A Survey of Security Vulnerabilities and Detection Methods for Smart Contracts

Jingqi Zhang[1], Xin Zhang[1], Zhaojun Liu[1], Fa Fu[1⋆],
Jianyu Nie[2], Jianqiang Huang[3], and Thomas Dreibholz[4]

[1] Hainan University, China, fufa@hainanu.edu.cn
[2] International Business Beijing Foreign Studies University, China
[3] China Telecom Corporation Limited, Hainan Branch, China
[4] Simula Metropolitan Centre for Digital Engineering, Norway, dreibh@simula.no

Abstract. At present, smart contracts cannot guarantee absolute security, and they have exposed many security issues and caused incalculable losses. Due to the existence of these security vulnerabilities, researchers have designed many detection and classification tools to identify and discover them. In this article, we present a classification of smart contract security vulnerabilities based on a large number of detailed articles. Then, we introduce the latest smart contract vulnerability detection methods, summarize the process model of detection tools based on artificial intelligence methods, and compare and analyze various detection tools. Finally, we provide an outlook on future research directions based on the current status of smart contract security.

Keywords: Blockchain, Smart Contracts, Detection Methods, Security Vulnerabilities

1 Introduction

Smart contracts are a special protocol used in the development of contracts within the blockchain. It allows for trusted transactions without a third party and ensures that these transactions are traceable and irreversible. The idea of Smart Contracts were introduced and defined by Nick Szabo [1] . Although smart contracts have a high level of security, there are still unscrupulous individuals who exploit the vulnerabilities of smart contracts to gain illegal benefits. For example,there is a case of an attack in 2021, Hackers have stolen some $600 million in cryptocurrency from the decentralized finance platform Poly Network[5]. This study explores the latest methods and tools for blockchain-based smart contract vulnerability detection. In addition, we analyze the features of these detection tools and compare them. Finally, based on the analysis, we discuss the research directions about the future security of smart contracts.

⋆ Corresponding author: Fa Fu
[5] Poly Network: https://edition.cnn.com/2021/08/11/tech/crypto-hack/index.html.

2 Security Issues

2.1 Smart Contract Platform Vulnerability

Oracle Oracle's security problem lies in the fact that its nodes are responsible for critical data stored securely on the blockchain. However, Oracle nodes are the only source of real data and are susceptible to attacks, manipulation, and compromise, so it is also a security issue for smart contracts [2].

IOTA (Internet of Things Application) is a distributed ledger technology (DLT) specifically designed for the Internet of Things (IoT). But it still carries the risk of being attacked, as well as its custom hashing algorithms. Some researchers found that there were collision issues with it. Bitcoin has Transaction malleability, and other popular platforms will be attacked as they get more activity [3].

2.2 Smart Contract Code Vulnerability

Unchecked Return Value In solidity language, some functions will generate unreasonable return values when executed or called, and these wrong return values will lead to logical errors in the operation of smart contracts.

tx.origin It is a solidity global variable that contains the address of the originator of the current transaction. However, using tx.origin may pose security issues. One potential security threat is spoofing smart contracts by spoofing the sender address of a transaction. If the smart contract relies on tx.origin to determine the sender of the transaction, this vulnerability can be exploited by the attacker [4].

Integer Overflow and Underflow Integer overflows are relatively common security vulnerability. If the integer variable exceeds the valid range during operation, an integer overflow error will occur. Among them, arithmetic, truncation, and signed overflow are the main problems of smart contracts [5].

2.3 Blockchain Vulnerability

Reentrancy Vulnerability It is a type of security vulnerability that can occur in smart contracts. It happens when an attacker exploits a contract's method that calls another contract without completing its own internal processing [6].

Timestamp Dependency Miners can manipulate the timestamps of transactions to exploit vulnerabilities. In order to benefit from transactions, miners can change the timestamps that are most favorable to their operations, so the different benefits for miners can lead to security issues [7].

Time Constraints In general, time constraints are implemented through timestamps, which require the consent of all miners. At the same time, all transactions within a block share the same timestamp. This mechanism enables all contracts to maintain a consistent state, but it also risks being attacked. Because the miner creating a new block can choose either timestamp, when the miner holds shares [8].

Generating Randomness Generating random numbers are intended to increase the security of smart contracts, but even then, there are still some problems of security vulnerabilities. The security issue of generating random numbers in smart contracts is of great importance, because the predictability of random numbers may leave vulnerabilities for hacking attacks [9].

3 Security Vulnerability Analysis Tools in Smart Contracts

The tamper-evident nature of smart contracts has both advantages and disadvantages. We analyze the security vulnerability analysis tools of smart contracts into two categories, static analysis methods [10–20] and dynamic analysis methods [21–29]. In addition, this paper also analyzes the dynamic static combined detection tool [30] was analyzed in this paper. The latest detection results are also summarized and analyzed according to this classification.

3.1 Static Analysis

Traditional smart contract vulnerability detection tools are basically based on fixed inspection rules [12], at the same time, traditional detection tools also have numerous problems. Therefore, in following search, a large proportion of tools using deep learning methods for vulnerability detection which can avoid this problem.

Xuesen Zhang et al. proposed a Bi-LSTM (Bi-directional Long Short-Term Memory) neural network based approach [10]. This method vectorizes the code of the smart contract and uses it as input for vulnerability detection. As LSTM is a forward network, the authors add backpropagation operators to obtain Bi-LSTM neural network operators. However, for new types of defects, the neural network needs to be retrained. On the other hand, Huiwen Yang et al. proposed a detection method [11] which is based on abstract syntax trees (AST). It extracts the features of smart contracts from AST, divides an AST into multiple ASTs based on the types of functions, state variables, and function modifiers, and then trains the model to detect vulnerabilities in smart contracts.

An approach to use graph neural networks (GNN) for smart contract vulnerability detection is suggested by Yuan Zhuang et al. [12], they proposed a degree-free graph convolutional neural network (DR-GCN) and a new temporal message propagation function (TMP) that learns from the normalized graph to vulnerability detection.

In contrast, Ziling Wang et al. proposed the GVD-net (Graph embedding-based Machine Learning Model for Smart Contract Vulnerability Detection) model [13] to detect vulnerabilities in smart contracts. The whole GVD-net is divided into a preprocessing part, a backbone network and a detection part. But it can only detect three types of vulnerabilities, namely arithmetic problems, access control and asset freezing.

There are still some methods proposed by researchers. Ran Guo et al. proposed a model based on twin networks (SCVSN) [14] for smart contract vulnerability detection. SCVSN combines twin networks with deep learning models and has a high level of accuracy for reentrant vulnerabilities. But it only uses reentrant vulnerabilities as case study, and it does not have the ability to detect integer overflow vulnerabilities and timestamp vulnerabilities.

Lejun Zhang et al. proposed a serial-parallel convolutional bidirectional gated recurrent network model (SPCBIG-EC) fusion integrated classifier [15] that can show excellent performance advantages in multi-task vulnerability detection. Meanwhile, the authors propose a CNN structure, string-parallel CNN (SPCNN), applicable to the above serial hybrid model. However, when the vulnerabilities are mixed, the accuracy is lower and the sensitivity is bad. They also proposed a hybrid model-based model called CBGRU [16], which combines different word embeddings and different deep learning methods to detect smart contract vulnerabilities. In the experiments, CBGRU performed excellent in detecting infinite loop vulnerabilities and so on. But the accuracy of the model in detecting smart contract vulnerabilities with obvious features is significantly higher than that of vulnerabilities with less obvious features.

Zhipeng Gao proposed a deep learning method called SmartEmbed [17] for detecting vulnerabilities in smart contracts, which detects code clones and problem points in smart contracts. In SmartEmbed, the information retrieval process has been accelerated in matrix computation, code embedding, and database access.

In Ethereum and some blockchain networks, the gas in smart contracts acts as an indicator to measure the computational effort and resource consumption. It ensures fair compensation for miners and helps maintain network security and efficiency. Asem Ghaleb et al. proposed a bytecode-based taint tracking detection tool, eTainter [18], which is a static analyzer that specifically targets gas-related vulnerabilities in smart contracts. But this tool has some limitations, as it cannot detect unbounded loop vulnerabilities that depend on data items, and some contracts have timeout issues.

Clara Schneidewind et al. proposed a tool called eThor [19], the first well-established automated static analyzer based on EVM (Ethereum Virtual Machine) bytecode. eThor takes as input the bytecode and contract-parameterized HoRSt specification, which is a framework for the specification and implementation of static analyses based on Horn clause resolution. And later goes through several stages of analysis to measure vulnerabilities.

Weimin Chen proposed a semantic-awareness-based tool, SADPonzi [20], to identify Ponzi schemes in Ether smart contracts. It uses a heuristic-guided sym-

bolic execution technique. Experimental tests show that SADPonzi outperforms the current so method with 100% accuracy and recall, but it also has the limitation that it cannot be used to detect new Ponzi smart contracts or less popular Ponzi schemes.

3.2 Dynamic Analysis

Dynamic analysis is used to detect, track, and analyze the running kind of smart contracts to understand their behavior, performance, and security. Common dynamic vulnerability detection methods are mainly dynamic symbolic execution, fuzzy testing, dynamic taint, etc. In recent years, with the development of deep learning, tools for detecting smart contract vulnerabilities using machine learning have also gradually emerged. In the following research, we will analyze and introduce several common dynamic detection tools as well as the latest detection tools.

A tool called Oyente is proposed by Loi Luuet al. [21], which uses symbolic execution for the detection of smart contract security vulnerabilities. This tool follows a modular design, which consists of four main components, CFGbuilder (Control Flow Graph), Explorer, CoreAnalysis, and Validator. It does not fully mimic the execution environment of Ether, so it does not yet reach the expected level.

Bo Jiang et al. proposed a tool called ContractFuzzer [22], which is the tool that first uses fuzzy testing to test the security vulnerabilities of Ethernet smart contracts. Compared with the Oyente, this tool can detect 7 types of vulnerabilities and its leakage rate is relatively reduced.

Another tool called EasyFlow [23] is a smart contract vulnerability detection tool based on taint analysis proposed by Jianbo Gao et al. It focuses on such vulnerabilities as integer overflows. It can not only classify smart contracts into secure contracts and contracts with overflows, but also automatically generate transactions that trigger overflows.

Besides, Yuhe Huang et al. proposed a tool called EOSFuzzer [24], a generic black-box fuzz testing framework, which is the first fuzzing framework for detecting vulnerabilities in EOSIO (Enterprise Operation System Input Output) smart contracts.

Mojtaba Eshghie et al. proposed a monitoring framework Dynamit, which is a tool that first uses machine learning to analyze the dynamic execution of smart contracts [25]. The tool consists of a monitor and a detector, where the monitor is used to collect information about the transaction and to discern whether the transaction is harmful or not. As the same time, Mengjie Ding et al. proposed HFContractFuzzer [26], a tool based on fuzzy technique for testing Hyperledger Fabric smart contracts. HFContractFuzzer is effective, but it still has drawbacks and limitations, such as performance degradation problems, inability to verify its superiority, etc.

Except for all the tools mentioned above, a novel reinforcement learning-based vulnerability guided fuzzifier RLF [27] was proposed by Jianzhong Su et al. , which is used to motivate vulnerable transaction sequences to detect

vulnerabilities in smart contracts. Meanwhile, the Park [28], a tool first using general framework for parallel forked symbolic execution of smart contracts, was proposed by Peilin Zheng et al. It proposes the use of multiple CPU cores to improve symbol execution efficiency based on symbol tools. And WASAI [29], a new concolic fuzzer for discovering vulnerabilities in Wasm (WebAssembly) smart contracts proposed by Weimin Chen et al. has been tested as a state-of-the-art tool, but it still has some shortcomings, such as the trade-off between scalability and efficiency, the seeds chosen by WASAI are not the most suitable, the benchmarks need to be improved, etc.

3.3 Dynamic and Static Combined Analysis

Currently, there are few tools that can combine static analysis with dynamic analysis, but HFCCT (Hyperledger Fabric Chaincodes Test) [30] is one of them. Peiru Li et al. proposed HFCCT, a vulnerability detection tool that combines dynamic symbolic execution and static abstract syntax trees for detecting Golang chain code vulnerabilities. After testing, HFCCT is significantly more efficient than HFContractFuzzer. HFCCT can detect more vulnerabilities compared to HFContractFuzzer, but it still has two kinds of vulnerabilities that cannot be detected, they are Reified Object Addresses and Cross Channel Chaincode Invocation. Besides, further optimization is needed to be improved.

3.4 Comprehensive Comparison

We conducted a comparative analysis of the detection tools mentioned above and found that the use of deep learning methods for smart contract security vulnerability detection has been increasing in recent years, and we summarized all the tools.

About 70% of the static analysis tools in our survey are based on machine learning or deep learning. We summarize the model of smart contract vulnerability detection tools using artificial intelligence technology, as shown in Figure 1. And we summarize the data of AI-based detection tools. All data are compared with the reentrancy vulnerability as the detection object, and the detection indicators include: Accuracy, Precision, Recall, and F1 as shown in Table 1.

First, these methods using artificial intelligence technology clean the source code of smart contracts, segment words, and convert the code into vectors to build a dataset. Then, the dataset is divided into training set and test set by labeling. Finally, the training set was used to train the constructed model, and the test set was used to obtain the analysis report.

[11, 13] improved the part of converting the code into vectors. Detection based on AST [11] improved the word embedding. The AST is divided into state variables, function modifiers, and functions, and all of them are converted into vectors for model training to obtain better training results. GVD-net [13] is an improvement on word embedding. The authors treat the variables and relationships of solidity code as a non-Euclidean graph, and use the Node2Vec [31] algorithm to construct vectors.

Table 1. Experimental data based on artificial intelligence detectors

Tool	Analysis Method	Accuracy	Precision	Recall	F1
Detection based on Bi-LSTM Neural Network [10]	Machine Learning	81.4%	100%	41.%	58.3%
Detection based on Abstract Syntax Tree [11]	Abstract Syntax Tree (AST)	99.6 %	98.8%	90.4%	94.4%
Detection Using Graph Neural Networks [12]	Machine Learning	84.4%	74.1%	82.6%	78.1%
SCVSN [14]	Deep Learning	96.0 %	94.2%	96.0%	94.8%
SPCBIG-EC [15]	Deep Learning	96.7%	94.6%	-	96.74%
CBGRU [16]	Deep Learning	93.3 %	96.3%	86.0%	90.9%
Dynamit [25]	Machine Learning	94.0%	-	94.0%	93.0%

[10, 14, 15, 17] improved the detection part. The Detection based on Bi-LSTM Neural Network [10] built their model based on Bi-LSTM network. SCVSN [14] improves the network structure and combines two networks to train the model. The SCVSN Siamese network structure is the combination of Siamese network and LSTM neural network. SPCBIG-EC [15] improves the network structure, combines the serial hybrid model of CNN and RNN, and combines the feature extraction advantages of CNN and the characteristics of RNN emphasizing the time dimension. In the detection part of the network, SmartEmbed [17] uses deep learning and similarity checking technology to unify clone detection and bug detection efficiently and accurately, so as to improve the efficiency and accuracy.

[12, 16] have improved the overall model. Detection Using Graph Neural Networks [12] made changes to word embeddings, The authors use a graph generation phase, which extracts the control flow and data flow semantics from the source code and explicitly models the fallback mechanism. Besides, they use a graph normalization phase inspired by k-partite graph. At the same time, they use graph neural networks for vulnerability detection to replace traditional neural networks such as CNNs. CBGRU [16] improves both the word embedding model and the detection model. The word embedding model uses Word2Vec and FastText, and the detection model combines five deep models CNN, LSTM, GRU, BiGRU and BiLSTM for detection, making full use of the advantages of five networks to improve its vulnerability detection ability.

In terms of dynamic analysis tools [21–23, 25, 27, 28], there is no unified process. The dynamic analysis method detects the contract in the execution process, so some dynamic analysis methods do not require source code [25]. EOSIO is a typical public blockchain platform. WASAI [29] and EOSFuzzer [24] analyzed EOSIO.

HFContractFuzzer [26] HFCCT [30] is aimed at Hyperledger Fabric platform for vulnerability detection, and HFCCT adopts the way of dynamic and static

collection. In HFCCT, the authors used 15 chain codes to compare with HFContractFuzzer. The test results show that the vulnerability detection efficiency of HFCCT is higher than that of HFContractFuzzer, and the display of errors is clearer. We can see that the dynamic and static analysis method is more efficient than the single analysis method.

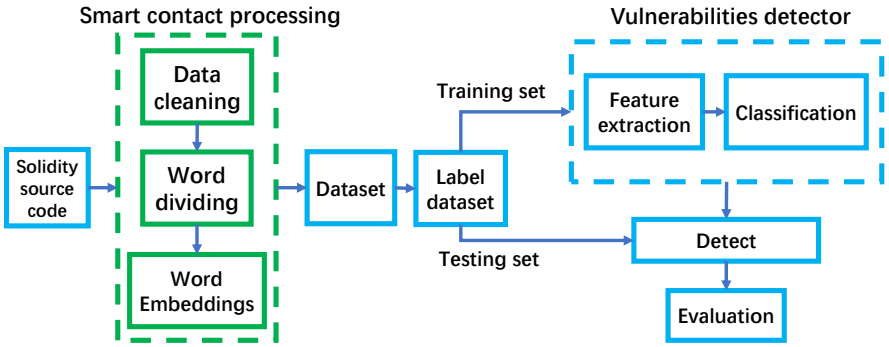

Fig. 1. Model of smart contract vulnerability detection tools using artificial intelligence

4 Development Direction

From the perspective of whether to run the contract, we divide all smart contract vulnerability detection tools into three categories. Through analysis and comparison, we draw the following conclusions:

4.1 Deep Learning

At present, machine learning and deep learning are increasingly used in vulnerability detection. In the future, Chat-GPT may be a powerful assistant for future smart contract vulnerability detection. However, a common drawback of these detection tools [13–15, 17] is that they detect fewer kinds of vulnerabilities, which may be related to the small number of existing large standard datasets.

4.2 Combination of Static and Dynamic Analysis

When the contract is complex, dynamic analysis has more advantages than static analysis, because the relationship between many vulnerabilities is complex, which can not be easily analyzed by simply inspecting the code files. Moreover, the coverage of dynamic analysis tools is not high. Therefore, how to improve the coverage of dynamic analysis tools is a big problem.

4.3 Platform and Language

Most of the current detection tools are aimed at the Ethereum platform and Solidity language. The running platform such as EOS, Hyperledger Fabric, etc., and the programming language such as Vyper language, etc. The smart contract security vulnerability of these platforms and languages are less studied, so it is also the direction of future research.

5 Conclusion

In the era when blockchain is more popular, smart contracts provide security for blockchain, but security of vulnerabilities in smart contracts cannot be ignored. We hope that the re-classification and summary of security vulnerabilities can also make people have a better understanding of the security issues in smart contracts. Through the analysis and comparison of detection tools, we believe that the security detection technology of smart contracts will become more and more mature, and they can also have higher efficiency and better vulnerability detection ability. Therefore, with the update and progress of technology, the security problem of blockchain will be more and more guaranteed.

Acknowledgment

This work was funded by Haikou Science and Technology Plan Project (2022-007).

References

1. Shuai Wang, Liwei Ouyang, Yong Yuan, Xiaochun Ni, Xuan Han, and Fei-Yue Wang. Blockchain-enabled smart contracts: architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(11):2266–2277, 2019.
2. Rohini Pise and Sonali Patil. A deep dive into blockchain-based smart contract-specific security vulnerabilities. In *2022 IEEE International Conference on Blockchain and Distributed Systems Security (ICBDS)*, pages 1–6. IEEE, 2022.
3. Purathani Praitheeshan, Lei Pan, Jiangshan Yu, Joseph Liu, and Robin Doss. Security analysis methods on ethereum smart contract vulnerabilities: a survey. *arXiv preprint arXiv:1908.08605*, 2019.
4. Chihiro Kado, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. An empirical study of impact of solidity compiler updates on vulnerabilities. In *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 92–97. IEEE, 2023.
5. Jinlei Sun, Song Huang, Changyou Zheng, Tingyong Wang, Cheng Zong, and Zhanwei Hui. Mutation testing for integer overflow in ethereum smart contracts. *Tsinghua Science and Technology*, 27(1):27–40, 2021.

6. Han Kun, Wu Bo, and Xin Dan. A return-value-unchecked vulnerability detection method based on property graph. In *Recent Developments in Intelligent Systems and Interactive Applications: Proceedings of the International Conference on Intelligent and Interactive Systems and Applications (IISA2016)*, pages 114–123. Springer, 2017.
7. Alexander Mense and Markus Flatscher. Security vulnerabilities in ethereum smart contracts. In *Proceedings of the 20th international conference on information integration and web-based applications & services*, pages 375–380, 2018.
8. Massimo Bartoletti and Livio Pompianu. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*, pages 494–509. Springer, 2017.
9. Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Weili Chen, Xiangping Chen, Jian Weng, and Muhammad Imran. An overview on smart contracts: Challenges, advances and platforms. *Future Generation Computer Systems*, 105:475–491, 2020.
10. Xuesen Zhang, Jianhua Li, and Xiaoqiang Wang. Smart contract vulnerability detection method based on bi-lstm neural network. In *2022 IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA)*, pages 38–41. IEEE, 2022.
11. Huiwen Yang, Jiaming Zhang, Xiguo Gu, and Zhanqi Cui. Smart contract vulnerability detection based on abstract syntax tree. In *2022 8th International Symposium on System Security, Safety, and Reliability (ISSSR)*, pages 169–170. IEEE, 2022.
12. Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. Smart contract vulnerability detection using graph neural networks. In *Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence*, pages 3283–3290, 2021.
13. Ziling Wang, Qinyuan Zheng, and Ye Sun. Gvd-net: Graph embedding-based machine learning model for smart contract vulnerability detection. In *2022 International Conference on Algorithms, Data Mining, and Information Technology (ADMIT)*, pages 99–103. IEEE, 2022.
14. Weijie Chen, Ran Guo, Guopeng Wang, Lejun Zhang, Jing Qiu, Shen Su, Yuan Liu, Guangxia Xu, and Huiling Chen. Smart contract vulnerability detection model based on siamese network. In *International Conference on Smart Computing and Communication*, pages 639–648. Springer, 2022.
15. Lejun Zhang, Yuan Li, Tianxing Jin, Weizheng Wang, Zilong Jin, Chunhui Zhao, Zhennao Cai, and Huiling Chen. Spcbig-ec: a robust serial hybrid model for smart contract vulnerability detection. *Sensors*, 22(12):4621, 2022.
16. Lejun Zhang, Weijie Chen, Weizheng Wang, Zilong Jin, Chunhui Zhao, Zhennao Cai, and Huiling Chen. Cbgru: A detection method of smart contract vulnerability based on a hybrid model. *Sensors*, 22(9):3577, 2022.
17. Zhipeng Gao. When deep learning meets smart contracts. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1400–1402, 2020.
18. Asem Ghaleb, Julia Rubin, and Karthik Pattabiraman. etainter: detecting gas-related vulnerabilities in smart contracts. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 728–739, 2022.
19. Clara Schneidewind, Ilya Grishchenko, Markus Scherer, and Matteo Maffei. ethor: Practical and provably sound static analysis of ethereum smart contracts. In *Pro-*

ceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pages 621–640, 2020.
20. Weimin Chen, Xinran Li, Yuting Sui, Ningyu He, Haoyu Wang, Lei Wu, and Xiapu Luo. Sadponzi: Detecting and characterizing ponzi schemes in ethereum smart contracts. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 5(2):1–30, 2021.
21. Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269, 2016.
22. Bo Jiang, Ye Liu, and Wing Kwong Chan. Contractfuzzer: Fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 259–269, 2018.
23. Jianbo Gao, Han Liu, Chao Liu, Qingshan Li, Zhi Guan, and Zhong Chen. Easyflow: Keep ethereum away from overflow. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 23–26. IEEE, 2019.
24. Yuhe Huang, Bo Jiang, and Wing Kwong Chan. Eosfuzzer: Fuzzing eosio smart contracts for vulnerability detection. In *Proceedings of the 12th Asia-Pacific Symposium on Internetware*, pages 99–109, 2020.
25. Mojtaba Eshghie, Cyrille Artho, and Dilian Gurov. Dynamic vulnerability detection on smart contracts using machine learning. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, EASE '21, page 305–312, New York, NY, USA, 2021. Association for Computing Machinery.
26. Mengjie Ding, Peiru Li, Shanshan Li, and He Zhang. Hfcontractfuzzer: Fuzzing hyperledger fabric smart contracts for vulnerability detection. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, EASE '21, page 321–328, New York, NY, USA, 2021. Association for Computing Machinery.
27. Jianzhong Su, Hong-Ning Dai, Lingjun Zhao, Zibin Zheng, and Xiapu Luo. Effectively generating vulnerable transaction sequences in smart contracts with reinforcement learning-guided fuzzing. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–12, 2022.
28. Peilin Zheng, Zibin Zheng, and Xiapu Luo. Park: Accelerating smart contract vulnerability detection via parallel-fork symbolic execution. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 740–751, 2022.
29. Weimin Chen, Zihan Sun, Haoyu Wang, Xiapu Luo, Haipeng Cai, and Lei Wu. Wasai: uncovering vulnerabilities in wasm smart contracts. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 703–715, 2022.
30. Peiru Li, Shanshan Li, Mengjie Ding, Jiapeng Yu, He Zhang, Xin Zhou, and Jingyue Li. A vulnerability detection framework for hyperledger fabric smart contracts based on dynamic and static analysis. In *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, pages 366–374, 2022.
31. Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.