Analyzing the Security of Smart Contracts Using Neural Networks

Student: Minghao Li

Partners: Hao Wu (Department of Computer Science and Technology, Nanjing University), Yue Duan (Department of Computer Science, Illinois Institute of Technology), and Mu Zhang (School of Computing, University of Utah)

Faculty Advisor: Elaine Shi, Department of Computer Science, Cornell University

Research Award Funding Source: John A. Swanson

August 15, 2020

Abstract

Decentralized Apps (DApps) are being widely deployed due to the accelerated adoption of their carrier – blockchains. DApps often handle cryptocurrencies and tokens transactions, tying their security directly to users' property safety. As most DApps are backed by smart contracts running on the blockchains, the core of developing secure DApps comes down to analyzing the security of smart contracts. In this paper, we approach smart contracts security analysis through clustering smart contract functions. We hypothesize that by generating a model graph embedding for each function using neural networks, we would be able to sort all functions according to their business logic by putting similar graph embeddings in one cluster. If a DApp claims to perform a certain business, but some smart contracts it uses have functions failing to fall in relevant business logic clusters, then the security of this DApp is questionable. Our experiment shows that functions from the same smart contract tend to end up in the same cluster, and the clustering result reflects the real-life distribution of DApps business categories. The result also indicates rooms for future improvements. We conclude that keep researching and developing our approach is worthwhile, and some changes may enhance its performance.

Analyzing the Security of Smart Contracts Using Neural Networks

I. Introduction

Decentralized applications (DApps) are applications that run on decentralized networks. With the recent surge of blockchains, DApps are being developed particularly on them. Blockchain was first designed by Satoshi Nakamoto as the distributed ledger recording transactions of bitcoin ("The great chain," 2015), and has been utilized by many other cryptocurrencies. Therefore, DApps nowadays often handle cryptocurrency payments. However, contrary to their rapid development and adoption, research on DApps security has been lagging. This is alarming considering the monetary nature of DApps.

A wide range of DApps is implemented by utilizing smart contracts, which are computer programs that operate on blockchains (Luu et al., 2016). Thus, a major component of DApps security analysis is ensuring that smart contracts fulfill their claimed functionalities. Our experiment explored the possibility of first generating a model graph for each function of a smart contract from its bytecode, and then clustering the model graphs to detect functions with similar functionality. The insight is that functions that correctly implement the same business logic are likely to have similar model graphs, and therefore are likely to fall in the same cluster through unsupervised learning and clustering.

The Materials and Methods section explains the smart contracts security analysis method we propose in detail. We experimented with 49681 functions from 2469 smart contracts, and present the experiment results in the Results section. We discuss our findings in the Discussion section. We conclude that our method is promising, and future developments are desired to make it more reliable.

II. Materials and Methods

Our approach consists of three steps: given the bytecode of a smart contract, we first construct a model graph for each function of the contract; next, we generate an embedding using neural networks for each graph; finally, we use the embeddings to cluster the graphs. Figure 1 gives an overview of our approach.

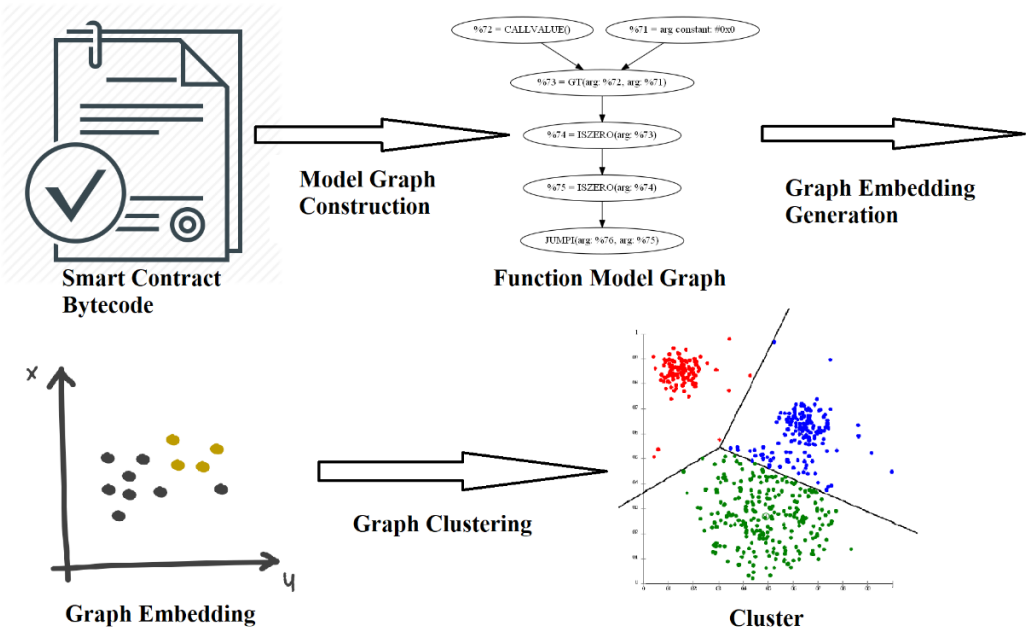
A. Collect Smart Contract Bytecodes

To gather smart contract bytecodes, we used the open-source web scraping tool Scrapy (Version 2.0.0, 2020) to extract bytecodes of smart contracts uploaded to and verified by Etherscan (https://etherscan.io/).

For our experiment, we used 2469 smart contracts.

Figure 1

Approach Overview



B. Function Model Graph Construction

A function model graph is a directed graph that has assembly instructions as nodes, and data flows and control flows as edges. We noticed that smart contract functions often have trivial statements that are responsible for non-essential jobs such as variables assignment and activities logging. To ensure that the function model graphs are of reasonable size, we decided to include only crucial statements that interact with the core factors of smart contracts while generating the graphs. We consider transaction properties (e.g., the sender of a transaction, the value of cryptocurrencies transferred, and the timestamp of a transaction), global variables, condition checks, and cryptocurrency transfer as core factors.

The model generation process was built on top of the open-source smart contracts bytecode analysis tool Octopus (Patrick Ventuzelo, 2020). With the bytecode of a smart contract, we use Octopus to convert it into assembly representation, translate the assembly representation into static single assignment format, and then perform control flow and data flow analysis. The final output is a series of function model graphs.

From the 2469 smart contracts we used, we constructed 49681 function model graphs.

C. Graph Embedding Generation

Our embedding generation method is inspired by DeepWalk (Perozzi et al., 2014). For each graph, we first perform five rounds of random walk on it. During a round of random walk, we start with each node, and then randomly traverse a path of five nodes. While traversing a path, we concatenate the assembly instruction each visited node stands for to form a sentence. We noticed that some assembly instructions contain memory addresses or numerical values. These addresses and values differ from one another, yet their roles in instructions are mostly the

same. To assure that these minor differences would not negatively affect our experiment result, we replace all memory addresses with “ADDR”, and replace all numerical values with “NUM” while forming a sentence.

After the random walks, we use the collected sentences to train a word2vec model (Mikolov et al., 2013) which uses neural networks to get the vector representation of each word in the collection of sentences. Then we get the embedding of a node by calculating the average value of the vector representations of all words that appear in the assembly instruction this node stands for (we again substitute memory addresses with “ADDR”, and numerical values with “NUM”). Finally, we assign the embedding of a graph to be the average value of its nodes’ embeddings.

D. Graph Clustering

We cluster the functions by clustering their model graph embeddings using k-means clustering (Lloyd, 1982). We set the number of clusters to be 100. After the clustering process finishes, each function will fall in one of the 100 clusters, with functions with close graph embeddings in one cluster.

III. Results

We clustered 49681 functions from 2469 smart contracts into 100 clusters. We assign cluster IDs in the decreasing order of the number of functions in each cluster. That is, we give the most populous cluster ID 0, and give the least populous cluster ID 99. We present the number of functions in each cluster in Figure 2. The x-axis of both subplots shows cluster ID numbers, and the y-axis of both subplots denotes the number of functions in a cluster. Since the size of

superclusters diminishes the height of bars representing smaller clusters, a subplot is created for clusters with ID between 20 and 99.

The five largest clusters each consists of 27642, 8922, 2683, 1693, and 592 functions. Together these five clusters have 41532 functions, which make up 83.6% of all the functions we clustered. There are 35 clusters with at least 100 functions. 26 clusters have less than 10 functions. And three clusters only contain one function.

Among the 49678 functions that are in a cluster with at least two functions, 47464 have the Euclidean distance between their graph embedding and the embedding of the closest same-cluster function to be less than 0.001.

Of the 2469 smart contracts, 1001 smart contracts have all of its functions in one cluster, and 1812 smart contracts have all of its functions spanning no more than five clusters. Figure 3 presents the number of contracts that have functions present in a specific number of clusters.

Figure 2

Number of Functions in Each Cluster

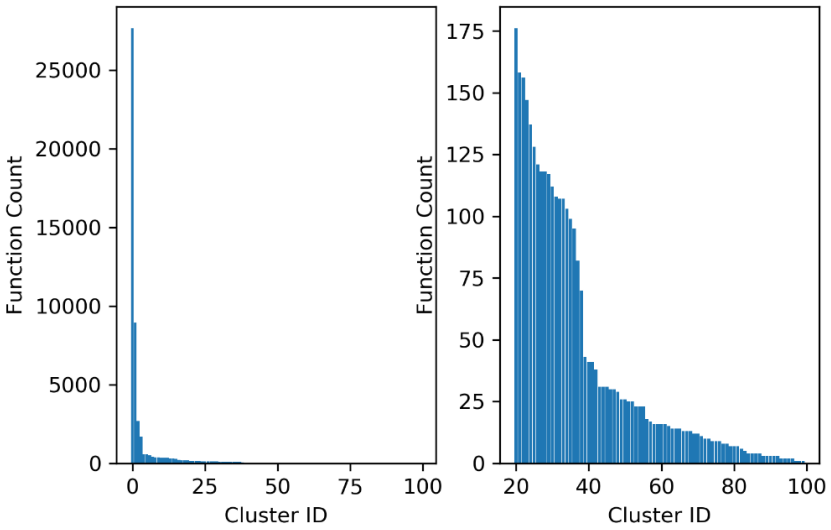
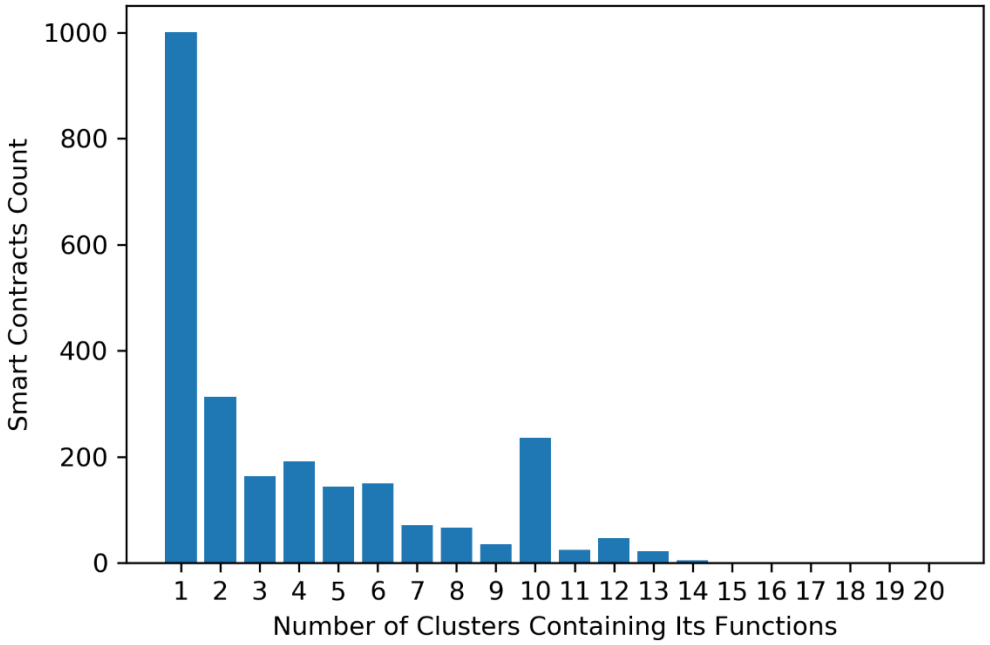


Figure 3

Number of Smart Contracts Having Its Functions Present in a Specific Number of Clusters



IV. Discussion

While reviewing DApps on the market, we noted that about 80% of the top 100 DApps from DApp Ranking on dapp.com (https://www.dapp.com/dapps) belongs to one of the following five business categories: auction, wallet, gambling, trading, and voting. We also noticed that for DApps with open-source smart contracts source code, function names were often closely related to their business category. For example, functions of an online auction DApp usually have the word “auction” or “bid” in their names. Therefore, during the preparation stage of the experiment, we first specified function name keywords for each of the aforementioned business categories as shown in Table 1. Next, we went through the source code of the 2469 smart contracts we used in our experiment and recorded for each business category the number of

function names containing its keyword(s). Figure 4 illustrates the result of this preliminary examination. The result indicates that a considerable portion of functions is likely utilized by wallet DApps. A probable explanation is that as previously discussed, DApps nowadays commonly operate on blockchains, which support numerous cryptocurrencies. Thus, a majority of DApps serve as wallets for holders of a certain cryptocurrency or token. In fact, 1908 of the 2469 smart contracts we used have at least one function name that consists of one or more keywords of the wallet business category.

The result of having a significant percentage of functions being placed in the top few clusters hence is expected. A large fraction of the smart contracts we collected are presumably used by wallet DApps, and consequently many functions are expected to carry out similar functionalities.

Moreover, it is reasonable to assume that functions from the same smart contract serve the same business category. Since 1001 out of the 2469 smart contracts can find all of its functions in the same cluster, and about 73.4% of the 2469 smart contracts have no more than five clusters containing its functions, we argue that our method performed well in identifying functions with similar tasks and assigning them to the same cluster.

In view of the preceding points, we believe that with the help of neural networks that generate model graph embeddings, clustering algorithms do have the potential to help us identify functions implementing the same business logic.

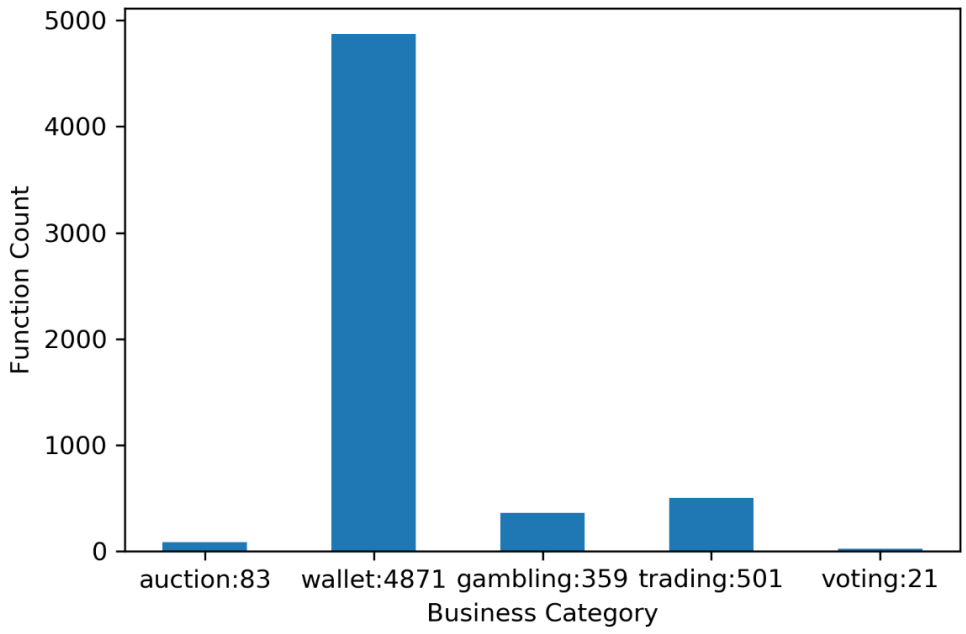
Table 1

Keywords for Business Categories

Business category	Keywords
Auction	auction, bid
Wallet	deposit, transfer, withdraw
Gambling	buy, bet, play
Trading	buy, sell
Voting	vote

Figure 4

Functions per Business Category Using Bag-of-Words Model



The phenomenon that 26 out of 100 clusters have less than 10 functions and three clusters comprise only one function is possibly caused by we selecting a relatively large value (100) as the number of clusters to generate. Considering that we only need five business categories to include about 80% of DApps, and each business category encompasses a limited number of business logic, the number of clusters needed to accommodate function implementations of non-trivial business logic could be less than 100. We would like to experiment with smaller cluster numbers in the future stage of our project.

However, there are two concerning aspects of our experiment. The first concern is reflected by the closeness of model graph embeddings of functions in the same cluster. We observed that almost all embeddings have a negligible Euclidean distance between them and their closest embedding, and in extreme cases, this distance is 0. This implies that our current method is only capable of distinguishing smart contract functions roughly. Modifications of established embedding generation and clustering algorithms may be required. The second concern is that we lack the ground truth to further evaluate our experiment results. We can manually check each function's source code to determine the exact business logic it implements, but that will be too time-consuming to be carried out on a large scale.

V. Conclusion

We designed a novel method to perform smart contract security analysis. Given a smart contract, we construct a model graph for each of its functions from its bytecode. We then generate an embedding for each function model graph and cluster the functions by clustering their graph embeddings. Functions in the same cluster are expected to fulfill the same task. If a function claims to accomplish a certain job but falls in an irrelevant function cluster, then we

should be suspicious about its security as well as the smart contract it appears in. We experimented with 49681 functions from 2469 smart contracts. We conclude that our idea is promising, but customization of existing algorithms may be needed to achieve the optimal result. We also look forward to the availability of smart contracts datasets or reliable smart contracts classification tools, so that solid evaluations of our experiment results will be possible.

References

Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory 28*(2), 129–137. https://doi.org/10.1109/TIT.1982.1056489

Luu, L., Chu, D., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*, 254–269. https://doi.org/10.1145/2976749.2978309

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. Retrieved August 12, 2020, from https://arxiv.org/abs/1301.3781v3

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: Online learning of social representations. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*, 701–710. https://doi.org/10.1145/2623330.2623732

Scrapy (Version 2.0.0) [Source Code]. (2020). Retrieved from https://github.com/scrapy/scrapy

The great chain of being sure about things; blockchains. (2015, Oct 31). *The Economist, 417*, 21–24. Retrieved from https://search-proquest-com.proxy.library.cornell.edu/docview/1728728735?accountid=10267

Ventuzelo, P. (2020). *Octopus* [Source Code]. Retrieved from https://github.com/pventuzelo/octopus