. This framework offers support for lightweight migration learning, addresses concealed security concerns and boasts scalability and simplification. The detection process of ESCORT consists of two main components: the first component extracts the features and semantics of the Ethernet smart contract; the second component obtains the feature inputs from the first component and consists of a multi-branch structure. Each branch in this multi-branch structure targets a specific security vulnerability.

CodeNet [137]: CodeNet, introduced in 2022, presents a novel code-oriented Convolutional Neural Network (CNN) architecture tailored for the security analysis and detection of smart contracts. CodeNet can detect vulnerable smart contracts without stepping over them while maintaining semantic and contextual relationships. CodeNet's vulnerability detection method is divided into two main steps: first data preprocessing, which converts the contract source code into bytecode, and then the bytecode is converted into an input image based on the smart contract used for the CNN architecture; then vulnerability detection, which analyzes the smart contract vulnerability detection based on the input image. CodeNet can detect 4 types of vulnerabilities: reentrant vulnerabilities, Tx.Origin, timestamp dependencies, and unchecked return values.

SC-VDM [138]: SC-VDM (Smart Contract Vulnerability Detection Model) is a lightweight intelligent contract vulnerability detection model based on Convolutional Neural Networks (CNN). It was introduced in 2021. This model takes the bytecode of smart contracts and converts it into grayscale matrix images, which are then used as input for the CNN-based vulnerability detection process. One notable advantage of SC-VDM is its ability to effectively detect common vulnerability types without requiring expert knowledge. Additionally, it is known for its fast detection speed, making it a valuable tool for quickly identifying vulnerabilities in smart contracts. This approach leverages the power of deep learning techniques and image-based representations to enhance the accuracy and efficiency of smart contract vulnerability detection.

4.2.3 Opcode

VSCL [139]: VSCL is an automated intelligent contract vulnerability detection framework that was released in 2021. VSCL achieves more accurate vulnerability detection by adding a metric learning module to the DNN network model, using bytecode as a dataset, converting the bytecode to a sequence of opcodes through a disassembler and an operational control flow graph (CFG), forming features with an n-gram, scoring the features through a TFIDF, and then feeding the features into a deep neural network model based on metric learning for detection and evaluation.

Table 2 provides an analysis and comparison of 20 deep learning-based vulnerability detection tools from various perspectives, including open-source nature, data input types, and feature extraction methods. Table 3 provides a detailed comparison of the types of vulnerability detection for deep learning tools. Specifically, feature extraction methods in Table 2 can be categorized into the following three classes.

Table 2: Comparison of deep learning detection tools analysis

Classify	Tools	Open source	Years	Solidity Code	EVM Bytecode	Opcode	Multiple vulnerability detections	Literature
Text processing	BLSTM-att	√	2020	√	×	×	×	[72]
	CBGRU	√	2022	√	×	×	√	[73]
	SVChecker	√	2022	√	×	×	√	[74]
	SPCBIG-EC	√	2022	√	×	×	√	[75]
	SCVDIE-ENSEMBLE	√	2022	×	×	√	√	[80]
	HAM-BiLSTM	×	2022	√	×	×	×	[81]
	HAM-BiGRU	×	2023	√	×	×	√	[82]
	VDDL	×	2023	√	×	×	√	[83]
	Multi-Task Learning	×	2022	√	×	×	√	[84]
	MODNN	×	2022	√	×	×	√	[85]
	ESCORT	×	2021	×	√	×	√	[87]
Static analysis	DeeSCVHunter	√	2021	√	×	×	√	[76]
	MANDO-GURU	√	2022	√	×	×	√	[77]
	CGE	√	2021	√	×	×	√	[78]
	GNNs	√	2020	√	×	×	√	[53]
	(B)ASGVul Detector	√	2022	√	√	×	√	[79]
	MTC	×	2021	√	×	×	×	[86]
	VSCL	×	2021	×	×	√	√	[90]

(Continued)

Table 2 (continued)

Classify	Tools	Open source	Years	Solidity Code	EVM Bytecode	Opcode	Multiple vulnerability detections	Literature
Image processing	CodeNet	×	2022	×	√	×	√	[88]
	SC-VDM	×	2021	×	√	×	√	[81]

Table 3: Comparison of deep learning detection tools analysis

Tool vulnerability type	Reent rancy	Integer over flow	Times tamp depen-dency	TOD	Call-stack depth	Denial of service	Access control	Infinite loop	Delegate call	Unchec ked call	Ponzi contr acts	Tx.Origin	Block parame-ter depen-dency
BLSTM-att	√												
DeeSCVHunter	√		√										
CBGRU	√	√	√		√			√					
SVChecker	√	√	√	√			√			√			
HAM-BiLSTM	√												
HAM-BiGRU	√	√	√							√		√	
VDDL	√	√											
MANDO-GURU													
SPCBIG-EC	√	√	√		√			√					
SCVDIE-ENSEMBLE	√	√	√	√	√				√				
MTC											√		
Multi-Task Learning	√	√											
MODNN	√	√	√	√	√							√	√
VSCL	√												
ESCORT	√			√		√	√						
GNNs	√		√					√					
(B)ASGVulDetector	√		√									√	√
CGE	√		√					√					
CodeNet	√		√							√		√	
SC-VDM	√					√				√			

(1) Text processing-based detection method: This method uses Natural Language Processing (NLP) techniques to extract semantic features of smart contracts, such as word embedding, attention mechanisms, etc. Then, models such as Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), etc., are used to classify or regress the smart contract to determine whether there is a vulnerability or to classify the type of vulnerability. This method can intuitively migrate the network model used in natural language processing to smart contract vulnerability detection and can achieve relatively good detection results, but this method ignores structural information such as function calls, data dependencies, and other information in the code, which may lead to false positives or missed positives.

(2) The static analysis-based detection method: Static analysis techniques extract structural information from smart contracts, such as Control Flow Graphs (CFGs), Data Flow Graphs (DFGs), Abstract Syntax Trees (ASTs), and more. Subsequently, models like Graph Neural Networks (GNNs) and Auto Encoders (AEs) are used for feature extraction and representation learning of smart

contracts. Finally, a classifier or clustering algorithm is applied for vulnerability detection. This method is effective in improving the accuracy of smart contract vulnerability detection and can adapt to various forms of smart contract representations.

(3) The image-based detection method: In this approach, smart contracts are transformed into pixel images. Image processing techniques are then applied to extract image pattern features, such as edge detection, filters, pooling layers, and more. Subsequently, Convolutional Neural Networks (CNNs) or other image classification models are used to classify these pixel images, ultimately achieving contract vulnerability classification.

5 Experimentation and Discussion

This paper provides a comprehensive summary of Ethereum smart contract vulnerability detection tools based on deep learning, covering 20 different methods. The timeframe for this review spans from 2020 to February 2023, and it includes the collection of 90 relevant articles and review materials, encompassing both journal papers and online resources. Fig. 13 illustrates common vulnerability types that can be detected using deep learning methods, with reentrancy vulnerabilities, timestamp dependencies, integer overflows, TOD, and call stack overflows being among the most commonly detected vulnerability types.

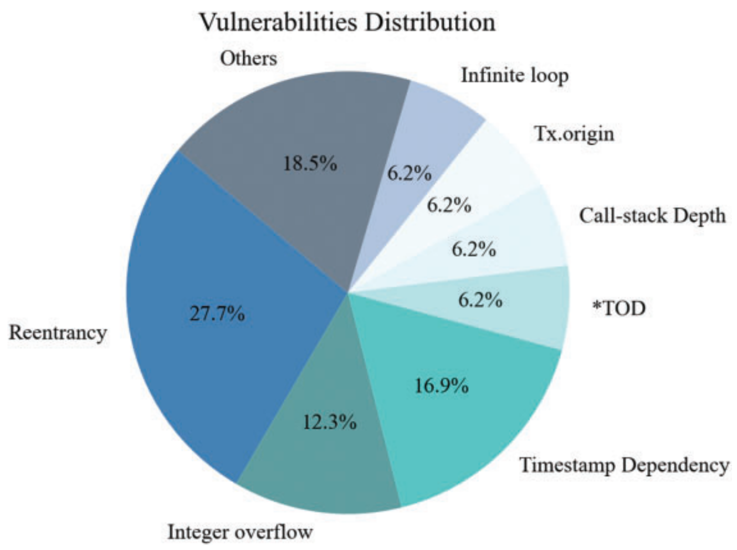


Figure 13: Percentage of contract vulnerability detection types based on deep learning

5.1 Detection Tool Performance Evaluation System

In order to visualize the performance of the detection tools more intuitively, this section evaluates these detection tools through accuracy and F1-Score in Table 3. The performance of the smart contract vulnerability detection tool is evaluated in two main ways:

(1) Accuracy

The accuracy rate serves as an intuitive metric that can to some extent reflect the performance strengths and weaknesses of detection tools. It is calculated based on four key indicators: true positive (TP), false positive (FP), false negative (FN), and true negative (TN), making it a crucial assessment index. The four indicators are defined as follows: "Positive" means that there is a vulnerability in the contract and "Negative" means that there is no vulnerability:

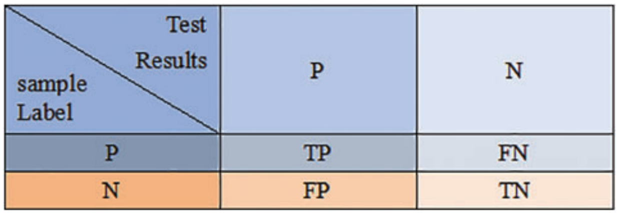
True Positive (TP): Denotes instances where the model correctly identifies a target sample as such. In essence, it signifies actual positive contracts being accurately detected as positive;

False Positive (FP): Represents cases where the model incorrectly identifies a non-target sample as a target sample. Essentially, it indicates situations where actual negative contracts are erroneously identified as positive;

True Negative (TN): Indicates when the model correctly identifies a non-target sample as a non-target sample. This refers to instances where actual negative contracts are appropriately identified as negative;

False Negative (FN): Signifies cases where the model mistakenly identifies a target sample as a non-target sample. It refers to scenarios where actual positive contracts are erroneously classified as negative.

In order to more visually represent the link between the evaluation indicators TP, FP, TN and FN, Fig. 14 shows the specific relationship between the sample situation and the evaluation indicators.



Test Results / sample Label	P	N
P	TP	FN
N	FP	TN

Figure 14: Relationship between sample conditions and evaluation indicators

Accuracy is calculated as in Eq. (1), where TP+FP+FN+TN is equal to the sum of test samples for all contracts.

$$Acc = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

(2) F1-Score

F1-Score is a very important metric in classification problems, belonging to a reasonably represented average between Precision and Recall, and is often used as a final performance evaluation metric for classification problems. The F1-Score is calculated as in Eqs. (2)–(4).

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

5.2 Experimental Validation

Among the 20 intelligent contract vulnerability detection tools based on deep learning methods mentioned earlier, this paper selected one tool from different categories of feature extraction methods, considering factors such as open-source availability and usability, to serve as representatives for analysis. The tool based on image processing for vulnerability detection is not open-source, so we chose the results from the SC-VDM paper as a reference. In the experimental validation in this paper, two representative detection methods SPCBIG-EC and GNNs based on text processing and based on static analysis are selected for experimental validation on the unified dataset CESC, which is a homemade dataset disclosed by the open-source method SPCBIG-EC in its original article, and the data are all derived from crawling the data of the contracts of the EtherNet website. In the text-based processing category, SPCBIG-EC exhibited superior detection performance and stability, making it the tool of choice for experimental comparisons. In the static analysis-based category, we selected GNNs from the available tools for comparison. For common vulnerabilities such as reentrancy and timestamp dependency, performance comparisons of vulnerability detection tools were conducted based on accuracy and F1-Score as the two key metrics.

From the perspective of detection types, most deep learning-based vulnerability detection tools support multiple types of vulnerability detection. Among the two open-source tools selected in this paper, both SPCBIG-EC and GNNs are capable of detecting multiple types of vulnerabilities. However, the majority of tools are designed to detect specific types of vulnerabilities, and a comprehensive detection tool that covers all vulnerability types has not yet emerged. For a more comprehensible evaluation of the effectiveness of deep learning-based smart contract vulnerability detection tools, we have chosen to compare them with three widely utilized and reasonably effective traditional tools: Oyente, SmartCheck, and Mythril.

In the experimental validation of SPCBIG-EC, we successfully replicated the detection of two vulnerability types: reentrancy and timestamp dependencies. The experimental results are presented in Table 4. For reentrancy vulnerability detection, SPCBIG-EC achieved an accuracy rate of 96.12%. Regarding timestamp dependency vulnerabilities, the detection accuracy reached 93.96%. On the same CESC dataset, the traditional tool Oyente demonstrated an accuracy rate of 69.56% for reentrancy vulnerability detection. In contrast, SmartCheck and Mythril exhibited relatively lower detection accuracies of 56.52% and 52.17%, respectively. For timestamp dependency vulnerabilities, SmartCheck had the highest accuracy among the traditional tools, at 61.53%, while Oyente and Mythril had accuracies of 57.69% and 46.15%, respectively. Furthermore, SPCBIG-EC's F1-Score also outperformed the traditional tools.

Table 4: Experimental validation of SPCBIG-EC, GNNs and conventional tools

Vulnerability type	Detection tools	Acc (%)	R (%)	P (%)	F1 (%)
Reentrancy	Oyente	69.56	62.50	55.55	58.79
	SmartCheck	56.52	36.14	38.96	37.50
	Mythril	52.17	34.42	29.41	31.72
	SPCBIG-EC	96.12	98.55	93.35	95.94
	GNNs	90.87	88.23	91.12	89.63
Timestamp dependency	Oyente	57.69	58.50	33.47	42.58
	SmartCheck	61.53	40.81	37.76	39.23
	Mythril	46.15	31.29	32.45	31.86
	SPCBIG-EC	93.96	95.83	92.87	94.25
	GNNs	87.82	83.92	81.93	85.12

We replicated the detection of reentrant and timestamp dependencies by GNNs on the same CESC dataset, as shown in Table 4. For reentrancy vulnerability detection, GNNs achieved a maximum accuracy of 90.87% and an F1-Score of 89.63%. Regarding timestamp dependency vulnerabilities, GNNs achieved an accuracy of 87.82% and an F1-Score of 85.12%. Among the traditional tools, SmartCheck had the highest detection accuracy at 61.53%, but its F1-Score was relatively lower. Notably, Mythril had an F1-Score of only 31.72%.

In the paper by Zhou et al. [138], they introduced a vulnerability detection method based on image processing, known as SC-SPP-CNN. The SD dataset comprises data collected autonomously from nearly 20,000 contracts on Ethereum, whereas the CD dataset relies on the dataset provided by reference [140]. As this approach has not been open-sourced, the experimental data are exclusively sourced from reference [138]. The experimental results of the SC-SPP-CNN model on the SD and CD datasets are presented in Table 5. SC-SPP-CNN achieved accuracy rates of over 80% for all four types of vulnerabilities, with the highest accuracy of 89.52% for reentrancy vulnerability detection. For three out of the four vulnerability types, SC-SPP-CNN achieved F1-Scores of over 90%, while the F1-Score for timestamp dependency detection was 86.58%. In terms of detection accuracy, SC-SPP-CNN outperformed the CNN model for all vulnerabilities. However, for the detection of unverified low-level call vulnerabilities, the CNN model had a slightly better F1-Score of 90.26% compared to SC-SPP-CNN.

Table 5: Experimental data for the SC-VDM model

Vulnerability type	Detection tools	SC-SPP-CNN				CNN			
		Acc (%)	R (%)	P (%)	F1 (%)	Acc (%)	R (%)	P (%)	F1 (%)
SD	Timestamp dependency	81.33	90.32	84.83	86.58	79.48	88.00	82.42	85.12
	Unchecked low-level call	85.63	93.85	86.64	90.10	85.20	98.32	83.41	90.26
BD	DOS	84.48	92.46	87.74	90.03	84.10	91.51	88.00	89.72
	Reentrancy	89.52	99.03	89.38	93.96	89.30	99.58	88.78	93.87

In summary, this paper has provided a synthesis of three deep learning-based vulnerability detection tools:

Indeed, SPCBIG-EC, as a vulnerability detection tool based on text processing methods, has demonstrated commendable performance in detecting reentrancy and timestamp dependency vulnerabilities during experimental validation. Both its accuracy rate and F1-Score outshine those of traditional tools significantly. This approach allows for the convenient adaptation of certain neural network models from natural language processing to smart contract vulnerability detection. However, it is important to acknowledge that smart contract code differs fundamentally from natural language text. Smart contract code conveys information with a distinct logical hierarchy, which presents a challenge for methods like SPCBIG-EC in capturing the sequential and logical aspects of smart contract code, including the order of execution and the logical structure.

GNNs is a vulnerability detection tool based on static analysis methods. Through experimental validation, GNNs have also exhibited excellent performance in detecting reentrancy and timestamp dependency vulnerabilities, surpassing traditional tools by over 20%. This class of methods demonstrates good scalability, as it leverages techniques like generating control flow graphs or syntax trees to effectively preserve the logical sequences within smart contracts. This preservation of contract logic contributes to enhancing the accuracy of smart contract vulnerability detection.

SC-VDM is a vulnerability detection tool employing image processing methods. In the experiments presented in reference [138], the SC-SPP-CNN model is capable of batch detection for various vulnerabilities by converting contract code into two-dimensional grayscale images and using image processing techniques for vulnerability detection. SC-VDM achieves an accuracy rate of over 80% for the detection of different types of vulnerabilities. However, it is worth noting that the approach of transforming code detection into image processing lacks a certain level of interpretability, and further research is needed in this area to gain a better understanding of the method.

6 Conclusion

Artificial intelligence has made rapid development in recent years, and blockchain technology has been fully recognized and trusted. In the field of smart contract security issues, deep learning-based vulnerability detection methods have gradually become popular. This paper provides a comprehensive review of smart contract vulnerability detection methods based on deep learning. Unlike previous overviews of vulnerability detection tools encompassing all categories, this paper delves into the research on deep learning methods in the field of smart contract vulnerability detection in a more detailed and comprehensive manner. The review categorizes these methods based on factors such as tool open-source availability, data input types, and differences in feature extraction. Then, different classes of tools are experimentally validated to evaluate their performance and effectiveness in smart contract vulnerability detection. In comparison with other relevant works, this review offers a more detailed examination of deep learning-based smart contract vulnerability detection tools.

7 Challenges and Future Prospects

With the continuous evolution of blockchain technology, the security challenges smart contracts in the future are anticipated to grow in complexity. Presently, deep learning-based method for detecting vulnerabilities in smart contracts confront notable limitations and hurdles. Such as most of these detection tools are not open source and cannot be quickly installed for direct detection. Furthermore, the absence of authoritative public datasets suitable for validating the efficacy of such tools poses a significant challenge. Additionally, individual tools exhibit limited coverage of smart contract vulnerability types. Moreover, prevailing detection methodologies predominantly rely on static models, lacking robustness in incorporating dynamic models. To address these challenges, the following recommendations are proposed as prospective directions for advancing the domain of smart contract vulnerability detection:

(1) Increase the open-source of relevant contract detection tools

At present, many deep learning-based smart contract vulnerability detection tools remain either closed-source or semi-open-source. To facilitate research and analysis in this field, it is highly advisable to increase the open-source availability of such detection methods. Improved open-source accessibility would enable better integration between detection tools and deep learning methods, thereby fostering the development and research of deep learning-based smart contract vulnerability detection.

(2) Specification of public datasets that cover the full range of vulnerabilities

In the context of vulnerability detection using deep learning methods, a substantial reliance is placed on powerful neural networks, with the training and testing of these networks being contingent upon extensive datasets. Presently, the majority of detection tools based on deep learning are trained and tested on custom-built datasets. This situation results in a lack of standardized evaluation metrics across different detection tools. Hence, the research field urgently requires a comprehensive and authoritative public dataset that encompasses a wide range of vulnerability types.

(3) Increase vulnerability type detection coverage

The deep learning-based smart contract vulnerability detection tools reviewed in this paper detect up to 10 types of vulnerabilities, and most of them target the more common and well-known vulnerability types. There are currently more than 20 known smart contract vulnerability types, and it is hoped that in the future we can integrate the tools for detecting all types of vulnerabilities together, establish a unified and standardized deep learning model, and improve the coverage of vulnerability type detection.

(4) Strengthen the combination of dynamic analysis and detection

At this stage, most of the contract detection tools, whether traditional or based on deep learning, are still based on static analysis of source code, bytecode, opcode, AST, or CFG. However, intelligent contract vulnerabilities manifest during runtime. In the future, integrating dynamic features from smart contract execution into the feature extraction process of neural network models can significantly enhance the efficiency and accuracy of vulnerability detection.

Acknowledgement: The authors are thankful to Zhengzhou University of Light Industry as well as Prof. Ning Hu of Guangzhou University for their support in this work.

Funding Statement: This research has been partially funded by the Major Public Welfare Special Fund of Henan Province (No. 201300210200), the Major Science and Technology Research Special Fund of Henan Province (No. 221100210400).

Author Contributions: All authors of this paper contributed equally to the research and writing of this paper, and all authors worked closely together throughout the research process, engaging in discussions and collaborating together to enhance this paper with a more in-depth review and examination of smart contract vulnerability detection tools based on deep learning methods. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: All the reviewed research literature and used data in this research paper consists of publicly available scholarly articles, conference proceedings, books, and reports. The references and citations are contained in the reference list of this manuscript and can be accessed through online databases, academic libraries, or by contacting the respective publishers.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Nakamoto, S., Bitcoin, A. (2008). A peer-to-peer electronic cash system. *Bitcoin, 4(2),* 15. https://doi.org/10.2139/ssrn.3440802
2. Swan, M. (2015). *Blockchain: Blueprint for a new economy*. 1005 Gravenstein Highway North, Sebastopol, O'Reilly Media, Inc. https://dl.acm.org/doi/10.5555/3006358
3. Zheng, Z., Xie, S., Dai, H. N., Chen, X., Wang, H. (2018). Blockchain challenges and opportunities: A survey. *International Journal of Web and Grid Services, 14(4),* 352–375. https://doi.org/10.1504/IJWGS.2018.095647

4. di Vaio, A., Hassan, R., Palladino, R. (2022). Blockchain technology and gender equality: A systematic literature review. *International Journal of Information Management, 68,* 102517. https://doi.org/10.1016/j.ijinfomgt.2022.102517
5. Andoni, M., Robu, V., Flynn, D., Abram, S., Geach, D. et al. (2019). Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews, 100,* 143–174. https://doi.org/10.1016/j.rser.2018.10.014
6. Xia, Q., Sifah, E. B., Smahi, A., Amofa, S., Zhang, X. (2017). Bbds: Blockchain-based data sharing for electronic medical records in cloud environments. *Information, 8(2),* 44. https://doi.org/10.3390/info8020044
7. Cao, S., Zhang, G., Liu, P., Zhang, X., Neri, F. (2019). Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain. *Information Sciences, 485,* 427–440. https://doi.org/10.1016/j.ins.2019.02.038
8. Wang, Y., Zhang, A., Zhang, P., Wang, H. (2019). Cloud-assisted EHR sharing with security and privacy preservation via consortium blockchain. *IEEE Access, 7,* 136704–136719. https://doi.org/10.1109/ACCESS.2019.2943153
9. Chen, L., Lee, W. K., Chang, C. C., Choo, K. K. R., Zhang, N. (2019). Blockchain based searchable encryption for electronic health record sharing. *Future Generation Computer Systems, 95,* 420–429. https://doi.org/10.1016/j.future.2019.01.018
10. Gupta, B. B., Li, K. C., Leung, V. C., Psannis, K. E., Yamaguchi, S. et al. (2021). Blockchain-assisted secure fine-grained searchable encryption for a cloud-based healthcare cyber-physical system. *IEEE/CAA Journal of Automatica Sinica, 8(12),* 1877–1890. https://doi.org/10.1109/JAS.2021.1004003
11. Christidis, K., Devetsikiotis, M. (2016). Blockchains and smart contracts for the Internet of Things. *IEEE Access, 4,* 2292–2303. https://doi.org/10.1109/ACCESS.2016.2566339
12. Bahga, A., Madisetti, V. K. (2016). Blockchain platform for industrial Internet of Things. *Journal of Software Engineering and Applications, 9(10),* 533–546. https://doi.org/10.1109/ACCESS.2016.2566339
13. Kshetri, N. (2017). Can blockchain strengthen the Internet of Things? *IT Professional, 19(4),* 68–72. https://doi.org/10.1109/MITP.2017.3051335
14. Da Xu, L., Viriyasitavat, W. (2019). Application of blockchain in collaborative Internet-of-Things services. *IEEE Transactions on Computational Social Systems, 6(6),* 1295–1305. https://doi.org/10.1109/TCSS.2019.2913165
15. Fernández-Caramés, T. M., Fraga-Lamas, P. (2018). A review on the use of blockchain for the Internet of Things. *IEEE Access, 6,* 32979–33001. https://doi.org/10.1109/ACCESS.2018.2842685
16. Saberi, S., Kouhizadeh, M., Sarkis, J., Shen, L. (2019). Blockchain technology and its relationships to sustainable supply chain management. *International Journal of Production Research, 57(7),* 2117–2135. https://doi.org/10.1080/00207543.2018.1533261
17. Abeyratne, S. A., Monfared, R. P. (2016). Blockchain ready manufacturing supply chain using distributed ledger. *International Journal of Research in Engineering and Technology, 5(9),* 1–10. https://api.semanticscholar.org/CorpusID:52086804
18. Chen, S., Shi, R., Ren, Z., Yan, J., Shi, Y. et al. (2017). A blockchain-based supply chain quality management framework. *2017 IEEE 14th International Conference on E-Business Engineering (ICEBE)*, Shanghai, China, IEEE. https://doi.org/10.1109/ICEBE.2017.34
19. Arunmozhi, M., Venkatesh, V., Arisian, S., Shi, Y., Sreedharan, V. R. (2022). Application of blockchain and smart contracts in autonomous vehicle supply chains: An experimental design. *Transportation Research Part E: Logistics and Transportation Review, 165,* 102864. https://doi.org/10.1016/j.tre.2022.102864
20. Natanelov, V., Cao, S., Foth, M., Dulleck, U. (2022). Blockchain smart contracts for supply chain finance: Mapping the innovation potential in Australia-China beef supply chains. *Journal of Industrial Information Integration, 30,* 100389. https://doi.org/10.1016/j.jii.2022.100389
21. Meng, Z., Morizumi, T., Miyata, S., Kinoshita, H. (2018). Design scheme of copyright management system based on digital watermarking and blockchain. *2018 IEEE 42nd Annual*

Computer Software and Applications Conference (COMPSAC), vol. 2. Tokyo, Japan, IEEE. https://doi.org/10.1109/COMPSAC.2018.10258

22. Holland, M., Nigischer, C., Stjepandić, J. (2017). Copyright protection in additive manufacturing with blockchain approach. In: *Transdisciplinary engineering: a paradigm shift*, pp. 914–921. Amsterdam: IOS Press. https://doi.org/10.3233/978-1-61499-779-5-914
23. Qian, P., Liu, Z., Wang, X., Chen, J., Wang, B. et al. (2019). Digital resource rights confirmation and infringement tracking based on smart contracts. *2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Singapore, IEEE. https://doi.org/10.1109/CCIS48116.2019.9073733
24. Pech, S. (2020). Copyright unchained: How blockchain technology can change the administration and distribution of copyright protected works. *Northwestern Journal of Technology and Intellectual Property, 18,* 1. https://doi.org/10.2139/ssrn.3578311
25. Yuldashov, A., Usmonov, V. (2022). Copyright protection in telecommunications networks: The example of blockchain technology. *European Journal of Interdisciplinary Research and Development, 4,* 22–29. http://www.ejird.journalspark.org/index.php/ejird/article/view/57
26. Mengelkamp, E., Notheisen, B., Beer, C., Dauer, D., Weinhardt, C. (2018). A blockchain-based smart grid: Towards sustainable local energy markets. *Computer Science-Research and Development, 33,* 207–214. https://doi.org/10.1007/s00450-017-0360-9
27. Pop, C., Cioara, T., Antal, M., Anghel, I., Salomie, I. et al. (2018). Blockchain based decentralized management of demand response programs in smart energy grids. *Sensors, 18(1),* 162. https://doi.org/10.3390/s18010162
28. Knirsch, F., Unterweger, A., Eibl, G., Engel, D. (2018). Privacy-preserving smart grid tariff decisions with blockchain-based smart contracts. *Sustainable Cloud and Energy Services: Principles and Practice,* 85–116. https://doi.org/10.1007/978-3-319-62238-5_4
29. Miglani, A., Kumar, N., Chamola, V., Zeadally, S. (2020). Blockchain for internet of energy management: Review, solutions, and challenges. *Computer Communications, 151,* 395–418. https://doi.org/10.1016/j.comcom.2020.01.014
30. Kirli, D., Couraud, B., Robu, V., Salgado-Bravo, M., Norbu, S. et al. (2022). Smart contracts in energy systems: A systematic review of fundamental approaches and implementations. *Renewable and Sustainable Energy Reviews, 158,* 112013. https://doi.org/10.1016/j.rser.2021.112013
31. Vitalik, B. (2015). Ethereum (ETH) blockchain explorer. https://etherscan.io/ (accessed on 05/04/2023).
32. Turing, A. M. (1936). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society, 42(1),* 230–265. https://doi.org/10.1112/plms/s2-42.1.230
33. Szabo, N. (1996). Smart contracts: Building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought, 16(2),* 28. https://api.semanticscholar.org/CorpusID:198956172 (accessed on 06/04/2023).
34. Zheng, Z., Xie, S., Dai, H. N., Chen, W., Chen, X. et al. (2020). An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems, 105,* 475–491. https://doi.org/10.1016/j.future.2019.12.019
35. Wang, S., Yuan, Y., Wang, X., Li, J., Qin, R. et al. (2018). An overview of smart contract: Architecture, applications, and future trends. *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, China, IEEE. https://doi.org/10.1109/IVS.2018.8500488
36. Pranata, A. R., Tehrani, P. M. (2022). The legality of smart contracts in a decentralized autonomous organization (DAO). In: *Regulatory aspects of artificial intelligence on blockchain*, pp. 112–131. Hershey, Pennsylvania, USA: IGI Global. https://doi.org/10.4018/978-1-7998-7927-5.ch006
37. Bui, V. C., Wen, S., Yu, J., Xia, X., Haghighi, M. S. et al. (2020). Evaluating upgradable smart contract. *2021 IEEE International Conference on Blockchain (Blockchain)*, Melbourne, Australia, IEEE. https://doi.org/10.1109/Blockchain53845.2021.00041

38. Zhao, X., Chen, Z., Chen, X., Wang, Y., Tang, C. (2017). The DAO attack paradoxes in propositional logic. *2017 4th International Conference on Systems and Informatics (ICSAI)*, Hangzhou, China, IEEE. https://doi.org/10.1109/ICSAI.2017.8248566
39. Perez, D., Livshits, B. (2021). Smart contract vulnerabilities: Vulnerable does not imply exploited. *30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, Canada.
40. Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., Lee, H. N. et al. (2022). Systematic review of security vulnerabilities in ethereum blockchain smart contract. *IEEE Access, 10,* 6605–6621. https://doi.org/10.1109/ACCESS.2021.3140091
41. Ren, X., Wu, Y., Li, J., Hao, D., Alam, M. (2023). Smart contract vulnerability detection based on a semantic code structure and a self-designed neural network. *Elsevier, 109,* 108766. https://doi.org/10.1016/j.compeleceng.2023.108766
42. Li, A., Choi, J. A., Long, F. (2020). Securing smart contract with runtime validation. *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, London, UK. https://doi.org/10.1145/3385412.3385982
43. Giesen, J. R., Andreina, S., Rodler, M., Karame, G. O., Davi, L. (2022). Practical mitigation of smart contract bugs. https://doi.org/10.48550/arXiv.2203.00364
44. Li, Z., Guo, W., Xu, Q., Xu, Y., Wang, H. et al. (2020). Research on blockchain smart contracts vulnerability and a code audit tool based on matching rules. *Proceedings of the. International Conference on Cyberspace Innovation of Advanced Technologies*, Guangzhou China. https://doi.org/10.1145/3444370.3444617
45. Prasad, B., Ramachandram, S. (2022). Prevention and detection mechanisms for re-entrancy attack and king of ether throne attack for ethereum smart contracts. *Ingénierie des Systèmes d'Information, 27(5).* https://doi.org/10.18280/isi.270505
46. Prasad, B., Ramachandram, S. (2021). Vulnerabilities and attacks on smart contracts over blockchain. *Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(11),* 5436–5449. https://doi.org/10.17762/turcomat.v12i11.6773
47. Samreen, N. F., Alalfi, M. H. (2021). A survey of security vulnerabilities in ethereum smart contracts. https://doi.org/10.48550/arXiv.2105.06974
48. Almakhour, M., Sliman, L., Samhat, A. E., Mellouk, A. (2020). Verification of smart contracts: A survey. *Pervasive and Mobile Computing, 67,* 101227. https://doi.org/10.1016/j.pmcj.2020.101227
49. Liu, J., Liu, Z. (2019). A survey on security verification of blockchain smart contracts. *IEEE Access, 7,* 77894–77904. https://doi.org/10.1109/ACCESS.2019.2921624
50. Tolmach, P., Li, Y., Lin, S. W., Liu, Y., Li, Z. (2021). A survey of smart contract formal specification and verification. *ACM Computing Surveys (CSUR), 54(7),* 1–38. https://doi.org/10.1145/3464421
51. Di Angelo, M., Salzer, G. (2019). A survey of tools for analyzing ethereum smart contracts. *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, Newark, CA, USA, IEEE. https://doi.org/10.1109/DAPPCON.2019.00018
52. Hu, B., Zhang, Z., Liu, J., Liu, Y., Yin, J. et al. (2021). A comprehensive survey on smart contract construction and execution: Paradigms, tools, and systems. *Patterns, 2(2).* https://doi.org/10.48550/arXiv.2008.13413
53. Ante, L. (2021). Smart contracts on the blockchain-a bibliometric analysis and review. *Telematics and Informatics, 57,* 101519. https://doi.org/10.1016/j.tele.2020.101519
54. He, D., Deng, Z., Zhang, Y., Chan, S., Cheng, Y. et al. (2020). Smart contract vulnerability analysis and security audit. *IEEE Network, 34(5),* 276–282. https://doi.org/10.1109/MNET.001.1900656
55. Kushwaha, S. S., Joshi, S., Singh, D., Kaur, M., Lee, H. N. (2022). Ethereum smart contract analysis tools: A systematic review. *IEEE Access, 10,* 57037–57062. https://doi.org/10.1109/ACCESS.2022.3169902

56. Jacsó, P. (2005). Google scholar: The pros and the cons. *Online Information Review, 29(2),* 208–214. https://doi.org/10.1108/14684520510598066
57. Qiu, J., Lv, H. (2014). An overview of knowledge management research viewed through the web of science (1993–2012). *Aslib Journal of Information Management, 66(4),* 424–442. https://doi.org/10.1108/AJIM-12-2013-0133
58. Kitchenham, B. A., Budgen, D., Brereton, O. P. (2011). Using mapping studies as the basis for further research-a participant-observer case study. *Information and Software Technology, 53(6),* 638–651. https://doi.org/10.1016/j.infsof.2010.12.011
59. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M. (2008). Systematic mapping studies in software engineering. *12th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, Trondheim, Norway. https://dl.acm.org/doi/10.5555/2227115.2227123
60. Wainer, J., de Oliveira, H. P., Anido, R. (2011). Patterns of bibliographic references in the ACM published papers. *Information Processing & Management, 47(1),* 135–142. https://doi.org/10.1016/j.ipm.2010.07.002
61. Lawrence, S., Giles, C. L. (1999). Searching the web: General and scientific information access. *IEEE Communications Magazine, 37(1),* 116–122. https://doi.org/10.1109/35.739314
62. Sotudeh, H., Ghasempour, Z., Yaghtin, M. (2015). The citation advantage of author-pays model: The case of springer and elsevier OA journals. *Scientometrics, 104,* 581–608. https://doi.org/10.1007/s11192-015-1607-5
63. Sutton, C., Gong, L. (2017). Popularity of arxiv. org within computer science. https://doi.org/10.48550/arXiv.1710.05225
64. Khan, S. N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., Bani-Hani, A. (2021). Blockchain smart contracts: Applications, challenges, and future trends. *Peer-to-Peer Networking and Applications, 14,* 2901–2925. https://doi.org/10.1007/s12083-021-01127-0
65. Wang, Z., Jin, H., Dai, W., Choo, K. K. R., Zou, D. (2021). Ethereum smart contract security research: Survey and future research opportunities. *Frontiers of Computer Science, 15,* 1–18. https://doi.org/10.1007/s11704-020-9284-9
66. Hu, T., Liu, X., Chen, T., Zhang, X., Huang, X. et al. (2021). Transaction-based classification and detection approach for ethereum smart contract. *Information Processing & Management, 58(2),* 102462. https://doi.org/10.1016/j.ipm.2020.102462
67. Bhardwaj, A., Shah, S. B. H., Shankar, A., Alazab, M., Kumar, M. et al. (2021). Penetration testing framework for smart contract blockchain. *Peer-to-Peer Networking and Applications, 14,* 2635–2650. https://doi.org/10.1007/s12083-020-00991-6
68. Min, T., Wang, H., Guo, Y., Cai, W. (2019). Blockchain games: A survey. *2019 IEEE Conference on Games (CoG)*, London, UK, IEEE. https://doi.org/10.1109/CIG.2019.8848111
69. Liu, C., Liu, H., Cao, Z., Chen, Z., Chen, B. et al. (2018). Reguard: Finding reentrancy bugs in smart contracts. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, Gothenburg, Sweden. https://doi.org/10.1145/3183440.3183495
70. Liao, J. W., Tsai, T. T., He, C. K., Tien, C. W. (2019). Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing. *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, Granada, Spain, IEEE. https://doi.org/10.1109/IOTSMS48152.2019.8939256
71. Frankenfield, J. (2022). Hard fork: What it is in blockchain, how it works, why it happens. https://www.investopedia.com/terms/h/hard-fork.asp (accessed on 12/04/2023).
72. Ayoade, G., Bauman, E., Khan, L., Hamlen, K. (2019). Smart contract defense through bytecode rewriting. *2019 IEEE International Conference on Blockchain (Blockchain)*, pp. 384–389. Atlanta, GA, USA, IEEE. https://doi.org/10.1109/Blockchain.2019.00059

73. Lai, E., Luo, W. (2020). Static analysis of integer overflow of smart contracts in ethereum. *Proceedings of the 2020 4th International Conference on Cryptography, Security and Privacy*, Nanjing, China. https://doi.org/10.1145/3377644.3377650

74. Sun, J., Huang, S., Zheng, C., Wang, T., Zong, C. et al. (2021). Mutation testing for integer overflow in ethereum smart contracts. *Tsinghua Science and Technology, 27(1),* 27–40. https://doi.org/10.26599/TST.2020.9010036

75. Fu, M., Wu, L., Hong, Z., Feng, W. (2019). Research on vulnerability mining technique for smart contracts. *Journal of Computer Applications, 39(7),* 1959. https://doi.org/10.11772/j.issn.1001-9081.2019010082

76. Atzei, N., Bartoletti, M., Cimoli, T. (2017). A survey of attacks on ethereum smart contracts (SoK). *POST 2017: Principles of security and trust*, pp. 164–186. Uppsala, Sweden: Springer. https://doi.org/10.1007/978-3-662-54455-6_8

77. Zhang, Y., Kasahara, S., Shen, Y., Jiang, X., Wan, J. (2018). Smart contract-based access control for the Internet of Things. *IEEE Internet of Things Journal, 6(2),* 1594–1605. https://doi.org/10.1109/JIOT.2018.2847705

78. Krupp, J., Rossow, C. (2018). TEETHER: Gnawing at ethereum to automatically exploit smart contracts. *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, USA. https://dl.acm.org/doi/10.5555/3277203.3277303

79. Chen, W., Zheng, Z., Ngai, E. C. H., Zheng, P., Zhou, Y. (2019). Exploiting blockchain data to detect smart ponzi schemes on ethereum. *IEEE Access, 7,* 37575–37586. https://doi.org/10.1109/ACCESS.2019.2905769

80. Fang, L., Zhao, B., Li, Y., Liu, Z., Ge, C. et al. (2020). Countermeasure based on smart contracts and AI against DoS/DDoS attack in 5G circumstances. *IEEE Network, 34(6),* 54–61. https://doi.org/10.1109/MNET.021.1900614

81. Singh, R., Tanwar, S., Sharma, T. P. (2020). Utilization of blockchain for mitigating the distributed denial of service attacks. *Security and Privacy, 3(3),* e96. https://doi.org/10.1002/spy2.96

82. Tang, X., Zhou, K., Cheng, J., Li, H., Yuan, Y. (2021). The vulnerabilities in smart contracts: A survey. *ICAIS 2021: Advances in Artificial Intelligence and Security*, pp. 177–190. Dublin, Ireland, Springer. https://doi.org/10.1007/978-3-030-78621-2_14

83. Baby, B., Sunil, A., Thomas, N. (2021). A review analysis on smart contract vulnerabilities using blockchain. *International Conference on Interllectual Property Rights*, Taichung, Taiwan.

84. Zhou, H., Milani Fard, A., Makanju, A. (2022). The state of ethereum smart contracts security: Vulnerabilities, countermeasures, and tool support. *Journal of Cybersecurity and Privacy, 2(2),* 358–378. https://doi.org/10.3390/jcp2020019

85. Chen, H., Pendleton, M., Njilla, L., Xu, S. (2020). A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys (CSUR), 53(3),* 1–43. https://doi.org/10.1145/3391195

86. Perez, D., Livshits, B. (2021). Smart contract vulnerabilities: Vulnerable does not imply exploited. *30th USENIX Security Symposium (USENIX Security 21)*, Vancouver, Canada. https://doi.org/10.48550/arXiv.1902.06710

87. Gill, P., Ray, I., Takami, A. L., Tripunitara, M. (2022). Finding unchecked low-level calls with zero false positives and negatives in ethereum smart contracts. *International Symposium on Foundations and Practice of Security*, Ottawa, ON, Canada, Springer. https://doi.org/10.1007/978-3-031-30122-3_19

88. Luu, L., Chu, D. H., Olickel, H., Saxena, P., Hobor, A. (2016). Making smart contracts smarter. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria. https://doi.org/10.1145/2976749.2978309

89. Zhou, E., Hua, S., Pi, B., Sun, J., Nomura, Y. et al. (2018). Security assurance for smart contract. *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Paris, France, IEEE. https://doi.org/10.1109/NTMS.2018.8328743

90. Tantikul, P., Ngamsuriyaroj, S. (2020). Exploring vulnerabilities in solidity smart. *Proceedings of the 6th International Conference on Information Systems Security and Privacy (ICISSP 2020)*, pp. 317–324.

91. Wöhrer, M., Zdun, U. (2018). Design patterns for smart contracts in the ethereum ecosystem. *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)*, Halifax, NS, Canada, IEEE. https://doi.org/10.1109/Cybermatics_2018.2018.00255
92. Zhao, H., Tan, J. (2022). A critical-path-based vulnerability detection method for tx.origin dependency of smart contract. *International Conference on Smart Computing and Communication*, New York, USA, Springer. https://doi.org/10.1007/978-3-031-28124-2_37
93. Chen, J., Xia, X., Lo, D., Grundy, J., Luo, X. et al. (2020). Defining smart contract defects on ethereum. *IEEE Transactions on Software Engineering, 48(1),* 327–345. https://doi.org/10.1109/TSE.2020.2989002
94. Gupta, B. C., Shukla, S. K. (2019). A study of inequality in the ethereum smart contract ecosystem. *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, Granada, Spain, IEEE. https://doi.org/10.1109/IOTSMS48152.2019.8939257
95. Sayeed, S., Marco-Gisbert, H., Caira, T. (2020). Smart contract: Attacks and protections. *IEEE Access, 8,* 24416–24427. https://doi.org/10.1109/ACCESS.2020.2970495
96. Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R., Bracciali, A. et al. (2018). Smart contracts vulnerabilities: A call for blockchain software engineering? *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, Campobasso, Italy, IEEE. https://doi.org/10.1109/IWBOSE.2018.8327567
97. Zhuang, Y., Liu, Z., Qian, P., Liu, Q., Wang, X. et al. (2021). Smart contract vulnerability detection using graph neural networks. *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, Yokohama, Japan. https://doi.org/10.24963/ijcai.2020/454
98. Yu, X. L., Al-Bataineh, O., Lo, D., Roychoudhury, A. (2020). Smart contract repair. *ACM Transactions on Software Engineering and Methodology (TOSEM), 29(4),* 1–32. https://doi.org/10.1145/3402450
99. Zhuang, Y., Liu, Z., Qian, P., Liu, Q., Wang, X. et al. (2021). Smart contract vulnerability detection using graph neural networks. *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, San Francisco, CA, USA. https://doi.org/10.24963/ijcai.2020/454
100. Zheng, H., Liu, Z. R., Huang, J. H., Qian, S. H. (2022). Verification of transaction ordering dependence vulnerability of smart contract based on CPN. *Journal of System Simulation, 34(7),* 1629–1638. https://doi.org/10.16182/j.issn1004731x.joss.21-0208
101. Wang, W., Song, J., Xu, G., Li, Y., Wang, H. et al. (2020). Contractward: Automated vulnerability detection models for ethereum smart contracts. *IEEE Transactions on Network Science and Engineering, 8(2),* 1133–1144. https://doi.org/10.1109/TNSE.2020.2968505
102. Praitheeshan, P., Pan, L., Doss, R. (2020). Security evaluation of smart contract-based on-chain ethereum wallets. *NSS 2020: Network and System Security*, pp. 22–41. Melbourne, VIC, Australia, Springer. https://doi.org/10.1007/978-3-030-65745-1_2
103. Praitheeshan, P., Pan, L., Yu, J., Liu, J., Doss, R. (2019). Security analysis methods on ethereum smart contract vulnerabilities: A survey. https://doi.org/10.48550/arXiv.1908.08605
104. Mossberg, M., Manzano, F., Hennenfent, E., Groce, A., Grieco, G. et al. (2019). Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, San Diego, CA, USA, IEEE. https://doi.org/10.1109/ASE.2019.00133
105. Shishkin, E. (2019). Debugging smart contracts business logic using symbolic model checking. *Programming and Computer Software, 45,* 590–599. https://doi.org/10.1134/S0361768819080164
106. Zhao, W., Zhang, W., Wang, J., Wang, H., Wu, C. (2020). Smart contract vulnerability detection scheme based on symbol execution. *Journal of Computer Applications, 40(4),* 947. https://doi.org/10.11772/j.issn.1001-9081.2019111919
107. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G. et al. (2016). Formal verification of smart contracts: Short paper. *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, Vienna, Austria. https://doi.org/10.1145/2993600.2993611

108. Bai, X., Cheng, Z., Duan, Z., Hu, K. (2018). Formal modeling and verification of smart contracts. *Proceedings of the 7th International Conference on Software and Computer Applications*, Kuantan, Malaysia. https://doi.org/10.1145/3185089.3185138
109. Abdellatif, T., Brousmiche, K. L. (2018). Formal verification of smart contracts based on users and blockchain behaviors models. *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Paris, France, IEEE. https://doi.org/10.1109/NTMS.2018.8328737
110. Takanen, A., Demott, J. D., Miller, C., Kettunen, A. (2018). *Fuzzing for software security testing and quality assurance*. Norwood, MA, USA: Artech House.
111. Liang, H., Pei, X., Jia, X., Shen, W., Zhang, J. (2018). Fuzzing: State of the art. *IEEE Transactions on Reliability, 67(3),* 1199–1218. https://doi.org/10.1109/TR.2018.2834476
112. Tsankov, P., Dan, A., Drachsler-Cohen, D., Gervais, A., Buenzli, F. et al. (2018). Securify: Practical security analysis of smart contracts. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, Toronto, Canada. https://doi.org/10.1145/3243734.3243780
113. Mueller, B. (2017). Practical smart contract security analysis and exploitation–Part 1. https://hackernoon.com/practical-smart-contract-security-analysis-and-exploitation-part-1-6c2f2320b0c (accessed on 11/04/2023).
114. Grishchenko, I., Maffei, M., Schneidewind, C. (2018). A semantic framework for the security analysis of ethereum smart contracts. In: *Principles of security and trust*, pp. 243–269. Thessaloniki, Greece: Springer. https://doi.org/10.1007/978-3-319-89722-6_10
115. Kalra, S., Goel, S., Dhawan, M., Sharma, S. (2018). Zeus: Analyzing safety of smart contracts. *Network and Distributed Systems Security (NDSS) Symposium 2018*, San Diego, CA, USA.
116. Amani, S., Bégel, M., Bortin, M., Staples, M. (2018). Towards verifying ethereum smart contract bytecode in Isabelle/hol. *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, Los Angeles, CA, USA. https://doi.org/10.1145/3167084
117. Jiang, B., Liu, Y., Chan, W. K. (2018). ContractFuzzer: Fuzzing smart contracts for vulnerability detection. *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Montpellier, France. https://doi.org/10.1145/3238147.3238177
118. Feist, J., Grieco, G., Groce, A. (2019). Slither: A static analysis framework for smart contracts. *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pp. 8–15. Montreal, QC, Canada, IEEE. https://doi.org/10.1109/WETSEB.2019.00008
119. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E. et al. (2018). Smartcheck: Static analysis of ethereum smart contracts. *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, Gothenburg, Sweden. https://doi.org/10.1145/3194113.3194115
120. Gao, J., Liu, H., Liu, C., Li, Q., Guan, Z. et al. (2019). Easyflow: Keep ethereum away from overflow. *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Montreal, QC, Canada, IEEE. https://doi.org/10.1109/ICSE-Companion.2019.00029
121. Qian, P., Liu, Z., He, Q., Zimmermann, R., Wang, X. (2020). Towards automated reentrancy detection for smart contracts based on sequential models. *IEEE Access, 8,* 19685–19695. https://doi.org/10.1109/ACCESS.2020.2969429
122. Zhang, L., Chen, W., Wang, W., Jin, Z., Zhao, C. et al. (2022). CBGRU: A detection method of smart contract vulnerability based on a hybrid model. *Sensors, 22(9),* 3577. https://doi.org/10.3390/s22093577
123. Yuan, Y., Xie, T. (2022). SvChecker: A deep learning-based system for smart contract vulnerability detection. *International Conference on Computer Application and Information Security (ICCAIS 2021)*, vol. 12260. Wuhan, China, SPIE. https://doi.org/10.1117/12.2637775
124. Zhang, L., Li, Y., Jin, T., Wang, W., Jin, Z. et al. (2022). SPCBIG-EC: A robust serial hybrid model for smart contract vulnerability detection. *Sensors, 22(12),* 4621. https://doi.org/10.3390/s22124621

125. Yu, X., Zhao, H., Hou, B., Ying, Z., Wu, B. (2021). Deescvhunter: A deep learning-based framework for smart contract vulnerability detection. *2021 International Joint Conference on Neural Networks (IJCNN)*, Shenzhen, China, IEEE. https://doi.org/10.1109/IJCNN52387.2021.9534324

126. Nguyen, H. H., Nguyen, N. M., Doan, H. P., Ahmadi, Z., Doan, T. N. et al. (2022). Mando-guru: Vulnerability detection for smart contract source code by heterogeneous graph embeddings. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Singapore. https://doi.org/10.1145/3540250.3558927

127. Liu, Z., Qian, P., Wang, X., Zhuang, Y., Qiu, L. et al. (2021). Combining graph neural networks with expert knowledge for smart contract vulnerability detection. *IEEE Transactions on Knowledge and Data Engineering, 35(2),* 1296–1310. https://doi.org/10.1109/TKDE.2021.3095196

128. Zhang, Y., Liu, D. (2022). Toward vulnerability detection for ethereum smart contracts using graph-matching network. *Future Internet, 14(11),* 326. https://doi.org/10.3390/fi14110326

129. Zhang, L., Wang, J., Wang, W., Jin, Z., Zhao, C. et al. (2022). A novel smart contract vulnerability detection method based on information graph and ensemble learning. *Sensors, 22(9),* 3581. https://doi.org/10.3390/s22093581

130. Xu, G., Liu, L., Zhou, Z. (2022). Reentrancy vulnerability detection of smart contract based on bidirectional sequential neural network with hierarchical attention mechanism. *2022 International Conference on Blockchain Technology and Information Security (ICBCTIS)*, Huaihua, China, IEEE. https://doi.org/10.1109/ICBCTIS55569.2022.00024

131. Wu, H., Dong, H., He, Y., Duan, Q. (2023). Smart contract vulnerability detection based on hybrid attention mechanism model. *Applied Sciences, 13(2),* 770. https://doi.org/10.3390/app13020770

132. Jiang, F., Cao, Y., Xiao, J., Yi, H., Lei, G. et al. (2022). VDDL: A deep learning-based vulnerability detection model for smart contracts. *International Conference on Machine Learning for Cyber Security*, Guangzhou, China, Springer. https://doi.org/10.1007/978-3-031-20096-0_6

133. Huang, J., Zhou, K., Xiong, A., Li, D. (2022). Smart contract vulnerability detection model based on multi-task learning. *Sensors, 22(5),* 1829. https://doi.org/10.3390/s22051829

134. Zhang, L., Wang, J., Wang, W., Jin, Z., Su, Y. et al. (2022). Smart contract vulnerability detection combined with multi-objective detection. *Computer Networks, 217,* 109289. https://doi.org/10.1016/j.comnet.2022.109289

135. Chen, Y., Dai, H., Yu, X., Hu, W., Xie, Z. et al. (2021). Improving ponzi scheme contract detection using multi-channel TextCNN and transformer. *Sensors, 21(19),* 6417. https://doi.org/10.3390/s21196417

136. Lutz, O., Chen, H., Fereidooni, H., Sendner, C., Dmitrienko, A. et al. (2021). Escort: Ethereum smart contracts vulnerability detection using deep neural network and transfer learning. https://doi.org/10.48550/arXiv.2103.12607

137. Hwang, S. J., Choi, S. H., Shin, J., Choi, Y. H. (2022). Codenet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection. *IEEE Access, 10,* 32595–32607. https://doi.org/10.1109/ACCESS.2022.3162065

138. Zhou, K., Cheng, J., Li, H., Yuan, Y., Liu, L. et al. (2021). SC-VDM: A lightweight smart contract vulnerability detection model. *DMBD 2021: Data Mining and Big Data*, pp. 138–149. Guangzhou, China, Springer. https://doi.org/10.1007/978-981-16-7476-1_13

139. Mi, F., Wang, Z., Zhao, C., Guo, J., Ahmed, F. et al. (2021). VSCL: Automating vulnerability detection in smart contracts with deep learning. *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, Sydney, Australia, IEEE. https://doi.org/10.1109/ICBC51069.2021.9461050

140. Durieux, T., Ferreira, J. F., Abreu, R., Cruz, P. (2020). Empirical review of automated analysis tools on 47,587 ethereum smart contracts. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, Seoul, South Korea. https://doi.org/10.1145/3377811.3380364