MANDO-GURU: Vulnerability Detection for Smart Contract Source Code by Heterogeneous Graph Embeddings

Hoang H. Nguyen
ehoang@l3s.de
L3S Research Center,
Leibniz Universität Hannover
Hannover, Germany

Nhat-Minh Nguyen
nmnguyen@smu.edu.sg
Singapore Management University
Singapore

Hong-Phuc Doan
phuc.dh194647@sis.hust.edu.vn
Hanoi University of
Science and Technology
Hanoi, Vietnam

Zahra Ahmadi
ahmadi@l3s.de
L3S Research Center,
Leibniz Universität Hannover
Hannover, Germany

Thanh-Nam Doan
me@tndoan.com
Independent Researcher
Atlanta, Georgia, USA

Lingxiao Jiang
lxjiang@smu.edu.sg
Singapore Management University
Singapore

ABSTRACT

Smart contracts are increasingly used with blockchain systems for high-value applications. It is highly desired to ensure the quality of smart contract source code before they are deployed. This paper proposes a new deep learning-based tool, MANDO-GURU, that aims to accurately detect vulnerabilities in smart contracts at both coarse-grained contract-level and fine-grained line-level. Using a combination of control-flow graphs and call graphs of Solidity code, we design new heterogeneous graph attention neural networks to encode more structural and potentially semantic relations among different types of nodes and edges of such graphs and use the encoded embeddings of the graphs and nodes to detect vulnerabilities. Our validation of real-world smart contract datasets shows that MANDO-GURU can significantly improve many other vulnerability detection techniques by up to 24% in terms of the F1-score at the contract level, depending on vulnerability types. It is the first learning-based tool for Ethereum smart contracts that identify vulnerabilities at the line level and significantly improves the traditional code analysis-based techniques by up to 63.4%. Our tool is publicly available at https://github.com/MANDO-Project/ge-sc-machine. A test version is currently deployed at http://mandoguru.com, and a demo video of our tool is available at http://mandoguru.com/demo-video.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**;
• **Security and privacy** → **Software security engineering**.

KEYWORDS

heterogeneous graphs, graph neural networks, vulnerability detection, smart contracts, Ethereum blockchain

ESEC/FSE '22, November 14–18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9413-0/22/11…$15.00
https://doi.org/10.1145/3540250.3558927

ACM Reference Format:
Hoang H. Nguyen, Nhat-Minh Nguyen, Hong-Phuc Doan, Zahra Ahmadi, Thanh-Nam Doan, and Lingxiao Jiang. 2022. MANDO-GURU: Vulnerability Detection for Smart Contract Source Code by Heterogeneous Graph Embeddings. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '22), November 14–18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3540250.3558927

1 INTRODUCTION

Smart contracts are increasingly used for creating and enforcing high-value business transactions, such as stock purchases, life insurance certificates, inventory management, and supply-chain payment and tracking since their first introduction in the 1990s by Nick Szabo [34]. Blockchain-based smart contracts provide append-only, non-repudiation and transparency for their executions, preventing double-spending or breaches of contracts. On the other hand, like traditional software programs, smart contracts can still contain programming bugs or vulnerabilities intentionally or unintentionally created by their programmers[1]. Such bugs or vulnerabilities may have more serious impact than those in traditional software as the buggy smart contracts, once deployed to a blockchain, are irreversible unless self-destructed, and may lead to huge financial losses if misused by attackers. Thus, it is highly desirable to have methods for detecting vulnerabilities in smart contract code during their early development and before deployment.

In this paper, we propose a new tool with a new method for representing smart contracts as specialized graphs and learning their patterns automatically via graph neural networks on a large scale to detect vulnerabilities at both the line-level and contract-level accuracy. In particular, (1) we represent Ethereum smart contract source code written in Solidity as *heterogeneous contract graphs* that combine control-flow graphs (CFGs) and call graphs (CGs) using unique properties of Solidity to capture contract code semantics, and (2) we design specialized metapaths for the graphs and build heterogeneous attention graph neural networks to learn multi-level embeddings of the contract code in various levels of granularity, which are then used together with known instances of smart contract vulnerabilities to train classifiers that can recognize

[1] In this paper, we use the two terms "bug" and "vulnerability" interchangeably.

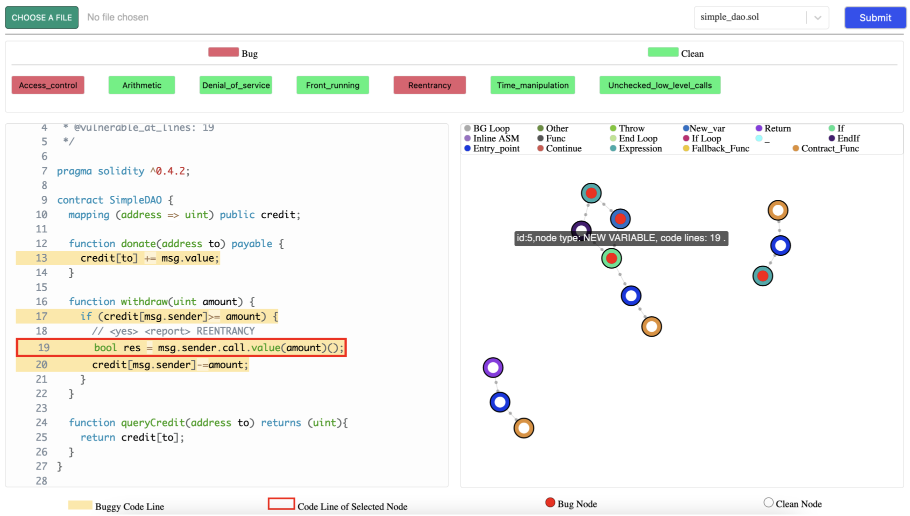

Figure 1: A sample vulnerability detection page of MANDO-GURU for an Ethereum smart contract includes summary detection results for seven bug types (Top), code snippet (Left), and its corresponding heterogeneous contract graph (Right). Line 19 with a yellow background is the root cause of a Reentrancy bug (Left); the nodes containing the Reentrancy bug are highlighted with red (Right).

vulnerabilities accurately in new smart contract code at both line-level and contract-level. Our tool is named MANDO-GURU. We have constructed a dataset containing both buggy and clean smart contracts, and compared MANDO-GURU with several state-of-the-art and conventional baselines. Our validation results show that MANDO-GURU outperforms the baselines in both contract- and line-level vulnerability detection with significant improvements.

2 RELATED WORK

Existing studies have proposed methods to detect vulnerabilities, either based on traditional program analysis, testing, verification techniques (e.g., [6, 9, 10, 13, 15, 26, 28, 32, 33, 36, 37]), or based on machine learning and deep learning (e.g., [2, 4, 5, 7, 8, 16, 19, 22–25, 31, 40, 41, 43–45]). Traditional techniques often rely on some bug patterns manually defined by experts, leading to low scalability (as the techniques can be slow in performing extensive checks on large codebases for complex patterns), low generalizability (as new patterns need to be manually defined for new types of bugs or new programming languages for smart contracts). Deep learning techniques alleviate the problem by automatically learning bug patterns from certain representations of existing code, such as syntax trees, data-/control-dependency graphs, etc. Still, the learning-based techniques have treated the trees/graphs as flattened sequences or conventional graphs disjointing each other and has not utilized particular kinds of control flow and call relations in the contract code to capture their semantics more comprehensively. Moreover, they often treat nodes and edges in the tree- and graph-representations of source code *homogeneously*, ignoring fine-grained differences in their types and locations. There may be only one recent study on using heterogeneous graphs for source code representation [42], but it has not yet been applied to smart contracts. As a result, they could only search for coarse-grained whole-graph-level smart contract vulnerabilities, which are not accurate enough to locate the line-level locations of vulnerabilities. Besides, some approaches also apply graph neural networks for vulnerability detection, such as Devign [44] and IVDETECT [21]. However, they are designed for other languages and unsuitable for Solidity.

3 USAGE

Figure 1 illustrates MANDO-GURU's main user interface and core features. More specifically, after a user submits a Solidity source file using the submit button on the top, MANDO-GURU scans the input and summarizes the coarse-grained contract-level detection results of seven bug types (the red/green buttons near top). A red button indicates a bug type detected for the contract, and users can click it to show the fine-grained line-level detection results. On the left side of the figure, the source code lines containing detected bugs would be highlighted with a yellow background. The right side visualizes the corresponding heterogeneous contract graph of the input contract. If a node is detected having a bug, it is colored red. When users hover the pointer over a node, the node details will be shown, and when they click a node, the code lines relevant to that node will be marked with the red border on the left.

Besides the core features, MANDO-GURU also provides various statistics charts for general analyses of the generated heterogeneous contract graphs. In particular, after getting the detection results, users could click the "Show Statistics" button to get three extended charts, including the number of clean and buggy nodes, running time for coarse-grained and fine-grained detection, and the density of each bug type. We explain in detail the charts in our demo video.

4 TOOL DESIGN & IMPLEMENTATION

Figure 2 illustrates an overview of MANDO-GURU with three main components: *Backend*, *RESTful APIs*, and *Frontend*. Backend plays a vital role with several core sub-components such as heterogeneous representation for the generated graphs from input smart contracts, heterogeneous graph fusion, custom multi-metapaths extraction, heterogeneous graph neural network, and vulnerability detections in coarse-grained and fine-grained levels. The technical details of *Backend* are described in [29]. The Frontend component services

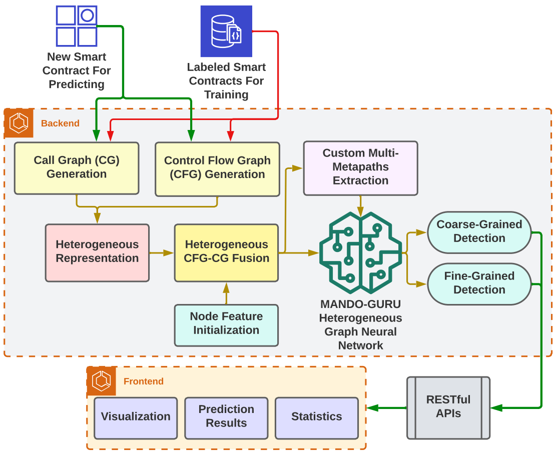

Figure 2: Overview of the MANDO-GURU Tool.

are used to visualize the prediction results and the statistics of the analyzed smart contracts. RESTful APIs are implemented as a bridge to communicate between the Backend and the Frontend.

4.1 Backend

4.1.1 Heterogeneous Representation for the Generated Control-Flow Graphs and Call Graphs. First, to generate the basic control-flow graphs and call graphs, we use Slither [13] to process the source code of each input Ethereum smart contract. Then, we convert the graphs into heterogeneous forms, called *heterogeneous control-flow graphs (HCFGs)* and *heterogeneous call graphs (HCGs)*, to represent the relations of different node and edge *types* and graph topologies. In particular, a heterogeneous graph is defined as a special graph consisting of multiple-type of nodes or edges. Unlike some recent studies [24, 45] that use only homogeneous graph structures and lead to loss of valuable information on the code semantics in smart contracts, one primary contribution of MANDO-GURU is to focus on capturing and retaining more structures and semantics of source code through our heterogeneous representations.

4.1.2 Fusion of Heterogeous Control-Flow Graphs and Heterogeous Call Graphs. An HCFG can represent each function in a smart contract, and it contains an entry node corresponding to the entry point/header of the function. Generally, a smart contract may be considered as a set of HCFGs since it consists of more than one function. The invocation relations among the functions in one contract or between contracts are represented by HCGs.

The structures of the heterogeneous graphs can be shared or combined to enrich information for graph learning. Hence, we design a sub-component as a core fusion of HCGs and HCFGs into a global graph. Accordingly, the HCG edges of a contract act as bridges to link the discrete HCFGs of the contract functions into a global fused graph. We call the fusion graphs as *heterogeneous contract graphs*. Intuitively, for each and every function node i in the call graph G_C, the function control-flow graph G^i_{CF} is attached to the function node i at the entry node of G^i_{CF}, and thus the call graph G_C is expanded with control-flow graphs to produce the heterogeneous contract graph G_{Fusion}. The heterogeneous graph generation also allows us to expand the generalizability of the proposed method to other programming languages (e.g., C/C++, Java) with minor modifications.

4.1.3 Node Feature Initialization. In the default setting of MANDO-GURU, the one-hot vectors based on node types are used to initialize node features. Besides, various state-of-the-art node embedding techniques can be plugged into MANDO-GURU to capture the graph topology and extract the node features. For a more comprehensive validation of the effectiveness of various initialization of node features, we use both embedding methods for homogeneous graphs (e.g., node2vec [18]) and embedding methods for heterogeneous graphs (e.g., metapath2vec [11]) (see Section 5).

4.1.4 Extraction of Custom Multi-Metapaths. A metapath θ is a path in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_l} A_{l+1}$, which defines relations R_i (i.e., edge types) from node types A_i to A_{i+1} in a heterogeneous graph. The **length** of θ is the number of relations in θ. We extract length-2 metapaths of each node type pair from a heterogeneous contract graph, since learning the extracted metapaths can be an effective way to learn the graph structures [11, 39]. Similar to the method used in HAN [39], we only focus on metapaths of length 2 to capture the relations between each node type pair and its neighbors and to prevent the explosion of metapaths when the generated heterogeneous contract graphs contain a dynamic number of node types (reaching eighteen in some large smart contracts, with five different connections per node type) and pre-defining all possible metapaths with any length according to all possible node types and edge types would lead to an exponential explosion of metapaths, increased data sparsity, and reduced accuracy in training data.

In addition, the heterogeneous contract graphs have mostly tree-like structures, with very few of their own back-edges induced by the LOOP-related statements in the smart contracts' source code, leading to the lack of metapaths connecting many types of leaf-node in the graphs. Therefore, we customize the length-2 metapaths by reflecting the relation R_i between adjacent nodes, from type A_i to type A_{i+1} and also from A_{i+1} to A_i to extract multiple-metapaths.

4.1.5 Heterogeneous Graph Neural Network. Our unique heterogeneous graph neural network learns to weigh the importance of every metapath and node by the node-level attention mechanism and can handle multiple dynamic custom metapaths without pre-defining the list of input metapaths. In particular, with the initialized node features (node embeddings) $e_i^{\phi_k}$ for each node i whose type is ϕ_k; then, we construct a corresponding weighted node feature by a linear transformation. Next, we measure the weight of the t-th metapath according to the node type ϕ_k of (i, j) pair by leveraging the self-attention mechanism [38] between i and j.

We concatenate all node embedding $M_i^{\phi_k}$ corresponding to all node type ϕ_k of all node i to generate a unified embedding vector for a node, which is used to train a *fine-grained* bug classifier. The average of all node embeddings in a graph is used as the graph embedding, which is used to train a *coarse-grained* bug classifier. Also, we employ the multi-layer perceptron (MLP) with a softmax activation function for predicting, with the inputs depending on the type of detection tasks. Moreover, the loss function for the training process is cross-entropy, and the parameters of our model are learned through back-propagation.

4.1.6 Coarse-Grained Detection and Fine-Grained Detection. First, MANDO-GURU classifies if a contract is clean or contains a type of vulnerabilities at the contract level by using coarse-grained graph classification. Next, MANDO-GURU identifies the actual locations of the vulnerabilities in the smart contract source code at the line level using fine-grained node classification. Providing line-level locations of vulnerabilities is one of our primary contributions, while the previous graph learning-based methods (e.g., [25, 45]) only report vulnerabilities at the contract or function level.

4.2 RESTful APIs and Frontend

MANDO-GURU is based on the FastAPI framework [30] to create our RESTful APIs as well as validation data to handle the requests and respond the detection results to the Frontend services. Also, we use a token for each request to validate and reduce the unexpected demands to our system via the basic HTTP authentication method. All RESTful APIs in MANDO-GURU are implemented and provided under POST methods. Besides, to ensure the MANDO's overall performance, we encode the source code of smart contracts to Base64 format before processing. Our APIs could be categorized into two groups depending on the request purposes from the Frontend component services: (1) Coarse-Grained requests for predicting whether a source code has any bug; and (2) Fine-Grained requests for getting the lines and nodes detected as having bugs.

Our Frontend web application is built on ReactJS [27] and ApexChartsJS [1] libraries. When users submit a source file to our web app, it scans through the file for a total of seven bug kinds supported and returns the summary and details of detection results for each bug type. We also provide some sample smart contracts in a dropdown menu, which may help the users who lack the Solidity source files to test MANDO-GURU more flexibly. The detection results are then visualized by interactive graphs and highlighted code snippets for users to double-check them easier.

5 TOOL VALIDATION

5.1 Setup

Our evaluaiton uses two tasks: (i) contract-level vulnerabilty detection; and (ii) line-level vulnerabilty detection. We combine the three following datasets for our training: **(1) Smartbugs Curated** [12, 14] **(2) SolidiFI-Benchmark** [17] and **(3) Clean Smart Contracts from Smartbugs Wild** [12, 14]. In total, we have 2,742 clean contracts and 493 annotated buggy contracts.

We use the following four state-of-the-art methods as the graph-based neural network comparison methods: *node2vec* [18]; *LINE* [35]; *Graph Convolutional Network (GCN)* [20]; and *metapath2vec* [11]. We use the output embeddings of the homogeneous and heterogeneous graph neural networks in two ways in our validation: First, directly as the baselines for the coarse-grained graph classification tasks and fine-grained node classification tasks. Second, each of the graph neural networks is plugged into MANDO-GURU as the topological graph neural network; the generated embeddings are considered the node features besides those based on the node-type one-hot vectors of the default setting and then fed to MANDO-GURU Heterogeneous Graph Neural Network (HGNN). We also used six detection tools built upon traditional software engineering techniques: *Manticore* [28]; *Mythril* [9]; *Oyente* [26]; *Securify* [37]; *Slither* [13]; and *Smartcheck* [36].

We use F1-score and Macro-F1 scores to measure the performance of our node/graph classification for the detection tasks. F1-score is used to validate the models' performance when finding bugs, and is also called *Buggy-F1*. Macro-F1 is considered to avoid biases in the clean and bug labels.

5.2 Empirical Results

Contract-Level Vulnerabilty Detection:[2]

- MANDO-GURU outperforms baseline GNNs. E.g., an improvement of 24% in both metrics is achieved by MANDO-GURU over the best baselines for detecting the Front Running type of bugs.
- The node feature generatation methods help MANDO-GURU outperform all the baselines; and it shows that our architecture is general for plugging in various kinds of GNNs.
- Being competible with Slither [13] makes MANDO-GURU more effectively with various versions of Solidity; MANDO-GURU is able to find newly-appeared bugs that graph learning methods [24, 45] struggle to achieve.

Line-Level Vulnerabilty Detection:

- MANDO-GURU outperforms conventional tools significantly with improvement up to 63.4% compared to the best performing tools for the Reentrancy type of bugs. It can be explained by (i) more CFG structures retained by our heterogeneous graphs; and (ii) the flexibility of our architecture.
- Our method beats the results of the baseline GNNs where the macro-F1 scores of our model is up to 20% higher than the ones of the baseline GNNs.
- Conventional detection tools perform well in detecting arithmetic bugs because they mostly use symbolic execution and such technique is suitable for detecting arithmetic bugs [3]. However, MANDO-GURU performance is still on par with the tools.

6 CONCLUSION

This paper presents a new tool, MANDO-GURU, for detecting vulnerabilities in Ethereum smart contracts written in Solidity. Our detection technique is new, based on a kind of *heterogeneous* attention graph neural networks that learn the embeddings of combined control-flow graphs (CFGs) and call graphs (CGs) of Solidity smart contract code. We can generate both node-level and graph-level embeddings of smart contracts and train classifiers to recognize various types of vulnerabilities in smart contracts at both the fine-grained line level and the coarse-grained contract level. Our validation on some datasets curated from the real world shows that MANDO-GURU can detect seven types of smart contracts more accurately on average than several baseline methods and thus is a promising complement to other vulnerability detection techniques.

Acknowledgments. This work was supported by the European Union's Horizon 2020 research and innovation program under grant agreement No. 833635 (project ROXANNE: Real-time network, text, and speaker analytics for combating organized crime, 2019-2022) and by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

[2] Due to the page limit, we only report highlights of our contributions and achievements here and shift more comprehensive evaluations to the arXiv version [29].

REFERENCES

[1] ApexCharts. 2022. *APEXCHARTS.JS: Modern & Interactive Open-source Charts.* https://apexcharts.com/

[2] Nami Ashizawa, Naoto Yanai, Jason Paul Cruz, and Shingo Okamura. 2021. Eth2Vec: learning contract-wide code representations for vulnerability detection on ethereum smart contracts. In *Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure.* 47–59.

[3] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. 2018. A Survey of Symbolic Execution Techniques. *ACM Comput. Surv.* 51, 3 (2018).

[4] Sicong Cao, Xiaobing Sun, Lili Bo, Ying Wei, and Bin Li. 2021. BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection. *Information and Software Technology* 136 (2021), 106576.

[5] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. 2021. Deep learning based vulnerability detection: Are we there yet. *IEEE Transactions on Software Engineering* (2021).

[6] Checkmarx. 2022. https://checkmarx.com/

[7] Xiao Cheng, Haoyu Wang, Jiayi Hua, Guoai Xu, and Yulei Sui. 2021. DeepWukong: Statically detecting software vulnerabilities using deep graph neural network. *ACM TOSEM* 30, 3 (2021), 1–33.

[8] Xiao Cheng, Haoyu Wang, Jiayi Hua, Miao Zhang, Guoai Xu, Li Yi, and Yulei Sui. 2019. Static detection of control-flow-related vulnerabilities using graph embedding. In *24th ICECCS*. IEEE, 41–50.

[9] ConsenSys. 2019. MythX Tech: Behind the Scenes of SmartContract Security Analysis. https://blog.mythx.io/features/mythx-tech-behind-the-scenes-of-smart-contract-analysis/. https://github.com/ConsenSys/mythril

[10] Coverity. 2022. https://scan.coverity.com/

[11] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 135–144.

[12] Thomas Durieux, João F Ferreira, Rui Abreu, and Pedro Cruz. 2020. Empirical review of automated analysis tools on 47,587 ethereum smart contracts. In *ACM/IEEE 42nd ICSE.* 530–541.

[13] Josselin Feist, Gustavo Grieco, and Alex Groce. 2019. Slither: a static analysis framework for smart contracts. In *IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain.* 8–15.

[14] João F Ferreira, Pedro Cruz, Thomas Durieux, and Rui Abreu. 2020. SmartBugs: a framework to analyze solidity smart contracts. In *35th IEEE/ACM ASE.* 1349–1352.

[15] Flawfinder. 2022. https://dwheeler.com/flawfinder/

[16] Zhipeng Gao, Lingxiao Jiang, Xin Xia, David Lo, and John Grundy. 2020. Checking smart contracts with structural code embedding. *IEEE TSE* (2020).

[17] Asem Ghaleb and Karthik Pattabiraman. 2020. How Effective Are Smart Contract Analysis Tools? Evaluating Smart Contract Static Analysis Tools Using Bug Injection. In *29th ACM SIGSOFT ISSTA.*

[18] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* 855–864.

[19] Sowon Jeon, Gilhee Lee, Hyoungshick Kim, and Simon S Woo. 2021. SmartConDetect: Highly Accurate Smart Contract Code Vulnerability Detection Mechanism using BERT. In *KDD Workshop on Programming Language Processing.*

[20] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[21] Yi Li, Shaohua Wang, and Tien N Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 292–303.

[22] Zhen Li, Deqing Zou, Shouhuai Xu, Zhaoxuan Chen, Yawei Zhu, and Hai Jin. 2021. VulDeeLocator: a deep learning-based fine-grained vulnerability detector. *IEEE Transactions on Dependable and Secure Computing* (2021).

[23] Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Yawei Zhu, and Zhaoxuan Chen. 2021. SySeVR: A framework for using deep learning to detect software vulnerabilities. *IEEE Transactions on Dependable and Secure Computing* (2021).

[24] Zhenguang Liu, Peng Qian, Xiang Wang, Lei Zhu, Qinming He, and Shouling Ji. 2021. Smart Contract Vulnerability Detection: From Pure Neural Network to Interpretable Graph Feature and Expert Pattern Fusion, In 13th IJCAI. *arXiv preprint arXiv:2106.09282.*

[25] Zhenguang Liu, Peng Qian, Xiaoyang Wang, Yuan Zhuang, Lin Qiu, and Xun Wang. 2021. Combining graph neural networks with expert knowledge for smart contract vulnerability detection. *IEEE TKDE* (2021).

[26] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *the ACM SIGSAC conference on computer and communications security (CCS).* 254–269.

[27] Meta Platforms, Inc. 2022. *React: A JavaScript library for building user interfaces.* https://reactjs.org/

[28] Mark Mossberg, Felipe Manzano, Eric Hennenfent, Alex Groce, Gustavo Grieco, Josselin Feist, Trent Brunson, and Artem Dinaburg. 2019. Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In *34th IEEE/ACM ASE.* 1186–1189.

[29] Hoang H. Nguyen, Nhat-Minh Nguyen, Chunyao Xie, Zahra Ahmadi, Daniel Kudendo, Thanh-Nam Doan, and Lingxiao Jiang. 2022. MANDO: Multi-Level Heterogeneous Graph Embeddings for Fine-Grained Detection of Smart Contract Vulnerabilities. https://doi.org/10.48550/ARXIV.2208.13252

[30] Sebastián Ramírez. 2022. *FastAPI framework, high performance, easy to learn, fast to code, ready for production.* Berlin, Germany. https://fastapi.tiangolo.com/

[31] Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. 2018. Automated vulnerability detection in source code using deep representation learning. In *17th IEEE international conference on machine learning and applications (ICMLA).* 757–762.

[32] SonarQube. 2022. https://www.sonarqube.org/

[33] SpotBugs. 2022. https://spotbugs.github.io/

[34] Nick Szabo. 1994. Smart Contracts: Building Blocks for Digital Markets.

[35] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW.*

[36] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. SmartCheck: Static Analysis of Ethereum Smart Contracts. In *the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain.* 9–16.

[37] Petar Tsankov, Andrei Dan, Dana Drachsler Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. 2018. Securify: Practical Security Analysis of Smart Contracts. In *25th ACM Conference on Computer and Communications Security.*

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems.* 5998–6008.

[39] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. 2019. Heterogeneous graph attention network. In *WWW.* 2022–2032.

[40] Hongjun Wu, Zhuo Zhang, Shangwen Wang, Yan Lei, Bo Lin, Yihao Qin, Haoyu Zhang, and Xiaoguang Mao. 2021. Peculiar: Smart Contract Vulnerability Detection Based on Crucial Data Flow Graph and Pre-training Techniques. In *the 32nd International Symposium on Software Reliability Engineering.*

[41] Yueming Wu, Deqing Zou, Shihan Dou, Wei Yang, Duo Xu, and Hai Jin. 2022. VulCNN: An Image-inspired Scalable Vulnerability Detection System. In *ICSE.*

[42] Kechi Zhang, Wenhan Wang, Huangzhao Zhang, Ge Li, and Zhi Jin. 2022. Learning to Represent Programs with Heterogeneous Graphs. In *ICPC.*

[43] Hui Zhao, Peng Su, Yihang Wei, Keke Gai, and Meikang Qiu. 2021. GAN-Enabled Code Embedding for Reentrant Vulnerabilities Detection. In *Knowledge Science, Engineering and Management.* 585–597.

[44] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *Advances in neural information processing systems* 32 (2019).

[45] Yuan Zhuang, Zhenguang Liu, Peng Qian, Qi Liu, Xiang Wang, and Qinming He. 2020. Smart Contract Vulnerability Detection using Graph Neural Network. In *IJCAI.* 3283–3290.