UDC 004.9:004.56.57:025.4 **doi:** https://doi.org/10.20998/2522-9052.2023.4.05

Oleksandr Shmatko[1], Oleksii Kolomiitsev[1], Nataliia Rekova[2], Nina Kuchuk[1], Oleksandr Matvieiev[2]

[1] National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine
[2] Technical University "Metinvest polytechnics", LLC, Zaporizhzhia, Ukraine

DESIGNING AND EVALUATING DL-MODEL FOR VULNERABILITY DETECTION IN SMART CONTRACTS

Abstract. Task features. Smart-contracts are programs that are stored in a distributed registry and execute code written in them in response to transactions addressed to them. Such smart- contracts are written in the Solidity programming language, which has a specific structure and syntax. The language was developed for the Ethereum platform. Having a specific structure, such languages are prone to certain vulnerabilities, the use of which can lead to large financial losses. **Task statement.** In this paper, a Deep Learning (DL) model is used to detect the vulnerabilities. Using the chosen approach and a properly specified input data structure, it is possible to detect complex dependencies between various program variables that contain vulnerabilities and bugs. **Research results.** Using well-defined experiments, this approach was investigated to better understand the model and improve its performance. The developed model classified vulnerabilities at the string level, using the Solidity corpus of smart-contracts as input data. The application of the DL model allows vulnerabilities of varying complexity to be identified in smart-contracts. **Conclusions.** Thus, the pipeline developed by us can capture more internal code information than other models. Information from software tokens, although semantically incapable of capturing vulnerabilities, increases the accuracy of models. The interpretability of the model has been added through the use of the attention mechanism. Operator accounting has shown significant performance improvements.

Keywords: blockchain; smart-contract; computer system; secure; vulnerability; deep learning.

Introduction

A software vulnerability is “the existence of a design flaw, weakness, or implementation that could lead to an unwanted event that compromises the operation of a software application, computer system, network, or protocol.” [1].

Since the popularity of smart-contracts and the amount of software associated with it increases every day, then the number of attacks and vulnerabilities that affect the operation of this software increases in direct proportion.

Such software must be protected to prevent irreversible consequences both financially and for society as a whole. Examples of such bugs are vulnerability problems, including Decentralized Autonomous Organization (DAO) [2, 3]. The harm from triggering vulnerabilities is one of the reasons for their detailed and thorough investigation. One of the causes of vulnerabilities is the presence of complex interactions between pieces of software code, and the mismatch between what a program should do and what it actually does. Since it is impossible to guarantee that vulnerabilities are not present when the code is written, methods or models must be developed to detect these vulnerabilities.

Currently there are tools [4–9] for statistical analysis that recognize bugs and vulnerabilities. Their main drawback is that they only detect a limited number of errors based on predefined rules. The results obtained for the detected vulnerabilities require the presence of a specialist who will analyze them. This often leads to erroneous conclusions when assessing smart-contracts for vulnerabilities.

Also, despite a lot of existing literature, which describes the main vulnerabilities and methods for their detection, there are still classes of vulnerabilities and errors that are very difficult or impossible to recognize, and therefore they continue to exist.

The object of the study is smart contracts in healthcare [10, 11].

The subject of the study is models and tools for vulnerability assessment of smart contracts.

The goal of the study is to improve the security of smart-contracts by developing software for assessing vulnerabilities using DL models.

The approach developed in this paper is based on the use of a large amount of data in the form of open source codes that located in repositories, as well as a technique that uses a big amount of input data to detect more vulnerabilities. As the initial data, the corpus of programs is used to detect global patterns of error manifestation, which cannot be done by static checking.

The developed approach will help to identify vulnerabilities of varying complexity that arise in the operation of smart-contracts, which will make it possible to more accurately determine whether a smart-contract is safe and suitable for execution or not.

A DL model was developed, which, using binary classification, determines whether a smart contract contains vulnerabilities and errors. The corpus of smart contract programs developed in the Solidity programming language was used as input data. The corpus of smart-contract corpus was view as an Abstract Syntax Tree (AST). The proposed model was considered in detail in order to fully understand how it would react to different types of input data under certain experimental conditions.

Literature review and problem statement. Research on error detection and software fixes has recently been an important area of study and analysis [12, 13]. And now this issue is becoming more and more important with the emergence of new programming languages in some areas, such as cryptocurrency. To achieve accurate predictions in the problem of detecting vulnerabilities, two problematic factors must be solved: first, the representation of the source code used as input must reflect the internal structure of the program and the relationship between the variables in it. Second, the model must be designed to take full advantage of the proposed input data structure.

At first, thoughts came to mind about the best initial data for solving problems with the detection of vulnerabilities. Some researchers tried to use the source code from Github.

Another way to look at the problem is to focus on transactions made by smart contracts: in order to exploit vulnerabilities, hackers mainly focus on functions that create a mismatch between the actual transfer amount and the amount reflected in the internal stored data. With the development of DL, it has been proven that the best source material for such a problem is a collection of several codes containing some vulnerabilities. In [14], a method for detecting vulnerabilities was developed, which showed that learning from artificially created errors allows one to obtain error detectors that are effective in detecting errors in real code. Another solution is to use existing detection tools, as done in [15].

In [16], a method was proposed based on the representation of labels in order to be able to directly interpret the source code. In [17], DL is used to take advantage of the specific structure of the source code. The shortcuts were also generated using a static analyzer. With all this research in mind, our method was designed to take advantage of this data-driven approach by combining a corpus of code as input with DL. Most, if not all, of the previously described methods fail to fully construct relationships between variables, whereas this research paper focuses on creating well-interpreted methods that capture the interactions of variables. This ability is key to creating accurate and interpretable models. An AST view was used to successfully complete this task.

The article [18], on combining DL using such a view, confirms the feasibility of this approach, and in [19], the advantages of this view are highlighted using the structure of a “bag-of-paths” as input for training. In [20], the Long Short-Term Memory (LSTM) mechanism was used to investigate the internal structure of source codes to detect errors. He has demonstrated a strong ability to analyze functions to identify vulnerabilities. This article presents a DL model composed of LSTM-networks using an AST path as an input representation.

The main section

In process researching and analyzing existing vulnerabilities in smart-contracts, the most common types were identified.

Integer overflow [21, 22]: Label 1: Integer overflow vulnerabilities arise when a computed value is too large for a type that has been assigned a value. The operations that can cause overflow are the instructions "add", "subtract", "multiply" and "exponentiate". Thus, if this operation is used in a conditional statement, the program will have random behavior.

Unverified call return value [23, 24]: Label 2: An external contract could take over control flow due to an unverified call to the return value. The consequence of this problem is that an attacker can cause the call to fail, causing unexpected behavior in subsequent program logic that could be exploited by the adversary. Since this can lead to unwanted interactions between different function calls, execution resumes even if the called contract throws an exception.

Exception state (invariant assertion) [25, 26]: Label 3: The code flow must never reach an erroneous assertion means it is not working as expected. In the event of this problem, a statement invariant statement is not followed, which means that there is an error in the contract or that the statement is being used incorrectly.

Algorithm of actions for the development of the proposed approach for detecting vulnerabilities in smart contracts:

- A corpus of Solidity smart-contracts was created and used as the source code.
- A code corpus view was presented as an AST for understanding the complex dependencies between variables in programs. This input structure uses an abstract syntax tree view to model a combination of control and data transformation paths.
- To evaluate the success of our approach and to understand our models, tests were conducted using various types of inputs to understand the operation and behavior of our models.
- DL model was implemented to detect patterns that cause vulnerabilities and to take full advantage of the code corpus representation.
- A thorough study of the models we developed was carried out to understand their behavior under various experimental conditions. Models with different architectures were tested in combination with input data of different informativeness. The result was an understanding of the causal relationship between the accuracy of our models.
- The use of the attention mechanism in the DL model, which is a kind of vector of importance, was also necessary in order to be able to add interpretability to prediction. This process allows us to understand the pattern of the code responsible for creating the vulnerability.

Fig. 1 illustrates architecture of the proposed model for assessing vulnerabilities, taking the Solidity corpus of contracts as input, were implemented for a successful binary classification of the problem of detecting vulnerabilities at a linear level. This means that one label L_{ij} is displayed for each line. As described earlier, the AST path representation is used to create one path P_{ijkl} for each token T_{ijk} belonging to each line L_{ij} for each contract D_i. This action is represented by steps 1 and 2.

The reason for using AST paths is to reflect the interaction between tokens within the same line and between tokens in previous lines, in order to emulate the actual flow of program execution. However, our approach should be designed to take full advantage of this input. For this, the key point that makes our model special is how the information from the previous row is used to classify the current row. During the construction of the described representation of the code, several pieces of information are stored in each line. More precisely, what are called endpoints stores indices that point to past rows. These indexes are used to relate each token to its previous use, which aims to add context to each line.

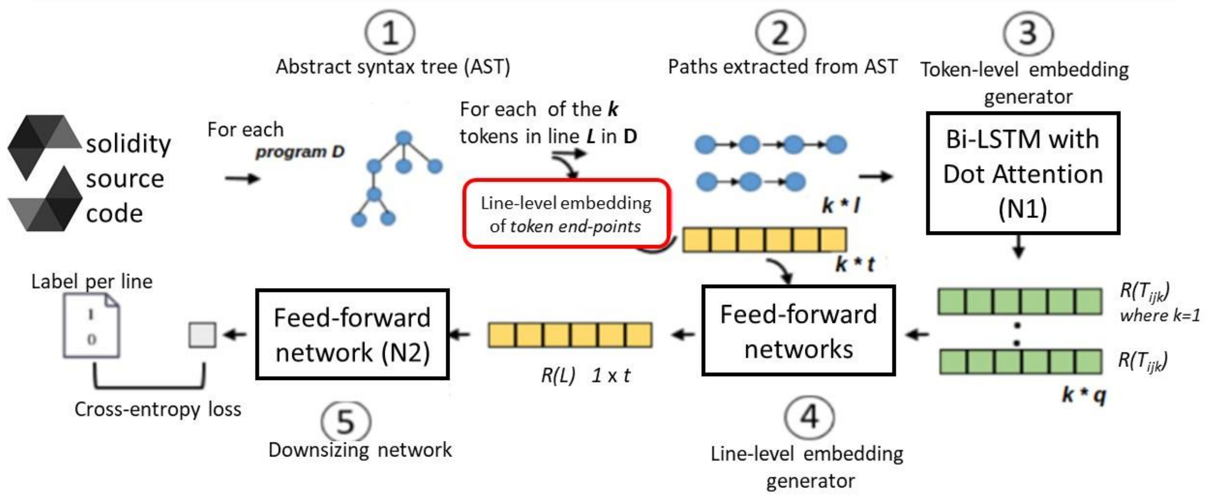
1
Abstract syntax tree (AST)
2
Paths extracted from AST
3
Token-level embedding generator
solidity source code
For each program D
For each of the k tokens in line L in D
Line-level embedding of token end-points
k * l
k * t
Bi-LSTM with Dot Attention (N1)
R(T_ijk) where k=1
R(T_ijk)
k * q
Label per line
Feed-forward network (N2)
R(L) 1 x t
Feed-forward networks
Cross-entropy loss
5
Downsizing network
4
Line-level embedding generator

Fig. 1. Architecture of the proposed model for assessing vulnerabilities of smart contracts

Then, in step 3, the view created after step 2 is used as an input to the LSTM- network in conjunction with the layer point attention. The consistent properties of the text were taken into account through the use of the LSTM-network, as proven in [27]. In the model we have developed, the attention layer invented in [29] plays a key role during the learning process, and is also the building block of the interpretability of our algorithm. The purpose of this layer of attention is to mimic the human attention mechanism. This step makes our model more capable of learning the context and relationships between tokens within the code.

A specific network using this concept of attention, defined as N1, used in step 3, creates a token-level embedding vector that is combined with endpoint information to feed into a simple feed forward network. The token-level attachments created in step 3 are a distributed representation of the q dimension and contain information corresponding to the Control Data Paths (CDP). The representations of each token $R(T_{ijk})$ forming a particular line of L_{ij} are pooled together as the researchers did in [28]. This concatenation is then used to create a linear representation of $R(T_{ijk})$ using the feedforward for the network defined in step 4.

This last row-level embedding is therefore used in another feedforward network to reduce the size of the vector to finally get one at a time binary label per line. The N2 network learns the relationship between the line and the assigned mark. Statistical analysis of attention weights evokes causal insight into the patterns implicit in vulnerabilities. So our models basically need two sources of information: AST paths at the token level and stored endpoint indices. They create two sets of inclusions in models: token-level inclusions and row-level inclusions. To create the latter, interspersed line level endpoints are needed, which are simply already created interspersed line level impregnations corresponding to past lines of previous use of tokens that form the predicted line. In terms of comparison with Natural Language Processing (NLP) methods, our algorithm can be compared to generating embeddings for each sentence and, ultimately, each paragraph, since the NLP input consists of several tokens per line, while a paragraph is a set of sentences. In this case, paragraph-level inlining is used to display line-level vulnerabilities.

The model based on the previous lines compares only the current and the previous line, not taking into account the lines several orders of magnitude higher.

Previous Line (PL) model is going through each line of this program sequentially, from the first line of the code to the last one. For each line, a collection of different CDPs with a defined length corresponding to the tokens forming the considered line is used as input. The collection corresponds to the 2D matrix. This representation is used in a bi-directional LSTM-network with local, multiplicative attention. Thus, this first network learns the different paths that can be associated with tokens in the scope of a program. The output of this first step is forming a line-level embedding of dimension (1, 100). This created embedding is stored in a look-up structure to be used if needed.

At that point, the model asks himself if the current computed line has a link with the previous one. A link exists to the previous line if at least one of the tokens used in the current line was used in the previous line. To know that, information was already processed during the path creation and stored in an array corresponding to the end-points data.

The model based on the endpoints (EP) is designed in such a way that it defines relationships between the tokens that are located in the lines several orders of magnitude higher.

The only difference between the PL model and the EP model is that the end-points information is not only linking tokens to the previous line but also to all the previous lines having dependencies in the entire code.

Metrics for assessment results of experiment:

False_Positive_Rate corresponds to the probability of falsely rejecting the null hypothesis for a particular test. *FP* is False Positive and *TN* is True negative. The ideal *FPR* is 0:

$$FPR = \frac{FP}{FP + TN}. \quad (1)$$

False_Negative_Rate corresponds to the proportion of people with a known positive condition for whom the test result is negative.

TP is True Positive and *FN* is False-negative. The ideal *FNR* is 0:

$$FNR = \frac{FN}{TP + FN} . \quad (2)$$

True_Positive_Rate or Recall calculates the ability of a model to find all the relevant cases within a dataset. The ideal *TPR* is 1:

$$TPR = \frac{TP}{TP + FN} . \quad (3)$$

Precision calculates how precise/accurate our model is out of those who are actually positive. Ideal P is 1:

$$P = \frac{TP}{TP + FP} . \quad (4)$$

F1 Score is needed when you want to seek a balance between precision and recall scores and when there is an unbalanced class distribution. The ideal F1 score is 1:

$$F1 = \frac{2 * P * TPR}{TPR + P} . \quad (5)$$

Average Precision = Area Under the Curve (AUC) of the Precision-Recall curve. The ideal score is 1.

ROC score = AUC of ROC Curves. The ideal score is 1.

Research questions that need to be answered in order to understand whether the developed approach is effective for detecting vulnerabilities in smart contracts.

Research results

RQ 1: Can a deep learning model work for the problem of detecting vulnerabilities?

RQ 1.1: How does the proposed method behave with the Solidity corpus of contracts as an input source?

Experimental hypothesis. The DL model does train if the learning loss decreases with epochs, if the test loss follows a similar evolution, and if the metric curves tend to ideal values before the plateau.

Experiment setup. The dataset used contains a sample of negative ones, which is a consequence of the extremely unbalanced distribution of labels. The goal of our model is to perform a row-level binary classification to predict the presence or absence of a vulnerability.

The input is a code representation was developed, where the main building blocks are the control data path for each token. These inputs were randomly split into 3 sets: a test set corresponding to 30% of that data set. The remaining 70% were further divided into 70%-30% of the training and validation sets: thus, the training set corresponds to 49%, and the validation set to 21%. the entire dataset. The loss used during training is the weighted cross-entropy loss, chosen for its ability to cope with unbalanced inputs. Our models were implemented using PyTorch version 1.0.

Results. As expected for a working DL model, learning loss decreases, and validation loss behaves similarly, but is noisier. By the definition of our validation set, this experimental observation makes sense. In addition, overfit is not observed. Thus, the first figure shows the expected shape of the learning curve.

The last two graphs in Fig. 2 show the evolution of the various metrics used to evaluate the performance of our model over an epoch.

Again, thanks to these visualizations, we can observe good learning behavior of our model. The rates rise rapidly during the first epoch and continue to rise at a slower rate in subsequent eras.

This means that our model is trained to discriminate between lines with and without vulnerabilities. The fourth graph shows other metrics that also have the expected behavior.

The FNR remains extremely low as our dataset is imbalanced. This specific metric illustrates the possible problems that you may encounter in work performed on a non-uniform dataset.

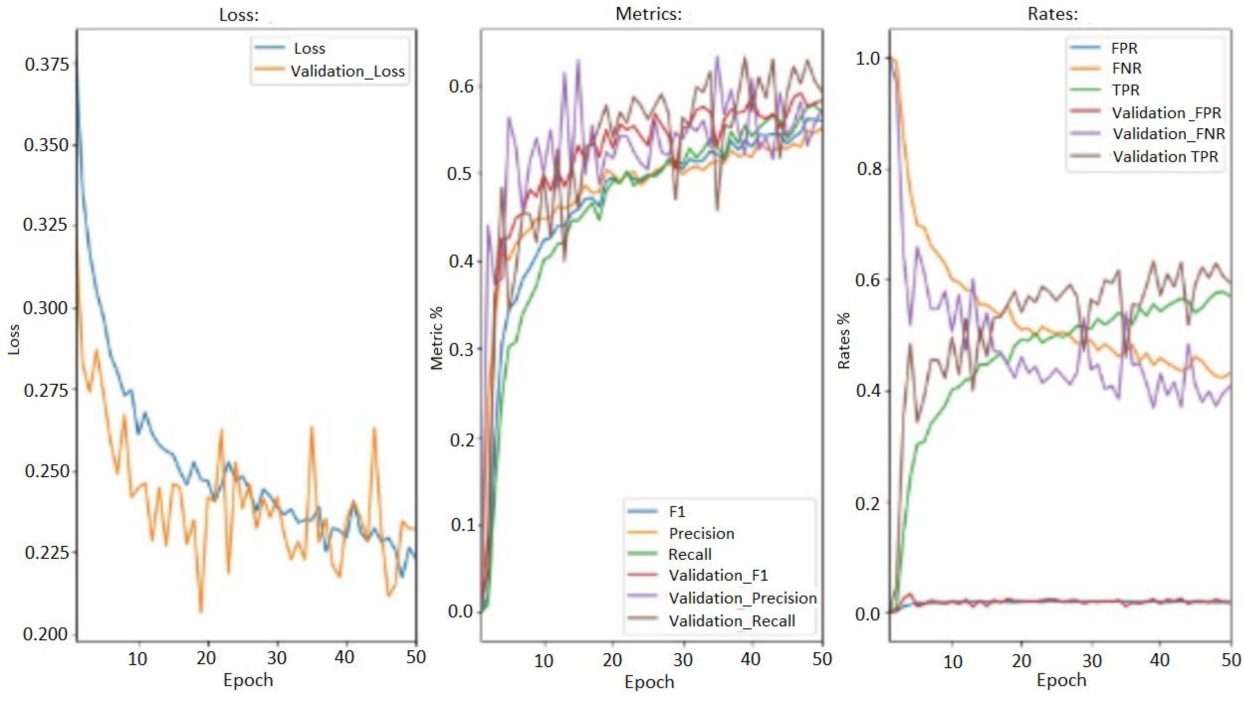

Fig. 2. Learning curves and estimates at the training stage of the EP model for the training and test set

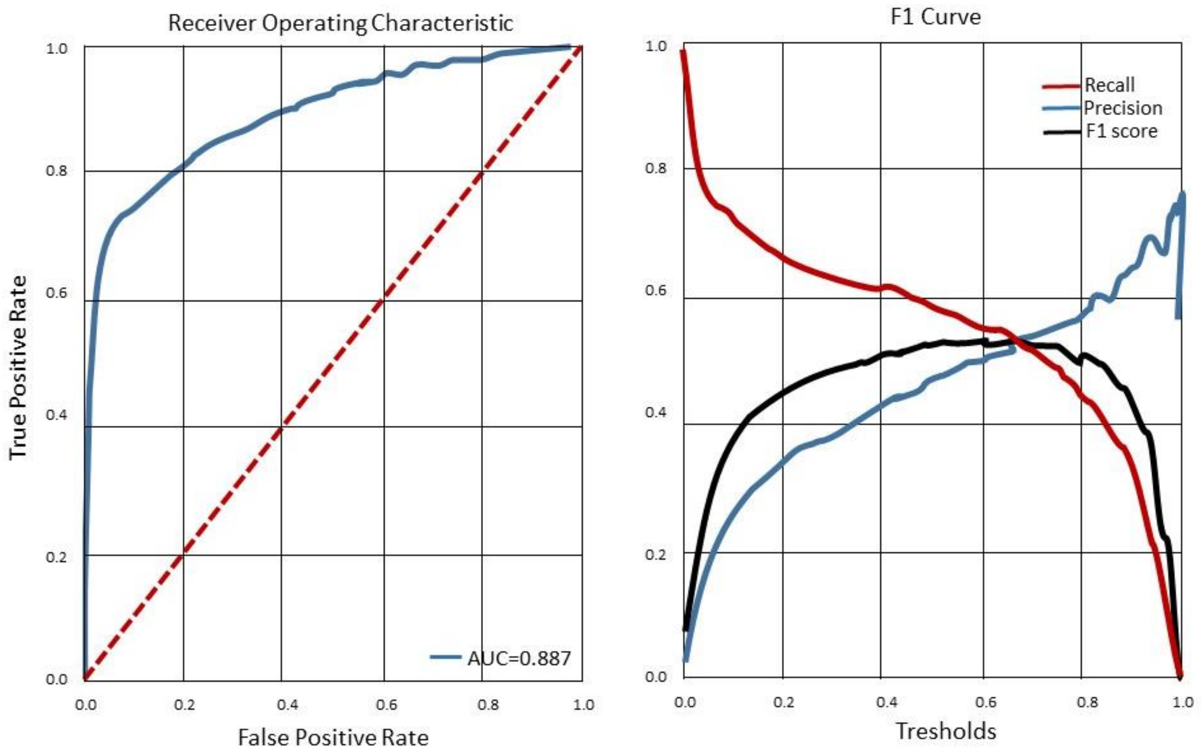

Fig. 3. ROC-curve of the EP model and Scores curves determined depending on the classification threshold

In conclusion, the graphs presented in Fig. 2 and 3 illustrate the skillful behavior of our model. These curves prove that our model is able to understand the input data and is able to learn statistical patterns in order to distinguish lines with vulnerabilities from lines without problems those that don't have any problems.

Fig. 3 is mainly due to the fact that our model predicts the probabilities for each class and then uses them in the softmax function to turn them into binary labels with a default threshold of 0.5. However, probabilities can be interpreted using different thresholds. Changing this parameter can change performance. The first graph in Fig. 3 shows TPR versus FPR. This Receiver Operation Characteristics (ROC) curve illustrates the trade-off between both metrics for a predictive model using different likelihood thresholds. The second graph shows the F1, precision and recall metrics versus threshold.

Finally, the graphs presented in Fig. 2 and 3 illustrate the skillful behavior of our model. These curves prove that our model is able to understand the input data and is able to learn statistical patterns to distinguish lines with vulnerabilities from lines that have no problems.

RQ 1.2: How does it scale if more information is added to the input dataset?

Experimental hypothesis. If the proposed DL algorithm works well, then the performance of the model on a richer dataset in terms of the amount of information should achieve more accurate predictions than on the standard input source.

This will confirm that the proposed approach is able to learn the context from the code.

Experiment setup: A new set of inputs is created. This input set qualifies as extended because more information is added to it.

Operators also act as tokens. In this new configuration, the operators simply become paths of length one, where the string that makes up the path is just the operator itself.

Padding is then used for formatting, since all CDPs require constant length. Our EP model is trained using a new extended data source to analyze the impact of this addition of information. To understand the changes caused by the addition of operators as tokens, a statistical survey of the CDP is being conducted.

Results. Tabl. 1 shows the impact of treating operators as tokens in terms of the number of tokens per line of code.

In fact, thanks to this change, the average number of tokens per row increased by 36%.

From the table shown in Tabl. 2, it can be seen that the amount of information presented in the data remains low, but more than for the default input data without operators. This also means that as more information is added, the difference in performance should also increase. Thus, one of the easy ways to improve the performance of the model is to account for more tokens.

Tabl. 3 shows the performance of our models tested on default and extended datasets.

Table 1 – **Collected statistics on the number of tokens per line in raw input and extended dataset**

	Mean	**Median**	**Std**
Dataset considering only variables as tokens	2.1	2.0	1.2
Dataset considering variables and operators as tokens	2.86	2.0	1.91

Table 2 – **The collection statistics of CDP aggregates corresponding to 4 tokens in each row are implied in negative and/or positive labels for the extended dataset, considering operators as tokens**

Vulnerability	Number of Aggr. of 4 CDP Paths	Number of Unique Aggregations of 4 CDP Paths
Lable 0	80944	12232
Lable 1	1614	420
Lable 2	441	257
Lable 3	1228	186
Intersection of paths aggregation implied in **Positive** Labels	316	(over 186 maximum possible paths); **~0.8% of the union of positive data are common between the 3 vulnerabilities type**
Union of paths aggregation implied in **Positive** Labels	3283	819; (over 863 maximum possible paths)
Intersection of paths aggregation implied in **Positive** and **Negative** Labels	2512	324; (over 819 maximum possible paths); **~39% of the data implied in positive labels are also implied in negative labels**

Table 3 – **Comparison of the results obtained using the EP model on the raw input and on the augmented dataset considering operators as a tokens**

Model Type	Max Var	F1	Std on F1 Score	Precision	Recall
Endpoints with Operators	4	0.53	1.8%	0.54	0.50
Endpoints to compare	4	0.47	1.8%	0.46	0.48
Endpoints with Operators	8	0.59	1.6%	0.59	0.60
Endpoints to compare	8	0.53	1.5%	0.51	0.55

RQ 1.3: Is the information added in the middle of the model using the endpoint data (corresponding to the previous lines) useful?

Experimental hypothesis. If information remains only at the endpoints of the corresponding rows, and if the rest of the information contained in the data is destroyed, the proposed DL approach should be able to classify a set of vulnerabilities proportional to the amount of information provided by the end user. indicates lines. According to our approach, the best results should be obtained using all the information from the previous lines.

Experiment setup. To answer this question, a synthetic dataset was created. The idea behind this new input source is to keep the same structure as our raw data, but change the amount of information encoded internally. To construct it, the raw input is modified during the formation of our view.

The method used is simple: the actual raw data is copied, the rows are fetched according to their labels, and the transformation is applied at the CDP scale, that is, at the token level. The endpoint indices remain unchanged so that they can be used to convey the same usual amount of information. The main purpose of this manipulation is to put information in one specific category and, accordingly, to study the behavior of our models.

An analysis of the behavior of the proposed method on the input, where the information remains only inside the endpoints, was performed by designing a special experiment. All paths to all inputs were randomized. This manipulation leads to the destruction of all information contained in the data. Iterations were then performed for each line of the program, and a constant pattern was introduced into the endpoints associated with the positively labeled current line, according to line labels (only positively labeled lines were converted), representing some kind of constant list of paths. This action allows the information to remain only on the paths that correspond to the end lines. Note that this process generates the current line only in random paths. In addition, the developed experiment distinguished between previous and subsequent lines.

The previous line represents short dependencies (using a token twice in two adjacent lines) between tokens, while farther lines represent long-term dependencies (calling a variable into another function) (Tabl. 4).

Table 4 – **Experimental Procedure used to build the different synthetic datasets**

Settings	Previous Line	Father Away Lines
Previous Line Experiment	Pattern1	Random
Both Method Experiment	Pattern1	Pattern1
Farther Away Lines Experiment	Random	Pattern1
Noise Experiment	Random	Random

Results. Before analyzing the performance of the model, a study of the distribution of endpoints is carried out to understand the amount of information they bring. Tabl.5 shows the relationship of the paths implicit in the various end-cases.

This analysis is critical to understanding the results of the various simulations. The line consists of 4 markers, and each of them refers to one endpoint index. This link is called endpoint information.

If the index matches the previous line, the corresponding path leads to the of the previous line category. If the index matches a previous line that is not the previous line, it jumps to the case of subsequent lines. If index is 0, it goes to zero register.

In Tabl. 5, you can observe the unbalanced property of the dataset. In addition, the empty proportion is about 60% within the positive set, which means that the amount of information encoded into the endpoints is small. The last interesting fact is that the proportion of cases of the previous lines is twice as large as the cases of the far line.

Table 5 – **Distribution of the end-points paths, which means at a token level, on the raw dataset**

Subset of Data	Specific Case of Endpoints information	Count	% inside the positive set	% in the entire set
Positive	Farther lines cases	4506	12.41%	0.46%
Positive	Previous lines cases	10386	28.61%	1.06%
Positive	Null cases	21412	58.98%	2.19%
Positive	Number of total positive Endpoints cases	36304	100.00%	3.71%
Negative	Total number of Negative Endpoints cases	941276		96.29%
Tot	Total number of Endpoints	977580		100.00%

Table 6 – **Experimental results coming from EP model trained on the different synthetic datasets**

Experiment	F1	Std on F1 Score	Preci-sion	Recall
Previous Line Experiment	0.23	1.2%	0.35	0.17
Both method Experiment	0.28	2.1%	0.33	0.21
Farther Away Lines Experiment	0.25	1.7%	0.37	0.19
Noise experiment	0	0%	0	0

Tabl. 6 shows the results of various experiments. The first observation is that the estimate obtained in the experiment on the far line is higher than in the experiment on the previous line. This means that the endpoints of farther lines contain more information than information about the previous line, even if the set of indices corresponding to the farthest lines is half the size. Long-term dependencies form easier patterns to recognize. In addition, based on the structure of our model, it is assumed that the EP model captures more contextual information and thus should overcome the characteristics PL modeled using Experiment with previous line. As shown in Tabl. 6, both methods experiment gets better results than the far line experiment and the previous line experiment. This observation is consistent with the expected behavior described in the experimental hypothesis. There is little difference between estimates obtained from both sources compared to estimates obtained with only one piece of endpoint information. This fact can be explained by overlapping information, which causes a lot of similarity within the main body of context brought by both sets. It can be concluded that the EP model works better with paternal line information, and because it is even better when the entire set of endpoints is specified, the information provided by the endpoints is definitely useful and allows for a better understanding of the code.

One way to improve the performance of our method would be to increase the proportion of endpoint information encoded in the input.

RQ 1.4: Does the model make interpretable results?

Experimental hypothesis. An interpretable model is a model that can explain its predictions. Therefore, in our case, if the model is interpretable, it should be able to identify some of the causes of the predicted vulnerabilities.

Experiment setup. In this experiment, attention weights were collected that correspond to the weights created by the attention layer within the model. To be precise, for each token within the 3 highest weights line, up to 3 nodes of the AST path were selected. Knowing that the maximum number of tokens in a row is 4, this means that for each row a number from 1 to 12 weights have been selected and tied to the node they represent using a dictionary.

In addition, the entire set of coefficients has been saved in another dictionary.

Thanks to this, the normalization of the weights was achieved. Each attention weight obtained during the testing phase of our model was divided by its total in the entire dataset. In fact, all quantities used by to obtain results are of relative importance. This method allowed for more importance to be given to tokens, which are less common in each type of vulnerability, while tokens that are always present are subject to more penalties.

This means that if a token is often present in all data, it will be less important than a token present only 10 times. In this process, more importance is attached to rare tokens. able to create vulnerability only because of their presence. Another selection criterion was established by examining the distribution of the entire set of weights: the minimum value that should have been achieved by the weights to be selected. This threshold was set at 0,075.

The purpose of this manipulation was to filter out meaningless weights. Then a classical statistical analysis was carried out on the collected collections.

Results. The histograms A, B, C represent the relative importance of the Top10 tokens implied for each vulnerability type (Fig. 4 and Fig. 5).

This means that from these graphs you can get a general idea of what tokens are causing and what types of vulnerabilities. Therefore, these 3 graphs summarize the causes of each vulnerability type, allowing the model to be interpreted. Indeed, some reasons can be identified: type 1 is mainly associated with expressions and root nodes, type 2 is mainly associated with trueBody nodes, and type 3 is mainly associated with statement nodes and parameters. In addition, a comparison was made of the relative importance of each token specified in the 0 label (which means no vulnerability). All tokens are mostly implied in label 0, which makes sense due to the unbalanced dataset.

This means that the causes found need to be detailed, as the markers identified by to create vulnerabilities are also building blocks of well-functioning code. This observation is logical because it corresponds to the intrinsic properties of the vulnerability discovery process and is the reason why this problem is difficult to solve.

In addition, investigation into the significance of certain tokens simultaneously for several types of vulnerabilities.

Histogram D illustrates this search, displaying a counter corresponding to the vulnerability type for each token (Fig. 5). This counter represents the number of times each token has been involved in the vulnerability. This means that the maximum rating that the token can have is 3, which means that this token is one of the reasons for three different types of vulnerabilities studied at the same time. If the counter is 2, it means that the token is implied in two types of vulnerabilities, so it can be implied in types 1 and 2, or types 1 and 3, or types 2 and 3. From this search, the importance of the identifier nodes is underlined in three types simultaneously.

In addition, by performing the same analysis for only two types of vulnerabilities, it is possible to identify some common causes for both types: the more important common causes of types 1 and 2 are operators and event call nodes of types 1 and 3.

Operator nodes and types 2 and 3 are the rigthExpression argument and expression.

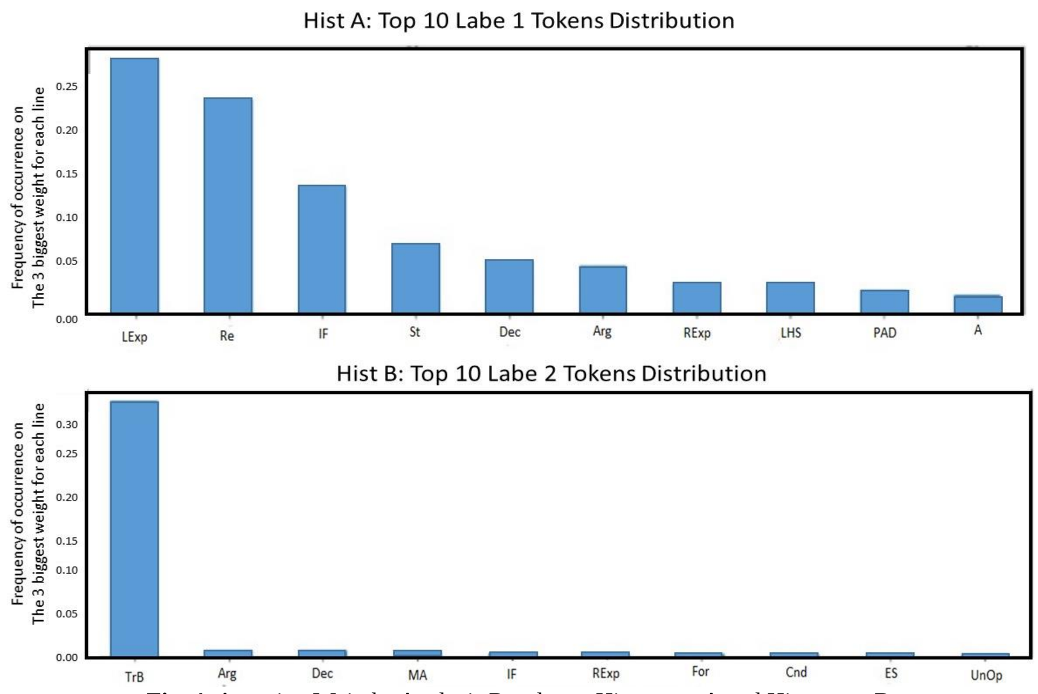

Fig. 4. Attention Weitghs Analysis Results on Histogram A and Histogram B

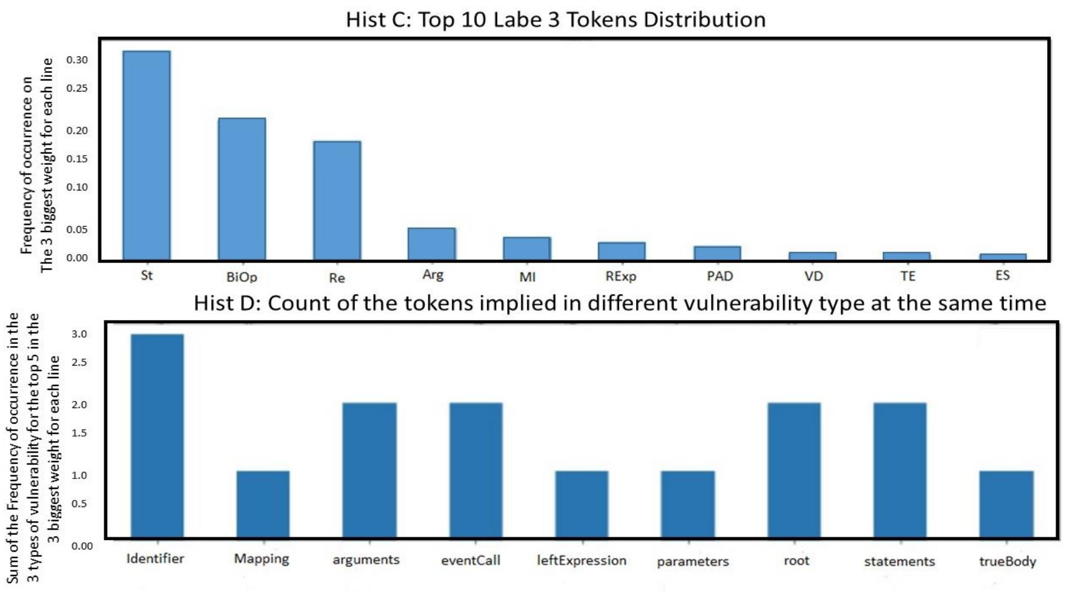

Fig. 5. Attention Weights Analysis Results on Histogram C and Histogram D

RQ 2: How can you improve the performance of your model?

RQ 2.1: How does the proposed program representation with an appropriate deep learning model affect the problem of detecting vulnerabilities? An equivalent question is to ask how does the EP model compare to baseline and similar vulnerability classifiers?

Experimental hypothesis: if the proposed method provides better performance than the baselines, the proposed program representation with the corresponding DL model has a positive effect on the vulnerability detection task.

Experiment setup. To answer this research question, the performance of our two models, PL and EP, is compared with different initial estimates. In fact, demonstrating that our developed models perform better than the baseline ones for the same task will prove the advantage of using our described method, created with an

AST based input representation mixed with endpoint information injected inside models. Several baselines have been tested on different types of inputs. Baselines studied and compared used to classify vulnerabilities were logistic regression model, random forest model, decision tree classifier, Gaussian naive Bayesian model. Results show only the best results for these baselines. The best performance is always achieved by a decision tree classifier. The ratings corresponding to each base model can be found in the appendix section. In conjunction with the baseline, two different input sources were used:

Bag of words is Path nodes (BOW node): The simplest input baseline under consideration was constructed using the set of tokens in each line, which means that a dictionary of all the unique nodes that make up the CDP was used. In this basic scenario, the input was only raw CDP information. There is no time frame to measure interest in time-related data [29].

Bag of Words is Paths (BOW-Path): Another input source is the same as BOW-Node one. However, instead of using a dictionary of all unique nodes, a dictionary containing all the unique paths of all strings appearing in the training set was used. This process measures the value of the CDP information in terms of its temporal structure. The BOW-Node is created at the node level and the BOW-Path is created at the path level. This can give us a general idea of the amount of information contained only in path objects.

A third baseline was also created to assess the skills of our model: Vulcan No end-points is Vul-NoEP: This model matches the pipeline we designed without using the information from the previous lines. Thus, it only matches the DL model, which consists of a bi-LSTM layer with an attention layer.

Results are shown in Tabl. 7.

Table 7 – **Scores comparison between implemented baselines on the BOW-Node input**

Baseline	F1	Preci-sion	Re-call	#0 labels	#1 labels
Logistic regression	0.17	0.10	0.71	54731	18588
Random forrest	0	0	0	73319	0
Decision Tree	0.41	0.56	0.32	71833	1486
Gaussian NB	0.07	0.03	0.98	3716	69603

The first conclusion to be drawn is that most baselines perform poorly with the BOW-Node input source (Tabl. 7), while one baseline stands out: the decision tree classifier. In fact, the F1 score of this model is quite high for the baseline. This means that a certain amount of information is contained in AST nodes even without temporary dependencies and is well understood by this basic classifier. However, this is not enough to develop a reliable tool. Tabl. 8 shows a comparison of the baseline estimates for each input source and the performance achieved with the EP and PL models. It can be concluded that the baseline, even if a richer set of words is used as input, is significantly superior to our EP and PL models. In addition, the Vul-NoEP score is, as expected, lower than the estimates for the PL and EP models. The positive value of our approach, taking into account the flow of information between different lines of code, as a consequence, is proven. Thus, the approach we developed, combining presentation with CDP and DL model, provides more information.

Table 8 – **Scores comparison between the implemented baselines on the different input sources and between the EP and PL models**

Model	F1	Std
Only Negative Prediction	0	0%
BOW-Node F1 Logistic Regression	0..17	0.8%
BOW-Node F1 Decision Tree	0.41	0.9%
BOW-Path F1 Logistic Regression	0.32	0.9%
BOW-Path F1 Decision Tree	0.43	1.1%
Vulcan-No Endpoints (Vul-NoEP)	0.47	1.3%
Previous Line Model (PL)	0.52	1.6%
Endpoints (EP)	0.53	1.8%

RQ 2.2: Is it useful to increase the complexity of the model with a default set of inputs? In other words, is the increase in complexity worth using the EP model, or should the PL model be used instead?

Experimental hypothesis: Is the increase in model complexity worthwhile? Thus, the performance of the EP model should be significantly higher than the estimates obtained with the PL model. If it is not, it means that the input data source is not is optimally structured to take advantage of the EP model.

Experiment setup: To answer this question, a CDP study is being conducted. We used the results of a statistical study conducted in RQ1.2 on the number of tokens in a row and the aggregation of unique paths on the input source, taking into account operators. In addition, the default simulations described in RQ1.1 are used to obtain the performance of EP and PL models. The run of each model was repeated 5 times so that the standard deviation of the characteristics could be calculated. Thanks to this pipeline, metrics and standard deviation F1 were obtained for both models. In this way, a reliable comparison becomes possible and the answer to the research question can be described.

Results are shown in Tabl. 9.

Table 9 – **Comparison of estimates between PL model and EP model on simulations made with default settings in an extended dataset**

Model	F1	Std of F1 Score
PL Model	0.52	1.6%
EP Model	0.53	1.8%

Both models are equally accurate when used with our operator-aware input source. Thus, it can be concluded that the increase in complexity due to the use of the EP model instead of the PL model is not worth it.

Hence, these different observations showed that the same information creates negative and positive labels, making our data extremely difficult to discern, even for a smart model. The overlapping information explains the similar performance of the PL and EP models. In fact, the amount of information added by the EP model is small because of the inherent similar data contained in the dataset. In conclusion, even when an extended dataset that is richer than the default is used as input, it is useless to use the EP model and therefore add complexity over the PL model. In fact, the current dataset does not have sufficient discriminatory power. Thus, the described research method allows the user to know when the EP model can be used in optimal conditions. This is really useful for tradeoffs between complexity and precision. Finally, adding information to the dataset can easily improve the performance of the model. The EP model can be used with some Domain Specific Language (DSL) that satisfies the described requirements (large number of tokens per line and variety in the path). In this case, our proposed pipeline can work very well.

Conclusions

This work represents an important first step in detecting vulnerabilities in domain-specific languages and analyzing programs written in Solidity. The presented method of semantically rich functions captures complex control and data dependencies and successfully classifies 3 types of vulnerabilities. The presentation presented using AST paths combined with a model using endpoint information has been shown to outperform the baseline. Endpoint information was proven with an experimental design using synthetic data and raw data. This allows you to better understand the natural structure of your code. Thus, the pipeline developed by us can capture more internal code information than other models. Information from software tokens, although semantically incapable of capturing vulnerabilities, increases the accuracy of models. The interpretability of the model has been added through the use of the attention mechanism. Operator accounting has shown significant performance improvements. This observation leads to the fact that as you add even more information to the data (for example, taking into account the function name as tokens), the difference in performance should also increase. Another way to increase the amount of sensitive data within the input is to use the encoding once method. In fact, operators are just paths of length one, where the string that makes up the path is just the operator itself.

References

1. (2020), "Vulnerabilities and Exploits", *European Union Agency for Cybersecurity*, available at: https://www.enisa.europa.eu/topics/incident-response/glossary/vulnerabilities-and-exploits
2. Luu, L., Chu, D.-H., Olickel, H., Saxena, P. and Hobor A. (2016), "Making smart contracts smarter", *Proc. of the 2016 ACM SIGSAC Conf. on Computer and Comm. Security*, ACM, pp. 254–269, doi: https://doi.org/10.1145/2976749.2978309
3. Mozhaev, O., Kuchuk, H., Kuchuk, N., Mykhailo, M. and Lohvynenko, M. (2017), "Multiservice network security metric", *2nd International Conference on Advanced Information and Communication Technologies*, AICT 2017 – Proceedings, pp. 133–136, doi: https://doi.org/10.1109/AIACT.2017.8020083
4. (2017), "Mythril", *ConsenSys*, 2 available at: https://github.com/ConsenSys/mythril
5. Kalra, S., Goel, S., Dhawan, M. and Sharma, S. (2018), "Zeus: Analyzing safety of smart contracts", *Network and Distributed System Security Symposium*, doi: https://doi.org/10.14722/ndss.2018.23092
6. Raskin, L., Sukhomlyn, L., Sokolov, D. and Vlasenko, V. (2023), "Evaluation of system controlled parameters informational importance, taking into account the source data inaccuracy", *Advanced Information Systems*, Vol. 7, no. 1, pp. 29–35, doi: https://doi.org/10.20998/2522-9052.2023.1.05
7. Mackey, T.K., Kuo, T.T., Gummadi, B., Clauson, K.A., Church, G., Grishin, D., Obbad, K., Barkovich, R. and Palombini, M. (2019), "'Fit-for-purpose?'—Challenges and opportunities for applications of blockchain technology in the future of healthcare", *BMC Med.*, 17, Article number 68, doi: https://doi.org/10.1186/s12916-019-1296-7
8. Dun, B., Zakovorotnyi, O. and Kuchuk, N. (2023), "Generating currency exchange rate data based on Quant-Gan model", *Advanced Information Systems*, Vol. 7, no. 2, pp. 68–74, doi: https://doi.org/10.20998/2522-9052.2023.2.10
9. (2018), "Manticore", *Trailofbits*, available at: https://github.com/trailofbits/manticore
10. Adomavicius, G. and Tuzhilin A. (2005), "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 6, pp. 734–749, doi: https://doi.org/10.1109/TKDE.2005.99
11. Hlavcheva, D., Yaloveha, V., Podorozhniak, A. and Kuchuk, H. (2021), "Comparison of CNNs for Lung Biopsy Images Classification", *2021 IEEE 3rd Ukraine Conference on Electrical and Computer Engineering*, UKRCON 2021 – Proceedings, pp. 1–5, doi: https://doi.org/10.1109/UKRCON53503.2021.9575305
12. Burke, R. (2002), "Hybrid Recommender Systems: Survey and Experiments", User Modeling and User-Adapted Interaction, Vol. 12, 4 (2002), pp. 331–370, doi: https://doi.org/10.1023/A:1021240730564
13. Kovalenko, A., Kuchuk, H., Kuchuk, N. and Kostolny, J. (2021), "Horizontal scaling method for a hyperconverged network", *2021 International Conference on Information and Digital Technologies* (IDT), Zilina, Slovakia, doi: https://doi.org/10.1109/IDT52577.2021.9497534
14. Amatriain, X., Pujol, J.M., Tintarev, N. and Oliver, N. (2009), "Rate it again: Increasing recommendation accuracy by user re-rating", Proc. of the 3rd Conf. on Recom. Syst., ACM Press, NY, pp. 173–180, doi: https://doi.org/10.1145/1639714.1639744
15. Basilico, J. and Hofmann, T. (2004), "Unifying collaborative and content-based filtering", *Proceedings of the 21st International Conference on Machine Learning* (ICML'04). ACM Press, New York, NY, 9, doi: https://doi.org/10.1145/1015330.1015394
16. Kuchuk, N., Mozhaiev, O., Mozhaiev, M. and Kuchuk, H. (2017), "Method for calculating of R-learning traffic peakedness", *2017 4th International Scientific-Practical Conference Problems of Infocommunications Science and Technology*, PIC S and T 2017 – Proceedings, pp. 359–362, doi: https://doi.org/10.1109/INFOCOMMST.2017.8246416
17. Bostandjiev, S., O'Donovan, J. and Hollerer, T. (2012), "TasteWeights: A Visual Interactive Hybrid Recommender System", Proc. of the 6th ACM Conference on Recommender Systems (RecSys). 35–42, doi: https://doi.org/10.1145/2365952.2365964
18. Lin, W., Li, Y., Feng, S. and Wang, Y. (2014), "The optimization of weights in weighted hybrid recommendation algorithm", *Proc. of the 2014 IEEE/ACIS 13th Int. Conf. on Comp. and Inf. Sc.* (ICIS) pp 415-18, doi: https://doi.org/10.1109/ICIS.2014.6912169
19. Sarwar, B., Karypis, J., Konstan, J., and Riedl, R. (2001), "Item-based Collaborative Filtering Recommendation Algorithms", Proc. of the 10th International Conference on World Wide Web, pp. 285-95, doi: https://doi.org/10.1145/371920.372071
20. Wu, H.T. and Tsai, C.W. (2018), "Toward blockchains for health-care systems: Applying the bilinear pairing technology to ensure privacy protection and accuracy in DS", *IEEE Consum. Electron. Mag.* 7, 65–71, doi: https://doi.org/10.1109/MCE.2018.2816306

21. Khezr, S., Moniruzzaman, M., Yassine, A. and Benlamri, R. (2019), "Blockchain technology in healthcare: A comprehensive review and directions for future research", *Appl. Sci.* 2019, 9, 1736, doi: https://doi.org/10.3390/app9091736
22. Zakharchenko, A. and Stepanets, O. (2023), "Digital twin value in intelligent building development", *Advanced Information Systems*, Vol. 7, no. 2, pp. 75–86, doi: https://doi.org/10.20998/2522-9052.2023.2.11
23. Vora, J., Nayyar, A., Tanwar, S., Tyagi, S., Kumar, N., Obaidat, M.S. and Rodrigues, J.J. (2018), "BHEEM: A Blockchain-Based Framework for Securing Electronic Health Records", *Proceedings of the 2018 IEEE Globecom Workshops* (GC Wkshps), Abu Dhabi, UAE, 9–13 December 2018, doi: https://doi.org/10.1109/GLOCOMW.2018.8644088
24. Datsenko, S. and Kuchuk, H. (2023), "Biometric authentication utilizing convolutional neural networks", *Advanced Information Systems*, Vol. 7, no. 2, pp. 87–91, doi: https://doi.org/10.20998/2522-9052.2023.2.12
25. Zhang, P., Schmidt, D.C., White, J. and Lenz, G. (2018), "Blockchain Technology Use Cases in Healthcare", *Advances in Computers*, Elsevier: Amsterdam, Netherlands, Vol. 111, pp. 1–41, doi: https://doi.org/10.1016/bs.adcom.2018.03.006
26. Kovalenko, A. and Kuchuk, H. (2022), "Methods to Manage Data in Self-healing Systems", *Studies in Systems, Decision and Control*, Vol. 425, pp. 113–171, doi: https://doi.org/10.1007/978-3-030-96546-4_3
27. Kumar, T., Ramani, V., Ahmad, I., Braeken, A., Harjula, E. and Ylianttila, M. (2018), "Blockchain Utilization in Healthcare: Key Requirements and Challenges", *Proceedings of the 2018 IEEE 20th International Conference on e-Health Networking, Applications and Services* (Healthcom), Ostrava, Czech Republic, https://doi.org/10.1109/HealthCom.2018.8531136
28. Salnikov, D., Karaman, D. and Krylova, V. (2023), "Highly reconfigurable soft-cpu based peripheral modules design", *Advanced Information Systems*, Vol. 7, no. 2, pp. 92–97, doi: https://doi.org/10.20998/2522-9052.2023.2.13
29. Luu, L., Chu, D.-H., Olickel, H., Saxena, P. and Hobor, A. (2016), "Making smart contracts smarter", *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 254–269, doi: https://doi.org/10.1145/2976749.2978309

Received (Надійшла) 30.08.2023
Accepted for publication (Прийнята до друку) 15.11.2023

ABOUT THE AUTHORS / ВІДОМОСТІ ПРО АВТОРІВ

Шматко Олександр Віталійович – доктор філософії, доцент, доцент кафедри Програмної інженерії та інтелектуальних технологій управління, Національний технічний університет «Харківський політехнічний інститут», Харків, Україна;
Oleksandr Shmatko – Doctor of Philosophy, Associate Professor, Associate Professor of Software Engineering and Intelligent Management Technologies Department, National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine;
e-mail: oleksandr.shmatko@khpi.edu.ua; ORCID ID: https://orcid.org/0000-0002-2426-900X.

Коломійцев Олексій Володимирович –доктор технічних наук, професор, професор кафедри Національного технічного університету «Харківський політехнічний інститут», Харків, Україна;
Oleksii Kolomiitsev – Doctor of Technical Sciences, Professor, Professor of Computer Engineering and Programming Department, National Technical University «Kharkiv Polytechnic Institute», Kharkiv, Ukraine;
e-mail: alexus_k@ukr.net; ORCID ID: https://orcid.org/0000-0001-8228-8404.

Рекова Наталія Юріївна – доктор економічних наук, професор, професор кафедри Цифрових технологій та проєктно-аналітичних рішень, ТОВ Технічний університет «Метінвест політехніка», Запоріжжя, Україна;
Nataliia Rekova – Doctor of Economics, Professor, Professor Department of Analysis and Project Decisions Department, Technical University "Metinvest Polytechnics", LLC, Zaporizhzhia, Ukraine;
e-mail: natarekova@gmail.com; ORCID ID: https://orcid.org/0000-0002-5961-3616.

Кучук Ніна Георгіївна – доктор технічних наук, професор, професор кафедри обчислювальної техніки та програмування, Національний технічний університет "Харківський політехнічний інститут", Харків, Україна;
Nina Kuchuk – Doctor of Technical Sciences, Professor, Professor of Computer Engineering and Programming Department, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;
e-mail: nina_kuchuk@ukr.net; ORCID ID: http://orcid.org/0000-0002-0784-1465.

Матвєєв Олександр Миколайович – доктор філософії, доцент, доцент кафедри Цифрових технологій та проєктно-аналітичних рішень, ТОВ Технічний університет «Метінвест політехніка», Запоріжжя, Україна;
Oleksandr Matvieiev – Doctor of Philosophy, Associate Professor, Associate Professor Department of Analysis and Project Decisions Department, Technical University "Metinvest olytechnics", LLC, Zaporizhzhia, Ukraine;
e-mail: matwei1970@gmail.com; ORCID ID: https://orcid.org/0000-0001-5907-3771.

Проектування та оцінка DL-моделі для виявлення вразливості в смарт-контрактах

О. В. Шматко, О. В. Коломійцев, Н. Ю. Рекова, Н. Г. Кучук, О. М. Матвєєв

Анотація. Особливості завдання. Смарт-контракти — це програми, які зберігаються в розподіленому реєстрі та виконують написаний у них код на відповідь на адресовані їм транзакції. Такі смарт-контракти написані на мові програмування Solidity, яка має специфічну структуру та синтаксис. Мова розроблена для платформи Ethereum. Маючи специфічну структуру, такі мови схильні до певних уразливостей, використання яких може призвести до великих фінансових втрат. **Постановка завдання.** У цій статті для виявлення вразливостей використовується модель глибокого навчання (DL). Використовуючи обраний підхід і правильно задану структуру вхідних даних, можна виявити складні залежності між різними програмними змінними, які містять вразливості та помилки. **Результати дослідження.** Використовуючи чітко визначені експерименти, цей підхід було досліджено, щоб краще зрозуміти модель і покращити її продуктивність. Розроблена модель класифікувала вразливості на рівні рядків, використовуючи як вхідні дані корпус смарт-контрактів Solidity. Застосування моделі DL дозволяє виявляти в смарт-контрактах уразливості різної складності. **Висновки.** Таким чином, розроблений підхід може фіксувати більше інформації про внутрішній код, ніж інші моделі. Інформація з програмних токенів, хоча семантично нездатна зафіксувати вразливі місця, підвищує точність моделей. Інтерпретативність моделі додана за рахунок використання механізму уваги.

Ключові слова: блокчейн; смарт-контракт; комп'ютерна система; безпека; вразливість; глибоке навчання.