Attention-based Machine Learning Model for Smart Contract Vulnerability Detection

Yuhang Sun, Lize Gu *

Department of Cybersecurity, Beijing University of Posts and Telecommunications, Beijing 100876, China

*Corresponding author’s e-mail: glzisc@bupt.com

Abstract. Ethereum attracts extensive attention due to its distinctive function of smart contract and decentralized applications (Dapps). Since the number of contracts on blockchain has increased vigorously, various security vulnerabilities come up. Researchers rely on static symbolic analysis method at first, and it seems to perform well in the accuracy of vulnerability detection. However, this method requires manual analysis in advance and it needs to traverse all the possible execution paths to find out the vulnerable ones. The deeper the path goes, the more time it costs to detect the contracts. This paper proposes an approach to detect smart contracts vulnerability on blockchain by using machine learning (ML) methods. This approach aims to build a general benchmark for new vulnerability detection in order to reduce the demand of expert manpower. Moreover, the high-speed-performance ML algorithm makes quick detection comes true. As long as we adjust the threshold of the model, it can work as a fast prefilter for the traditional symbolic analysis tools in further improvement of accuracy.

1. Introduction
Since blockchain technologies have been gaining increasing popularity and acceptance by a wider community, smart contract recently received a considerable amount of attention[1]. Smart contracts provide virtual commitments by using cryptographic and other security mechanisms. Notably, smart contracts preserve many algorithmically specifiable relationships from breach by principals, and from eavesdropping or malicious interference by third parties[2].

Ethereum is well-known for smart contracts with a business scale up to $22 billion [3]. A smart contract is an agreement which could be activated by any involved party without a trusted third party whenever the intended conditions are satisfied. Due to the immutability of the blockchain, vulnerabilities are largely irreversible[4]. Above 3.6 million Ether were stolen from a decentralized investment fund called The DAO (Decentralized Autonomous Organization) in June 2016, with a loss of $70 million [6]. The hacks were aroused by exploitable logic within smart contract itself, and these incidents strengthened an urgent need for the security of smart contracts[5]. To detect and analyze underlying vulnerabilities in contract, especially before deployment is of vital importance.

Solidity is a turing complete programming language in Ethereum for contract programming. The source code written by solidity can be of great help for traditional manual detection method. However, most contracts on blockchain are only bytecodes. In other words, they are unreadable. According to the latest records, among the 1 million smart contracts running on Ethereum, only 2% open their source code[7]. Thus, a method that can effectively detect vulnerabilities via bytecode instead of source code is needed, which motivates this paper.
In this paper, we propose an approach of CNN combining model with self-attention mechanism to realize the vulnerability detection of smart contracts. Besides, our method combines an improved one-hot encoder, sensitive word sharding and stop word list through feature engineering.

2. Background

2.1. Vulnerabilities

(1) Reentrancy

Reentrancy occurs when external contract calls are allowed to make new calls to the calling contract before the initial execution is complete. For a function, this means that the contract state may change in the middle of its execution as a result of a call to an untrusted contract or the use of a low level function with an external address. The reentrancy attack resulted in a hard fork of Ethereum in a grand larceny for the first time. The discovery of reentrancy surprised everyone for the first time which made it one of the most famous Ethereum vulnerabilities.

(2) Arithmetic Issues(Integer Overflow/Underflow)

Some developers are accustomed to simple int types that are usually merely signed integers, while unsigned integers are widespread. In an arithmetic issue, a fixed size variable is required to store a number while the number is beyond the range of the variable's data type. BECToken was attacked via an integer overflow in 2018, resulting in a sharp drop of the token price.

(3) Time manipulation(Timestamp Dependence)

Contracts sometimes rely on the current time from locking a token sale to unlocking funds at a specific time for a transaction, which is often done via timestamp of block. Now that miners is able to affect the block's timestamp, they can insert their own transaction with changes to the related contracts before the victim’s transaction in the block.

2.2. Machine Learning Algorithm

(1) Convolutional Neural Network(CNN)

CNN is a feedforward neural network with deep structure containing convolution calculation[9]. CNN is a typical deep-learning algorithm[10], which mainly consists of convolutional-layer, pooling-layer and full-connection-layer. The algorithm performs well in many fields because of its unique linear computing characteristics.

(2) Self-attention

The self-attention model can select states that are similar to the given ones from the set in the network, and then do subsequent information extraction[11]. Since the word information is feedback in turn in the traditional time correlation text classification model based on CNN, it costs much more time. Words can be directly correlated by using self-attention.

Fig.1 Overview of the method
3. Method

3.1. Preprocessing

3.1.1. Stop Word List

Among the opcode of all kinds, there are several types of instructions that operate directly on data in the stack. On consider of the distinct version of compiler, these instructions may vary from one type to another. This change has a negative impact on the accuracy of vulnerability detection.

Fortunately, these stack-operating instructions have little to do with the behavior of the source code. We believe that the role of these instructions can be neglected in the process of vulnerability detection. The Table shows the stop word list made up of the instructions to remove before detecting.

| Type | Instructions |
|------|--------------|
| DUPx | DUP1, DUP2, DUP3, ……, DUP16 |
| SWAPx | SWAP1, SWAP2, SWAP3, ……, SWAP16 |
| PUSHx | PUSH1, PUSH2, PUSH3, ……, PUSH20 |
| POP | POP |
| LOGx | LOG0, LOG1, LOG2, LOG3, LOG4, …… |

3.1.2. Sensitive Word Sharding

By reviewing the source code, we realized that some vulnerabilities usually gather in one function. Hence, the corresponding instructions of the vulnerable sequence has a concentrated distribution. In the paper, a sensitive word sharding method is proposed. The opcodes of a contract are cut into slices according to a sensitive words list. We select the return type as the segmentation point for the reason that these words usually mark the end of the function. The return type opcode includes returndatasize, return, revert, selfdestruct, invalid.

3.1.3. Label Normalization

In a contract, the logic is reflected by the opcode[8]. It seems that to extract the feature of opcodes is instrumental in detecting. In this paper, there are 70 different opcode instructions in the dataset. The instructions are divided into 9 types according to their function. Since the stack type is in the stop word list, there are 8 types left for the later detection.

| Type | Instructions |
|------|--------------|
| Compute | add(x, y), sub(x, y), mul(x, y), div(x, y), mod(x, y), exp(x, y), sdiv(x, y), addmod(x, y, m), mulmod(x, y, m), smod(x, y), signextend(i, x) |
| Compare | lt(x, y), gt(x, y), eq(x, y), slt(x, y), sgt(x, y), iszero(x) |
| Bit_operation | not(x), and(x, y), or(x, y), xor(x, y), byte(n, x), shl(x, y), shr(x, y), sar(x, y) |
| Transaction_data | caller, gas, origin, address, balance |
| Memory | mload, mstore, sload, mload, sstore, msize, create, delegatecall, staticcall, call |
| Calldata&Codedata | callcode, calldataload, calldata, calldatasize, extcodecopy, codecopy, codesize |
| Block_data | gasprice, gaslimit, difficulty, number, timestamp, coinbase, blockhash(b), keccak256 |
| Jump&Stop | stop, jump, junpi, pc, returndatasize, return, revert, invalid, selfdestruct, returndatasize, |
| Stack | push, pop, swap, dup, log |
3.2. Model Implementation
Source code, a set of structured and sequential instructions, inherently reflects human intent: it encodes the way we command a machine to perform a task. In other words, source code follows the same distributional regularities that a natural language observes to some degree [12]. Although the opcodes compiled from source-code are based on machine language, the correlation between the opcodes remains. Hence, the sequence of opcodes can be treated as text of natural language with strong correlation.

The classification model proposed in this paper contains self-attention layer, convolutional layer, pooling layer and softmax layer. In attention layer, the input word-vector matrix is turned into the one where adjacent words are connected. In convolutional layer, more advanced text features are obtained using different convolution kernels. The most important characteristics of a sequence is captured at the pooling layer. Finally, the classification is implemented as output in the softmax layer.

3.2.1. WordEmbedding based on Self-attention
The traditional embedding method like bag of words is based on word frequency statistics which ignores the connection between words. We tackle this inefficiency by introducing attention model to extract characteristics of words connection in long distance.

First of all, each instruction of opcodes is transferred into a vector with 78 entries via one-hot encoder, as there are 70 kinds and 8 types of opcode instructions. For each opcode, one-hot vector \( v_i \) is embedded via

\[
W_{\text{word}} \in \mathbb{R}^{d \times n}
\]

\( d \) for the dimension of a word vector, \( n \) for number of opcodes in one sharding). In the sequence matrix \( D = \{w_1, w_2, w_3, \ldots, w_n\} \), vectors are fully connected and activated by Relu function, and the matrix \( X \in \mathbb{R}^{n \times d} \) is demonstrated.

\[
\begin{align*}
    w_i^l &= W_{\text{word}} v^l \\
    X &= \sigma(W_{\text{word}} D + b) \\
    \sigma: f(x) &= \max(0,x) \\
    B &= DW_{\text{word}} X
\end{align*}
\]

In the attention matrix \( A \in \mathbb{R}^{n \times n} \), each element \( a_{ij} \) is classified via softmax from each row in matrix \( B \in \mathbb{R}^{n \times n} \). Then, matrix \( C \in \mathbb{R}^{n \times d} \) is demonstrated from cross-multiplication of \( A \) and \( D \).

\[
\begin{align*}
    A &= \text{softmax}(B) \\
    a_{ij} &= \frac{e^{b_{ij}}}{\sum_{k=1}^{n} e^{b_{ik}}} \\
    C &= A \otimes D \\
    C_i &= \sum_{j=1}^{n} a_{ij} D_j
\end{align*}
\]

3.2.2. Feature Extraction based on CNN
In the model proposed by this paper, the next is the convolutional layer. The multi-dimensional feature matrix \( C \) is operated by the convolution kernel \( m \in \mathbb{R}^{k \times d \times w} \) to generate a new feature map \( H \in \mathbb{R}^{n-\kappa+1} \). In each feature \( h_i \), \( f \) represents a nonlinear activation function; \( b_c \in \mathbb{R} \) represents offset matrix; \( i \in [1, n-\kappa+1] \).

\[
\begin{align*}
    h_i &= f(m \cdot C_{i:i+k-1:d\omega} + b_c) \\
    H &= \{h_1, h_2, \ldots, h_{n-\kappa+1}\} \\
    M &= \sum_{i=1}^{n+k-1} \max(h_i)
\end{align*}
\]

In order to capture the high-level features at different levels of the sequence, multiple convolution kernels of the same latitude are further adopted. To compress the multiple feature map and extract the main features, maxpooling is used in pooling layer.
3.2.3. Softmax Classification

Last but not the least, the model classifies the contracts from \( M \) in softmax layer. On account that \( M \) is a set of vector, it is impossible to directly judge whether a contract is vulnerable. Therefore, softmax is used for normalization processing where probability of matrix \( D \)'s possible classification is obtained.

\[
p(i|D; \theta) = \frac{e^{m_i}}{\sum_{k=1}^{s} e^{m_k}} \tag{12}
\]

\[
\hat{m} = \arg\max_{i} (p(i|D; \theta)) \tag{13}
\]

This model intends to minimize the loss of cross entropy of classification \( J(\theta) \). \( p(m_i) \) represents probability estimation of each category. Besides, an index describing the complexity of the model is added to the loss function to avoid overfitting. \( \lambda \) represents the ratio of model complex losses to total, that is weight of the regularization.

\[
J(\theta) = -\frac{1}{s} \sum_{i=1}^{s} t_i \log (p(m_i)) \tag{14}
\]

\[
J(\theta) = -\frac{1}{s} \sum_{i=1}^{s} t_i \log (p(m_i)) + \lambda \parallel \omega \parallel^2 \tag{15}
\]

4. Test Results and Discussions

We collected 8632 verified smart contracts from Etherscan as dataset[7]. The dataset is split into 3 groups, 60% as training data, 20% as validation, 20% as test. The performance measures of the model compared with static analysers are shown in Table.3. The value of precision measures the ability of a method not to mislabel an innocent contract as vulnerable, while the value of recall measures the ability not to miss any vulnerable contract. Besides, the value of F1-score takes into account both the precision and recall.

| Vulnerability       | Tool   | Precision | Recall | F1-score | Error rate | Missing rate | Average time |
|---------------------|--------|-----------|--------|----------|------------|--------------|--------------|
| Reentrancy          | ABCNN  | 85%       | 96%    | 90%      | 15%        | 4%           | <1s          |
|                     | Oyente | 95%       | 95%    | 95%      | 5%         | 5%           | 102s         |
|                     | Mythril| 63%       | 99%    | 77%      | 37%        | 1%           | 315s         |
| Arithmetic Issues   | ABCNN  | 85%       | 90%    | 87%      | 15%        | 10%          | <1s          |
|                     | Oyente | 97%       | 95%    | 96%      | 3%         | 5%           | 298s         |
|                     | Mythril| 84%       | 83%    | 83%      | 16%        | 17%          | 432s         |
| Time manipulation   | ABCNN  | 80%       | 93%    | 86%      | 20%        | 7%           | <1s          |
|                     | Oyente | 90%       | 88%    | 88%      | 10%        | 12%          | 214s         |
|                     | Mythril| 90%       | 83%    | 86%      | 10%        | 17%          | 358s         |

As the table shows, the attention-based CNN model(ABCNN) performs well in detecting three types of vulnerabilities. By adjusting parameters like threshold, the model lays stress on the positive samples to reduce the missing rate. The accuracy of our model is not inferior to the static symbolic methods. Nevertheless, the model costs far less time compared with oyente[13] and mythril[14].

5. Conclusion and Future Directions

In this paper, we propose a method based on CNN model combining with self-attention mechanism for smart contract vulnerabilities detection. Our model detects three types of Ethereum smart contract vulnerabilities and performs well in accuracy. The highlights of the model are lower missing rate and average time which enables high speed of detection. We believe that it can work as a fast prefilter for the traditional symbolic analysis tools in large-scale smart contracts security detection.

In future work, we expect to extend the single binary classification models to a multinomial classification model that can detect different types of vulnerabilities at one time. Furthermore, our
The proposed method will be implemented into other vulnerabilities detection combining with formal verification method to improve the accuracy.

Acknowledgments
This work was financially supported by the National Key R&D Program of China (2017YFB0803002).

References
[1] Nakamoto, S. (2019). Bitcoin: A peer-to-peer electronic cash system. Manubot.
[2] Szabo, N. (1997) Formalizing and Securing Relationships on Public Networks. First Monday, 2(9).
[3] Buterin, V. (2014). A next-generation smart contract and decentralized application platform. White Paper, 3, 37.
[4] Hacking Distributed. (2020). The Biggest Crowdfunding Project Ever—the DAO—Is Kind of a Mess. https://www.wired.com/2016/06/biggest-crowdfunding-project-ever-dao-mess.
[5] Zhou, Y., et al. (2018). Erays: Reverse engineering ethereum’s opaque smart contracts. In: 27th USENIX security symposium. pp. 1371–1385.
[6] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. (2016). Making smart contracts smarter. In: Proc 23rd ACM Conference on Computer and Communication Security (CCS). Vienna. pp. 254-269.
[7] Etherscan. (2020). Verified contract list. https://etherscan.io/contractsVerified.
[8] Chen, T., et al. (2017). Under-optimized smart contracts devour your money. In: IEEE 24th international conference on software analysis, evolution and reengineering (SANER). Austria. pp. 442–446.
[9] Gu, J., et al. (2018). Recent advances in convolutional neural networks. Pattern Recognition, 77, 354–377.
[10] LeCun, Y., Yoshua, B., & Geoffrey, H. (2015). Deep learning. Nature, 521, 436.
[11] Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). Attention is all you need. In: Proc 31st International Conference on Neural Information Processing Systems (NIPS). New York. pp. 6000-6010.
[12] Loyola, P., Marrese-Taylor, E., Balazs, J., et al. (2018). Content Aware Source Code Change Description Generation. In: Proceedings of the 11th International Conference on Natural Language Generation. Netherlands.
[13] Luu, L., et al. (2016). Making smart contracts smarter. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. Vienna. pp. 254–269.
[14] B. Mueller. (2018). Smashing ethereum smart contracts for fun and real profit. In: Proc 9th Annual HITB Sec. Conf. Amsterdam.
[15] Kalra, S., et al. (2018). ZEUS: Analyzing safety of smart contracts. In: The Network and Distributed System Security Symposium. California.
[16] Bhargavan, K., et al. (2016) Formal verification of smart contracts: Short paper. In: Proc of the 2016 ACM workshop on programming languages and analysis for security. France. pp. 91–96.