UDC 004.4'24

IDENTIFICATION AND LOCALIZATION OF VULNERABILITIES IN SMART CONTRACTS USING ATTENTION VECTORS ANALYSIS IN A BERT-BASED MODEL

Tereshchenko O. I. – Postgraduate student of the Department of Software Engineering, Odesa Polytechnic National University, Odesa, Ukraine.

Komleva N. O. – PhD, Associate Professor, Head of the Department of Software Engineering, Odesa Polytechnic National University, Odesa, Ukraine.

ABSTRACT

Context. With the development of blockchain technology and the increasing use of smart contracts, which are automatically executed in blockchain networks, the significance of securing these contracts has become extremely relevant. Traditional code auditing methods often prove ineffective in identifying complex vulnerabilities, which can lead to significant financial losses. For example, the reentrancy vulnerability that led to the DAO attack in 2016 resulted in the loss of 3.6 million ethers and the split of the Ethereum blockchain network. This underscores the necessity for early detection of vulnerabilities.

Objective. The objective of this work is to develop and test an innovative approach for identifying and localizing vulnerabilities in smart contracts based on the analysis of attention vectors in a model using BERT architecture.

Method. The methodology described includes data preparation and training a transformer-based model for analyzing smart contract code. The proposed attention vector analysis method allows for the precise identification of vulnerable code segments. The use of the CodeBERT model significantly improves the accuracy of vulnerability identification compared to traditional methods. Specifically, three types of vulnerabilities are considered: reentrancy, timestamp dependence, and tx.origin vulnerability. The data is preprocessed, which includes the standardization of variables and the simplification of functions.

Results. The developed model demonstrated a high F-score of 95.51%, which significantly exceeds the results of contemporary approaches, such as the BGRU-ATT model with an F-score of 91.41%. The accuracy of the method in the task of localizing reentrancy vulnerabilities was 82%.

Conclusions. The experiments conducted confirmed the effectiveness of the proposed solution. Prospects for further research include the integration of more advanced deep learning models, such as GPT-4 or T5, to improve the accuracy and reliability of vulnerability detection, as well as expanding the dataset to cover other smart contract languages, such as Vyper or LLL, to enhance the applicability and efficiency of the model across various blockchain platforms.

Thus, the developed CodeBERT-based model demonstrates high results in detecting and localizing vulnerabilities in smart contracts, which opens new opportunities for research in the field of blockchain platform security.

KEYWORDS: smart contracts, vulnerabilities, blockchain, machine learning, attention vector analysis, transformers, code security, code audit.

ABBREVIATIONS

NN is a Neural Network;
BERT is a Bidirectional Encoder Representations from Transformers;
GPT is a Generative Pre-trained Transformer;
GRU is a Gated Recurrent Unit;
LSTM is a Long Short-Term Memory;
RNN is a Recurrent Neural Network;
CNN is a Convolutional Neural Network;
AUC is an Area Under the Curve;
ROC is a Receiver Operating Characteristic;
RGB is Red, Green, Blue;
TP is a True Positive;
FN is a False Negative;
FP is a False Positive;
TN is a True Negative.

NOMENCLATURE

C is a set of all smart contracts;
V is a set of all possible vulnerabilities;
T is a set of tokens in a smart contract code;
t_i is the i-th token in a smart contract code;
A is a set of attention weights for tokens;
a is an attention weights for the token;
$f()$ is a model that maps a smart contract to a set of vulnerabilities;
w_{ij} is an attention weight from token t_i to t_j token;
D is a dataset;
Q is a query matrix;
K is a key matrix;
V is a value matrix;
P is a value of precision;
R is a value of recall;
d_k is a key dimension;
F_β is an F score, which is a weighted harmonic mean of precision and recall;
β is a weight of recall in the F_β score;
windows_size is a size of the sliding window;
token_att_mean is a mean attention value for a token across all axes;
$windows_sum_j$ is a sum of averaged attentions, where j is the index of the start of the window in the sequence of tokens;
start_index is a starting index from which the total windows_sum is calculated;
FPR_i is a False Positive Rate value at the i-th point;
TPR_i is a True Positive Rate value at the i-th point.

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

INTRODUCTION

With the development of blockchain technology and the increasing popularity of smart contracts, the need to ensure their security has grown. Smart contracts, which are automatically executed when predetermined conditions are met, have become the foundation for numerous applications, ranging from financial transactions to voting systems. However, like any software code, smart contracts are susceptible to vulnerabilities that can lead to significant financial losses and a loss of trust in the technology. According to industry research, companies lose billions of dollars each year due to smart contract breaches, highlighting the critical need to improve methods for their protection. For example, the 2016 hack of TheDAO, in which 3.6 million Ether were stolen due to a reentrancy vulnerability, led to the split of the Ethereum blockchain and underscores the critical necessity for early detection of vulnerabilities [1].

The object of study is the security of smart contracts deployed on blockchain networks.

Detecting and eliminating vulnerabilities in smart contracts before they are deployed on the blockchain is a critically important task. Traditional code auditing methods, including manual analysis and automated static and dynamic analysis tools, often fail to fully ensure the security of smart contracts due to their limitations in identifying complex and non-obvious vulnerabilities.

The subject of study includes the methods of vulnerability detection and localization within smart contracts using machine learning techniques, with a focus on transformer-based models like BERT.

Modern methods for detecting vulnerabilities in smart contracts include symbolic execution, fuzzing, and formal verification. Well-known tools for vulnerability detection, such as Oyente, Mythril, Securify, Slither, and Smartcheck, automatically analyze contract code and can identify common types of vulnerabilities, including reentrancy issues, incorrect authorization via tx.origin, timestamp dependencies, and unhandled exceptions. However, these tools may produce false positives or miss real threats due to their reliance on predefined rules, which cannot accurately interpret complex code logic. Additionally, these preset rules quickly become outdated and cannot adapt or generalize to new data that continually evolves in the smart contract domain. Unlike these methods, deep learning approaches extract knowledge from data and can continuously update, maintaining their relevance. In recent years, researchers have been actively exploring the application of machine learning methods for software code analysis. Transformer-based models, such as BERT and its adaptations for code like CodeBERT, have shown promising results in understanding code semantics and identifying potential vulnerabilities.

The purpose of the work is to develop and validate a novel method for identifying and localizing vulnerabilities in smart contracts using attention vector analysis implemented through a CodeBERT-based model, improving the accuracy and efficiency of smart contract audits.

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

1 PROBLEM STATEMENT

The task of identifying vulnerabilities in smart contracts can be formalized as follows. Let C be the set of all smart contracts, and V be the set of all possible vulnerabilities. It is necessary to build a model $f: C \rightarrow V^*$, where $f(c)$ returns the set of vulnerabilities for each contract $c \in C$.

For each contract c a set of tokens $T=\{t_1, t_2, ..., t_n\}$, is determined, where t_i is the i-th token of the code. The analysis of attention vectors allows determining the attention weights $A=\{a_1, a_2, ..., a_n\}$, where a_i is the attention weight for token t_i

For each smart contract, it is necessary to identify vulnerable code segments using attention vectors. Let w_{ij} be the attention weight from token t_i to token t_j. Then the overall attention weight for token t_i is defined as:

$$a_i = \sum_{j=1}^{n} w_{ij}.$$

2 REVIEW OF THE LITERATURE

In this section, we analyze scientific works dedicated to identifying vulnerabilities in smart contracts using deep learning technologies.

Huang et al. [2] developed a model for detecting vulnerabilities in smart contracts using convolutional neural networks. This model transforms the binary representation of vulnerable code into RGB images, complicating the preservation of syntactic and semantic information and leading to a high level of false negatives, despite improving accuracy in some cases.

Liao et al. [3] utilized N-gram modeling and tf-idf feature vectors for analyzing the source code of smart contracts. They trained traditional machine learning models to identify 13 types of vulnerabilities using real-time fuzz testing. However, designating some critical operational codes as stop-words may lead to missed vulnerabilities and false negatives.

Yu et al. [4] presented the first systematic and modular framework for detecting vulnerabilities in smart contracts based on deep learning. They introduced the concept of a “Vulnerability Candidate”, focused on analyzing dependencies between different data elements and control flow. Experiments showed a significant improvement in efficiency by 25.76% in F1 score. However, for vulnerabilities with limited data and control flow dependencies, no substantial improvement was observed.

Gao et al. [5] proposed an automated method based on word embedding representations for studying the features of smart contracts in the Solidity language. Zhang et al. [6] developed a vulnerability detection method that combines information graphs and integrated learning to extract features from smart contracts. Sendner et al. [7] were the first to propose a migration learning-based method for vulnerability detection, which uses a universal feature extractor to analyze smart contract bytecode and independent branches to analyze each type of vulnerability. Zhuang et al. [8] were the first to propose using a contract graph to represent the syntactic and semantic structures of

OPEN ACCESS

smart contracts and applied graph convolutional neural networks for analyzing vulnerabilities based on this graph.

The analysis of existing scientific works shows that static analysis-based tools suffer from false positives and false negatives due to their reliance on predefined rules [9]. These tools are incapable of performing deep syntactic and semantic analysis, and predefined rules quickly become outdated, unable to adapt or generalize to new data. Unlike them, deep learning methods do not require predefined detection rules and can adaptively learn the characteristics of vulnerabilities during the training process.

3 MATERIALS AND METHODS

Transformers represent a neural network architecture that was first introduced in the paper "Attention is All You Need" in 2017 [10]. The main innovation of transformers is the attention mechanism, which allows models to dynamically focus on different parts of the input data, making it particularly effective for natural language processing tasks. This architecture differs from previous approaches, such as RNN and CNN, in that it is entirely based on attention without the need for sequential data processing. This significantly accelerates training and improves the handling of long dependencies in text [11].

BERT is one of the most well-known implementations of the transformer architecture, developed by Google in 2018. The main difference between BERT and preceding transformer models lies in its ability to process texts in a bidirectional manner. Traditional models, such as GPT, process text either left-to-right or right-to-left, limiting the context available to each word in a sentence. In contrast, BERT analyzes context in both directions, enabling it to better understand the contextual relationships between words. Figure 1 provides an example of the general architecture of source code classification models based on CodeBERT [12, 13].

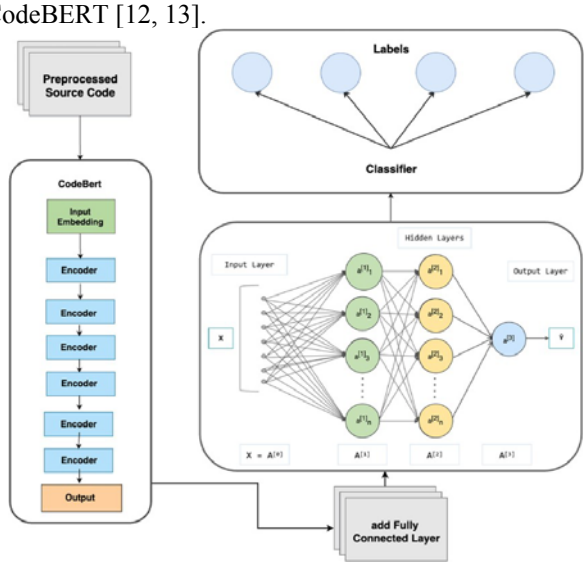

Figure 1 – General architecture of classification models based on BERT

In the context of smart contract analysis, BERT can be used for various tasks, including classification, vulnerability detection, and automatic code correction. Thanks to its ability to capture complex dependencies in the data, BERT effectively handles the syntactic and semantic features of smart contract programming languages such as Solidity [14]. This makes it an ideal tool for identifying potential vulnerabilities and errors in the code, which is critically important for ensuring the security and reliability of blockchain platforms.

An important stage of our research is data preparation, which underpins the training of a machine learning model for detecting vulnerabilities in smart contracts. To this end, we collected a dataset of 2000 Solidity smart contract source codes, each of which was analyzed using the static code analysis tool Oyente, designed to identify vulnerabilities and issues in smart contracts written in Solidity.

The dataset was divided into two categories: vulnerable and non-vulnerable smart contracts. The analysis determined that approximately 80% of the smart contracts do not contain vulnerabilities, while the remaining 20% contain one or more vulnerabilities identified using Oyente. We selected three types of vulnerabilities from the Oyente analysis results: reentrancy, timestamp dependence, and tx.origin vulnerability.

To enhance the efficiency and accuracy of model training, special attention was paid to the preprocessing of smart contract source code. In particular, the following normalization strategies were implemented:

1. Normalization of variables: All variables in the source code were renamed to a standardized format (e.g., VAR1, VAR2, ..., VARN). This reduced the diversity of the model's input data and minimized the risk of overfitting to specific or unique variable names that do not carry functional significance. This approach promotes more generalized model training, enabling it to better adapt to new, previously unseen smart contracts.

2. Simplification of functions: Auxiliary functions that do not impact the core logic of the contracts, such as logging functions or helper functions used for code simplification, were excluded. This reduces the complexity of the code input to the model and allows it to focus on functions that directly affect the contract's state and security. The remaining functions were standardized to eliminate variability in naming and approaches to performing similar operations, which also contributes to more stable and predictable model training.

To ensure effective model training, each smart contract was transformed into a format suitable for neural network processing. This included tokenizing the text of the smart contracts using the pre-trained tokenizer associated with the CodeBERT model, specifically the RobertaTokenizer from the Transformers library. However, processing long texts has always been a challenging task in the field of deep learning. Therefore, the maximum input sequence length was limited to 256 tokens, which allowed for a balance between the detail of data representation and computational resource requirements. If the extracted code contained more than 256 tokens, it was

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

truncated, and code with fewer than 256 tokens was padded with zeros. This approach ensures standardized input data length, adheres to the length limits set for the CodeBERT model, and preserves the features of the vulnerabilities.

To evaluate the model's effectiveness, the dataset D was divided into three subsets: training, validation, and test:

$$D = D_{train} \cup D_{val} \cup D_{test} \,. \tag{1}$$

The split was performed randomly while maintaining the proportions between vulnerable and non-vulnerable smart contracts. The validation subset, comprising 20% of the total examples, was used for hyperparameter tuning and evaluating intermediate training results, while the test subset, also comprising 20% of the total examples, was used for the final assessment of the model's performance.

As the basis for vulnerability detection, we used the pre-trained CodeBERT model from Microsoft, which is an adaptation of the BERT model specifically designed for working with source code. The model was further trained on the prepared smart contract dataset to adapt it to the specifics of the task.

Smart contracts in text form are fed into the model. These texts represent the source code of smart contracts written in Solidity. The input text is transformed into a sequence of tokens using the pre-trained tokenizer associated with the CodeBERT model. The tokenizer converts the source code into a set of tokens that the model can effectively process [15]. The token sequence is fed into the BERT model (CodeBertModel). CodeBERT processes the tokens using attention mechanisms and transformers to extract contextualized vector representations of the tokens.

Instead of relying solely on the hidden state of the [CLS] token, which aggregates information across the entire sequence and is traditionally used for classification, it is proposed to use all tokens in the sequence. These tokens are fed into a single-layer Bidirectional GRU. The use of Bidirectional GRU allows the model to better capture the context in both directions (left-to-right and right-to-left), which improves the understanding of contextual relationships between tokens in a broader context [16]. This can lead to a more accurate understanding of the meaning of the entire input text and, consequently, to improved classification accuracy.

The output from the bidirectional GRU is then fed into a fully connected layer (nn.Linear), which converts the GRU output into logits for each class (in our case, binary classification into vulnerable or non-vulnerable smart contracts). The logits are converted into probabilities using the softmax function, and the class with the highest probability is chosen as the model's prediction. The model continues to minimize the CrossEntropyLoss function during training, which helps measure the difference between the model's predictions and the true class labels. The model's effectiveness is evaluated on the validation and test datasets to confirm its ability to generalize to new data [17]. Figure 2 shows the overall architecture of the proposed model.

To detect and localize vulnerabilities in smart contracts, we used attention vector analysis generated by the CodeBERT model. The attention vectors were analyzed to identify the tokens and code fragments that the model focused on most during classification. This approach allowed us not only to detect potential vulnerabilities but also to pinpoint specific locations in the code that require further analysis and corrections. This process includes several key stages:

Stage 1 involves obtaining predictions and attention vectors. The model processes the input data and returns logits for each token and attention weights. The attention weights indicate how much attention the model pays to each token while analyzing other tokens. The attention weights are calculated as follows:

$$Attention(Q,K,V) = soft\max\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2}$$

Stage 2 involves the selection of attention vectors from the last layer. Attention vectors from the last layer of the transformer are selected because they reflect the model's highest-level understanding of the context.

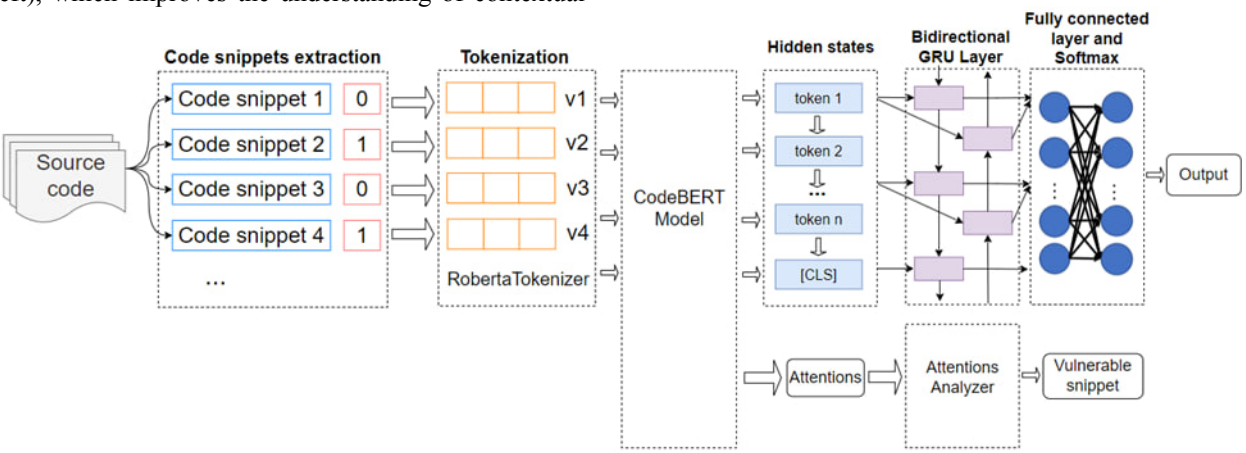

Figure 2 – General architecture of the proposed model

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

Stage 3 focuses on averaging attention across heads. Attention is averaged across all heads of the attention mechanism in the last layer to obtain a generalized representation of how the model distributes its attention among tokens.

Stage 4 includes additional averaging across tokens. After averaging attention across the heads of the attention mechanism, additional averaging is performed across all tokens in the sequence. This averaging helps produce a single attention vector for the entire input set, simplifying the analysis and interpretation of which aspects of the input data the model pays the most attention to overall.

Stage 5 entails the exclusion of special tokens. The first and last tokens are excluded from the analysis because they usually contain meta-information ([CLS], [SEP]) and are not related to the substantive part of the smart contract code.

Figure 3 shows the stages of transforming the attention matrix of the last layer.

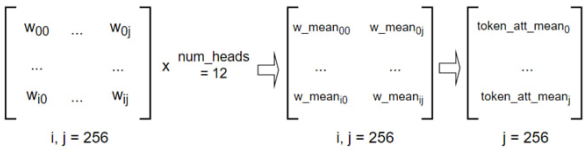

Figure 3 – Stages of transformation of the attention matrix of the last layer

Stage 6 determines important code segments. By iterating through all possible windows of a given size, the sum of averaged attention in each window is calculated. The window with the maximum sum of averaged attention is considered the most important code segment.

For each possible window of size window_size tokens, the sum of averaged attention is calculated using the following formula:

$$windows_sum_j = \sum_{i=j}^{j+window_size} token_att_mean[i]. \quad (3)$$

Next, the indices of the tokens corresponding to the window with the highest attention are determined. These tokens represent the code segment that the model considers most significant or potentially vulnerable.

The starting index of the window is calculated as follows:

$$start_index = \arg\max_j (window_sum_j). \quad (4)$$

Based on the starting index of the window and the window size, the tokens corresponding to the most significant code segment are selected. These tokens are then converted back to text using the tokenizer to represent the important code fragments. The result is a segment of the smart contract code that the model considers most likely to contain vulnerabilities.

This approach provides a deeper understanding of which parts of the code attract the most attention from the model, potentially indicating the presence of vulnerabilities or other critical aspects of the code.

4 EXPERIMENTS

For training the model and conducting experiments, the following software and technical resources were used. Development and testing were carried out in the Python programming language, providing flexibility and powerful capabilities for working with machine learning algorithms. The primary framework used for working with the model was Pytorch Lightning, which structured the model training process, making it cleaner, more modular, and scalable.

All development and testing were conducted on the Windows 11 operating system.

The technical configuration of the computer used for training and experiments included the following specifications:

Processor: Intel Core i9-12900K, providing high performance with its 16 cores and 24 threads, and a maximum clock speed of 5.2 GHz.

RAM: 32 GB DDR4, allowing efficient handling of large data volumes and complex models without memory constraints.

Graphics Card: NVIDIA GeForce RTX 3090 with 24 GB of GDDR6X memory.

Storage: 1 TB SSD, ensuring fast data access and efficient storage of extensive datasets and experimental results.

This configuration provided the necessary computational power and speed required for handling complex machine learning tasks and data analysis.

The model was trained using the AdamW optimizer and a learning rate scheduler, which effectively adapted the learning rate depending on the training stage. A batch size of 32 was used during training, which was conducted over 8 epochs, each consisting of 4 steps. The model included one GRU layer with a hidden state size of 16. These parameters helped avoid overfitting while achieving the best results. During training, metrics such as accuracy and loss on the training and validation sets were monitored.

Thanks to the careful tuning of parameters and the model architecture, an accuracy of 98.67% on the training data and 97.34% on the validation data was achieved (Figures 4 and 6). These results underscore the high effectiveness and adequacy of the chosen approach for solving the task.

The dynamics of the validation loss values for the training and validation data are visualized in Figures 5 and 7, respectively. These graphs illustrate how the model gradually minimized errors throughout the training process, achieving progressively lower loss values.

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

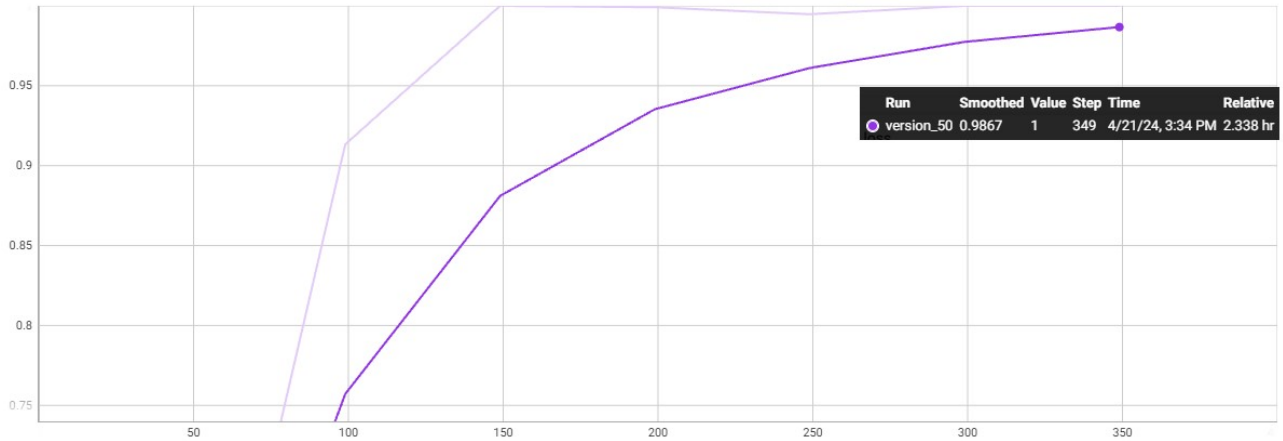

Figure 4 – Change in model accuracy during training

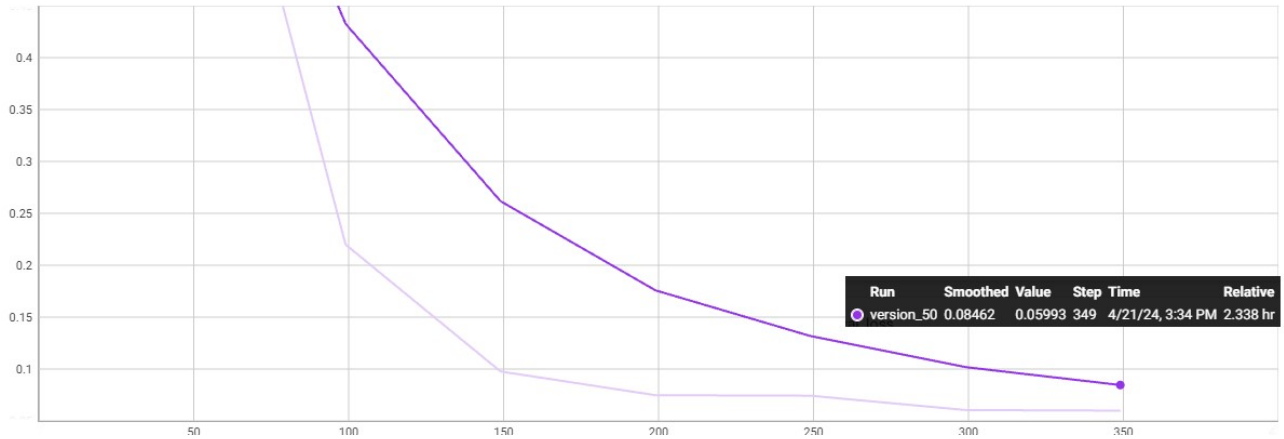

Figure 5 – Training Loss over epochs

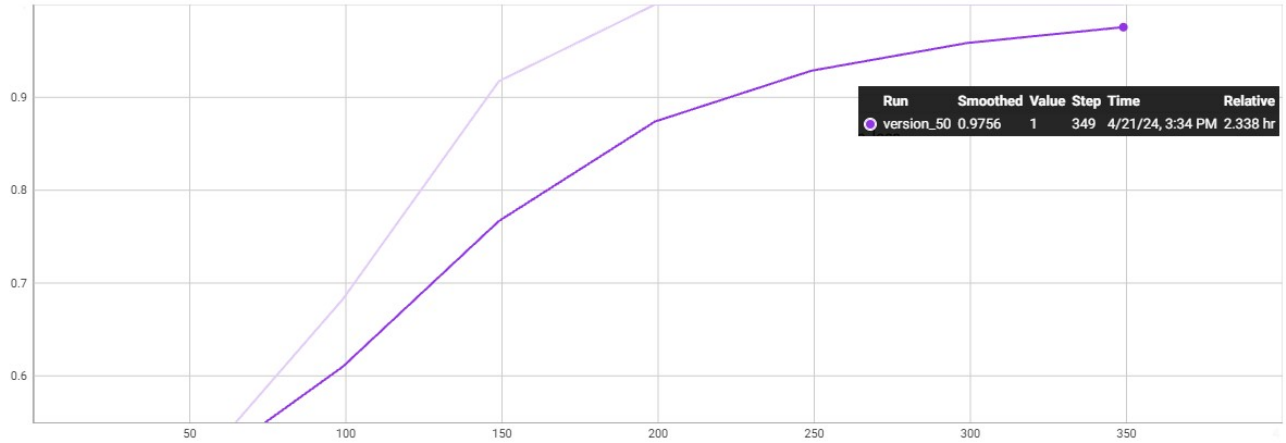

Figure 6 – Change in model accuracy during validation

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

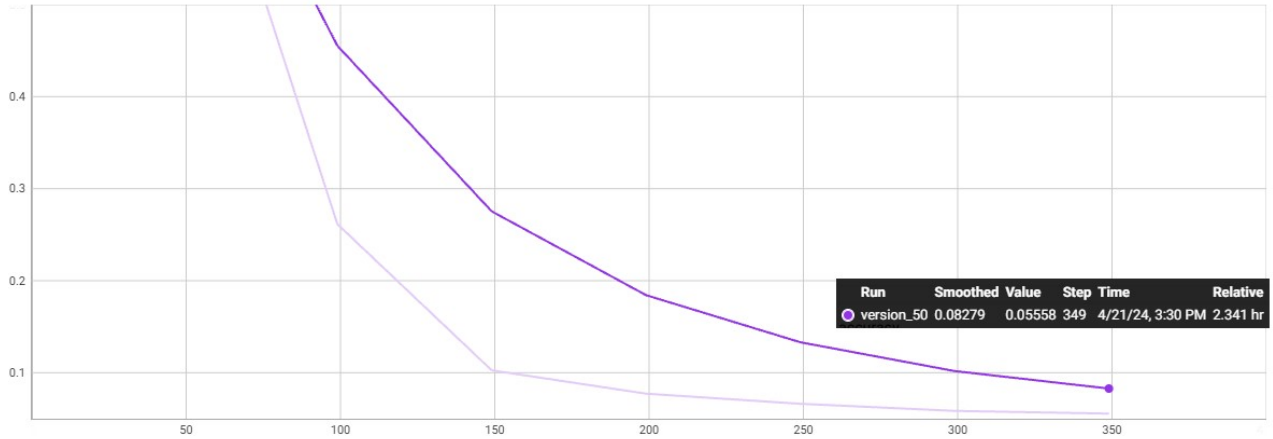

Figure 7 – Validation Loss over epochs

Figure 8 shows the confusion matrix for the test data. As can be seen from the matrix, only 7 examples were classified as False Positives and 5 as False Negatives, indicating the model's high capability to accurately identify positive cases [18]. Using the obtained confusion matrix, the model's accuracy can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = 0.97. \quad (5)$$

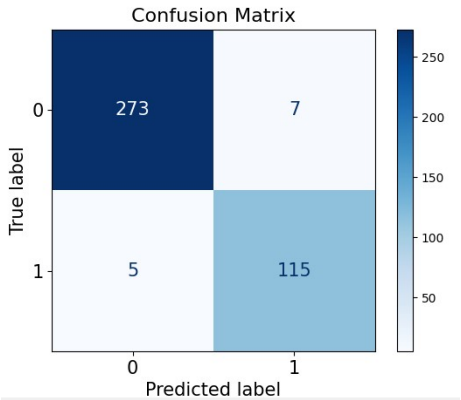

Figure 8 – Confusion matrix for test data

Furthermore, Figure 9 shows the ROC curve, with an AUC reaching 0.97, which is close to 1. This indicates that the model has excellent discriminative ability and can effectively distinguish between classes.

The area under the curve is calculated using the following formula:

$$AUC = \sum_{i=1}^{n-1} ((FPR_{i+1} - FPR_i) \frac{TPR_{i+1} + TPR_i}{2}). \quad (6)$$

Thus, the training and testing results confirm that the developed model is a reliable prediction tool capable of providing high accuracy and excellent generalization ability on new data.

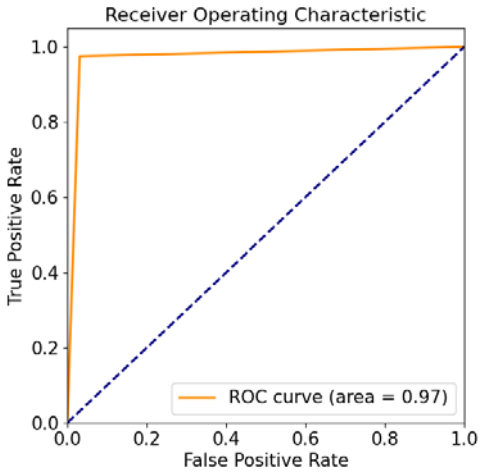

Figure 9 – ROC-curve

5 RESULTS

In our research, we analyzed various smart contracts for vulnerabilities using modern static code analysis tools. Figure 10 shows an example of Solidity code that demonstrates the classic reentrancy vulnerability.

```
contract MyContract {
    event Withdrawal(address indexed user, uint256 amount);
    event Deposit(address indexed user, uint256 amount);
    mapping(address => uint256) public balances;
    function deposit() public payable {
        balances[msg.sender] += msg.value;
        emit Deposit(msg.sender, msg.value);
    }
    function withdraw(uint256 amount) public {
        require(balances[msg.sender] >= amount, "Insufficient balance");
        (bool success, ) = msg.sender.call{value: amount}("");
        require(success, "Withdrawal failed");
        balances[msg.sender] -= amount;
        emit Withdrawal(msg.sender, amount);
    }
    function getBalance() public view returns (uint256) {
        return balances[msg.sender];
    }

}
```

Figure 10 – Example of vulnerable code

This smart contract code contains a withdraw function that may be vulnerable to reentrancy attacks due to the sequence of operations. The function first checks the balance, then makes an external call to send funds (msg.sender.call{value: amount}("")), and only after that decreases the balance. This leaves room for an attacker to repeatedly call the withdraw function during the execution of the external call, potentially allowing them to withdraw more funds than they are entitled to if the at-

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

tacker controls the calling address. This violates the recommended "checks-effects-interactions pattern" design pattern, which dictates that state changes should be made before external calls [19].

To eliminate this vulnerability and enhance the security of the smart contract, it is recommended to restructure the operation logic, ensuring that all state changes are performed before calling external contracts.

During the analysis, our tool highlighted the following code segment, shown in Figure 11, as potentially vulnerable.

```
Vulnerable code snippet:

(bool success, ) = msg.sender.call{value: amount}("");
require(success, "Withdrawal failed");
balances[msg.sender] -= amount;
```

Figure 11 – Detected vulnerable code

Confirming this code segment as vulnerable not only demonstrates the risks associated with improper use of external calls in smart contracts but also validates the effectiveness of our analysis method. This highlights the importance of applying strict security patterns when developing smart contracts and the necessity of using static code analysis tools to identify and eliminate potential vulnerabilities before deploying contracts on the network.

In our research, we used heatmap visualization to analyze the attention matrices obtained from the implemented model. The heatmap provides a clear representation of which tokens in the smart contract text attract the most attention from the model.

Figure 12 shows a graph where bright vertical stripes indicate that certain tokens on the X-axis receive significant attention from many other tokens in the sequence. This suggests that such tokens may play a key role in understanding the context or contain critically important information.

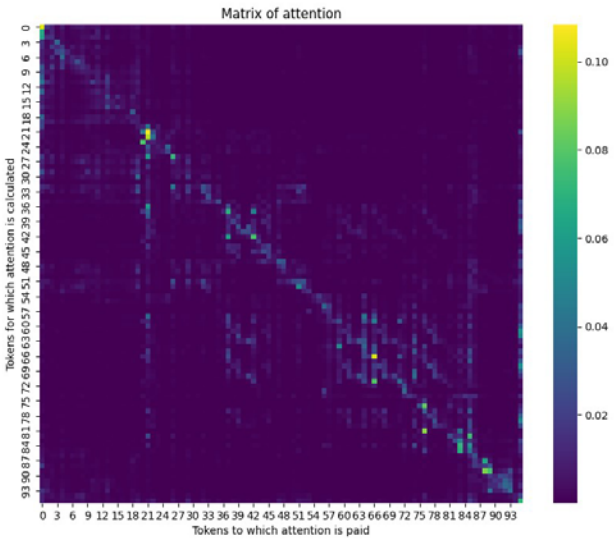

Figure 12 – Heat map of attention matrix

The model's attention to these tokens can help identify potential vulnerabilities or important aspects of the smart contract's logic, making this visualization method particularly valuable for analyzing and improving smart contract security.

In our research, we selected 50 smart contracts in which the Oyente static analysis tool identified vulnerabilities related to reentrancy issues. In each of these contracts, lines were marked where the incorrect order of method calls occurs, potentially leading to vulnerabilities. The analysis revealed that in 41 of these smart contracts, the code areas marked by the analyzer indeed contained the indicated lines, corresponding to an accuracy of 82%. This confirms the effectiveness of the applied analysis method for identifying potential vulnerabilities.

Table 1 – Experimental results

Total number of vulnerable contracts	Number of vulnerabilities correctly identified by the developed analyzer	Accuracy, %
50	41	82

In the paper [20], we evaluated various models based on key metrics. For the analysis, we selected models based on Simple RNN, LSTM, Bidirectional LSTM (BLSTM), Bidirectional GRU (BGRU), and Bidirectional LSTM with Attention Mechanism (BLSTM-ATT). The models were evaluated using the metrics precision, recall, and F-beta:

$$P = \frac{TP}{TP + FP}, R = \frac{TP}{TP + FN}, \quad (7)$$

$$F_\beta = \frac{(1+\beta^2)(P * R)}{(\beta^2 * P + R)}. \quad (8)$$

The developed CodeBERT-GRU model demonstrates significantly higher results, which are presented in Table 2.

Table 2 – Comparison of the developed model with alternative models

Model	**Precision, %**	**Recall, %**	**F-score (β=2), %**
Simple RNN	66.34	64.85	65.14
LSTM	73.28	76.33	75.70
BLSTM	86.12	87.97	87.59
BGRU	86.05	88.10	87.68
BLSTM-ATT	89.87	90.66	90.50
BGRU-ATT	90.03	91.76	91.41
CodeBERT-GRU	94.26	95.83	95.51

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

In the course of the study, metrics such as precision, recall, and F-measure (β=2) were used to evaluate the effectiveness of the models. Among the considered models, the developed CodeBERT-GRU model showed the best results, highlighting its high efficiency in data processing. The precision of the CodeBERT-GRU model was 94.26%, recall was 95.83%, and the F-measure reached 95.51%. These indicators are significantly higher than those of other analyzed models, such as BLSTM-ATT and BGRU-ATT, which also showed high results with F-measures of 90.50% and 91.41%, respectively. This indicates that the integration of CodeBERT with LSTM not only improves the processing of contextual information but also provides a significant improvement in recognition and classification accuracy compared to traditional models based on RNN, LSTM, and GRU.

6 DISCUSSION

The presented study offers a novel approach to identifying and localizing vulnerabilities in smart contracts through the analysis of attention vectors in a BERT-based model, specifically using CodeBERT. Our method has demonstrated a significantly higher F-score of 95.51% compared to traditional approaches like the BGRU-ATT model, which achieved an accuracy of 91.41%. Moreover, our approach for localizing reentrancy vulnerabilities has shown an accuracy of 82%, underscoring the effectiveness of the proposed method.

In comparing our results with those of other authors, it becomes evident that the application of transformer-based models, particularly CodeBERT, provides a substantial improvement in detecting and understanding code semantics.

Gao et al. [5] and Zhang et al. [6] explored automated methods based on word embeddings and integrated learning to extract features from smart contracts. Sendner et al. [7] and Zhuang et al. [8] proposed using migration learning and graph convolutional neural networks for vulnerability analysis, respectively. While these methods have shown promise, our approach's integration of CodeBERT with bidirectional GRU layers enhances the model's ability to contextualize and understand code, leading to superior performance metrics.

A major issue with previous machine learning-based methods, including those described in ESCORT [21], was their inability to precisely pinpoint the vulnerable code segments; they could only indicate whether the code was vulnerable or not. ESCORT, for instance, leverages a multi-output neural network architecture with a common feature extractor and multiple branch structures, achieving an average F1-score of 95% on six vulnerability types and 93% when extended to new types. However, it still primarily focuses on whether a contract is vulnerable and lacks the precise localization of vulnerabilities within the code. Our method addresses this limitation by enabling precise localization of the vulnerable code segments, providing more detailed and actionable insights for developers.

Our model's performance metrics, with a precision of 94.26% and a recall of 95.83%, indicate a well-balanced approach to vulnerability detection. The high precision value signifies that the model is highly effective in minimizing false positives, ensuring that the identified vulnerabilities are indeed present in the smart contracts. This reduces the likelihood of unnecessary alarm and enables developers to focus on actual issues. Meanwhile, the high recall value demonstrates the model's capability to identify the majority of actual vulnerabilities, minimizing false negatives and ensuring that most vulnerabilities are detected. This balance between precision and recall reflects the robustness of our model in maintaining high accuracy while effectively reducing both false positives and false negatives.

The limitations of our study include the focus on Solidity smart contracts and the reliance on static code analysis. Expanding the dataset to include other smart contract languages such as Vyper and LLL could enhance the generalizability of our model. Furthermore, incorporating dynamic analysis techniques alongside static analysis could provide a more comprehensive understanding of the contract's behavior, thus increasing the detection rate of complex vulnerabilities that manifest only during execution.

Practically, the results of our research can be applied to improve the security auditing processes for smart contracts. By integrating our model into existing auditing tools, developers can identify and address vulnerabilities more effectively before deployment, reducing the risk of financial losses and enhancing trust in blockchain technologies.

Future research directions include the integration of more advanced deep learning models such as GPT-4 or T5, which could further improve the accuracy and robustness of vulnerability detection. Additionally, expanding the dataset to cover a wider variety of smart contract languages and incorporating dynamic analysis techniques could provide a more holistic approach to smart contract security.

In conclusion, our study demonstrates that the application of transformer-based models like CodeBERT significantly enhances the detection and localization of vulnerabilities in smart contracts. This approach offers a promising direction for future research and practical applications in the field of blockchain security.

CONCLUSIONS

In our study, we presented an innovative approach to identifying and localizing vulnerabilities in smart contracts using attention vector analysis in the CodeBERT model. This method not only effectively determines the presence of vulnerabilities but also precisely points to the areas in the code that require developers' attention. This has been made possible by the deep understanding of the context and semantics of the code, which is a significant advantage over traditional auditing methods.

We successfully achieved an accuracy of 97.34% with the developed CodeBERT model, which is significantly

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

higher compared to the accuracy of the BGRU-ATT model, which was 90.03%. Furthermore, the vulnerability localization method demonstrated an accuracy of 82% in identifying reentrancy vulnerabilities, confirming the effectiveness of this approach for detecting specific types of vulnerabilities in smart contracts.

The confirmation of the effectiveness of our approach is reflected in the significant improvement in vulnerability detection results compared to existing methods, as highlighted in our experimental results. The use of the CodeBERT model for analyzing smart contracts has opened new opportunities for research in the field of blockchain security.

Finally, the results of our study can serve as a foundation for further development of machine learning methods in the field of blockchain platform cybersecurity. They also emphasize the importance of continuing research in this area, aimed at improving auditing technologies and smart contract development to ensure their reliability and security.

The scientific novelty of our research is rooted in the development and validation of a novel approach for identifying and localizing vulnerabilities in smart contracts using attention vector analysis within a BERT-based model. Unlike traditional methods, our approach leverages the deep contextual understanding provided by CodeBERT, significantly enhancing the model's accuracy and robustness. A major issue with previous machine learning-based methods was their inability to precisely pinpoint the vulnerable code segments; they could only indicate whether the code was vulnerable or not. Our innovative method addresses this limitation by enabling precise localization of the vulnerable code segments, offering more detailed and actionable insights for developers. This advancement marks a significant step forward in the application of transformer-based models to the field of blockchain security.

The practical significance of our research lies in its potential to enhance security auditing processes for smart contracts. By integrating our model into existing auditing tools, developers can more effectively identify and address vulnerabilities before deployment, thereby reducing the risk of financial losses and enhancing trust in blockchain technologies.

Prospects for further research include several promising directions based on the results obtained in this study. First, the integration of more advanced deep learning models, such as transformers with enhanced attention mechanisms like GPT-4 or T5, could further improve the accuracy and robustness of vulnerability detection and localization in smart contracts. Second, expanding the dataset to include a wider variety of smart contract languages beyond Solidity, such as Vyper or LLL, could generalize the model's applicability and effectiveness across different blockchain platforms. Additionally, incorporating dynamic analysis techniques alongside the static analysis employed in this study could provide a more comprehensive understanding of the contract's behavior, thereby increasing the detection rate of complex vulnerabilities that manifest only during execution. These future directions hold the potential to greatly advance the field of smart contract security and contribute to the broader adoption and trust in blockchain technologies.

ACKNOWLEDGEMENTS

We would like to express our sincere appreciation to the Department of Software Engineering at Odesa Polytechnic National University for their invaluable support throughout this research.

REFERENCES

1. Komleva N. O., Tereshchenko O. I. Requirements for the development of smart contracts and an overview of smart contract vulnerabilities at the Solidity code level on the Ethereum platform, *Herald of Advanced Information Technology*, 2023, Vol. 6, № 1, pp. 54–68. DOI: 10.15276/hait.06.2023.4
2. Huang T. H.-D. Hunting the ethereum smart contract: Color-inspired inspection of potential attacks [Electronic resource]. Access mode: https://arxiv.org/abs/1807.01868
3. Liao J.-W., Tsai T.-T., He C.-K. et al. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing, *IOTSMS 2019 : Sixth International Conference on Internet of Things: Systems, Management and Security, Granada, 22–25 October 2019 : proceedings*. New York, NY, IEEE Press, 2019, pp. 458–465. DOI: 10.1109/IOTSMS48152.2019.8939256
4. Yu X., Hou B., Ying Z. et al. Deep learning-based solution for smart contract vulnerabilities detection, *Scientific Reports,* 2023, Vol. 13, P. 20106. DOI: 10.1038/s41598-023-47219-0
5. Gao Z., Jiang L., Xia X. et al. Checking Smart Contracts With Structural Code Embedding, *IEEE Transactions on Software Engineering,* 2021, Vol. 47, № 12, pp. 2874–2891. DOI: 10.1109/TSE.2020.2971482
6. Zhang L., Wang J., Wang W. et al. A Novel Smart Contract Vulnerability Detection Method Based on Information Graph and Ensemble Learning, *Sensors,* 2022, Vol. 22, P. 3581. DOI: 10.3390/s22093581
7. Sendner C., Chen H., Fereidooni H. et al. Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning, *Network and Distributed System Security : Symposium 2023, San Diego, 27–03 February–March 2023 : proceedings*. Reston, VA: The Internet Society, 2023.
8. Zhuang Y., Liu Z., Qian P. et al. Smart Contract Vulnerability Detection using Graph Neural Network [Electronic resource], *IJCAI'20: Twenty-Ninth International Joint Conference on Artificial Intelligence, 07–15 January 2021 : proceedings*. Electronic resource, IJCAI, 2021, pp. 3283–3290. DOI: 10.24963/ijcai.2020/454
9. Park D., Zhang Y., Saxena M. et al. A formal verification tool for ethereum vm bytecode, *ESEC/FSE '18: 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista FL, 04–09 November 2018 : proceeding*s. New York, Association for Computing Machinery, 2018, pp. 912–915. DOI: 10.1145/3236024.3264591
10. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser L., Polosukhin I. Attention Is All You Need [Electronic resource]. Access mode: https://arxiv.org/abs/1706.03762. DOI: 10.48550/arXiv.1706.03762

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

11. Tereshchenko O. I., Komleva N. O. Vulnerability Detection of Smart Contracts Based on Bidirectional GRU and Attention Mechanism, *Information and Communication Technologies in Education, Research, and Industrial Applications 2023: 18th International Conference, Ivano-Frankivsk, 18–22 September 2023 : proceedings*. Berlin, Springer, 2023, Vol. 1980, pp. 276–287. DOI: 10.1007/978-3-031-48325-7_21
12. Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., Stoyanov V. Roberta: A robustly optimized bert pretraining approach [Electronic resource]. Access mode: https://arxiv.org/abs/1907.11692
13. Yu X., Zhao H., Hou B. et al. DeeSCVHunter: A deep learning-based framework for smart contract vulnerability detection, *IJCNN '21 : 2021 International Joint Conference on Neural Networks, Shenzhen, 18–22 July 2021 : proceedings*. New York, NY, IEEE Press, 2021, pp. 1–8. DOI: 10.1109/IJCNN52387.2021.9534324
14. Harer J. A., Ozdemir O., Lazovich T. et al. Learning to repair software vulnerabilities with generative adversarial networks, *NeurIPS 2018 : 32nd Conference on Neural Information Processing Systems, Montreal, 03–08 December 2018 : proceedings*. Red Hook, NY, Curran Associates Inc., 2018, pp. 7933–7943. DOI: 10.48550/arXiv.1805.07475.
15. Zhou Y., Liu S., Siow J. et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks, NeurIPS 2019 : 33rd Conference on Neural Information Processing Systems, Vancouver, 08–14 December 2019 : proceedings. Red Hook, NY: Curran Associates Inc., 2019, Vol. 32. DOI: 10.48550/arXiv.1909.03496
16. Tsankov P., Dan A., Drachsler-Cohen D. et al. Securify: Practical security analysis of smart contracts, *CCS '18 : 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, 15–19 October 2018 : proceedings*. New York, NY, ACM, 2018, pp. 67–82. DOI: 10.1145/3243734.3243780
17. Huang J., Han S., You W. et al. Hunting vulnerable smart contracts via graph embedding based bytecode matching, *IEEE Transactions on Information Forensics and Security,* 2021, Vol. 16, pp. 2144–2156. DOI: 10.1109/TIFS.2021.3050051
18. Yuan X., Lin G., Tai Y. et al. Deep neural embedding for software vulnerability discovery: Comparison and optimization, *Security and Communication Networks*, 2022, pp. 1–12. DOI: 10.1155/2022/5203217
19. Feist. J., Greico G., Groce A. Slither: A static analysis framework for smart contracts, *WETSEB '19: 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, Monteal, Quebec, 27 May 2019 : proceedings*, 2019. New York, NY, IEEE Press, 2019, pp. 8–15. DOI: 10.48550/arXiv.1908.09878
20. Lutz O., Chen H., Fereidooni H., Sendner C. ESCORT: Ethereum Smart COntRacTs Vulnerability Detection using Deep Neural Network and Transfer Learning [Electronic resource]. Access mode: https://arxiv.org/abs/2103.12607. DOI: 10.48550/arXiv.2103.12607
21. Rodler M., Li W., Karame G. O. et al. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks, *Network and Distributed System Security : Symposium 2019, San Diego, 24–27 February 2019 : proceedings*. Reston, VA, The Internet Society, 2023. DOI: 10.14722/ndss.2019.23413

Received 05.06.2024.
Accepted 12.08.2024.

УДК 004.4'24

ІДЕНТИФІКАЦІЯ ТА ЛОКАЛІЗАЦІЯ ВРАЗЛИВОСТЕЙ У СМАРТ-КОНТРАКТАХ З ВИКОРИСТАННЯМ АНАЛІЗУ ВЕКТОРІВ УВАГИ В МОДЕЛІ НА ОСНОВІ BERT

Терещенко О. І. – аспірант кафедри Інженерії програмного забезпечення Національного університету «Одеська політехніка», Одеса, Україна.

Комлева Н. О. – канд. техн. наук, доцент, завідувач кафедри Інженерії програмного забезпечення Національного університету «Одеська політехніка», Одеса, Україна.

АНОТАЦІЯ

Актуальність. З розвитком технології блокчейн та зростанням використання смарт-контрактів, які автоматично виконуються в блокчейн-мережах, значущість безпеки цих контрактів стала надзвичайно актуальною. Традиційні методи аудиту коду часто виявляються неефективними для виявлення складних уразливостей, що може призвести до значних фінансових втрат. Наприклад, уразливість повторного входу, яка призвела до атаки на DAO в 2016 році, спричинила втрату 3,6 мільйона ефірів та поділ блокчейн-мережі Ethereum. Це підкреслює необхідність раннього виявлення уразливостей.

Мета роботи – розробка та апробація новаторського підходу до виявлення та локалізації уразливостей у смарт-контрактах на основі аналізу векторів уваги в моделі, що використовує архітектуру BERT.

Метод. Описується методика, яка включає підготовку даних та навчання трансформерної моделі для аналізу коду смарт-контрактів. Запропонований метод аналізу векторів уваги дозволяє точно ідентифікувати уразливі ділянки коду. Використання моделі CodeBERT значно покращує точність ідентифікації уразливостей порівняно з традиційними методами. Зокрема, розглядаються три типи уразливостей: повторний вхід, залежність від часу та уразливість tx.origin. Дані попередньо нормалізуються, що включає стандартизацію змінних та спрощення функцій.

Результати. Розроблена модель продемонструвала високий F-score на рівні 95,51%, що значно перевищує результати сучасних підходів, таких як модель BGRU-ATT з F-score 91,41%. Точність методу у завданні локалізації уразливості повторного входу склала 82%.

Висновки. Проведені експерименти підтвердили ефективність запропонованого рішення. Перспективи подальших досліджень включають інтеграцію більш просунутих моделей глибокого навчання, таких як GPT-4 або T5, для покращення точності та надійності виявлення уразливостей, а також розширення набору даних для охоплення інших мов смарт-контрактів, таких як Vyper або LLL, для підвищення застосовності та ефективності моделі на різних блокчейн-платформах.

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15

Таким чином, розроблена модель на основі CodeBERT демонструє високі результати у виявленні та локалізації уразливостей у смарт-контрактах, що відкриває нові можливості для досліджень у сфері безпеки блокчейн-платформ.

КЛЮЧОВІ СЛОВА: смарт-контракти, вразливості, блокчейн, машинне навчання, аналіз векторів уваги, трансформери, безпека коду, аудит коду.

ЛІТЕРАТУРА

1. Komleva N. O. Requirements for the development of smart contracts and an overview of smart contract vulnerabilities at the Solidity code level on the Ethereum platform / N. O. Komleva, O. I. Tereshchenko // Herald of Advanced Information Technology. – 2023. – Vol. 6, № 1. – P. 54–68. DOI: 10.15276/hait.06.2023.4
2. Huang T. H.-D. Hunting the ethereum smart contract: Color-inspired inspection of potential attacks [Electronic resource] / T. H.-D. Huang. – Access mode: https://arxiv.org/abs/1807.01868
3. Soliaudit: Smart contract vulnerability assessment based on machine learning and fuzz testing / [J.-W. Liao, T.-T. Tsai, C.-K. He et al.] // IOTSMS 2019 : Sixth International Conference on Internet of Things: Systems, Management and Security, Granada, 22–25 October 2019 : proceedings. – New York, NY : IEEE Press, 2019. – P. 458–465. DOI: 10.1109/IOTSMS48152.2019.8939256
4. Deep learning-based solution for smart contract vulnerabilities detection / [X. Yu, B. Hou, Z. Ying et al.] // Scientific Reports. – 2023. – Vol. 13. – P. 20106. DOI: 10.1038/s41598-023-47219-0
5. Checking Smart Contracts With Structural Code Embedding / [Z. Gao, L. Jiang, X. Xia et al.] // IEEE Transactions on Software Engineering. – 2021. – Vol. 47. – № 12. – P. 2874–2891. DOI: 10.1109/TSE.2020.2971482
6. A Novel Smart Contract Vulnerability Detection Method Based on Information Graph and Ensemble Learning / [L. Zhang, J. Wang, W. Wang et al.] // Sensors. – 2022. – Vol. 22. – P. 3581. DOI: 10.3390/s22093581
7. Smarter Contracts: Detecting Vulnerabilities in Smart Contracts with Deep Transfer Learning / [C. Sendner, H. Chen, H. Fereidooni et al.] // Network and Distributed System Security : Symposium 2023, San Diego, 27–03 February–March 2023 : proceedings. – Reston, VA: The Internet Society, 2023.
8. Smart Contract Vulnerability Detection using Graph Neural Network [Electronic resource] / [Y. Zhuang, Z. Liu, P. Qian et al.] // IJCAI'20: Twenty-Ninth International Joint Conference on Artificial Intelligence, 07–15 January 2021 : proceedings. – Electronic resource: IJCAI, 2021. – P. 3283–3290. DOI: 10.24963/ijcai.2020/454
9. A formal verification tool for ethereum vm bytecode / [D. Park, Y. Zhang, M. Saxena et al.] // ESEC/FSE '18: 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista FL, 04–09 November 2018 : proceedings. – New York: Association for Computing Machinery, 2018. – P. 912–915. DOI: 10.1145/3236024.3264591
10. Vaswani A. Attention Is All You Need [Electronic resource] / A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin. – Access mode: https://arxiv.org/abs/1706.03762. DOI: 10.48550/arXiv.1706.03762
11. Tereshchenko O. I. Vulnerability Detection of Smart Contracts Based on Bidirectional GRU and Attention Mechanism / O. I. Tereshchenko, N. O. Komleva // Information and Communication Technologies in Education, Research, and Industrial Applications 2023: 18th International Conference, Ivano-Frankivsk, 18–22 September 2023 : proceedings. – Berlin: Springer, 2023. – Vol. 1980. – P.276–287. DOI: 10.1007/978-3-031-48325-7_21
12. Roberta: A robustly optimized bert pretraining approach [Electronic resource] / [Y. Liu, M. Ott, N. Goyal et al.]. – Access mode: https://arxiv.org/abs/1907.11692
13. DeeSCVHunter: A deep learning-based framework for smart contract vulnerability detection / [X. Yu, H. Zhao, B. Hou et al.] // IJCNN '21 : 2021 International Joint Conference on Neural Networks, Shenzhen, 18–22 July 2021 : proceedings. – New York, NY: IEEE Press, 2021. – P. 1–8. DOI: 10.1109/IJCNN52387.2021.9534324
14. Learning to repair software vulnerabilities with generative adversarial networks / [J. A. Harer, O. Ozdemir, T. Lazovich et al.] // NeurIPS 2018 : 32nd Conference on Neural Information Processing Systems, Montreal, 03–08 December 2018 : proceedings. – Red Hook, NY: Curran Associates Inc., 2018. – P. 7933–7943. DOI: 10.48550/arXiv.1805.07475.
15. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks / [Y. Zhou, S. Liu, J. Siow et al.] // NeurIPS 2019 : 33rd Conference on Neural Information Processing Systems, Vancouver, 08–14 December 2019 : proceedings. – Red Hook, NY: Curran Associates Inc., 2019. – Vol. 32. DOI: 10.48550/arXiv.1909.03496
16. Securify: Practical security analysis of smart contracts / [P. Tsankov, A. Dan, D. Drachsler-Cohen et al.] // CCS '18 : 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, 15–19 October 2018 : proceedings. – New York, NY: ACM, 2018. – P. 67–82. DOI: 10.1145/3243734.3243780
17. Hunting vulnerable smart contracts via graph embedding based bytecode matching / [J. Huang, S. Han, W. You et al.] // IEEE Transactions on Information Forensics and Security. – 2021. – Vol. 16. – P. 2144–2156. DOI: 10.1109/TIFS.2021.3050051
18. Deep neural embedding for software vulnerability discovery: Comparison and optimization / [X. Yuan, G. Lin, Y. Tai et al.] // Security and Communication Networks. – 2022. – P. 1–12. DOI: 10.1155/2022/5203217
19. Feist. J. Slither: A static analysis framework for smart contracts / J. Feist, G. Greico, A. Groce // WETSEB '19: 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain, Monteal, Quebec, 27 May 2019 : proceedings. – 2019. – New York, NY: IEEE Press, 2019. – P. 8–15. DOI: 10.48550/arXiv.1908.09878
20. ESCORT: Ethereum Smart COntRacTs Vulnerability Detection using Deep Neural Network and Transfer Learning [Electronic resource] / [O. Lutz, H. Chen, H. Fereidooni, C. Sendner]. – Access mode: https://arxiv.org/abs/2103.12607. DOI: 10.48550/arXiv.2103.12607
21. Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks / [M. Rodler, W. Li, G. O. Karame et al.] // Network and Distributed System Security : Symposium 2019, San Diego, 24–27 February 2019 : proceedings. – Reston, VA: The Internet Society, 2023. DOI: 10.14722/ndss.2019.23413

© Tereshchenko O. I., Komleva N. O., 2024
DOI 10.15588/1607-3274-2024-3-15