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

Hoang H. Nguyen
ahoang@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@sisc.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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ESEC/FSE ’22, November 14–18, 2022, Singapore, Singapore
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9413-0/22/11...
$15.00
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 programmers1. Such bugs or vulnerabilities may have more serious impact than those in traditional software such 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

1In this paper, we use the two terms “bug” and “vulnerability” interchangeably.
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
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 \( \theta \) is a path in the form of \( A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \ldots \xrightarrow{R_k} A_{i+1} \), which defines relations \( R_i \) (i.e., edge types) from node types \( A_i \) to \( A_{i+1} \) in a heterogeneous graph. The \textit{length} of \( \theta \) is the number of relations in \( \theta \). 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) \( \phi_i \) for each node \( i \) whose type is \( \phi_k \); 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 \( \phi_k \) of (i, j) pair by leveraging the self-attention mechanism [38] between \( i \) and \( j \).

We concatenate all node embedding \( M^\phi_i \) corresponding to all node type \( \phi_k \) of all node \( i \) to generate a unified embedding vector for a node, which is used to train a \textit{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 \textit{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.

Figure 2: Overview of the MANDO-GURU Tool.
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 evaluation uses two tasks: (i) contract-level vulnerability detection; and (ii) line-level vulnerability 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 Vulnerability Detection:
- 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 generation methods help MANDO-GURU outperform the baseline; and it shows that our architecture is general for plugging in various kinds of GNNs.
- Being compatible 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 Vulnerability 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.

Footnote. Due to the page limit, we only report highlights of our contributions and achievements here and shift more comprehensive evaluations to the arXiv version [20].
REFERENCES

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