When Deep Learning Meets Smart Contracts

Zhipeng Gao
Monash University, Australia

ABSTRACT

Ethereum has become a widely used platform to enable secure, Blockchain-based financial and business transactions. However, many identified bugs and vulnerabilities in smart contracts have led to serious financial losses, which raises serious concerns about smart contract security. Thus, there is a significant need to better maintain smart contract code and ensure its high reliability.

In this research: (1) Firstly, we propose an automated deep learning based approach to learn structural code embeddings of smart contracts in Solidity, which is useful for clone detection, bug detection and contract validation on smart contracts. We apply our approach to more than 22K solidity contracts collected from the Ethereum blockchain, results show that the clone ratio of solidity code is at around 90%, much higher than traditional software. We collect a list of 52 known buggy smart contracts belonging to 10 kinds of common vulnerabilities as our bug database. Our approach can identify more than 1000 clone related bugs based on our bug databases efficiently and accurately. (2) Secondly, according to developers' feedback, we have implemented the approach in a web-based tool, named SMARTEMBED, to facilitate Solidity developers for using our approach. Our tool can assist Solidity developers to efficiently identify repetitive smart contracts in the existing Ethereum blockchain, as well as checking their contract against a known set of bugs. which can help to improve the users' confidence in the reliability of the contract. We optimize the implementations of SMARTEMBED which is sufficient in supporting developers in real-time for practical uses. The Ethereum ecosystem as well as the individual Solidity developer can both benefit from our research.

SMARTEMBED website: http://www.smartembed.tools
Demo video: https://youtu.be/o9ylyOpYFq8
Replication package: https://github.com/beyondacm/SmartEmbed

1 INTRODUCTION

In recent years, along with the adoption and development of cryptocurrencies on distributed ledgers (a.k.a., blockchains), *smart contracts* [22] has attracted more and more attention with Ethereum blockchain platform. A Smart contract is a computer program that can be triggered to execute any task when specifically predefined conditions are satisfied. A major concern in the Ethereum platform is the security of smart contracts. A smart contract in the blockchain often involves cryptocurrencies worthy of millions of USD (e.g.,

DAO [7], Parity [21] and many more). Moreover, different from a traditional software program, the smart contract code is immutable after its deployment. They cannot be changed but may be killed when any security issue is identified within the smart contracts. This introduces challenges to blockchain maintenance and gives much incentive to hackers for discovering and exploiting potential problems in smart contracts, and there is a very significant need to check and ensure the robustness of smart contracts before deployment.

Many prior works have investigated characteristics of bugs in smart contracts and underlying blockchain systems (e.g., [1, 2, 6, 15, 26]) and detection of smart contract bugs (e.g., [3, 5, 8, 17, 20, 23, 24]). A major disadvantage is that all these existing tools require certain bug patterns or specification rules defined by human experts. Considering the high stakes in smart contracts and race between attackers and defenders, it can be far too slow and costly to write new rules and construct new checkers in response to new bugs and exploits created by attackers. Recently, there are also studies on clones and clone detection for Ethereum smart contracts (e.g., [11, 16]). However, they use expensive symbolic transaction sketches or pairwise comparisons which affect their efficiency and they are limited to clone detection.

In this research, we aim to address these problems by exploring deep learning techniques to learn vector representation for smart contracts. Our main idea is two folds: (1) Structural Code Embeddings: code and bug patterns, including their lexical, syntactical, and even some semantic information, can be automatically encoded into numerical vectors via techniques adapted from deep learning and word embeddings (e.g., [4, 18, 19, 25, 29]). (2) Similarity Checking: code checking can be essentially done through similarity checking among the numerical vectors representing various kinds of code elements of various levels of granularity in smart contracts.

In our previous work [10], we proposed an deep learning based approach that can efficiently and effectively check smart contracts with structural code embeddings. ur approach can be general enough to be applied for various code debugging and maintenance tasks, which include repetitive contract detection, detection of specific kinds of bugs in a large contract corpus, or validation of a contract against a set of known bugs. Moreover, our approach can easily add new bug checking rules by generating code embeddings for the changing bug patterns, without extra manual efforts in defining the bug specifications. According to the feedback of Solidity developers in Github, we further implemented our approach as a web-based tool, named SMARTEMBED, which can help Solidity developers check code clones and bugs for their own smart contracts [9]. Furthermore, to meet the efficiency requirements as an online web tool, we improve the efficiency of SMARTEMBED in three ways: (i) replace multiple loop structure calculation with matrix computation. (ii) put the code embeddings into cache to reduce redundant data loading. (iii) create indexes for smart contracts in our database to speed up information retrieval process.
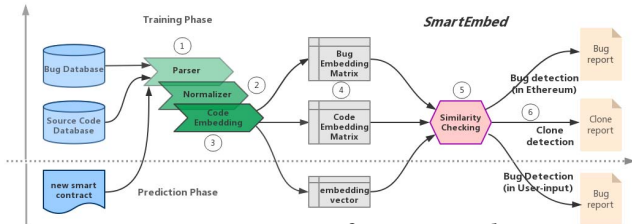
Figure 1: Overview of Our Approach

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents the overall workflow of our approach and details of each step. Section 4 shows the experimental results. Section 5 concludes the contribution of our work.

2 RELATED WORK

Both clone detection and bug detection have been extensively studied in the literature for both traditional software programs and Ethereum smart contracts. Clone detection techniques utilize expensive symbolic transaction sketch [16] or pair-wise signature comparison [11]. They can be applied to compiled contract bytecode, but efficiency is limited for checking contract-level similarities, instead of more fine-grained levels of granularity. More bug detection techniques have been developed for smart contracts (e.g., [3, 5, 8, 17, 20, 23, 24]), while they often require manual efforts in defining specifications and/or bug patterns needed for validation and/or bug checking.

Machine learning and deep learning techniques have been used for clone detection (e.g., [14, 27, 30]) and bug detection problems (e.g. [13, 28]) in traditional software programs too, but little has been applied for smart contracts. Our approach is unique in that it utilizes deep learning and similarity checking techniques to unify clone detection and bug detection together efficiently and accurately for Ethereum smart contracts.

3 APPROACH

Fig.1 illustrates the overall framework of Our approach. Based on code embeddings and similarity checking, our approach targets two tasks in a unified approach: clone detection and bug detection. For clone detection, our approach can identify similar smart contracts. For bug detection, based on our bug database, our approach can detect bugs in the existing contracts in the Ethereum blockchain and/or in any smart contract given by solidity developers that are similar to any known bug in the bug database. Our approach contains two phases: a model training phase and a prediction phase.

There are mainly 4 steps during the model training phase. We built a custom Solidity parser for smart contract source code. The parser generates an abstract syntax tree (ASTs) for each smart contract in our collected dataset, and serializes the parse tree into a stream of tokens depending on the types of the tree nodes (step 1). After that, the normalizer reassembles the token streams to eliminate nonessential differences (e.g., the stop word, values of constants or literals) between smart contracts (step 2). The output token streams are then fed into our code embedding learning module, and each code fragment is embedded into a fixed-dimension numerical vector (step 3). After the code embedding learning step, all the source code is encoded into the source code embedding

matrix; in the meanwhile, all the bug statements we collected are encoded into the bug embedding matrix (step 4).

In the prediction phase, any given new smart contract is turned into embedding vectors by going through the steps 1,2,3 and utilizing the learned embedding matrices. Similarity comparison is performed between the embeddings for the given contract and those in our collected database (step 5), and similarity thresholds are used to govern whether a code fragment in the given contract will be considered as code clones or clone-related bugs (step 5-6).

4 EVALUATION RESULTS

Clone Detection. Our approach can effectively identify many repetitive Solidity codes in Ethereum blockchain. Our experimental results against more than 22K smart contracts show the clone ratio of solidity code is at around 90%, much higher than traditional software, which reveals homogeneous of the Ethereum ecosystem. Our approach can detect more semantic clones accurately than the commonly used clone detection tool Deckard [12]. The relatively high ratio of code clones in smart contracts may cause severe threats, such as security attacks, resource wastage, etc. Finding such clones can enable significant applications such as vulnerability discovery and deployment optimization (reduce contract size and duplication), hence contributing to the overall health of the Ethereum ecosystem. Our work in identifying clones can also help Solidity developers to check for plagiarism in smart contracts, which may cause a huge financial loss to the original contract creator.

Bug Detection. For bug detection, we first collect a list of 52 known buggy smart contracts belonging to 10 kinds of common vulnerabilities. The bug detection results show that our approach can efficiently and accurately identify more than 1,000 clone-related bugs based on our bug database in Ethereum blockchain, which can enable efficient checking of smart contracts with changing code and bug patterns. For contract validation, our approach can capture bugs similar to known ones with low false positive rates, the query for a clone or a bug is quite efficient which can be sufficient for practical uses. Our web-based tool can be easily updated with the newly added contracts and bug patterns in Ethereum blockchain and further support developers for using our approach.

5 SUMMARY AND CONTRIBUTIONS

This paper presented a deep learning based approach, SMARTEMBED, for detecting code clones and bugs in smart contracts accurately and efficiently. It develops a code embedding technique for tokens and syntactical structures in Solidity code and utilizes similarity checking to search for "similar" code satisfying certain thresholds. Our approach can be easily updated to recognize new contract clones and new kinds of bugs when the contract code and bugs evolve. The approach is automated on the contract and bug data collected from the Ethereum blockchain. It helps developers to find repetitive contract code and clone-related bugs in existing contracts, which helps to improve developers' confidence in the reliability of their contracts. It also helps to efficiently validate given smart contracts against known set of bugs without the need of manually defined bug patterns. The clone detection and bug detection results can benefit the smart contract community as well as individual Solidity developers.

REFERENCES

[1] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A survey of attacks on ethereum smart contracts (sok). In *Principles of Security and Trust*. Springer, 164–186.

[2] Massimo Bartoletti and Livio Pompianu. 2017. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International Conference on Financial Cryptography and Data Security*. Springer, 494–509.

[3] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. 2016. Formal Verification of Smart Contracts: Short Paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS '16)*. ACM, New York, NY, USA, 91–96. https://doi.org/10.1145/2993600.2993611

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* (2016).

[5] Chad E Brown, Ondrej Kuncar, and Josef Urban. 2017. Formal Verification of Smart Contracts (Poster). In *8th International Conference on Interactive Theorem Proving*.

[6] Ting Chen, Xiaoqi Li, Xiapu Luo, and Xiaosong Zhang. 2017. Under-optimized smart contracts devour your money. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 442–446.

[7] DAO. 2018. The DAO (organization). https://en.wikipedia.org/wiki/The_DAO_(organization)

[8] Kevin Delmolino, Mitchell Arnett, Ahmed Kosba, Andrew Miller, and Elaine Shi. 2016. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In *International Conference on Financial Cryptography and Data Security*. Springer, 79–94.

[9] Zhipeng Gao, Vinoj Jayasundara, Lingxiao Jiang, Xin Xia, David Lo, and John Grundy. 2019. SmartEmbed: A Tool for Clone and Bug Detection in Smart Contracts through Structural Code Embedding. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 394–397.

[10] Zhipeng Gao, Lingxiao Jiang, Xin Xia, David Lo, and John Grundy. 2020. Checking Smart Contracts with Structural Code Embedding. *IEEE Transactions on Software Engineering* (2020).

[11] Ningyu He, Lei Wu, Haoyu Wang, Yao Guo, and Xuxian Jiang. 2019. Characterizing Code Clones in the Ethereum Smart Contract Ecosystem. *arXiv:1905.00272* (2019).

[12] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. 2007. Deckard: Scalable and accurate tree-based detection of code clones. In *Proceedings of the 29th International Conference on Software Engineering (ICSE)*. 96–105. https://github.com/skyhover/Deckard

[13] J. Li, P. He, J. Zhu, and M. R. Lyu. 2017. Software Defect Prediction via Convolutional Neural Network. In *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 318–328.

[14] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder. 2017. CCLearner: A Deep Learning-Based Clone Detection Approach. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 249–260. https://doi.org/10.1109/ICSME.2017.46

[15] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. 2017. A survey on the security of blockchain systems. *Future Generation Computer Systems* (2017).

[16] Han Liu, Zhiqiang Yang, Chao Liu, Yu Jiang, Wenqi Zhao, and Jiaguang Sun. 2018. EClone: Detect Semantic Clones in Ethereum via Symbolic Transaction Sketch. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. ACM, New York, NY, USA, 900–903. https://doi.org/10.1145/3236024.3264596

[17] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 254–269.

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[20] Bernhard Mueller. 2018. Smashing Smart Contracts for Fun and Real Profit. In *9th annual HITB Security Conference (HITBSecConf)*. Consensys, Amsterdam. https://github.com/ConsenSys/mythril

[21] Parity. 2017. Parity Security Alert. https://paritytech.io/security-alert-2/

[22] Nick Szabo. 1994. Smart Contracts. http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

[23] Sergei Tikhomirov, Ekaterina Voskresenskaya, Ivan Ivanitskiy, Ramil Takhaviev, Evgeny Marchenko, and Yaroslav Alexandrov. 2018. SmartCheck: Static Analysis of Ethereum Smart Contracts. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB) (WETSEB '18)*. ACM, New York, NY, USA, 9–16. https://doi.org/10.1145/3194113.3194115

[24] 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 (CCS). *arXiv preprint arXiv:1806.01143*.

[25] Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 384–394.

[26] Zhiyuan Wan, David Lo, Xin Xia, and Liang Cai. 2017. Bug Characteristics in Blockchain Systems: A Large-scale Empirical Study. In *Proceedings of the 14th International Conference on Mining Software Repositories (MSR) (MSR '17)*. IEEE Press, Piscataway, NJ, USA, 413–424. https://doi.org/10.1109/MSR.2017.59

[27] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. 2016. Deep learning code fragments for code clone detection. In *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 87–98.

[28] Xinli Yang, David Lo, Xin Xia, Yun Zhang, and Jianling Sun. 2015. Deep Learning for Just-in-Time Defect Prediction. In *IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 17–26.

[29] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*. ACM, 404–415.

[30] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. 2019. A Novel Neural Source Code Representation based on Abstract Syntax Tree. In *ICSE*.