IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 29, 2022, accepted April 20, 2022, date of publication April 22, 2022, date of current version June 3, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3169902

Ethereum Smart Contract Analysis Tools: A Systematic Review

SATPAL SINGH KUSHWAHA[1], (Member, IEEE), SANDEEP JOSHI[1], (Senior Member, IEEE), DILBAG SINGH[2], (Senior Member, IEEE), MANJIT KAUR[2], (Member, IEEE), AND HEUNG-NO LEE[2], (Senior Member, IEEE)

[1]Department of Computer Science and Engineering, Manipal University Jaipur, Jaipur 303007, India
[2]School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju 61005, South Korea

Corresponding author: Heung-No Lee (heungno@gist.ac.kr)

This work was supported in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant by the Korean Government through the Ministry of Science and ICT (MSIT) (Development of of decentralized consensus composition technology for large- scale nodes) under Grant 2021-0-00118; and in part by MSIT, South Korea, through the Information Technology Research Center (ITRC) Support Program supervised by IITP under Grant IITP-2021-0-01835.

ABSTRACT Blockchain technology and its applications are gaining popularity day by day. It is a ground-breaking technology that allows users to communicate without the need of a trusted middleman. A smart contract (self-executable code) is deployed on the blockchain and auto executes due to a triggering condition. In a no-trust contracting environment, smart contracts can establish trust among parties. Terms and conditions embedded in smart contracts will be imposed immediately when specified criteria have been fulfilled. Due to this, the malicious assailants have a special interest in smart contracts. Blockchains are immutable means if some transaction is deployed or recorded on the blockchain, it becomes unalterable. Thus, smart contracts must be analyzed to ensure zero security vulnerabilities or flaws before deploying the same on the blockchain because a single vulnerability can lead to the loss of millions. For analyzing the security vulnerabilities of smart contracts, various analysis tools have been developed to create safe and secure smart contracts. This paper presents a systematic review on Ethereum smart contracts analysis tools. Initially, these tools are categorized into static and dynamic analysis tools. Thereafter, different sources code analysis techniques are studied such as taint analysis, symbolic execution, and fuzzing techniques. In total, 86 security analysis tools developed for Ethereum blockchain smart contract are analyzed regardless of tool type and analysis approach. Finally, the paper highlights some challenges and future recommendations in the field of Ethereum smart contracts.

INDEX TERMS Ethereum, smart contract, blockchain, cryptocurrency, decentralized, dapp, vulnerabilities, security, analysis tool.

I. INTRODUCTION

Blockchain technology [1] gained the interest of the research community in the year 2008 when a white paper was published by Satoshi Nakamoto [2] on a double-spending problem in peer-to-peer decentralized network [3], [6]. Now the popularity of blockchain technology is rapidly increasing day by day, such that countries and giant financial institutions are planning to deploy their operational processes on the same technology [4], [5]. In blockchain technology transactions, the trusted third parties are removed with the help of a consensus mechanism. Smart contracts work on the application layer of blockchain. Blockchain technology became popular after Satoshi Nakamoto's white paper on the double-spending problem in a peer-to-peer network. In blockchain technology transactions, the trusted third parties are removed with the help of a consensus mechanism. Today, Ethereum is the most widely used blockchain platform. Ethereum is Turing complete to code smart contracts with developers' constraints.

A smart contract [8], [9] is a contractual agreement embedded in a self-enforceable piece of code. The parties in the agreement agree to interact with each other based on some predefined constraints, such that whenever a condition is met, the predefined operations will execute automatically. Smart Contracts provide higher transparency without the need for

The associate editor coordinating the review of this manuscript and approving it for publication was Luca Cassano.

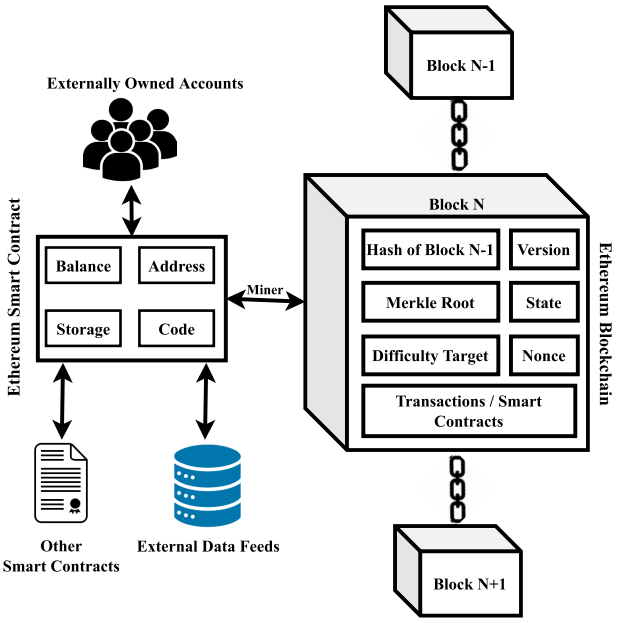

FIGURE 1. Ethereum blockchain based smart contract.

trusted third parties. Figure1 depicts the structure [12], [14] of Ethereum smart contract.

The smart contract has many use cases in various real-life areas. Following are some of the use cases:

- **Financial Contracts:** The governing rules of a financial product or service can be coded in the form of a smart contract to facilitate claims settlements and automated financial transactions. The DeFi (Decentralized Finance) services or applications can address a large market of financial transactions without a bank. The DeFi applications can be more advantageous than traditional ones regarding round-the-clock borderless availability.
- **Prediction Markets:** The growth in the prediction market is not according to time due to the involved risk in the same. Smart contracts can revolutionize the field of the prediction market. There are several gaming fields where the players have been betting for a long time without trusting the third party. So, the concept of the Ethereum smart contract can be utilized in the prediction of the auction, election, and any betting game.
- **Digital Identity:** In traditional systems, identity management and trust management are facilitated by Public Key Infrastructure or PKI. The certificate-based PKIs have a problem with the certificate tree isolation. The Ethereum smart contracts can manage digital identities and build trust.
- **Supply Chain Management:** The Ethereum smart contracts can reduce the complexity in the supply chain by using the automatic verification process. The traditional supply chain system lacks transparency and traceability, which blockchain-based smart contracts can effectively improve.
- **Health Care Industry:** The smart contract can be applied in various application areas in the healthcare industry like health insurance, medical research, patient data management, and drugs supply chain management. Smart contracts can help effectively manage patients' medical history data management.
- **Tokenization ICO/DAICO:** The ERC-20 is one of the essential Ethereum smart contract standards. The ERC-20 is a set of rules to keep track of all types of fungible Ethereum tokens. ERC -20 is short of Ethereum Request for Comments 20. It employs an application programming interface within smart contracts. ERC-721 is one other type of token which is non-fungible. The ERC-20 token represents a single entity, whereas ERC-721 represents a set of resources.

But smart contracts [7], [10], [11] are vulnerable to attacks due to security flaws present in there due to several reasons like features of blockchain, coding issues, etc. As smart contracts store cryptocurrencies as their balances, attackers can take benefit of these security vulnerabilities [35]–[38], which can result in enormous losses. For analyzing smart contracts, several security analysis tools [43], [47] have been developed. Our survey will focus only on analysis tools associated with the Ethereum blockchain smart contract. We present a detailed review of 86 analysis tools for the Ethereum blockchain-based smart contract, covering all the analysis tools present in the literature or on the web, irrespective of their type and analysis approach.

A. RELATED WORK

Many review articles have been published by researchers in this domain with different-different viewpoints. Harz et.al. [15] examined ten verification tools along with their respective languages and verification methods. Angelo et. al. [16] surveyed 27 smart contract analysis tools with different-different points of view like open-source availability, development, working methodology, and security vulnerabilities. Liu and Liu [17], surveyed 53 papers for security vulnerabilities and correctness aspects. They discussed 18 tools in different-different categories like semantic analysis, behavioral analysis, formal verification, etc. Tang *et al.* [44] surveyed 15 analysis tools and their related vulnerabilities. Ante [18] studied the smart contracts concerning citation statistics distribution of keywords of several smart contract platforms and discussed very few analysis tools like Oyente and SmartCheck. Almakhour *et al.* [19] surveyed smart contract analysis tools by categorizing them into verification tools and vulnerability analysis tools for Ethereum blockchain smart contracts. They discussed 25 tools in two categories: formal verification for correctness and Vulnerability detection for security assurance. T. Durieux et. al. [55] performed a pragmatic survey of 9 automatic analysis tools on 47587 Ethereum smart contracts and found that 97% of contracts are vulnerable. Ghaleb et. al. [48] focused only on static analysis tools and proposed a technique named

SolidiFI for evaluating the performance of static analysis tools. Tolmach et. al. [49] studied various verification tools by considering the formal modeling and verification techniques. D. He et. al. [50] studied security vulnerabilities related to Ethereum smart contract and their defense mechanism and some of the security audit methods. They discussed only three analysis tools: Oyente, Porosity, and Mythrill. Grishchenko et. al. [51] surveyed 11 security and verification tools but focused their discussion on the static analysis tool named EtherTrust and formal verification tools. Anna Veca et. al. [52] surveyed 26 analysis tools for Ethereum smart contracts concerning smart contract testing and code analysis. Pinna et.al. [53] presented n pragmatic study on the specific type of Ethereum smart contract (with the topmost number of transactions means financial smart contract) deployed on Ethereum blockchain but covers a little about analysis tools. Bin Hu et. al. [54] surveyed 39 analysis tools concerning methodology, input, and availability of source code. All the above surveys discussed analysis tools related to specific vulnerabilities or specific fields like verification tools. None of the above surveys covers all the Ethereum smart contract analysis tools associated with the analysis of the Ethereum smart contract. This paper presents 86 analysis tools for the Ethereum blockchain-based smart contract to cover this research gap.

B. MOTIVATION

In a no-trust contracting environment, smart contracts can establish trust among parties. Terms and conditions embedded in smart contracts will be imposed immediately when specified criteria have been fulfilled. So, the smart contract, which is just a piece of code, executes the terms and conditions without the need of any third person. Thus, smart contracts must be analyzed to ensure zero security vulnerabilities or flaws before deploying them on the blockchain because a single vulnerability can lead to terrific losses [35]. Thus, it becomes necessary to analyze the security vulnerabilities of smart contracts to develop safe and secure smart contracts.

The existing review articles have discussed only a limited set of Ethereum smart contract analysis tools. Even most of the existing review articles are limited to specific types of tools. Hu et. al. [54] discussed 39 tools, which was the highest among all review articles. Therefore, this paper presents a detailed systematic survey of smart contract analysis tools for the Ethereum blockchain. The overall objective is to discuss maximum analysis tools to highlight some challenges and future recommendations in Ethereum smart contracts.

C. RESEARCH QUESTIONS

Smart contracts [39]–[42] can be developed on various blockchain platforms, which have their features and challenges. Still, Ethereum is mainly used as a very prominent smart contract development platform, so we focus only on analysis tools for smart contracts related to Ethereum blockchain and systematized these analyses tools regardless of their type or analysis approach. The literature lacks an organized survey of Ethereum blockchain-based analysis tools covering all the tools. Systematic study methods of Kitchenham *et al.* [12] and Peterson *et al.* [13] are used for defining the following research questions:

- **Research Question 1:** What are the static analysis tools available for Ethereum blockchain smart contracts?
- **Research Question 2:** Which dynamic analysis tools are available for the Ethereum blockchain smart contract?
- **Research Question 3:** For Ethereum blockchain smart contracts, what kind of analysis approaches are employed by static/dynamic analysis tools?
- **Research Question 4:** What are the five most common vulnerabilities detected by analysis tools?

D. INCLUSION AND EXCLUSION OF ARTICLES

To address the research questions, we identified 670 research articles from Web of Science (WoS) that are published between 2016 to 2021. This search was performed in peer-reviewed scientific research databases like Springer, ACM, IEEE, Elsevier, and Willey. Out of these research articles, 525 articles are excluded based on exclusion criteria, and 132 articles are included based on inclusion criteria. Duplicated, survey, and review articles are excluded from the selected articles. Also, articles in which only tools comparisons are presented are also avoided. All the phases of the methodology for inclusion and exclusion of research articles are depicted in Figure 2.

E. CONTRIBUTIONS

This paper contributes a systematic review of analysis tools for Ethereum blockchain smart contracts from 2016 to December 2021. This work provides a thorough understanding of the analysis tools for Ethereum smart contracts. The main contributions of this paper are as follows:

1) A systematic review of Ethereum smart contracts analysis tools is presented.
2) The analysis tools are categorized into static and dynamic analysis categories. These categories are further divided into subcategories based on the input type of the tools, such as solidity code, EVM byte code, or both.
3) Different sources code analysis techniques are studied, such as taint analysis, symbolic execution, and fuzzing techniques.
4) In total, 86 security analysis tools in Ethereum blockchain smart contract are analyzed regardless of tool type and analysis approach.
5) Finally, the paper highlights some challenges and future recommendations in the field of Ethereum smart contracts.

F. PAPER OUTLINE

The remaining structure of the article is as follows: Section II briefly describes some famous vulnerabilities associated with

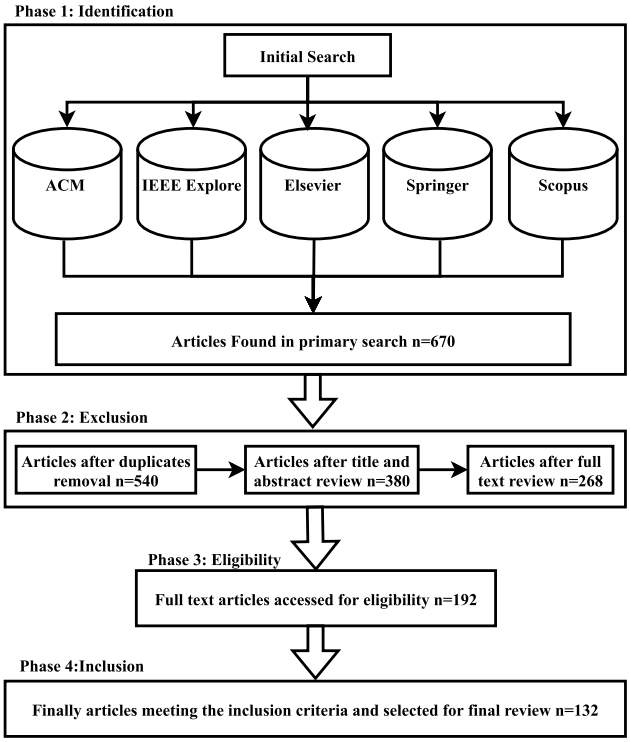

FIGURE 2. Identification, exclusion, eligibility, and inclusion methodology.

Ethereum blockchain smart contracts, Section III illustrates a detailed description of all Ethereum smart contract analysis tools, Section IV presents a comparison with related work, limitations of the present survey, and possible future research directions. Finally, Section V concludes this study by exhibiting an outline of the contributions.

II. SECURITY VULNERABILITIES IN ETHEREUM BLOCKCHAIN SMART CONTRACT

Several researchers presented many types of vulnerabilities [46] associated with Ethereum blockchain smart contract [31]–[34] in literature. Li *et al.* [29] surveyed 20 different-different vulnerabilities, Zhu *et al.* [30] studied 11 types of vulnerabilities, Luu *et al.* [9] presented security vulnerabilities in their survey, Atzei *et al.* [142] studied some specific security vulnerabilities, Tang *et al.* [44] presented 15 security vulnerabilities, Huashan Chen et. al [45] presented 40 vulnerabilities under several root causes. Following are some of the most discussed Ethereum smart contract vulnerabilities

- **Timestamp Dependency [9]:** It occurs when the block timestamp is used to trigger a condition to initiate the execution of a critical operation. If the block timestamp is used to create randomness, it can be compromised by the malicious attacker.
- **Reentrancy [23]–[25]:** It is one of the most famous vulnerabilities related to the Ethereum smart contract. It was first reported in 2016 from the renowned DAO attack [26], which caused a loss of 60 million US dollars. It occurs due to repeatedly calling of a function of the caller contract by the callee contract before the function completed its execution. Due to this, the state variables of the function are not updated after each function call and create a very serious issue.
- **Transaction Ordering Dependency [16]:** It occurs due to concurrent order of transaction execution. The miners decide the transaction execution sequence. A malicious miner may select or not select a specific transaction to mine, which ultimately results in wrong execution results if the transactions are dependent on each other.
- **tx.origin [8]:** The "tx.origin" is used for authorization purposes. Still, the attacker can utilize the same for a phishing attack. "msg.sender" should be used in place of "tx.origin" for authentication purposes.
- **Block-hashBlock Number [8], [20]:** It also occurs when the block has, or block number is used to generate randomness by generating random numbers. But a miner can act maliciously to manipulate or modify the same for its benefit.
- **Gas Related Issues [14], [26], [27]:** There can be several gas-related issues like sending a transaction with insufficient gas, useless code in the smart contract, or gas costly loops present in the contract. Gas is used as a transnational fee to execute instructions of the smart contract like each type of operation requires a different gas, which is charged in Ether (Wei-smallest unit of Ether).
- **Delegate Call [28]:** It was first reported in one of the other famous attacks on the Ethereum smart contract Parity wallet. It occurs because of using EVM opcodes maliciously by the callee contract to update the state variables of the caller's contract.
- **Arithmetic UnderflowOverflow [21]:** It occurs due to solidity data type range, which means values of arithmetic operation cross the range limit of data type on upside or downside and give a chance to the attacker to manipulate the values of state variables. It was first reported in attacks on BEC tokens.
- **Freezing Ether [44]:** It was also reported the first attack against the Parity wallet. It occurs because the user of the contract cannot spend money due to the dependency on other contract's money spending function, and the function doesn't allow to spend the money.
- **Unchecked Call [45]:** It occurs due to improper exception handling in the solidity code. When the return value of execution is not adequately checked and proper measures are not taken, the malicious user can benefit from that.
- **Self Destruct [29]:** The "self destruct" is a method the owner uses to kill its contract to delete its byte code and free the storage. But the attacker can kill a contract if there is poor authentication in the contract. It was first reported in the Parity wallet bug.
- **Access Control [30]:** It is the case when inadequate authorization or authentication is used while coding the

smart contract. An attacker can maliciously use the same to access the critical functions.

- **Denial of Service [14], [22]:** It occurs due to the malicious intention of the user to disrupt the execution of another user's caller contract by reverting the call every time.

III. ANALYSIS TOOLS FOR ETHEREUM SMART CONTRACT

Smart contracts must execute according to the user's need or owner of the smart contract. Security vulnerabilities [29], [30], [44], [45] or bugs may not allow the smart contract to perform its operation for which it was coded and be the reason for tremendous losses. Analysis tools [66], [72], [112], [124] are necessary to check and analyze the smart contract for any security flaw. Because the immutability [99] nature of the blockchain does not allow any type of alteration in the code of smart contract after deployment of the same on the blockchain. This Section presents the categorization of Ethereum smart contract analysis tools into two main categories.

Further, these two main categories are divided into two subcategories based on the initial input on the tool for analysis purposes. Some of the tools take both Solidity and Byte code as input. One tool named FSolidM [95] generates solidity code by taking input some formal specifications. Figure 3 shows the categorization of Analysis tools for smart contracts associated with the Ethereum blockchain.

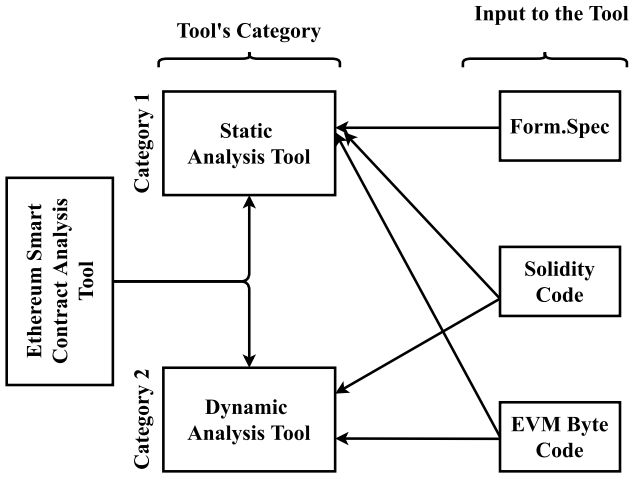

FIGURE 3. Categorization of analysis tools based on type of analysis and type of input to the tool.

Several analysis tools have been invented since 2016. We have considered the tools invented up to November 2021. Figure 4 shows the year-wise evolution of analysis tools for Ethereum blockchain-based smart contracts. The majority of the tools developed to date are static analysis tools. Authors justified in their articles the benefits of their tool's analysis approach.

Figure 5 depicts different-different tools in each category, and figure 6 shows the share of each category of the tool among the total analysis tools invented for Ethereum blockchain-based smart contract. Data used in the figure 6 is collected in our survey.

Now we will give a detailed overview of each tool under each category. First, we will describe static analysis tools under subcategory input to the tool is Solidity code

A. CATEGORY 1: STATIC ANALYSIS TOOLS

1) INPUT TO THE TOOL: SOLIDITY CODE

- **ContractWard [76]:** It is a static analysis tool that takes solidity as input and was invented in 2019. It is an automatic vulnerability finding tool. Wei Wang etàl proposed this system for detecting vulnerabilities at a large level with machine learning algorithms. ContractWard detects six vulnerabilities: Timestamp Dependency, Re-entrancy, Arithmetic Overflow and Underflow, Call-stack Depth, and Transaction-Ordering Dependence. It depends on the Oyente tool for label generation for each contract with six labels. ContractWard works in the following six steps: Step 1: Collection of smart contracts from Ethereum's official website. Step 2: Transformation of source code to opcode for simplification. Step 3: 1619 bigram features are extracted from the simplified opcodes of step 2. Then each smart contract is labeled with six labels corresponding to each type of vulnerability from C1 to C6. Step 4: For multi-label classification, the OvR algorithm is employed. Step 5: Classification and balancing are done in this step. Step 6: Balanced training sets are used for creating detection models.
- **Echidna [80]:** It is a publicly available open-source static analysis tool that takes solidity or viper code as input and was invented in 2020. It is an Ethereum smart contract fuzzer developed in Haskell, which supports three properties such as user-defined properties, assertion checking, and gas use estimation. Echidna works in two steps: 1. Pre-processing: In this step, it leverages Slither to analyze smart contracts. 2. Fuzzing Campaign: In this step, random transactions are generated, and property violations are detected. Echidna is very easy to use and supports most contract development frameworks. It is very fast to produce results very quickly.
- **Eth2Vec [84]:** It is a command-line-based static analysis tool, invented in 2021. It employs the machine learning approach for analyzing smart contracts to learn the features of vulnerable EVM byte code. It creates a model for feature extraction by training the tool using training data. Then, matching the similarity in the code of EVM and target EVM detects the vulnerabilities.
- **Ethainter [85]:** It is a static analysis tool invented in the year 2020. It analyses information flow with data sanitization in Ethereum smart contracts. It enhances the tainted information flow by tainting the guard conditions. Ethainter efficiently detects Seif Destruct, Delegate Cal, Unchecked, variable tainting type vulnerabilities.
- **EthVer [140]:** It is a static analysis tool invented in the year 2020. It performs automatic formal verification of

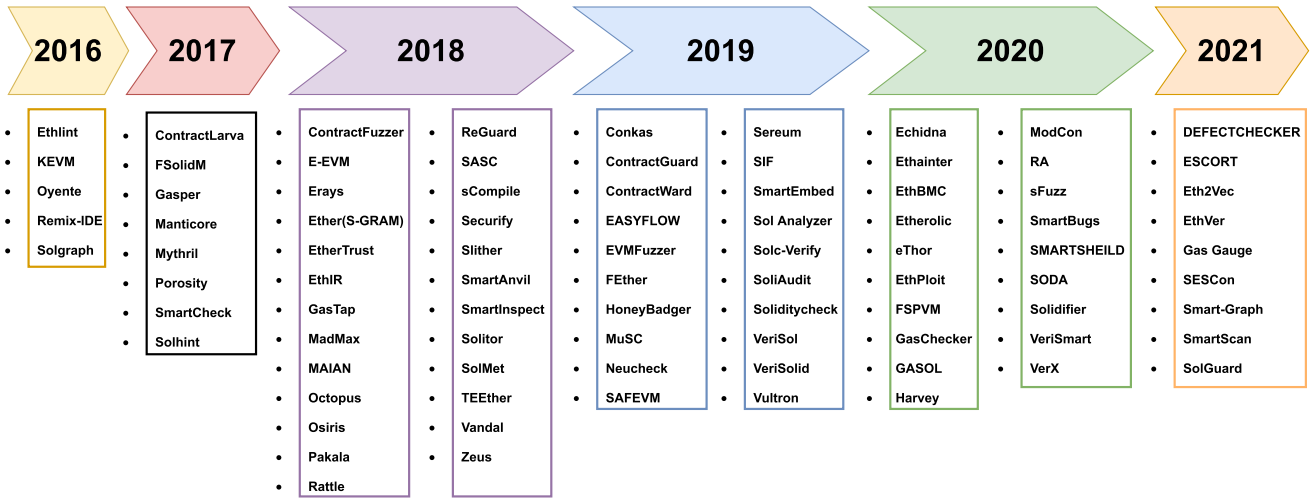

FIGURE 4. Year wise evolution of analysis tools for ethereum blockchain based smart contract.

smart contracts. Then tool translates the smart contract into formal models known as the Markov decision process and then verifies the same using a formal verification tool known as PRISM model checker.

- **FEther [94]:** It is a publicly available static analysis tool implemented in Coq and invented in 2019. It takes an input of solidity code and analyzes the same using a combination of symbolic execution and high order logic theorem proving. FEther's functional correctness is verified in Coq. FEther's processing is divided into three parts: The first one is a Parser, the second one is an ISA based on Lolisa semantics, and the third one is a validation checking mechanism.
- **FSPVM [96]:** It is a static analysis tool implemented in Coq and invented in 2020. It supports ERC-20 token standard. FSPVM symbolically analyses the Ethereum smart contract solidity code and checks for security vulnerabilities by employing Hoare style logic in Coq. FSPVM combines the virtual machine platforms with static security issues checking, based on an extension of Curry-Howard isomorphism.
- **GasGauge [97]:** It is a static analysis tool developed in the year 2021. It employs fuzz testing to detect vulnerabilities. It efficiently detects out of gas denial of service vulnerability in Ethereum blockchain-based smart contracts. The tool is divided into three phases: the Detection phase, Identification phase, and Correction phase. All the stages of the tool can work alone or together to analyze Ethereum smart contracts.
- **Gastap [101]:** It is a static analysis tool developed in 2018. This tool takes input for analyzing the EVM byte code or disassembled EVM byte code or solidity code. It deduces the gas bounds requirements for its functions and finally compares the deduced gas requirement with the genuine gas limit paid by the user. The difference between the deduced gas limit and the actual gas limit paid will show the gas-related vulnerabilities.
- **MuSc [109]:** It is a publicly available open-source static mutation testing tool implemented in Java and invented in 2019. MuSc generates mutants of smart contracts at the abstract syntax tree level. These AST-generated mutants are converted to source code for compilation, execution, and testing. It also supports the user-defined test net.
- **NeuCheck [110]:** Ning Lu et. al. proposed this static analysis tool in 2019, which takes solidity as input. NeuCheck is developed in Java. To avoid missing semantics and the transformation of solidity code to an intermediate representation, NeuCheck introduced the syntax tree in the syntactical analyzer. For this transformation, NeuCheck takes use of a solidity parser which is developed in ANTLR. NeuCheck works in three steps: 1. Parse the source code to an intermediate representation. 2. Then, the second step for analyzing the syntax tree utilizes the open-source XML library working. 3. The third and last step notifies users about the location of security issues in the smart contract.
- **Pakala [114]:** It is a publicly available open-source static analysis tool implemented in Python and invented in 2018. It is a symbolic execution tool for Ethereum Virtual Machine byte code. It used Z3 as an abstraction layer and added a SHA3 layer on top of it. It employs two steps to detect vulnerabilities. In the first step, it executes byte code to find outcomes, and in the second step, it analyses the outcome to find something terrible corresponding to a vulnerability.
- **Remix IDE [115]:** It is an open-access tool, which provides an easy way to write and analyze smart contracts in solidity code. It is a JavaScript implementation of Ethereum Virtual Machine with a browser-based user's interface, invented in 2016 and written in JavaScript. It can be used either in the web version or desktop version. Its source code is available in GitHub Repository. Presently there are 21 analysis modules under

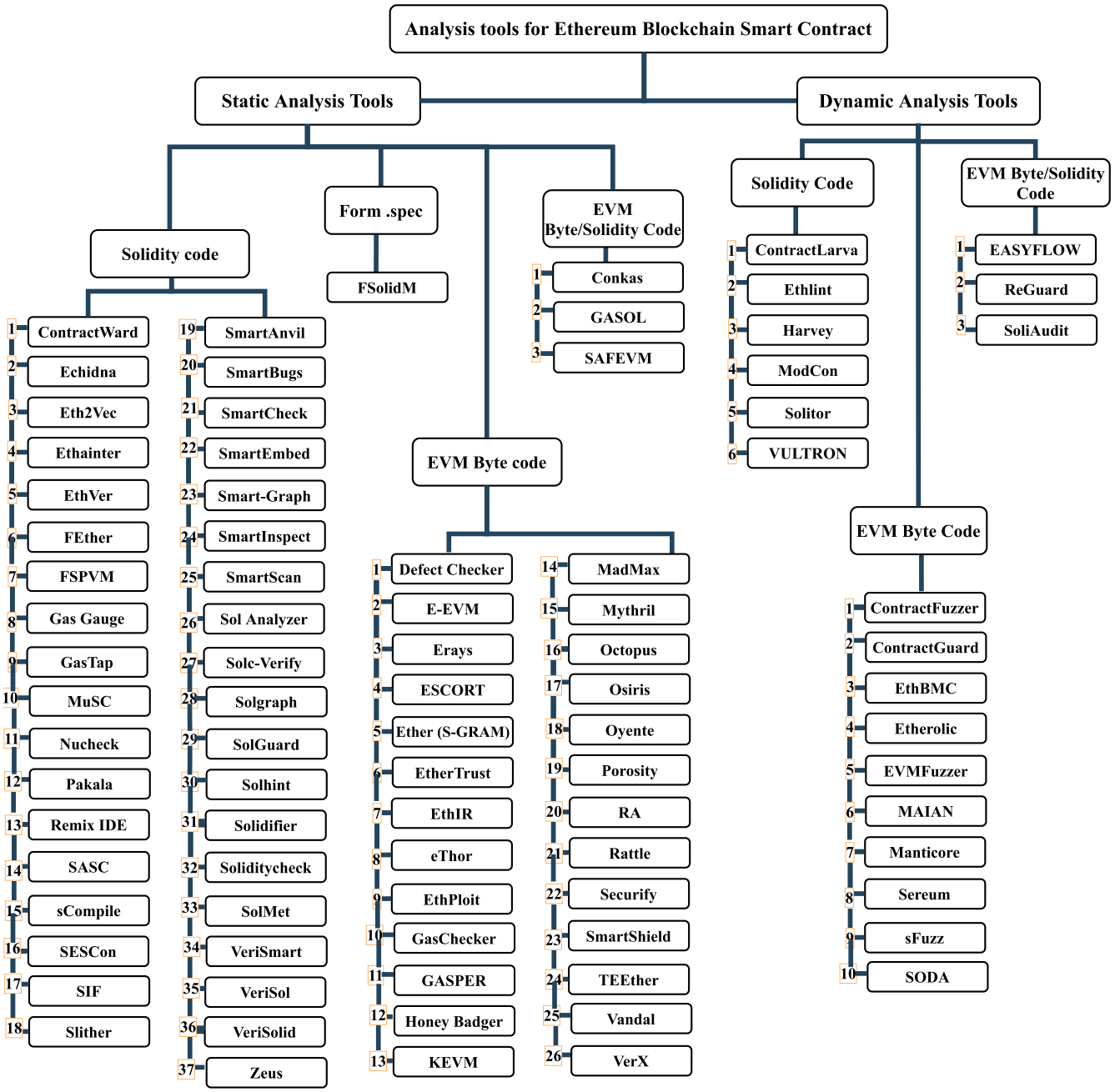

FIGURE 5. Categorywise ethereum smart contract analysis tools.

four categories are listed in Remix IDE v0.10.1, which are Security, Gas & Economy, ERC & Miscellaneous. Remixanalyser is a library that works beneath of Remix-IDE Solidity Static Analysis plugin.

- **SASC [126]:** It is a static analysis tool invented in 2018 and developed in C (83%), Python (15.7%), Mark file(0.5%). It comprises two main functions a) Invocation Relationship Analysis b)Logic Risk Expansion and Location. This tool detects timestamp dependency, tx.origin, divide by zero vulnerabilities.
- **sCompile [127]:** It is a static analysis tool invented in 2018 and developed in Python. This tool is based on automatically identifying monetary transaction's related critical program paths and concentrating on those that possibly contravene essential properties.
- **SESCon [130]:** It is a static analysis tool invented in the year 2021. It employs taint analysis to analyze the Ethereum smart contracts. The tool has three main modules: XPath module, Vulnerable patterns modules, and Taint module. The XPath module takes an input of the solidity code and converts the same to an abstract syntax tree. The Vulnerable pattern module creates vulnerable patterns from vulnerable smart contracts. The Taint module takes input

from the above modules and products the vulnerability report.

- **SIF [132]:** It is a publicly available open-source static analysis tool implemented in C++ and invented in 2019. Apart from analysis, SIF also supports query instrumentation and code generation at the abstract syntax tree level. It takes an input of abstract syntax tree generated by solidity compiler and user-defined query for code instrumentation and finally generates instrumented solidity code. It detects arithmetic Overflow or Underflow.
- **Slither [133]:** It is a publicly available open-source static analysis framework invented in 2018 and developed in Python. It takes solidity code as input to analyze. It uses an intermediate representation known as SlithIR. The Slither employs data flow analysis and taints tracking approaches to detect vulnerabilities. It can be used for automated vulnerabilities detection, automated optimization detection, code understanding, and assisted code review. The open-source version of this tool detects approximately 20 bugs like shadowing, uninitialized variables, re-entrancy, suicidal contracts, locked ether, or arbitrary sending of ether.
- **SmartAnvil [134]:** It is a publicly available open-source platform invented in 2018 and developed in Smalltalk. It is constructed around various modules to cover the multiple aspects of smart contract analysis. SmartAnvil platform contains three components' tools: 1) SmaCC-Solidity: a parser used to represent or support solidity smart contract's static code. 2) SmartInspect: It is used to inspect the internal state of the Solidity smart contract. 3) Ukulele: It is a query language that helps to fetch required data from the blockchain.
- **SMARTBUGS [135]:** It is a publicly available open-source static analysis framework implemented in Python and invented in 2020. It supports ten tools for analyzing the smart contract. This tool comprises 5 components: command-line interpreter, tool's configuration, docker's image of tools, dataset, and SMARTBUGS runner. Apart from the command line interface, the SMARTBUGS also has a web interface to interact.
- **SmartCheck [136]:** It is a publicly available open-source static analysis tool invented in 2017 and developed in Java. SmartCheck employs a lexical and syntactical analysis approach to analyze the smart contract. An XML parse tree is generated as an intermediate representation using ANTLR (a parser generator) and a custom Solidity grammar. XPath queries are used to process intermediate representation for detecting vulnerabilities patterns. It detects approximately 20 types of vulnerabilities like implicit visibility level, compiler version not fixed, arithmetic division, style guide violation, etc.
- **SmartEmbed [137]:** It is a publicly available open-source static analysis tool implemented in JavaScript and invented in 2019. It is a web-based service tool that detects repetitive contracts. The core component of SmartEmbed is the similarity checker, which takes an input of bug embedding matrix, code embedding matrix, and embedding vector and finally outputs the bug report and clone report.
- **Smart-Graph [138]:** It is a static analysis tool invented in the year 2021. It takes an input of the solidity source code and creates a graphical visualization. The tool has a web-based graphical user interface that is very easy to use and can be accessed using any browser. At the web GUI the tool takes the smart contract address and generates the graphical diagram of the same.
- **SmartInspect [56]:** It is a static analysis tool implemented in Pharo and invented in 2018. It is a mirror-based reflection system. It inspects the known smart contracts at the source code level to analyze the instructiveness and distribution. The reflective approach of SmartInspect permits the user to view the content of any contract without needing to redeploy it.
- **SmartScan [57]:** It is a static/dynamic analysis tool invented in the year 2021. For detecting the denial of service or DoS vulnerability, it combines static and dynamic analysis. SmartScan works in two steps: First, it statically analyses the smart contract to detect denial of service vulnerability-related patterns. Then, the second step uses dynamic analysis to confirm their exploitability.
- **SolAnalyzer [60]:** It is an open-source static analysis framework invented in 2019 and developed in GO. It allows fully automatic analysis of Ethereum smart contracts and reduces false positives. SolAnalyzer detects vulnerabilities in three phases Phase 1). Instrumentation with assertion via Solidity Instrumentation Framework (SIF). Phase 2). Input generation for instrumented smart contracts. Phase 3). Execution in the Ethereum virtual machine and analysis of instrumented contracts. For checking the efficiency and effectiveness

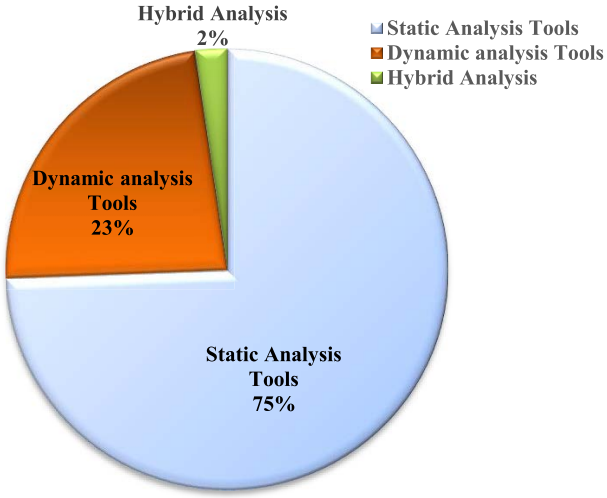

FIGURE 6. Category-wise share of ethereum smart contract analysis tools.

of SolAnalyzer, there is another component in this framework named MuContract, which creates several faulty versions of original smart contracts by seeding artificial vulnerabilities.

- **Solc-Verify [61]:** It is a publicly available open-source static analysis tool invented in 2019 and developed in C++ and Solidity. It employs formal verification methods to analyze Ethereum blockchain-based smart contracts. Very easy to integrate with other developer tools. It verifies Solidity smart contracts with a modular software verification approach. Solc-Verify can be employed as an add-on to the open-source Solidity compiler. Solc-Verify detects re-entrancy and integer overflow/underflow vulnerabilities.
- **Solgraph [116]:** It is a publicly available open-source static analysis command-line tool invented in 2016 and developed in JavaScript. Solgraph visualizes the control flow of the function in a Solidity smart contract. It generates a DOT graph to depict the control flow of the functions. DOT is a graphical description language used to visualize functions' control flow and show the relations between the objects. Solgraph uses this DOT graph to identify potential security vulnerabilities in solidity smart contracts.
- **SolGuard [141]:** It is a static analysis tool invented in the year 2021. It was developed by extending the existing static analysis tool named Solhint. It mainly detects external call-related vulnerabilities by checking the order of the state variables, address type parameters, delegate call invocations, and patterns related to denial of service in the smart contracts related to decentralized robotic applications.
- **Solhint [119]:** It is a publicly available open-source static analysis command-line tool invented in 2017 and developed in Java. Solhint uses an antlr4-based implementation of the Solidity parser that enables efficient parsing and validation performance. The tool has flexible configuration options like using a predefined set of rules, a default customized rule set, and code-level configuration rule management. Solent uses three major commands 1) **/*.sol: by this command, it receives a list of file patterns to analyze. 2)stdin: It provides validating source code to standard input. 3) init config: It creates a basic configuration file, which can also be customized if needed.
- **Solidifier [62]:** It is a static analysis tool developed in the year 2020. This tool takes input for analysis of the solidity code. Rather than finding specific behavioral patterns, it detects errors and bad states that do not conform to the developer's intent or detects the falsifying behavioral properties, which the developers can correct.
- **Soliditycheck [63]:** It is a publicly available static analysis tool invented in 2019 and developed in C++. It uses regular expressions to locate security problems in smart contracts. Soliditycheck's main processing is divided into four steps: 1)Formatting the codes. 2) Keywords filtering from the formatted code. 3) Detection and prevention. 4) Detection report and preventive contract. At the end of the fourth step, it shows a detection report of 18 types of security problems except for re-entrancy and integer overflow problem and outputs contract that prevent problems after inserting code.
- **SolMet [65]:** It is a publicly available open-source static analysis tool invented in 2018 and developed in Java. It is a static source code metrics generator to measure smart contracts' size and complexity attributes. Parsing the solidity source code uses a generated parser which the modified version of antlr4 grammar [https://github.com/solidityj/solidity-antlr4]. SolMet proposes the following source code metrics for a smart contract: SLOC, LLOC, CLOCNF, WMC, and NL.
- **VeriSmart [68]:** It is a publicly available open-source static analysis tool implemented in OCaml and invented in 2020. It is an automatized and scalable analysis tool based on a domain-specific smart contract verification language. It starts analysis from basic path construction, the generation of verification conditions, then collecting unproven paths, then performing domain-specific refinement, then processing, and at last efficient validity/invalidity checking. It detects all CVE related vulnerabilities.
- **VeriSol [69]:** It is a publicly available open-source static analysis tool invented in 2019 and developed in C#. It is a general-purpose solidity verifier used to check the assertion in a solidity smart contract. VeriSol converts the Solidity program's semantics to Boogie, a low-level intermediate verification language. It uses the CORRAL [24], a bounded model checking tool that helps Boogie generate witnesses to assertion violations.
- **VeriSolid [70]:** It is a static analysis tool implemented in JavaScript and invented in 2019. It is a formal verification framework that allows the creation of solid source code from validated prototypes which ultimately allows the correct design development of the smart contract. It is constructed on top of another static analysis tool named FSolidM.
- **Zeus [73]:** It is a company tool, developed in 2018 by IBM Research India for static analysis of the solidity smart contracts. It is a symbolic model checking framework consisting of three components: policy builder, source code translator, and verifier. Solidity smart contract and policies against which the smart contract is to be verified are taken as input. It then inserts policies predicates as assertions at correct program points. Then this policy asserted code is converted to LLVM bytecode. Then at the end the verifier checks for policy violations.

Table 1 presents a comparative analysis of static analysis tools with solidity code as input. The comparative analysis is based on some criteria like either the tool is a company tool

or academic tool, source code of the tool is available or not (means source code is freely available to access or evaluate the tool on Github or some other web location), the tool has either command-line interface or web interface, the year of advent and the implementation language or development language of the tool.

2) INPUT TO THE TOOL: EVM BYTE CODE

- **DEFECTCHECKER [78]:** It is a static analysis tool invented in the year 2021 and developed in Java. It is based on symbolic execution and has four processing sections: Inputter, CFG Builder, Feature Detector, and Defect Identifier. It takes as input the byte code and then extracts the opcodes from that. Then all the opcodes are categorized into different-different categories for symbolic execution. Then a control flow graph is constructed to detect the defects in the smart contract.
- **E-EVM [81]:** It is a publicly available open-source static analysis tool invented in 2018 and developed in Python. It visualizes the emulated execution of the smart contract on Ethereum Virtual Machine. It works on the byte code of the smart contract by displaying control flow, opcode, and stack for each step of the contract's program execution. The front end of E-EVM is written in JavaScript, and the back end is written in Python.
- **Erays [82]:** It is a publicly available open-source static analysis tool invented in 2018 and developed in Python. It is a reverse engineering tool that analyses EVM byte code of Ethereum blockchain smart contracts. It generates a high-level pseudo-code for the EVM byte code. Erays works in eight steps starting from disassembly from hex string to EVM instructions, then basic blocks, then control flow graph is recovered from these basic blocks, then EVM's stack-based instruction are lifted to registered based instruction. Then it performs data flow optimizations following the aggregation to an intermediate representation. Then control flow structure is recovered using structural analysis algorithms. Then validation is performed to transform the contract into more readable expressions. Erays has limitations like it cannot capture operation on complex types.
- **ESCORT [83]:** It is a static analysis tool invented in the year 2021. It employs a Deep Neural Network (DNN)-based approach to analyze Ethereum blockchain smart contracts vulnerability detection framework. It supports lightweight transfer learning on invisible security issues, thus is extensible and oversimplified. The ESCORT is composed of two components: (i) the First component extracts the features and semantics of the Ethereum smart contract (ii) The second component takes an input of features from the first component and consists of Multiple branch structures. Each branch from this multiple branch structure works on a specific security vulnerability.
- **Ether (S-GRAM) [87]:** It is a semantic-aware security-aware framework. It was developed in Python based on the S-Gram artifact in 2018. To detect vulnerabilities, it works in two phases: the model construction phase and the security auditing phase. It uses a combination of N-gram language modeling and lightweight static semantic labeling to learn statistical regularities of contract tokens and finally capture high-level semantics to predict potential vulnerabilities.
- **EtherTrust [89]:** It is a publicly available open-source automated static analysis tool invented in 2018 and developed in JavaScript. Its analysis is based on the horn clause. For discharging proof obligations, it relies on Z3 theorem prover. It shows a formal guarantee and supports the analysis of EVM byte code. It detects Single entrance and independence from the transaction environment.
- **EthIR [90]:** It is a publicly available open-source static analysis tool invented in 2018 and developed in Python. It is an Ethereum byte code analyzer and depends on Oyente to generate the control flow graph. It converts the control flow graph to a rule-based intermediate representation. It then uses SACO, a high-level static analyzer, to analyze its intermediate rule-based representation of EVM byte code.
- **eThor [91]:** It is an EVM byte code static analyzer invented in JavaScript in 2020. The author employs the HoRSt (specification and implementation framework for static analysis) to implement eThor. eThor is built on the top of the reachability analysis realized by horn clause resolution, which abstracts the contract's execution behavior to query about the abstracted property over abstract executions instead.
- **EthPloit [92]:** It is a static analysis tool invented in the year 2020. This tool uses a fuzzing approach for exploit generation in smart contracts exploit generators based on fuzzing. The workflow of EthPloit is divided into five parts starting from Static analysis, test-case generation, test case execution, trace analysis, and feedback handling. It generates exploits related to Unchecked Transfer Value, Vulnerable Access Control, Exposed secret, etc.
- **GasChecker [98]:** It is a static analysis tool invented in the year 2020. GasChecker mainly works on gas-related bugs in smart contracts. The tool analyses smart contracts based on ten gas inefficient codes or programming patterns. Symbolic execution is employed as GasChecker's analysis approach to detect gas-related security issues in the Ethereum virtual machine byte code.
- **Gasper [100]:** It is a static analysis tool invented in Python in the year 2017. Gasper analyzes the EVM byte code of the smart contract to identify the gas costly pattern. Gasper identifies 7 gas costly patterns in two categories: unnecessary code-related patterns and gas costly loop-related patterns. So, Gasper is a gas costly pattern checker based on symbolic execution and work on byte code.

TABLE 1. Comparative analysis of static analysis tools (with solidity code as input).

Tool	Source Code	Organization	Academic	Command Line Interface	Web Interface	Year	Platform
ContractWard	✗	✗	✓	✓	✗	2019	-
Echidna	✓	✗	✗	✓	✗	2020	Haskel
Eth2Vec	✗	✗	✓	✓	✗	2021	-
Ethainter	✗	✗	✓	✓	✗	2020	-
EthVer	✗	✗	✓	✓	✗	2021	-
FEther	✓	✗	✗	✓	✗	2019	Coq
FSPVM	✗	✗	✓	✓	✗	2020	Coq
Gas Gauge	✗	✗	✓	✓	✗	2021	-
GasTap	✓	✗	✓	✗	✓	2018	Python
MuSC	✓	✗	✗	✓	✗	2019	JAVA
Neucheck	✗	✗	✓	✓	✗	2019	JAVA
Pakala	✓	✗	✗	✓	✗	2018	Python
Remix-IDE	✓	✗	✗	✗	✓	2016	Java Script
SASC	✗	✗	✓	✓	✗	2018	C (83%), Python (15.7%), Markfile(0.5%)
sCompile	✗	✗	✓	✓	✗	2018	Python
SESCon	✗	✗	✓	✓	✗	2021	-
SIF	✓	✗	✗	✓	✗	2019	C++
Slither	✓	✗	✗	✓	✗	2018	Python
SmartAnvil	✓	✗	✗	✓	✗	2018	-
SmartBugs	✓	✗	✗	✓	✗	2020	Python
SmartCheck	✓	Smart Check	✗	✓	✗	2017	JAVA
SmartEmbed	✓	✗	✗	✗	✓	2019	Java Script
Smart-Graph	✗	✗	✓	✗	✓	2021	-
SmartInspect	✗	✗	✓	✓	✗	2018	Pharo
SmartScan	✗	✗	✓	✗	✓	2021	-
Sol Analyzer	✓	✗	✓	✓	✗	2019	GO
Solc-Verify	✓	✗	✗	✓	✗	2019	C++, Solidity
Solgraph	✓	✗	✗	✓	✗	2016	Java Script
SolGuard	✗	✗	✓	✓	✗	2021	-
Solhint	✓	Protofire	✗	✓	✗	2017	JAVA
Solidifier	✓	✗	✓	✓	✗	2020	-
Soliditycheck	✓	✗	✓	✓	✗	2019	C++
SolMet	✓	✗	✗	✓	✗	2018	JAVA
VeriSmart	✓	Software Analysis Laboratory, Korea University	✓	✓	✗	2020	Ocaml
VeriSol	✓	✗	✗	✓	✗	2019	C#
VeriSolid	✓	✗	✓	✗	✓	2019	Java Script
Zeus	✗	✗	✓	✓	✗	2018	-

- **HoneyBadger [103]:** It is a publicly available open-source static analysis tool invented in 2019 and developed in Python. It performs a systematic analysis of honeypot smart contracts. It employs symbolic execution with a defined heuristic for exposing honeypots by investing their pervasiveness, actions, and influence on the Ethereum blockchain. Its structure contains three types of analysis pipeline named symbolic analysis, cash flow analysis, and honeypot analysis. Each type of analysis uses Z3 SMT solver to check the satisfiability of constraints.
- **KEVM [104]:** It is a publicly available open-source static analysis tool invented in 2016 and developed with a mixture of markdown syntax and k specification language. The KEVM formation is divided into two components states. The first one is the active VM state or virtual machine state for executing transactions and contracts. Another one is the network state which records a log of account information.
- **MadMax [105]:** It is a publicly available open-source static analysis tool invented in 2018. It uses Gigahose IR to perform static analysis. Gigahose IR is a lifter that converts low-level Ethereum virtual machine byte code into high-level IR (intermediate representations). MadMax automatically detects gas-focused vulnerabilities. MadMax employs a combination of two analysis approaches. The first is a control-flow-analysis-based decompiler, and the second is declarative program-structure queries. This approach identifies high-level area-specific concepts.
- **Mythril [120]:** It is a publicly available open-source analysis tool implemented in Python programming language in 2017 by ConsenSys, a software engineering leader in the blockchain space. Ethereum Virtual Machine byte code is given as input for analysis in this tool. Mythril not only analyses Ethereum blockchain-based smart contracts but also works for other blockchain platforms. Mythril uses three approaches for analyzing smart contracts: symbolic execution, SMT solving, and taint analysis. It can also be used in combination with other tools.
- **Octopus [121]:** It is a publicly available open-source static analysis tool implemented in Python and invented in 2017 and sponsored by QuoScient Technologies. It is a vulnerability detection model for WASM and blockchain smart contracts. Octopus can work as a disassembler to translate bytecode into assembly representation. It can generate a control flow graph and call flow graph. It employs symbolic execution to find new paths into the program.
- **Osiris [111]:** It is a publicly available open-source static analysis tool implemented in Python and invented in 2018. For detecting arithmetic vulnerabilities, it employs a combination of two approaches, which are symbolic execution and taint analysis. It can detect three types of integer vulnerabilities arithmetic, truncation, and signedness bugs.
- **Oyente [9]:** It is a publicly available open-source static analysis tool implemented in Python and invented in 2016. It is one of the oldest tools which employs symbolic execution for analyzing smart contracts and statically analyzing the program code path by a path. Its main architectural components are CFG builder, Explorer, Core analyzer, Validator. The Explorer and Validator use Z3 bit-vector solver to eliminate provably infeasible traces from consideration.
- **Porosity [113]:** It is a publicly available open-source static analysis tool implemented in C++ and invented in 2017 by Comae technology. It is a decompiler and vulnerability analysis tool for Ethereum blockchain and generates readable solidity syntax contracts. As per the author, the tool is not maintained for a long time, and he suggested using another tool.
- **RA [117]:** It is a static analysis tool invented in the year 2020. The complete form of the RA is ''Re-entrancy Analyser.'' So, as per the name, it is a re-entrancy attack hunter. It employs a combinational approach of symbolic execution and constraints solver in two ways, i.e., symbolic simulation of re-entrancy vulnerability and then verification of the same. It also supports the analysis of inter-contract behavior. Its architectural design contains three components CFManager, VM, and Verifier. The CFManager conducts the symbolic simulation process of re-entrancy vulnerability. Verifier assisted by VM executes the vulnerability verification process.
- **Rattle [118]:** It is a publicly available open-source static analysis tool implemented in Python and invented in 2018 Trail of Bits. It is a binary static analysis framework and can analyze deployed smart contracts. It recovers the original control flow graph by a flow-sensitive analysis of EVM byte code. Then it converts the control flow graph into SSA (Single Static Assignment) register form. At last, optimizing SSA removes DUPs, SWAPs, PUSHs, and POPs.
- **Securify [128]:** It is a publicly available open-source static analysis tool implemented in Java and invented in 2018. It is a fully automated security analyzer and indicates the behavior of a smart contract associated with a given feature to check it is either safe or not. The input to this tool is EVM byte code and a set of security patterns. Security employs two steps analysis process. The first step performs the symbolic execution of the contracts dependency graph and extracts specific semantic data. The second step checks for compliance and violation pattern for proving if a property holds or not.
- **SmartSheild [58]:** It is a static analysis tool invented in the year 2020. It is a type of Ethereum virtual machine byte code rectification tool. It rectifies three security vulnerabilities: missing checks for failing external calls, missing checks for out-of-bound arithmetic operations,

TABLE 2. Comparative analysis of static analysis tools (with EVM byte code as input).

Tool	Source Code	Organization	Academic	Command Line Interface	Web Interface	Year	Platform
DEFECTCHECKER	✗	✗	✓	✓	✗	2021	JAVA
E-EVM	✓	✗	✗	✓	✗	2018	Python
Erays	✓	✗	✗	✓	✗	2018	Python
ESCORT	✗	✗	✓	✓	✗	2021	-
Ether (S-GRAM)	✗	✗	✓	✓	✗	2018	Python (Based on S-gram artifact)
EtherTrust	✓	✗	✗	✓	✗	2018	Java, Javascript
EthIR	✓	✗	✗	✓	✗	2018	Python
eThor	✗	✗	✓	✓	✗	2020	Java Script
EthPloit	✗	✗	✓	✓	✗	2020	-
GasChecker	✗	✗	✓	✓	✗	2020	-
Gasper	✗	✗	✓	✓	✗	2017	Python
HoneyBadger	✓	✗	✗	✓	✗	2019	Python
KEVM	✓	✗	✗	✓	✗	2016	Mixture of markdown syntax and K specification language.
MadMax	✓	✗	✗	✓	✗	2018	GogaHose IR
Mythril	✓	ConsenSys	✗	✓	✗	2017	Python
Octopus	✓	quoscient	✗	✓	✗	2018	Python
Osiris	✓	✗	✗	✓	✗	2018	Python
Oyente	✓	✗	✗	✓	✗	2016	Python
Porosity	✓	Camae	✗	✓	✗	2017	C++
RA	✓	✗	✗	✓	✗	2020	Python
Rattle	✓	Trail of Bits	✗	✓	✗	2018	Python
Securify	✓	Chain Security (Closed Version)	✗	✓	✗	2018	JAVA
SMARTSHEILD	✗	✗	✓	✓	✗	2020	-
TEEther	✗	✗	✓	✓	✗	2018	Python
Vandal	✓	✗	✗	✓	✗	2018	Python
VerX	✗	✗	✓	✗	✓	2020	-

and state changes after external calls. It extracts semantic information related to Ethereum virtual machine byte code from abstract syntax tree generated from source code and non-rectified EVM byte code. It performs byte code relocation and validation and, in the end, produces rectification reports and rectified contracts by using the information from control flow transformation and data guard insertion.

- **TEEther [14]:** It is a static analysis tool implemented in Python and invented in the year 2018. It is an exploit generation tool for smart contracts. The process of exploit generation is divided into five modules. The first module is a CFG generation module, the second module is a path generation module, the third is a constraint generation module, and the last is an exploit generation module. TEEther uses Z3 as a constraint solver.
- **Vandal [67]:** It is a publicly available open-source static analysis tool implemented in Python and invented in the year 2018. Its analysis pipeline transforms the Ethereum virtual machine byte code to semantic logic relations. It phrases the vulnerability analysis into souffle‘ which is a declarative language. Vandal's analysis pipeline process is divided into several stages: scrapper, disassembler, decompiler, and extractor, which finally produces the logic relations.
- **VerX [71]:** It is a static analysis tool invented in the year 2020. It verifies the temporal properties of smart contracts. It employs the combination of symbolic execution and abstraction during transaction execution. A new symbolic execution engine for Ethereum virtual machine is used by the VerX, which avoids the shortcomings of the existing execution engine.

3) INPUT TO THE TOOL: EVM BYTE CODE/SOLIDITY CODE

- **Conkas [122]:** It is a publicly available open-source static analysis tool implemented in Python and invented in 2019. The cookies analysis methodology is based on symbolic execution. It takes an input of solidity code or Ethereum virtual machine byte code. Cookies use Z3 as an SMT solver and Rattle for intermediate representation.
- **GASOL [139]:** It is a static analysis tool invented in the year 2020. GASOL is an analysis tool and optimization tool for gas-related issues. After analyzing the selected types of EVM instructions, GASOL returns the upper bound of the cost of execution of the function. This tool estimates the gas cost of a running function and informs the users about any vulnerability related to gas in the same.
- **SAFEVM [125]:** It is a publicly available open-source static analysis tool implemented in Python and invented in 2019. It is a verification tool with a verification engine for the C program. It can take Solidity or EVM byte code input for analysis purposes along with assert and required authentication annotation and outputs a verification result. It depends on Oyente for generating control flow graphs and EthIR framework for generating rule-based representation. It can detect the division by zero, out-of-bounds access vulnerabilities efficiently.

Table 2 presents a comparative analysis of static analysis tools with EVM byte code as input.

4) INPUT TO THE TOOL: FORM. SPEC

- **FSolidM [95]:** It is a publicly available open-source static analysis tool implemented in JavaScript and invented in 2017. It allows defining contract as FSM (Finite State Machine) with the precise and clear-out specification. It is a web-based tool built on top of the WebGME. This tool provides a security plugins mechanism to prevent security vulnerabilities in smart contracts. It is the only tool that generates solidity code according to the specifications defined by the user.

Table 3 summarizes the same type of information as in Tables 1 and 2 for static analysis tools like a year of advent, development programming language, source code availability, etc. Table 4 summarizes the same type of information for the tools with input both solidity code and EVM byte code. Some companies develop some tools. The respective company name is mentioned in front of the respective tool's name for such tools.

Table 4 presents the relationship between the tools (with solidity code as input, EVM byte code, and both respectively) and their checked or detected vulnerabilities and their employed analysis approach. Some tools are only analysis tools. They do not detect or check any vulnerability but analyze the smart contract by code transformation into some other form or by visualizing it into some kind of graphical representation, which makes the analysis very easier. Table 5 depicts the same type of information as presented in Table 4 but for the tools with input either only as solidity code or both the solidity code and EVM byte code.

B. CATEGORY 2: DYNAMIC ANALYSIS TOOL

1) INPUT TO THE TOOL: SOLIDITY CODE

- **ContractLarva [75]:** It is a publicly available open-source dynamic analysis tool implemented in Haskell and TeX and invented in 2017. It is a runtime verification tool and works on solidity code. This tool instruments the Ethereum smart contract using event triggering and monitoring logic. It uses dynamic event automata for specifying properties that help monitor the events. The tool captures two types of events: control flow events and data flow events.
- **Ethlint [123]:** It is a publicly available open-source dynamic analysis tool implemented in JavaScript and invented in 2016. It was formally known as Solium. It checks the solidity code for style and security issues. It derives ideas from ESLint, a static analyzer for JavaScript code and Solidity Parser.
- **Harvey [102]:** It is a dynamic analysis tool invented in the year 2020. Harvey is a grey-box fuzzer for smart contracts, which is nothing but a lightweight test generation approach to detect vulnerabilities and security bugs. Harvey mainly detects two types of bugs, the first one is assertion violations defined in SWC 110 and the other one is memory access errors defined in SWC 124.
- **ModCon [108]:** It is a dynamic analysis tool implemented in JavaScript and invented in 2020. It is a model-based testing framework, defines test oracles using user-defined models, and works for both permissioned and permission-less blockchain platforms. ModCon has a web-based frontend and JavaScript-based back-end. It takes an input of the smart contract and a test model specification from the user.
- **Solitor [64]:** It is a dynamic analysis tool implemented in Java and invented in 2018. Solitor is short for Solidity monitor. In this tool, the user can specify the behavior using annotations. These annotations can be used at run-time to check whether specific properties hold or not.
- **Vultron [27]:** It is a publicly available open-source dynamic analysis tool implemented in JavaScript and invented in 2019. It proposes an approach to building an oracle to detect irregular transactions over a normal one. A broad spectrum of downstream analysis techniques like testing, fuzzing, verification, and symbolic execution can be enabled by this oracle. Its other name is ContraMaster.

Tables 6, 7, and 8 presents a comparative analysis of dynamic analysis tools with solidity code, EVM byte code and both as input respectively.

TABLE 3. Comparative analysis of static analysis tools (with both solidity code and EVM byte code as input).

Tool	Source Code	Organization	Academic	Command Line Interface	Web Interface	Year	Platform
Conkas	✓	✗	✗	✓	✗	2019	Python
GASOL	✗	✗	✓	✓	✗	2020	-
SAFEVM	✓	✗	✗	✗	✓	2019	Python
FSolidM	✓	✗	✓	✗	✓	2017	Java Script

2) INPUT TO THE TOOL: EVM BYTE CODE

- **ContractFuzzer [74]:** It is a publicly available open-source dynamic analysis tool implemented in GO and invented in 2018. It is a fuzzing tool to detect security vulnerabilities in Ethereum smart contracts. It uses ABI specifications of smart contracts for generating fuzzing inputs. It classifies test oracles to detect security vulnerabilities. Smart contract's run time behavior is logged by instrumenting the Ethereum virtual machine. Finally, these logs are analyzed to find out security vulnerabilities.
- **ContractGuard [77]:** It is a dynamic analysis tool implemented in JavaScript and invented in 2019. It employs a practical anomaly-based intrusion detection system approach. It raises the alarm to the administrators when detecting some abnormal behavior and rolls back the changes in the smart contracts state to the previous safe state.
- **EthBMC [86]:** It is a dynamic analysis tool developed in the year 2020. This tool takes input for the analysis of EVM byte code. It is an automatized vulnerability detector based on symbolic execution. It explores the available state space a program can reach. It encodes the attackers' goal using some constraints, and then that constraint is solved by using the SMT solver. It works efficiently on parity bug vulnerabilities.
- **Etherolic [88]:** It is a dynamic analysis tool implemented in Rust and invented in 2020. Etherolic's analysis methodology is based on the combination of dynamic taint tracking and console testing to analyze Ethereum virtual machine byte code. It identifies vulnerabilities as well as generates exploits to trigger unknown errors. It can detect Integer Overflow/Underflow, Bad Randomness, Re-entrancy, Locked Ether, Unhandled Exceptions, Denial of Service, Short addresses, etc.
- **EVMFuzzer [93]:** It is a publicly available open-source dynamic analysis tool implemented in Python and invented in 2019. For detecting vulnerabilities, it uses differential fuzzing techniques. It feeds seed contracts into a benchmark EVM and targets EVM to discover the discrepancies in the outcomes. These discrepancies are finally used to detect vulnerabilities by cross-referencing outputs.
- **MAIAN [106]:** It is a publicly available open-source dynamic analysis tool implemented in Python and invented in the year 2018. It employs symbolic analysis and a concrete validation approach. It uses systematic approaches for discovering a violation of specific properties in smart contracts. The specific properties are safety properties and liveness properties.
- **Manticore [107]:** It is a publicly available open-source dynamic analysis tool implemented in Python in 2017 by Trell of Bits. It is a dynamic symbolic execution analysis tool. User customization is allowed in Manticore for analysis purposes. Its architecture is divided into primary and secondary components. The primary components are the Ethereum execution modules and the core engine. The secondary components include the API module, event system module, and SMT-LIB module.
- **Sereum [129]:** It is a dynamic analysis tool implemented in JavaScript and implemented in 2019. Sereum is short of secured Ethereum. It protects deployed smart contracts against re-entrancy vulnerability attacks. It employs the run-time monitoring and validation approach in a backward-compatible way. This tool also employs taint tracking to monitor the execution of a smart contract and monitor the data flow from storage variables.
- **sFuzz [131]:** It is a publicly available open-source dynamic analysis tool implemented in C++ and invented in 2020. It employs the approach of feedback-guided adaptive fuzzing. The test generation problem is transformed into an optimization problem in this approach. Then this optimization problem is solved by using some form of feedback. A genetic algorithm is employed at the top level by this tool.
- **SODA [59]:** It is a publicly available open-source dynamic analysis tool implemented in GO and invented in 2020. SODA is compatible with any blockchain compatible with Ethereum Virtual Machine. It is embedded with eight apps with new methods which can easily detect major vulnerabilities in Ethereum blockchain-based smart contracts like invalid input data, re-entrancy incorrect check for authorization, no check after contract, unexpected function invocation.

3) INPUT TO THE TOOL: EVM BYTE CODE/SOLIDITY CODE

Following dynamic analysis tools analyze the Ethereum smart contract by taking as input both the solidity code or EVM byte code.

- **EasyFlow [79]:** It is a publicly available open-source dynamic analysis tool implemented in the

TABLE 4. Static analysis tool (with input as solidity code) checked vulnerabilities and analysis approach.

	Checked Vulnerability													Analysis Approach										
Tool	Timestamp Dependancy	Reentrency	*TOD	tx.origin	Blockhash/Block Number	Gas Related Isssues	Delegate Call	**Underflow/Overflow	Freezing Ether	Unchecked Call	Self Destruct	Access Control	Denial of Service	Symbolic Execution	Dis-assembler	Graphic Visualizer	Fuzz Testing	Constraint Solving	Machine Learning	Code Instrumentation	Mutation Testing	Code Transformation	Formal Verification	Abstract Interpretation
ContractWard	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Echidna	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
Eth2Vec	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Ethainter	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
EthVer	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
FEther	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
FSPVM	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✓	✗
Gas Gauge	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
GasTap	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
MuSC	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
Neucheck	✓	✓	✗	✓	✗	✗	✗	✓	✗	✓	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Pakala	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Remix-IDE	✓	✓	✗	✓	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
SASC	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
sCompile	✗	✓	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
SESCon	✓	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗
SIF	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Slither	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗
SmartAnvil	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
SmartBugs	✗	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
SmartCheck	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
SmartEmbed	✓	✓	✗	✓	✗	✓	✗	✓	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
Smart-Graph	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗
SmartInspect	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✓	✗	✗
SmartScan	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Sol Analyzer	✓	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Solc-Verify	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Solgraph	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗
Solguard	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
Solhint	✗	✓	✗	✓	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Solidifier	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Soliditycheck	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
SolMet	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗
VeriSmart	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
VeriSol	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
VeriSolid	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Zeus	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗

*TOD-Transaction Ordering Dependency, **Related to Arithmetic

TABLE 5. Static analysis tool (with input as only solidity code or both solidity code and EVM byte code) checked vulnerabilities and analysis approach.

	Checked Vulnerability													Analysis Approach										
Tool	Timestamp Dependancy	Reentrency	*TOD	tx.origin	Blockhash/Block Number	Gas Related Issues	Delegate Call	**Underflow/Overflow	Freezing Ether	Unchecked Call	Self Destruct	Access Control	Denial of Service	Symbolic Execution	Dis-assembler	Graphic Visualizer	Fuzz Testing	Constraint Solving	Machine Learning	Code Instrumentation	Mutation Testing	Code Transformation	Formal Verification	Abstract Interpretation
Conkas	✗	✓	✓	✗	✓	✗	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
DEFECTCHECKER	✓	✓	✗	✗	✓	✗	✗	✗	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
E-EVM	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Erays	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
ESCORT	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
Ether (S-GRAM)	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
EtherTrust	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓
EthIR	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
eThor	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
EthPloit	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗
FSolidM	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
GasChecker	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
GASOL	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
Gasper	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
HoneyBadger	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
KEVM	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
MadMax	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Mythril	✓	✓	✓	✓	✗	✓	✗	✓	✗	✓	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Octopus	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗
Osiris	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Oyente	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Porosity	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
RA	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Rattle	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
SAFEVM	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Securify	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
SMARTSHEILD	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗
TEEther	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Vandal	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
VerX	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗

*TOD-Transaction Ordering Dependency, **Related to Arithmetic

GO programming language and invented in 2019. It employs the taint analysis approach to track the propagation of involved units. It monitors the transaction processes using the taint analysis component and declares the smart contract vulnerable in three categories which are ''safe'', ''overflow'' and ''potential overflow''.

- **ReGuard [24]:** It is a dynamic analysis tool implemented in Python and invented in 2018. It is a fuzzing-based analyzer for Ethereum smart contracts. It performs fuzz testing on the smart contract and generates diverse transactions iteratively. It records critical execution traces during the run time and dynamically

TABLE 6. Comparative analysis of dynamic analysis tools with solidity code as input.

Tool	Source Code	Organization	Academic	Command Line Interface	Web Interface	Year	Platform
ContractLarva	✓	✗	✗	✓	✗	2017	Haskell(70%) TeX(30%)
Ethlint	✓	✗	✗	✓	✗	2016	JavaScript
Harvey	✗	✗	✓	✓	✗	2020	-
ModCon	✗	✗	✓	✗	✓	2020	JavaScript
Solitor	✓	✗	✓	✓	✗	2018	Java
VULTRON	✓	✗	✗	✓	✗	2019	Java Script

TABLE 7. Comparative analysis of dynamic analysis tools with EVM byte code as input.

Tool	Source Code	Organization	Academic	Command Line Interface	Web Interface	Year	Platform
ContractFuzzer	✓	✗	✗	✓	✗	2018	Go
ContractGuard	✗	✗	✓	✓	✗	2019	Java Script
EthBMC	✓	✗	✗	✓	✗	2020	Rust
Etherolic	✗	✗	✓	✓	✗	2020	Rust
EVMFuzz	✓	✗	✓	✓	✗	2019	Python
MAIAN	✓	✗	✗	✓	✓	2018	Python
Manticore	✓	Trail of Bits	✗	CLI and Python API	✗	2017	Python
Sereum	✗	✗	✓	✓	✗	2019	-
sFuzz	✓	✗	✗	✓	✗	2020	C++
SODA	✓	✗	✗	✓	✗	2020	Go

TABLE 8. Comparative analysis of dynamic analysis tools with both solidity code and EVM byte code as input.

Tool	Source Code	Organization	Academic	Command Line Interface	Web Interface	Year	Platform
EASYFLOW	✓	✗	✗	✓	✓	2019	Go
ReGuard	✗	✓	✗	✓	✗	2018	Python
SoliAudit	✗	✗	✓	✓	✗	2019	-

identifies re-entrancy vulnerability by feeding them to re-entrancy automata.

- **SoliAudit [25]:** It is a dynamic testing tool invented in the year 2019. SoliAudit is a fuzzing and vulnerability analysis tool based on machine learning and fuzz testing approach. It analyses solidity code in machine code, i.e., opcode, to verify 13 kinds of top vulnerabilities: access control, denial of service, bad randomness, front running, arithmetic, time manipulation, unchecked low-level calls, short addresses, re-entrancy. The approach used by this tool does not require expert knowledge or predefined patterns.

Tables 7, 8, and 9 present a comparative analysis of the same type of information as presented in Tables 2, 3, and 4. Table 7 presents the comparative analysis for dynamic analysis tools with solidity code as input. Table 8 presents the comparative analysis for dynamic analysis tools with EVM byte code as input. Table 9 presents the comparative analysis for dynamic analysis tools with input both solidity code and EVM byte code.

IV. DISCUSSION

This paper presents a systematic review of Ethereum blockchain-based smart contract analysis tools irrespective of their type and analysis approach. For this purpose, we covered 86 analysis tools and referred to 145 research papers from the literature and other online resources from 2016 to 2021. This section covers the comparison of this survey with the related work, limitations of the survey, and the future research directions, which help the researchers and smart contract tools developers to set future research directions in this domain. The most popular top five vulnerabilities checked or detected by most of the tools are re-entrance, arithmetic overflow/underflow, gas-related, timestamp dependency, and transaction ordering dependency. Figure 7 depicts the share of each of these vulnerabilities concerning checking or detecting by static or dynamic analysis tools.

TABLE 9. Dynamic analysis tools checked vulnerabilities and analysis approach.

	Checked Vulnerability													Analysis Approach									
Tool	Timestamp Dependancy	Reentrency	*TOD	tx.origin	Blockhash/Block Number	Gas Related Issues	Delegate Call	**Underflow/Overflow	Freezing Ether	Unchecked Call	Self Destruct	Access Control	Denial of Service	Symbolic Execution	Dis-assembler	Visualizer	Fuzz Testing	Taint analysis	Machine Learning	Code Instrumentation	Mutation Testing	Model Based Testing	Formal Verification
ContractFuzzer	✓	✓	✗	✗	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
ContractGuard	✗	✓	✗	✓	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗
ContractLarva	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
EASYFLOW	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
EthBMC	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Etherolic	✗	✓	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
Ethlint	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
EVMFuzz	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Harvey	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
MAIAN	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
Manticore	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
ModCon	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
ReGuard	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
Sereum	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗
sFuzz	✓	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗
SODA	✓	✓	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
SoliAudit	✓	✓	✗	✗	✗	✗	✗	✓	✗	✓	✗	✓	✗	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗
Solitor	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗
VULTRON	✗	✓	✗	✗	✗	✓	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗	✗	✗	✗	✗	✗

*TOD-Transaction Ordering Dependency, **Related to Arithmetic

The most popular top five analysis approaches employed by most tools are symbolic execution, fuzz testing, constraint solving, code instrumentation, and code transformation. Figure 8 depicts the share of each of these analysis approaches concerning the use by static or dynamic analysis tools.

We selected some tools from each category and performed an experimental analysis. Some of these chosen tools have already been practically evaluated in the past on different data sets. We used a group of 30 contracts tagged with the top 5 vulnerabilities mentioned in Figure 7 and downloaded them from the SolidiFI Benchmark [143] data set. The SolidiFI Benchmark is a remote data set smart contracts with 9369 tagged vulnerabilities. Figure 9 depicts the average execution time of each selected tool on the 30 smart contracts. The Slither, Solhint, and Smart check perform better in average execution time. The Average Execution time of the Manticore is very high as compared to other tools. Figure 10 shows the detected vulnerabilities results by each tool, and Figure 11 shows the false-positive results of each selected tool on 30 contracts. The performance of the Slither, Mythril, and Oyente is better than other tools chosen for comparison in terms of vulnerability detection. The Slither and Mythril detect the maximum number of vulnerabilities related to re-entrency and arithmetic underflow/overflow issues. The false-positive rate of the Mythril and Securify is high than other selected tools. The false-positive rate of the SmartCheck, and HoneyBadger is low as compared to other tools. VeriSmart, RA, sFuzz, SODA, and VeriSolid are some of the latest tools which have not been explored so much in the past. The false-positive rate of these tools is meager as compared to other tools. The RA and VeriSolid are specific tools for Re-entrency vulnerabilities. The VeriSmart is a particular tool for arithmetic overflow/underflow vulnerabilities.

In case of an attack on smart contracts, no one is liable for any losses due to the decentralized nature of blockchain. Involved parties need to be bound for any loss that occurred due to the legitimate design of the smart contract because the

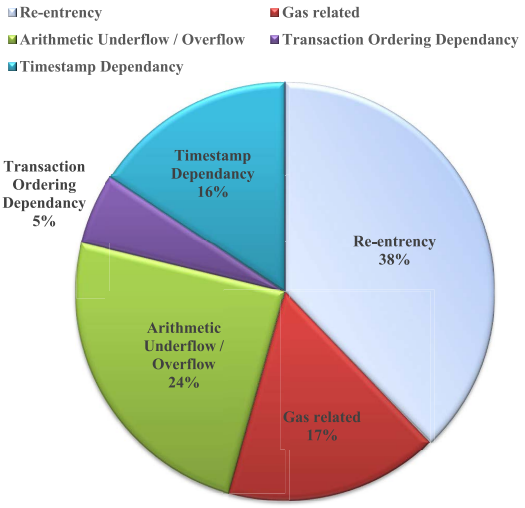

FIGURE 7. Top five vulnerabilities share checked or detected by static or dynamic analysis.

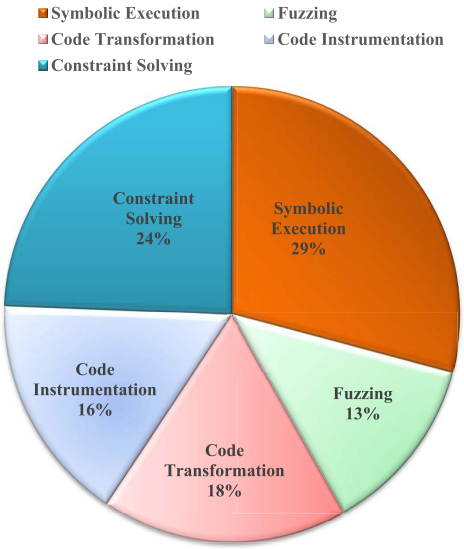

FIGURE 8. Top five analysis approaches share with respect to use by static or dynamic analysis.

smart contract code permits the breach. The DAO (Decentralized Autonomous Organization) was the first and one of the famous attacks on a smart contract which caused the loss of 60 Million US dollars. The existing tools are unable to deal with the liability issues. Because of this, smart contracts must be analyzed for security issues in advance before being deployed on the blockchain.

A. COMPARISON WITH RELATED WORK

This systematic review can be considered an extension to the existing surveys in the Ethereum blockchain-based smart contract analysis tools. To fill the research gap, this paper covers a deep insight of 86 analysis tools divided into two main categories static analysis and dynamic analysis. Then the tools are further divided into sub-categories based on input to the tool for analysis. It is found that this review covers most of the tools as compared to any other survey

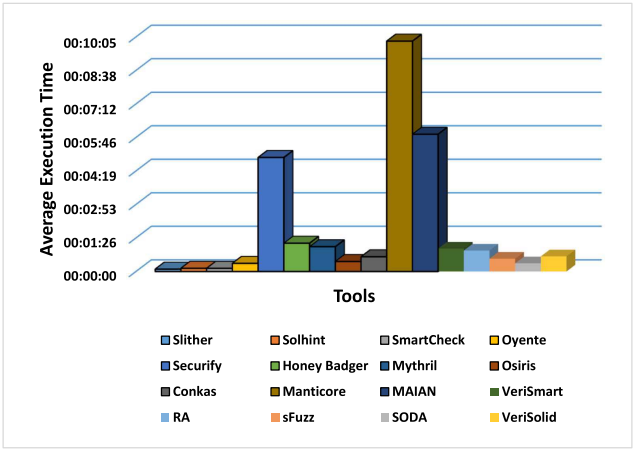

FIGURE 9. Average execution time of each tool on 30 contracts.

in this domain. To highlight the future research direction in this domain we considered the most recent literature. Figure 9 depicts the comparison of the present survey and related work in terms of the number of tools discussed.

B. LIMITATIONS

The Ethereum blockchain is the most prominent one, and most of the research has the maximum share of the same. So, our discussion is mainly related to the Ethereum blockchain-based smart analysis tools. But other blockchain platforms also have essential concerns. Our discussion doesn't consider those analysis approaches that give any tool name to their approach. The same is left for future research. Following are some limitations of this review:

1) Our discussion compares the tools with different-different aspects like source code availability, development platform, checked vulnerabilities, etc., but does not cover the high-level description related to the full flesh working of the tools.
2) There are several other analysis tools for other blockchain-based smart contracts like Solana, Hyperledger, etc. Our discussion mainly covers the analysis tools related to Ethereum blockchain smart contract because it has the maximum share in the literature related to smart contracts.
3) This review should not be considered complete because several new analysis tools and approaches are being proposed and developed day by day. Our discussion does not cover the analysis tools proposed in the preprints articles.
4) Some research articles don't have any name for the detection approach or model proposed. Such articles are not covered in this review.

C. FUTURE RESEARCH DIRECTIONS

Blockchain technology is changing day by day at a very fast pace. So, several new functionalities and features will be added to the Ethereum blockchain in the near future. So, more new security vulnerabilities will be discovered. So, it may

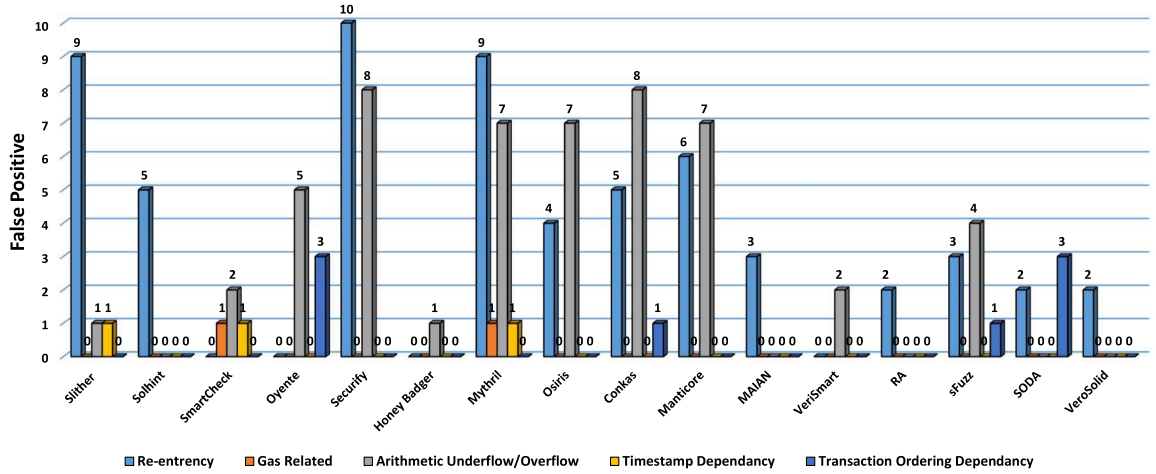

FIGURE 10. Top 5 vulnerabilities detection performance of each selected tools.

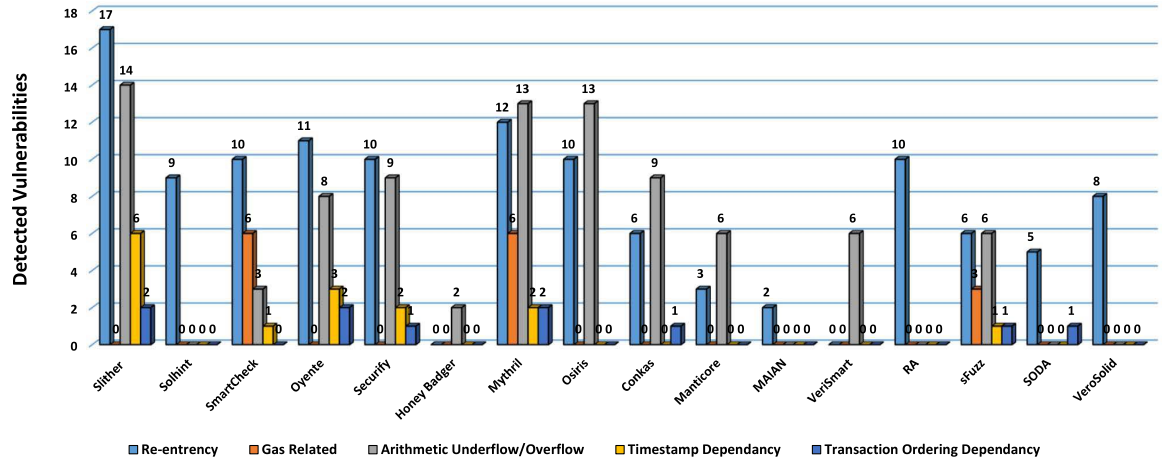

FIGURE 11. False positive rate of each tool on top 5 vulnerabilities.

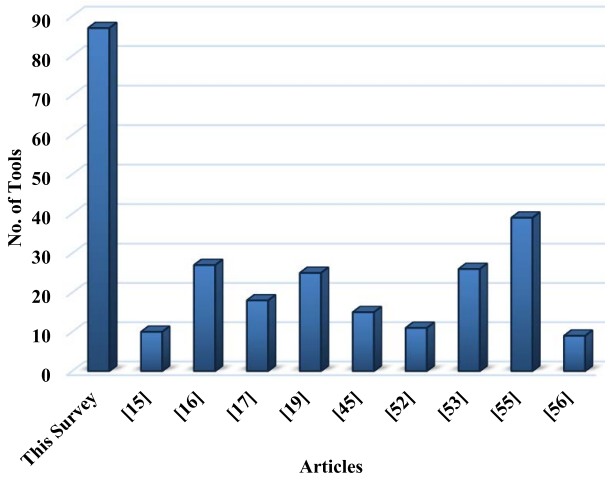

FIGURE 12. Comparison of present survey and related work in terms of number of tools discussed.

lead to the development of new and advanced analysis tools for detecting such types of vulnerabilities. No tool is found in the survey which checks or detects all the vulnerabilities presented in the literature. A lot of work is suggested to be done in this direction. Following are some of the future research directions:

1) We found that most of the tools' source code is not openly accessible for evaluation purposes. So, for building trust among the research community, the source code must be openly available for evaluation.
2) Most of the Ethereum smart contract analysis tools mostly check or detect only some of the famous Ethereum smart contract vulnerabilities. A standard benchmark should be suggested by the research community for evaluating the effectiveness of any analysis tool.
3) The majority of the Ethereum smart contract analysis tools employ the static analysis approach. But for the complete analysis of a smart contract, both static and dynamic analysis is necessary to detect or check all the vulnerabilities.
4) The research lacks very few tools for designing and creating new smart contracts. Such types of tools should be developed that will be an aid for the research community for developing safe smart contracts.

V. CONCLUSION

The Ethereum smart contracts security analysis is of essential concern, and various analysis tools have been developed for creating safe and secure smart contracts. This paper presented a systematic review of Ethereum smart contracts analysis tools. Initially, 670 articles were selected from various databases such as ACM, IEEE explores, Elsevier, Springer, and Scopus. 132 articles were selected by using various inclusion and exclusion criteria. Besides it, 13 additional articles and online sources were also utilized. 86 security analysis tools in the Ethereum blockchain smart contract were analyzed regardless of tool type and analysis approach. These tools were categorized into static and dynamic analysis tools. After that, different source code analysis techniques were studied, such as taint analysis, symbolic execution, and fuzzing techniques. It was found that the most popular vulnerability checked and detected by most of the static and dynamic analysis tools is 're-entrancy'. The most popular analysis methodology employed by static analysis tools is symbolic execution and fuzz testing by dynamic analysis tools. Most of the tools have utilized static analysis, and some tools were found that employ a combination of static and dynamic analysis, i.e., hybrid analysis. It is concluded that hybrid analysis-based tools have considered more than 95% of security flaws. Finally, the paper highlights some challenges and future recommendations in Ethereum smart contracts.

REFERENCES

[1] A. Averin and O. Averina, "Review of blockchain technology vulnerabilities and blockchain-system attacks," in *Proc. Int. Multi-Conf. Ind. Eng. Modern Technol. (FarEastCon)*, Oct. 2019, pp. 1–6, doi: 10.1109/FarEastCon.2019.8934243.

[2] C. S. Wright, "Bitcoin: A peer-to-peer electronic cash system," *SSRN Electron. J.*, p. 9, Aug. 2019, doi: 10.2139/ssrn.3440802.

[3] C. K. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in *Proc. IEEE 1st Int. Workshops Found. Appl. Self Syst. (FAS-W)*, Sep. 2016, pp. 210–215.

[4] M. In and F. Of, "Dubai aims to be a city built on blockchain where financial regulation goes in a republican era," *Wall Street J.*, vol. 4, pp. 3–6, Apr. 2017. [Online]. Available: https://www.wsj.com/articles/dubai-aims-to-be-a-city-built-on-blockchain-1493086080

[5] R. Liu, "Study on single-valued neutrosophic graph with application in shortest path problem," *CAAI Trans. Intell. Technol.*, vol. 5, no. 4, pp. 308–313, 2020.

[6] M. Singh and S. Kim, "Blockchain technology for decentralized autonomous organizations," in *Advances in Computers*, vol. 115. Amsterdam, The Netherlands: Elsevier, 2019, pp. 115–140.

[7] M. F. Aljunid and M. D. Huchaiah, "Multi-model deep learning approach for collaborative filtering recommendation system," *CAAI Trans. Intell. Technol.*, vol. 5, no. 4, pp. 268–275, 2020.

[8] M. Wöhrer and U. Zdun, "Design patterns for smart contracts in the Ethereum ecosystem," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Aug. 2018, pp. 1513–1520, doi: 10.1109/Cybermatics_2018.2018.00255.

[9] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM Conf. Comput. Commun. Secur.*, vols. 24–28, 2016, pp. 254–269, doi: 10.1145/2976749.2978309.

[10] J. Jiang and L. Hu, "Decentralised federated learning with adaptive partial gradient aggregation," *CAAI Trans. Intell. Technol.*, vol. 5, no. 3, pp. 230–236, 2020.

[11] S. Jatsun, A. Malchikov, A. Yatsun, A. M. Khalil, and A. S. M. Leon, "Simulation of a walking robot-exoskeleton movement on a movable base," *J. Artif. Intell. Technol.*, vol. 1, no. 4, pp. 207–213, Oct. 2021.

[12] B. A. Kitchenham, D. Budgen, and O. P. Brereton, "Using mapping studies as the basis for further research—A participant-observer case study," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 638–651, 2011, doi: 10.1016/j.infsof.2010.12.011.

[13] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *Proc. 12th Int. Conf. Eval. Assessment Softw. Eng. (EASE)*, Jun. 2008, pp. 1–10, doi: 10.14236/ewic/ease2008.8.

[14] J. Krupp and C. Rossow, "teEther: Gnawing at Ethereum to automatically exploit smart contracts," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1317–1333.

[15] D. Harz and W. Knottenbelt, "Towards safer smart contracts: A survey of languages and verification methods," 2018, *arXiv:1809.09805*.

[16] M. di Angelo and G. Salzer, "A survey of tools for analyzing Ethereum smart contracts," in *Proc. IEEE Int. Conf. Decentralized Appl. Infrastruct. (DAPPCON)*, Newark, CA, USA, Apr. 2019, pp. 69–78.

[17] J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," *IEEE Access*, vol. 7, pp. 77894–77904, 2019.

[18] L. Ante. (2020). *Smart contracts on the Blockchain—A Bibliometric Analysis and Review*. [Online]. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3576393

[19] M. Almakhour, L. Sliman, A. E. Samhat, and A. Mellouk, "Verification of smart contracts: A survey," *Pervasive Mobile Comput.*, vol. 67, Sep. 2020, Art. no. 101227.

[20] I. C. Lin and T. C. Liao, "A survey of blockchain security issues and challenges," *Int. J. Netw. Secur.*, vol. 19, no. 5, pp. 653–659, 2017, doi: 10.6633/IJNS.201709.19(5).01.

[21] E. Mik, "Smart contracts: Terminology, technical limitations and real world complexity," *Law, Innov. Technol.*, vol. 9, no. 2, pp. 269–300, Jul. 2017, doi: 10.1080/17579961.2017.1378468.

[22] W. Chen, Z. Zheng, E. C.-H. Ngai, P. Zheng, and Y. Zhou, "Exploiting blockchain data to detect smart Ponzi schemes on Ethereum," *IEEE Access*, vol. 7, pp. 37575–37586, 2019, doi: 10.1109/ACCESS.2019.2905769.

[23] T. Min, H. Wang, Y. Guo, and W. Cai, "Blockchain games: A survey," in *Proc. IEEE Symp. Comput. Intell. Games (CIG)*, Aug. 2019, pp. 1–8, doi: 10.1109/CIG.2019.8848111.

[24] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding reentrancy bugs in smart contracts," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 65–68, doi: 10.1145/3183440.3183495.

[25] J.-W. Liao, T.-T. Tsai, C.-K. He, and C.-W. Tien, "SoliAudit: Smart contract vulnerability assessment based on machine learning and fuzz testing," in *Proc. 6th Int. Conf. Internet Things, Syst., Manage. Secur. (IOTSMS)*, Oct. 2019, pp. 458–465, doi: 10.1109/IOTSMS48152.2019.8939256.

[26] B. Ghaleb, A. Al-Dubai, E. Ekonomou, M. Qasem, I. Romdhani, and L. Mackenzie, "Addressing the DAO insider attack in RPL's Internet of Things networks," *IEEE Commun. Lett.*, vol. 23, no. 1, pp. 68–71, Jan. 2019, doi: 10.1109/LCOMM.2018.2878151.

[27] H. Wang, Y. Li, S. W. Lin, L. Ma, and Y. Liu, "VULTRON: Catching vulnerable smart contracts once and for all," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., New Ideas Emerg. Results (ICSE-NIER)*, May 2019, pp. 1–4, doi: 10.1109/ICSE-NIER.2019.00009.

[28] B. C. Gupta and S. K. Shukla, "A study of inequality in the ethereum smart contract ecosystem," in *Proc. 6th Int. Conf. Internet Things, Syst., Manage. Secur. (IOTSMS)*, Oct. 2019, pp. 441–449, doi: 10.1109/IOTSMS48152.2019.8939257.

[29] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Gener. Comput. Syst.*, vol. 107, pp. 841–853, Jun. 2020, doi: 10.1016/j.future.2017.08.020.

[30] L. Zhu, B. Zheng, M. Shen, F. Gao, H. Li, and K. Shi, "Research on the security of blockchain data: A Survey," *CoRR*, vol. abs/1812.02009, pp. 1–48, Dec. 2018, doi: 10.1007/s11390-020-9638-7.

[31] S. N. Khan, F. Loukil, C. Ghedira-Guegan, E. Benkhelifa, and A. Bani-Hani, "Blockchain smart contracts: Applications, challenges, and future trends," *Peer-to-Peer Netw. Appl.*, vol. 14, no. 5, pp. 2901–2925, Sep. 2021, doi: 10.1007/s12083-021-01127-0.

[32] Z. Wang, H. Jin, W. Dai, K.-K.-R. Choo, and D. Zou, "Ethereum smart contract security research: Survey and future research opportunities," *Frontiers Comput. Sci.*, vol. 15, no. 2, Apr. 2021, doi: 10.1007/s11704-020-9284-9.

[33] T. Hu, X. Liu, T. Chen, X. Zhang, X. Huang, W. Niu, J. Lu, K. Zhou, and Y. Liu, "Transaction-based classification and detection approach for Ethereum smart contract," *Inf. Process. Manage.*, vol. 58, no. 2, Mar. 2021, Art. no. 102462, doi: 10.1016/j.ipm.2020.102462.

[34] A. Bhardwaj, S. B. H. Shah, A. Shankar, M. Alazab, M. Kumar, and T. R. Gadekallu, "Penetration testing framework for smart contract blockchain," *Peer-to-Peer Netw. Appl.*, vol. 14, no. 5, pp. 2635–2650, Sep. 2021, doi: 10.1007/s12083-020-00991-6.

[35] A. Vacca, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges," *J. Syst. Softw.*, vol. 174, Apr. 2021, Art. no. 110891, doi: 10.1016/j.jss.2020.110891.

[36] J. Liu, C. Sun, and Y. Lai, "A data transmission approach based on ant colony optimization and threshold proxy re-encryption in WSNs," *J. Artif. Intell. Technol.*, vol. 2, no. 1, pp. 23–31, Dec. 2021.

[37] H. Rameder, "Systematic review of Ethereum smart contract security vulnerabilities, analysis methods and tools," M.S. thesis, Dept. Comput. Eng., Technische Universität Wien, Vienna, Austria, 2021, doi: 10.34726/hss.2021.86784.

[38] Y. Xu, Y. Li, and C. Li, "Electric window regulator based on intelligent control," *J. Artif. Intell. Technol.*, vol. 1, no. 4, pp. 198–206, Sep. 2021.

[39] Z. Wan, X. Xia, D. Lo, J. Chen, X. Luo, and X. Yang, "Smart contract security: A practitioners' perspective," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 1410–1422, doi: 10.1109/ICSE43902.2021.00127.

[40] T. Palina, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Comput. Surv.*, vol. 54, no. 7, p. 38, 2021, doi: 10.1145/3464421.

[41] S. Ji, D. Kim, and H. Im, "Evaluating countermeasures for verifying the integrity of Ethereum smart contract applications," *IEEE Access*, vol. 9, pp. 90029–90042, 2021, doi: 10.1109/ACCESS.2021.3091317.

[42] N. A. N. Aidee, M. G. M. Johar, M. H. Alkawaz, A. I. Hajamydeen, and M. S. H. Al-Tamimi, "Vulnerability assessment on Ethereum based smart contract applications," in *Proc. IEEE Int. Conf. Autom. Control Intell. Syst. (I2CACIS)*, Jun. 2021, pp. 13–18, doi: 10.1109/I2CACIS52118.2021.9495892.

[43] R. Gupta, M. M. Patel, A. Shukla, and S. Tanwar, "Deep learning-based malicious smart contract detection scheme for Internet of Things environment," *Comput. Electr. Eng.*, vol. 97, Jan. 2022, Art. no. 107583, doi: 10.1016/j.compeleceng.2021.107583.

[44] X. Tang, K. Zhou, J. Cheng, H. Li, and Y. Yuan, "The vulnerabilities in smart contracts: A survey," in *Advances in Artificial Intelligence and Security* (Communications in Computer and Information Science), vol. 1424, X. Sun, X. Zhang, Z. Xia, and E. Bertino, Eds. Cham, Switzerland: Springer, 2021, doi: 10.1007/978-3-030-78621-2_14.

[45] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on Ethereum systems security," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–43, May 2021, doi: 10.1145/3391195.

[46] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H.-N. Lee, "Systematic review of security vulnerabilities in Ethereum blockchain smart contract," *IEEE Access*, vol. 10, pp. 6605–6621, 2022, doi: 10.1109/ACCESS.2021.3140091.

[47] R. Lawler. (Dec. 3, 2021). *Someone Stole $ 120 Million in Crypto by Hacking a DeFi Website, The Verge*. Accessed: Dec. 11, 202. [Online]. Available: https://www.theverge.com

[48] A. Ghaleb and K. Pattabiraman, "How effective are smart contract analysis tools? Evaluating smart contract static analysis tools using bug injection," 2020, *arXiv:2005.11613*.

[49] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, "A survey of smart contract formal specification and verification," *ACM Comput. Surv.*, vol. 54, no. 7, pp. 1–38, Sep. 2022.

[50] D. He, Z. Deng, Y. Zhang, S. Chan, Y. Cheng, and N. Guizani, "Smart contract vulnerability analysis and security audit," *IEEE Netw.*, vol. 34, no. 5, pp. 276–282, Sep. 2020, doi: 10.1109/MNET.001.1900656.

[51] I. Grishchenko, M. Maffei, and C. Schneidewind, "Foundations and tools for the static analysis of Ethereum smart contracts," in *Computer Aided Verification* (Lecture Notes in Computer Science), vol. 10981, H. Chockler and G. Weissenbacher, Eds. Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-319-96145-3_4.

[52] A. Vacca, A. Di Sorbo, C. A. Visaggio, and G. Canfora, "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges," *J. Syst. Softw.*, vol. 174, Apr. 2021, Art. no. 110891.

[53] A. Pinna, S. Ibba, G. Baralla, R. Tonelli, and M. Marchesi, "A massive analysis of Ethereum smart contracts empirical study and code metrics," *IEEE Access*, vol. 7, pp. 78194–78213, 2019.

[54] B. Hu, Z. Zhang, J. Liu, Y. Liu, J. Yin, R. Lu, and X. Lin, "A comprehensive survey on smart contract construction and execution: Paradigms, tools, and systems," *Patterns*, vol. 2, no. 2, Feb. 2021, Art. no. 100179, doi: 10.1016/j.patter.2020.100179.

[55] T. Durieux, J. F. Ferreira, R. Abreu, and P. Cruz, "Empirical review of automated analysis tools on 47,587 Ethereum smart contracts," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 530–541.

[56] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "SmartInspect: Solidity smart contract inspector," in *Proc. Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Mar. 2018, pp. 9–18, doi: 10.1109/IWBOSE.2018.8327566.

[57] N. F. Samreen and M. H. Alalfi, "SmartScan: An approach to detect denial of service vulnerability in ethereum smart contracts," in *Proc. IEEE/ACM 4th Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, May 2021, pp. 17–26.

[58] Y. Zhang, S. Ma, J. Li, K. Li, S. Nepal, and D. Gu, "SMARTSHIELD: Automatic smart contract protection made easy," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2020, pp. 23–34, doi: 10.1109/SANER48275.2020.9054825.

[59] T. Chen, R. Cao, T. Li, X. Luo, G. Gu, Y. Zhang, Z. Liao, H. Zhu, G. Chen, Z. He, Y. Tang, X. Lin, and X. Zhang, "SODA: A generic online detection framework for smart contracts," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 449–461, doi: 10.14722/ndss.2020.24449.

[60] S. Akca, A. Rajan, and C. Peng, "SolAnalyser: A framework for analysing and testing smart contracts," in *Proc. 26th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2019, pp. 482–489, doi: 10.1109/APSEC48747.2019.00071.

[61] Á. Hajdu and D. Jovanovic, "SOLC-VERIFY: A modular verifier for solidity smart contracts," in *Proc. Working Conf. Verified Softw., Theories, Tools, Exp.*, in Lecture Notes in Computer Science : Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 12031, 2020, pp. 161–179, doi: 10.1007/978-3-030-41600-3_11.

[62] P. Antonino and A. W. Roscoe, "Solidifier: Bounded model checking solidity using lazy contract deployment and precise memory modelling," in *Proc. 36th Annu. ACM Symp. Appl. Comput.*, Mar. 2021, pp. 1788–1797, doi: 10.1145/3412841.3442051.

[63] P. Zhang, F. Xiao, and X. Luo, "SolidityCheck : Quickly detecting smart contract problems through regular expressions," 2019, *arXiv:1911.09425*.

[64] L. Stegeman, "Solitor: Runtime verification of smart contracts," M.S. thesis, Univ. Twente, Enschede, The Netherlands, 2018.

[65] P. Hegedűs, "Towards analyzing the complexity landscape of solidity based Ethereum smart contracts," in *Proc. 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain*, May 2018, pp. 35–39, doi: 10.1145/3194113.3194119.

[66] S. S. Kushwaha and S. Joshi, "An overview of blockchain-based smart contract," in *Computer Networks and Inventive Communication Technologies* (Lecture Notes on Data Engineering and Communications Technologies), vol. 58, S. Smys, R. Palanisamy, Á. Rocha, and G. N. Beligiannis, Eds. Singapore: Springer, 2021, doi: 10.1007/978-981-15-9647-6_70.

[67] L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, and B. Scholz, "Vandal: A scalable security analysis framework for smart contracts," 2018, *arXiv:1809.03981*.

[68] S. So, M. Lee, J. Park, H. Lee, and H. Oh, "VERISMART: A highly precise safety verifier for Ethereum smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1678–1694, doi: 10.1109/SP40000.2020.00032.

[69] Y. Wang, S. K. Lahiri, S. Chen, R. Pan, I. Dillig, C. Born, and I. Naseer, "Formal specification and verification of smart contracts for azure blockchain," 2018, *arXiv:1812.08829*.

[70] A. Mavridou, A. Laszka, E. Stachtiari, and A. Dubey, "VeriSolid: Correct-by-design smart contracts for Ethereum," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, in Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11598, 2019, pp. 446–465, doi: 10.1007/978-3-030-32101-7_27.

[71] A. Permenev, D. Dimitrov, P. Tsankov, D. Drachsler-Cohen, and M. Vechev, "VerX: Safety verification of smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1661–1677, doi: 10.1109/SP40000.2020.00024.
[72] T. M. Hewa, Y. Hu, M. Liyanage, S. S. Kanhare, and M. Ylianttila, "Survey on blockchain-based smart contracts: Technical aspects and future research," *IEEE Access*, vol. 9, pp. 87643–87662, 2021, doi: 10.1109/ACCESS.2021.3068178.
[73] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 18, 2018, pp. 5489–5501, doi: 10.14722/ndss.2018.23082.
[74] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 259–269, doi: 10.1145/3238147.3238177.
[75] S. Azzopardi, J. Ellul, and G. J. Pace, "Monitoring smart contracts: ContractLarva and open challenges beyond," in *Proc. Int. Conf. Runtime Verification*, in Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11237, Nov. 2019, pp. 113–137, doi: 10.1007/978-3-030-03769-7_8.
[76] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: Automated vulnerability detection models for Ethereum smart contracts," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1133–1144, Apr. 2021, doi: 10.1109/TNSE.2020.2968505.
[77] X. Wang, J. He, Z. Xie, G. Zhao, and S.-C. Cheung, "ContractGuard: Defend Ethereum smart contract with embedded intrusion detection," *Chin. J. Netw. Inf. Secur.*, vol. 6, no. 2, pp. 35–55, 2020, doi: 10.1109/TSC.2019.2949561.
[78] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "DEFECTCHECKER: Automated smart contract defect detection by analyzing EVM bytecode," *IEEE Trans. Softw. Eng.*, early access, Jan. 27, 2021, doi: 10.1109/TSE.2021.3054928.
[79] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, "EASYFLOW: Keep Ethereum away from overflow," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2019, pp. 23–26, doi: 10.1109/icse-companion.2019.00029.
[80] G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, "Echidna: Effective, usable, and fast fuzzing for smart contracts," in *Proc. 29th ACM SIGSOFT Int. Symp. Softw. Test. Anal. (ISSTA)*, Jul. 2020, pp. 557–560, doi: 10.1145/3395363.3404366.
[81] R. Norvill, B. B. F. Pontiveros, R. State, and A. Cullen, "Visual emulation for Ethereum's virtual machine," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Apr. 2018, pp. 1–4, doi: 10.1109/NOMS.2018.8406332.
[82] Y. Zhou, D. Kumar, S. Bakshi, J. Mason, A. Miller, and M. Bailey, "Erays: Reverse engineering Ethereum's opaque smart contracts," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 1371–1385.
[83] O. Lutz, H. Chen, H. Fereidooni, C. Sendner, A. Dmitrienko, A. Sadeghi, and F. Koushanfar, "ESCORT: Ethereum smart contracts vulnerability detection using deep neural network and transfer learning," *CoRR*, vol. abs/2103.12607, 2021.
[84] N. Ashizawa, N. Yanai, J. P. Cruz, and S. Okamura, "Eth2Vec: Learning contract-wide code representations for vulnerability detection on Ethereum smart contracts," in *Proc. 3rd ACM Int. Symp. Blockchain Secure Crit. Infrastruct.*, May 2021, pp. 47–59, doi: 10.1145/3457337.3457841.
[85] L. Brent, N. Grech, S. Lagouvardos, B. Scholz, and Y. Smaragdakis, "Ethainter: A smart contract security analyzer for composite vulnerabilities," in *Proc. 41st ACM SIGPLAN Conf. Program. Lang. Design Implement. (PLDI)*, Jun. 2020, pp. 454–469, doi: 10.1145/3385412.3385990.
[86] J. Frank, C. Aschermann, and T. Holz, "ETHBMC: A bounded model checker for smart contracts," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 2757–2774.
[87] H. Liu, C. Liu, W. Zhao, Y. Jiang, and J. Sun, "S-gram: Towards semantic-aware security auditing for Ethereum smart contracts," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 814–819, doi: 10.1145/3238147.3240728.
[88] M. Ashouri, "Etherolic: A practical security analyzer for smart contracts," in *Proc. 35th Annu. ACM Symp. Appl. Comput.*, Mar. 2020, pp. 353–356, doi: 10.1145/3341105.3374226.
[89] I. Grishchenko, M. Maffei, and C. Schneidewind, "EtherTrust: Sound static analysis of Ethereum bytecode," Tech. Rep., 2018. [Online]. Available: https://pdfs.semanticscholar.org/26c2/b7e7479336d44891aadda6b5eaae2ca2ee91.pdf
[90] E. Albert, P. Gordillo, B. Livshits, A. Rubio, and I. Sergey, "EthIR: A framework for high-level analysis of Ethereum bytecode," in *Proc. Int. Symp. Automated Technol. Verification Anal.*, in Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11138, 2018, pp. 513–520, doi: 10.1007/978-3-030-01090-4_30.
[91] C. Schneidewind, I. Grishchenko, M. Scherer, and M. Maffei, "EThor: Practical and provably sound static analysis of Ethereum smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 621–640, doi: 10.1145/3372297.3417250.
[92] Q. Zhang, Y. Wang, J. Li, and S. Ma, "EthPloit: From fuzzing to efficient exploit generation against smart contracts," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2020, pp. 116–126, doi: 10.1109/SANER48275.2020.9054822.
[93] Y. Fu, M. Ren, F. Ma, H. Shi, X. Yang, Y. Jiang, H. Li, and X. Shi, "EVMFuzzer: Detect EVM vulnerabilities via fuzz testing," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Aug. 2019, pp. 1110–1114, doi: 10.1145/3338906.3341175.
[94] Z. Yang and H. Lei, "FEther: An extensible definitional interpreter for smart-contract verifications in Coq," *IEEE Access*, vol. 7, pp. 37770–37791, 2019, doi: 10.1109/ACCESS.2019.2905428.
[95] A. Mavridou and A. Laszka, "Tool Demonstration: FSolidM for designing secure Ethereum smart contracts," in *Proc. Int. Conf. Princ. Secur. Trust*, Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 10804, 2018, pp. 270–277, doi: 10.1007/978-3-319-89722-6_11.
[96] Z. Yang, H. Lei, and W. Qian, "A hybrid formal verification system in Coq for ensuring the reliability and security of Ethereum-based service smart contracts," *IEEE Access*, vol. 8, pp. 21411–21436, 2020, doi: 10.1109/ACCESS.2020.2969437.
[97] B. Nassirzadeh, H. Sun, S. Banescu, and V. Ganesh, "Gas gauge: A security analysis tool for smart contract out-of-gas vulnerabilities," M.S. thesis, Univ. Waterloo, Waterloo, ON, Canada, Aug. 2021.
[98] T. Chen, Y. Feng, Z. Li, H. Zhou, X. Luo, X. Li, X. Xiao, J. Chen, and X. Zhang, "GasChecker: Scalable analysis for discovering gas-inefficient smart contracts," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1433–1448, Jul. 2021, doi: 10.1109/TETC.2020.2979019.
[99] F. Hofmann, S. Wurster, E. Ron, and M. Böhmecke-Schwafert, "The immutability concept of blockchains and benefits of early standardization," in *Proc. ITU Kaleidoscope, Challenges Data-Driven Soc. (ITU K)*, Nov. 2017, pp. 1–8.
[100] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," in *Proc. IEEE 24th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2017, pp. 442–446, doi: 10.1109/SANER.2017.7884650.
[101] E. Albert, P. Gordillo, A. Rubio, and I. Sergey, "Running on fumes: Preventing out-of-gas vulnerabilities in Ethereum smart contracts using static resource analysis," in *Proc. Int. Conf. Verification Eval. Comput. Commun. Syst.*, in Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11847, 2019, pp. 63–78, doi: 10.1007/978-3-030-35092-5_5.
[102] V. Wüstholz and M. Christakis, "Harvey: A greybox fuzzer for smart contracts," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1398–1409, doi: 10.1145/3368089.3417064.
[103] C. F. Torres and M. Steichen, "The art of the scam: Demystifying honeypots in Ethereum smart contracts," in *Proc. 28th USENIX Secur. Symp.*, 2019, pp. 1591–1607.
[104] E. Hildenbrandt, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, B. Moore, D. Park, Y. Zhang, A. Stefanescu, and G. Rosu, "KEVM: A complete formal semantics of the Ethereum virtual machine," in *Proc. IEEE 31st Comput. Secur. Found. Symp. (CSF)*, Jul. 2018, pp. 204–217, doi: 10.1109/CSF.2018.00022.
[105] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, and Y. Smaragdakis, "MadMax: Surviving out-of-gas conditions in Ethereum smart contracts," *Proc. ACM Program. Lang.*, vol. 2, pp. 1–27, Nov. 2018, doi: 10.1145/3276486.
[106] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, Dec. 2018, pp. 653–663, doi: 10.1145/3274694.3274743.

[107] M. Mossberg, F. Manzano, E. Hennenfent, A. Groce, G. Grieco, J. Feist, T. Brunson, and A. Dinaburg, "Manticore: A user-friendly symbolic execution framework for binaries and smart contracts," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 1186–1189, doi: 10.1109/ASE.2019.00133.

[108] Y. Liu, Y. Li, S.-W. Lin, and Q. Yan, "ModCon: A model-based testing platform for smart contracts," in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1601–1605, doi: 10.1145/3368089.3417939.

[109] Z. Li, H. Wu, J. Xu, X. Wang, L. Zhang, and Z. Chen, "MuSC: A tool for mutation testing of Ethereum smart contract," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 1198–1201, doi: 10.1109/ASE.2019.00136.

[110] N. Lu, B. Wang, Y. Zhang, W. Shi, and C. Esposito, "NeuCheck: A more practical Ethereum smart contract security analysis tool," *Softw., Pract. Exp.*, vol. 51, no. 10, pp. 2065–2084, Oct. 2021, doi: 10.1002/spe.2745.

[111] C. F. Torres and J. Schütte, "Osiris," *Nature*, vol. 143, no. 3625, pp. 674–675, 1939, doi: 10.1038/143674d0.

[112] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Gener. Comput. Syst.*, vol. 105, pp. 475–491, Apr. 2020.

[113] M. Suiche, "Porosity: A decompiler for blockchain-based smart contracts bytecode," in *Proc. DEFCON*, vol. 25, 2017, p. 30.

[114] (Aug. 2021). *Pakala*. [Online]. Available: https://github.com/palkeo/pakala

[115] (Jun. 2021). *Remix-IDE*. [Online]. Available: https://github.com/ethereum/remix-ide

[116] (Aug. 2021). *Solgraph*. [Online]. Available: https://github.com/raineorshine/solgraph

[117] Y. Chinen, N. Yanai, J. P. Cruz, and S. Okamura, "RA: Hunting for re-entrancy attacks in Ethereum smart contracts via static analysis," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Nov. 2020, pp. 327–336, doi: 10.1109/Blockchain50366.2020.00048.

[118] (2018). *RECON Montreal 2018*. [Online]. Available: https://github.com/trailofbits/publications

[119] (Aug. 2021). *Solhint*. [Online]. Available: https://protofire.github.io/solhint

[120] (Aug. 2021). *Mythril*. [Online]. Available: https://github.com/ConsenSys/mythril

[121] (Nov. 2021). *Octopus*. [Online]. Available: https://github.com/pventuzelo/octopus

[122] (Dec. 2021). *Conkas*. [Online]. Available: https://github.com/nveloso/conkas

[123] (Dec. 2021) *Ethlint*. [Online]. Available: https://github.com/duaraghav8/Ethlint

[124] C. G. Harris, "The risks and challenges of implementing Ethereum smart contracts," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2019, pp. 104–107.

[125] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio, "SAFEVM: A safety verifier for Ethereum smart contracts," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2019, pp. 390–393, doi: 10.1145/3293882.3338999.

[126] E. Zhou, S. Hua, B. Pi, J. Sun, Y. Nomura, K. Yamashita, and H. Kurihara, "Security assurance for smart contract," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5, doi: 10.1109/NTMS.2018.8328743.

[127] J. Chang, B. Gao, H. Xiao, J. Sun, Y. Cai, and Z. Yang, "sCompile: Critical path identification and analysis for smart contracts," in *Proc. Int. Conf. Formal Eng. Methods*, in Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11852, 2019, pp. 286–304, doi: 10.1007/978-3-030-32409-4_18.

[128] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 67–82, doi: 10.1145/3243734.3243780.

[129] M. Rodler, W. Li, G. O. Karame, and L. Davi, "*Sereum*: Protecting existing smart contracts against re-entrancy attacks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2019, pp. 1–15, doi: 10.14722/ndss.2019.23413.

[130] A. Ali, Z. U. Abideen, and K. Ullah, "SESCon: Secure Ethereum smart contracts by vulnerable patterns' detection," *Secur. Commun. Netw.*, vol. 2021, Sep. 2021, pp. 1–14, doi: 10.1155/2021/2897565.

[131] T. D. Nguyen, L. H. Pham, J. Sun, Y. Lin, and Q. T. Minh, "sFuzz: An efficient adaptive fuzzer for solidity smart contracts," in *Proc. Int. Conf. Softw. Eng.*, 2020, pp. 778–788, doi: 10.1145/3377811.3380334.

[132] C. Peng, S. Akca, and A. Rajan, "SIF: A framework for solidity contract instrumentation and analysis," in *Proc. 26th Asia–Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2019, pp. 466–473, doi: 10.1109/APSEC48747.2019.00069.

[133] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. IEEE/ACM 2nd Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, May 2019, pp. 8–15, 2019, doi: 10.1109/WETSEB.2019.00008.

[134] M. Denker, *Blockchain and Web 3.0*. Evanston, IL, USA: Routledge, 2019, doi: 10.4324/9780429029530.

[135] J. F. Ferreira, P. Cruz, T. Durieux, and R. Abreu, "SmartBugs: A framework to analyze solidity smart contracts," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Dec. 2020, pp. 1349–1352, doi: 10.1145/3324884.3415298.

[136] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in *Proc. 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain*, May 2018, pp. 9–16, doi: 10.1145/3194113.3194115.

[137] Z. Gao, V. Jayasundara, L. Jiang, X. Xia, D. Lo, and J. Grundy, "SmartEmbed: A tool for clone and bug detection in smart contracts through structural code embedding," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2019, pp. 394–397, doi: 10.1109/ICSME.2019.00067.

[138] G. A. Pierro, "Smart-graph: Graphical representations for smart contract on the Ethereum blockchain," in *Proc. IEEE Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Mar. 2021, pp. 708–714, doi: 10.1109/SANER50967.2021.00090.

[139] E. Albert, J. Correas, P. Gordillo, G. Román-Díez, and A. Rubio, "GASOL: Gas analysis and optimization for Ethereum smart contracts," in *Tools and Algorithms for the Construction and Analysis of Systems* (Lecture Notes in Computer Science), vol. 12079, A. Biere and D. Parker, Eds. Cham, Switzerland: Springer, 2020, doi: 10.1007/978-3-030-45237-7_7.

[140] L. Mazurek, "EthVer: Formal verification of randomized Ethereum smart contracts," in *Financial Cryptography and Data Security. FC 2021 International Workshops* (Lecture Notes in Computer Science), vol. 12676, M. Bernhard, Ed. Berlin, Germany: Springer, 2021, doi: 10.1007/978-3-662-63958-0_30.

[141] P. Praitheeshan, L. Pan, X. Zheng, A. Jolfaei, and R. Doss, "SolGuard: Preventing external call issues in smart contract-based multi-agent robotic systems," *Inf. Sci.*, vol. 579, pp. 150–166, Nov. 2021, doi: 10.1016/j.ins.2021.08.007.

[142] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on Ethereum smart contracts (SoK)," in *Principles of Security and Trust* (Lecture Notes in Computer Science), vol. 10204, M. Maffei and M. Ryan, Eds. Berlin, Germany: Springer, 2017, doi: 10.1007/978-3-662-54455-6_8.

[143] (Mar. 2021). *SolidiFI Benchmark*. [Online]. Available: https://github.com/smartbugs/SolidiFI-benchmark

SATPAL SINGH KUSHWAHA (Member, IEEE) received the B.E. degree in information technology from the University of Rajasthan, in 2006, and the M.Tech. degree in computer science and engineering from Rajasthan Technical University, Kota, in 2012. He is currently a Ph.D. Research Scholar with the Department of Computer Science and Engineering, Manipal University Jaipur. As an author, he published three books and researching multiple areas in network and security, information retrieval, and image processing. He also published several research papers in reputed conferences and journals. During his tenure, he has enhanced the classroom's learning environment, academic, and professional programs to cater to changing industry needs in a dynamic business environment through a good mix of theoretical and industry training modules. His research interests include blockchain technology and ethereum smart contract. He is an Active Member of IEI. He has cleared various international certification programs, such as Microsoft Certified Professional using C# (MCP).

SANDEEP JOSHI (Senior Member, IEEE) received the B.E., M.Tech., and Ph.D. degrees in computer science and engineering. He is currently a Senior Academician with experience in providing visionary leadership to students & imparting quality management education to a wide range of aspiring professionals. He mentored and guided the students towards the successful future as a Professor at Manipal University Jaipur, India. He possess profound knowledge of computer networks, network protocol, computer network & security, advance data structure & algorithms, web technology, and system software engineering with passion for learning. He always tries to keep myself up to date with the latest trends and techniques of the industry. During his tenure, he has enhanced the learning environment in the classroom, designed academic/professional programs to cater to changing industry needs in a dynamic business environment through a good mix of theoretical and industry training modules and worked with lecturers to develop and maintain high curriculum standards, develop mission statements, and set performance goals and objectives. He has proven competencies in designing course curriculum and implementing innovative instructional methodologies and equipped to provide authoritative leadership to both academic as well as administrative personnel in the wider issues of running/managing an academic institution of higher learning.

DILBAG SINGH (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the Thapar Institute of Engineering and Technology, India, in 2019. He worked as an Assistant Professor in three well-known universities of India, such as Chandigarh University, Manipal University Jaipur, and Bennett University. He has published more than 80 research articles in SCI/SCIE indexed journals. He has also submitted five patents, published three books, and two book chapters. His H-index is 32. His research interests include image processing, computer vision, deep learning, metaheuristic techniques, and information security. He has also acted as a lead guest editor/editorial board member of many SCI/SCIE indexed journals. He was in the top 2% list issues by "World Ranking of Top 2% Scientists" in 2021.

MANJIT KAUR (Member, IEEE) received the Master of Engineering degree in information technology from Punjab University, Chandigarh, India, in 2011, and the Ph.D. degree from the Thapar Institute of Engineering and Technology, Patiala, India, in 2019. She worked as an Assistant Professor in three well-known universities of India, such as Chandigarh University, Mohali, India, Manipal University Jaipur, Jaipur, India, and Bennett University, Greater Noida, India. In 2021, she moved to the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju, South Korea, where she is currently affiliated. Her research interests include post-quantum cryptography, fully homomorphic encryption, and privacy-preserving machine learning, wireless sensor networks, digital image processing, and metaheuristic techniques. She was in the top 2% list issues by "World Ranking of Top 2% Scientists," in 2021. She was part of the 14 Web of Science/Scopus indexed conferences.

HEUNG-NO LEE (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of California at Los Angeles, Los Angeles, CA, USA, in 1993, 1994, and 1999, respectively. He was with HRL Laboratories, LLC, Malibu, CA, USA, as a Research Staff Member, from 1999 to 2002. From 2002 to 2008, he was an Assistant Professor with the University of Pittsburgh, Pittsburgh, PA, USA. In 2009, he moved to the School of Electrical Engineering and Computer Science, Gwangju Institute of Science and Technology, Gwangju, South Korea, where he is currently affiliated. His research interests include information theory, signal processing theory, blockchain, communications/networking theory, and their application to wireless communications and networking, compressive sensing, future internet, and brain–computer interface. He has received several prestigious national awards, including the Top 100 National Research and Development Award in 2012, the Top 50 Achievements of Fundamental Researches Award in 2013, and Science/Engineer of the Month (January 2014).