UDC 004.49 **doi:** https://doi.org/10.20998/2522-9052.2022.1.16

Zhang Liqiang[1], Nataliia Miroshnichenko[2]

[1] Neijiang Normal University, Neijiang, China
[2] National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine

THE SOFTWARE SECURITY DECISION SUPPORT METHOD DEVELOPMENT

Abstract. The actuality of the power to improve the accuracy of the results was determined in order to make a decision about the process of testing the software security. An analysis of the methods of support for making a decision was carried out. The necessity and feasibility of improving the accuracy of the results was determined in case of further software security inconsistencies in the minds of the fuzziness of input and intermediate data. With this method, on the basis of the mathematical apparatus of fuzzy logic, the method of support for making a decision about the security of software security has been developed. The main feature of this method is the synthesis of an improved method of generating the initial vibration in the process of starting a piece of neural string. Within the framework of the model, the next stages of follow-up are reached. For the mathematical formalization of the process of accepting the decision and designation of the input data, the model of forming the vector in the input data was developed. Depending on this model for shaping the input data, an anonymous sign of potential inconsistencies and undeclared possibilities of the PP is valid until the data of PVS-Studio Analysis Results. To improve the accuracy of the classification of data collected, the method of creating a piece of neural array has been improved, which is modified by the method of generating a sample, which is being developed. This generation method includes three equal generations: generation of the initial vibration, generation of the initial butt and generation of a specific value of the safety characteristic. This made it possible to increase the accuracy of classification and acceptance of the solution by 1.6 times for positive elements in the selection by 1.2 times for negative elements in the selection. To confirm the effectiveness of the development of the method of support for the decision on how to ensure software security, a ROC-analysis was carried out over the course of the above procedures. The results of the experiment confirmed the hypothesis about the efficiency of the divided method of support to make a decision about the security of PZ up to 1.2 times equal to the methods, which are based on the position of discriminant and cluster analysis.

Keywords: software security inconsistency; security testing; decision support; fuzzy logic; cyberthreat.

Problem statement and literature analysis

Studies of the scheme for software vulnerability research and evaluation of the results of mathematical modeling allow to conclude that the data analyzed in the system for confirming potential vulnerabilities is complex [1, 2]. In this case, the summary security indicator of the software the object under study F_{sec}, can be represented as a sequence of the following data:

– vector $F_{sec}=(f_1,...,f_m)$ of initial characteristics of the composition of potential vulnerabilities, and also a list of vulnerabilities and undeclared capabilities, for a complete, comprehensive assessment of the security of the objects under study;

– vector $Y_{spec}=(y_1,...,y_m)$ of individual indicators representing functions $Q=q(f_i;i)$, $i=l,...,m$, of the corresponding initial characteristics and evaluate the object under study using m different criteria;

– function $X(Y_{spec})$, that compares the vector of individual indicators $Y_{spec}=(y_1,...,y_m)$ with a summary assessment (summary indicator) $X=X(Y_{spec})$, characterizing the object under study in terms of compliance with the stated security requirements.

The assessment is based on the security vector, which combines many characteristics of the composition of potential vulnerabilities, and also a list of vulnerabilities and undeclared features that show the degree of compliance of the software with a certain security criterion and the encoded value of the compliance control level [3, 4]. Based on the features of the chosen mathematical apparatus, the solution of the verification problem must be reduced to solving the following subtasks:

1. Definition of the initial vector F_{sec}.
2. Calculation of the classification features of the vector Y_{spec}.
3. Choice of decision-making method X.

The developed method should ensure the formation of correct output signals in the entire space of the composition of potential vulnerabilities, and also a list of vulnerabilities and undeclared capabilities F_{sec}.

1. Scheme of a method to support decision making on software security

The software security decision support method can be represented as a combination of three stages: initial data preparation, intellectual processing, and decision making. The sequence of stages is shown schematically in Fig. 1.

At the stage of preparing the initial data, a vector of software security characteristics is formed, which includes the TOP 20 composition of potential vulnerabilities and also a list of vulnerabilities and undeclared capabilities.

At the stage of intellectual processing, the previously obtained vector of characteristics of potential vulnerabilities and also the list of vulnerabilities and undeclared capabilities is processed using an artificial neural network. The result of this processing is the classification vector Y_{spec}, on the basis of which the final decision is made on the compliance of the software with security requirements at the third stage.

2. Model for the formation of input data vectors

The initial data for the decision-making method is a set of characteristics of potential vulnerabilities $V1$ and also a list of vulnerabilities and undeclared capabilities $V2$:

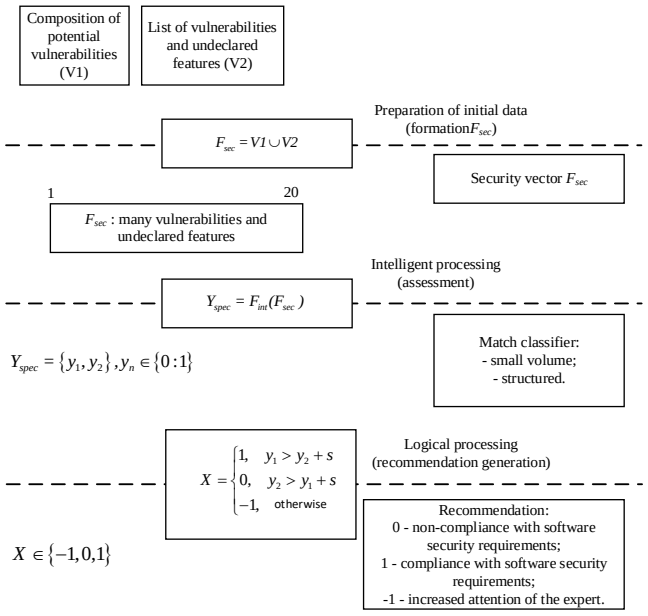


Fig. 1. Scheme of a method to support decision making on software security

$$F_{sec} = V1 \cup V2 . \quad (1)$$

The TOP 25 CWE lists provide data on the assessment of the potential danger of software vulnerabilities that change, and are statistically evaluated every year by MITER specialists [5]. These scores are distributed, accompanied by signs of potential vulnerabilities and undeclared features of programming languages distributed across classes.

At the same time, it is advisable to take into account the specifics of the apparatus of neural networks. In particular, to reduce the probability of misclassification, the requirement of linear separability of the input data must be met [6].

Vector F_{sec} contains a union of all signs of potential vulnerabilities and undeclared features, distributed across classes of programming languages. From the statistical data published in the articles by PVS-Studio Analysis Results [7], signs of potential vulnerabilities and undeclared features can be distinguished. Their list is presented in Table 1.

It should be noted that in Table 1, a limited set of features is presented. The full set of features is presented in the MITER reports. The number and essence of features can be updated over time. Presented in Table 1 data correspond to the statistics of 2021.

Table 1 – **List of signs of potential vulnerabilities and undeclared software features according to PVS-Studio Analysis Results**

№	ID	Description
1	V512	Potential error related to filling, copying or comparing memory buffers
2	V557	Potentially possible memory access outside the array
3	V582	Potential error when working with a fixed size container
4	V645	Potential error related to string concatenation
5	V3106, V6025	Index out of range
6	V5610	Potentially corrupted data that can be used to execute a malicious script
7	V739	EOF constant is compared to a variable of type 'char' or 'unsigned char'
8	V781	At the beginning, the value of the variable is used as the size or index of the array. And then this value is compared with 0 or with the size of the array.
9	V1010, V5009	Use of data obtained from outside without prior verification
10	V1024	Possible use of incorrect data when reading them
11	V5608	Creation of an SQL command from data received from an external source without prior validation
12	V623	Possible error when working with a ternary operator '?:'
13	V723	The function returns a pointer to the internal string buffer of the local object
14	V758	Detection of a link that may become invalid
15	V774	Using a pointer that points to a freed area of memory
16	V1017	Detecting the initialization of an instance of a class 'std::string_view' temporary object or assignment to an instance of a class 'std::string_view' a temporary object
17	V629	Detection of a potential error in an expression containing a shift operation
18	V683	Detecting a potential error in a loop
19	V1028	Detection of suspicious type casting. The result of a binary operation on 32-bit is converted to a 64-bit type
20	V5305	Detection in the code of data that may be confidential
21	V3125	Detection of a potential bug that could lead to null reference access

Thus, when conducting a static software analysis, as part of the certification procedure, 20 features can be used, and the result of the initial data preparation stage is a vector F_{sec} containing 20 elements

$$F_{\text{sec}} = \{v_1, v_2, \ldots v_n\}, \quad v_2 = \{1, 0\}, \quad (2)$$

where *n* - number of evaluated security features.

After the vector of security characteristics of the studied software is formed, it is necessary to evaluate its compliance with the stated requirements on the basis of this vector, i.e. determine which security class it belongs to.

The result of evaluating the security characteristics is the classifier Y_{spec}, that includes two elements (3):

– y_1 shows the degree of compliance with the stated requirements;

– y_2 shows the degree of non-compliance with the stated requirements.

$$Y_{spec} = \{y_1, y_2\}, y_n \in \{0:1\} . \quad (3)$$

Vector F_{sec} is an *j*-th prototype (i.e. separate implementation) *m*- dimensional random vector

$V1 \cup V2$, to be classified by a multilayer perceptron. Each *i*-th of the *K* possible classes of vulnerabilities to which the given input signal belongs, denote C_i Let $y_i^{(j)}$ – *i*-th output of the network generated in response to the prototype v_i:

$$y_i^{(j)} = F_i\left(v_j\right), \quad i = 1,2,\ldots K, \tag{4}$$

where $F_i(\ )$ – a function that determines the mapping that the neural network learns when passing the input example to the *i*-th input.

Therefore, the decision rule R, used to classify the inputs of the network, after its training should be based on the value of the vector function:

$$F : R^{(m)} v \to y \in R^K . \tag{5}$$

In general, about a vector function $F(\ )$ it can be fixed that this is a continuous function minimizing the empirical risk functional:

$$G = \frac{1}{2N} \sum_{j=1}^{N} \left\| d_j - F\left(v_j\right) \right\|^2 , \tag{6}$$

where d_j – expected output for the prototype v_j; $\| \ \|$ – Euclidean norm of a vector; *N* – total number of examples presented to the network for training.

As can be seen from 6, the physical meaning of the minimizing empirical risk functional is reduced to a cost function. Function vector $F(\ )$ strictly depends on the choice of examples (d_j, v_i), used to train the network. This means that different values of the pairs (d_j, v_i) will lead to the construction of various vector functions $F(\ )$.

The purpose of the decision stage is to form the final decision G based on the classifier obtained at the previous stage Y_{spec}.

Based on the main problem solved by the developed method for supporting decision-making on software security, the following options are possible:

- Software complies with security requirements;
- Software does not meet security requirements.

It should be noted that neglecting the opinion of an expert can lead to negative consequences in verification procedures. Therefore, in cases of a controversial decision, in order to increase the reliability of the results, it is necessary to provide for the possibility of expert intervention. Therefore, the final decision G can take one of three values:

- 1 – complies with safety requirements;
- 0 – does not comply with safety requirements;
- -1 – disputed decision.

To calculate the final solution G it is necessary to determine the position of the current solution in the space of possible solutions, relative to the reference positive and negative solutions.

$$\begin{aligned} G_{tr} &= \sqrt[2]{(1-y_1)^2 + (0-y_2)^2}; \\ G_{fl} &= \sqrt[2]{(0-y_1)^2 + (1-y_2)^2}, \end{aligned} \tag{7}$$

where y_1, y_2 – the values of the elements of the previously calculated compliance classifier for the analyzed software.

The evaluation of the quality of the obtained solutions is proposed to be calculated based on the calculation of the standard deviation (SD) for the sets of positive and negative solutions.

3. Development of a neural network architecture for solving the problem of supporting decision making on software security

After performing one of the important steps - extraction of security features, which is usually performed without a teacher, the second important step is the choice of a reasonably small number of features that concentrate the most significant information about the input (classified) data. Despite the fact that an artificial neural network can independently carry out classification, it is recommended to supplement it with a training scheme with a teacher to improve performance.

It is also recommended that the artificial neural network be able to scale as new security features or sets of security requirements are added. All vector elements F_{sec} must be normalized with respect to the range of possible values of the chosen neuron model. On Fig. 2 a model of an artificial neural network is presented that corresponds to the general scheme of the method for supporting decision making on software security.

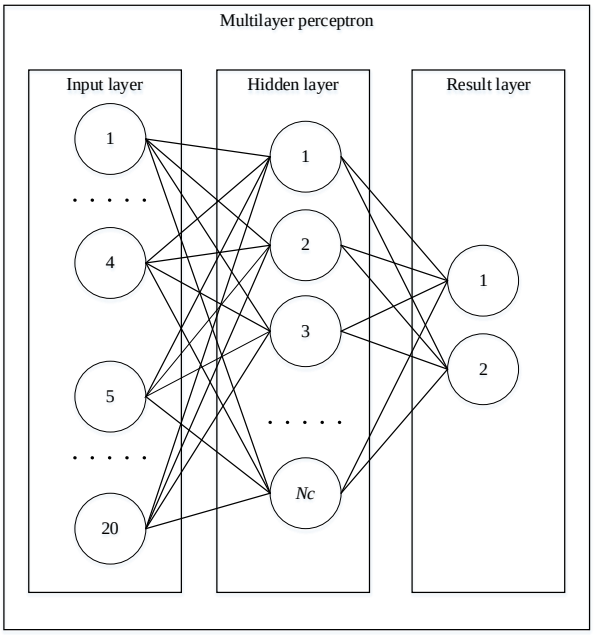


Fig. 2. Artificial neural network model

To solve the problem, one hidden layer of neurons is enough [8]. To build the final model of an artificial neural network, it is necessary to determine the number of neurons in the hidden layer, i.e. calculate its power.

To estimate the number of neurons in the hidden layers of homogeneous neural networks, you can use the formula for estimating the required number of synaptic weights L_w (in a multilayer network with sigmoidal transfer functions) [9]:

$$\frac{mN}{1+\log_2 N} \le L_w \le m\left(\frac{N}{m}+1\right)(m+n+1)+m, \quad (8)$$

where n – размерность входного вектора; m – dimension of the input vector; N – number of training sample elements.

Having estimated the required number of weights, we can calculate the number of neurons in the hidden layers. For a neural network with one hidden layer, the calculation is carried out according [10]^

$$L = \frac{L_w}{n+m}. \quad (9)$$

The dependence of the number of neurons in the hidden layers of the network on the number of weight connections is shown in Fig. 3.

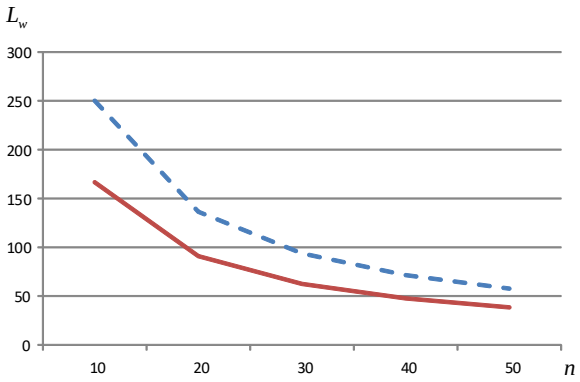


Fig. 3. Dependence of the number of neurons in the hidden layers of the network on the number of weight connections

It should be noted that the number of neurons in an artificial neural network should not be less than the number of classes, and since the exact number of classes may not be known in advance, the number of neurons is set with a certain margin. "Superfluous" neurons, whose weights will change chaotically during the learning process, can be removed at the end of this process.

4. Improvement of the artificial neural network training method

The conducted studies have shown that one of the most common and proven methods for training neural networks is the method of back propagation of an error. [11]. It is based on calculating the difference between the existing weight of the neural network and the necessary one to obtain the required result on a predetermined set of input signals, which is called training. The backpropagation method is aimed at minimizing the difference between the actual and expected network outputs by changing the weights of synapses.

The main requirement for applying the backpropagation method is the generation of such a set of pairs (v_p, y_p) input and output signals, training an artificial neural network on which will correctly solve the verification problem. At the same time, it is necessary that the artificial neural network has the properties of learning and generalization, and does not go in cycles around training examples.

In general, an artificial neural network training algorithm consists of the following steps:

1. The weights of the artificial neural network are assigned averaged initial values.
2. A training pair is selected (v_p, y_p) from the training set. Vector F_{sec} is fed to the input of the artificial neural network.
3. The result of the work of an artificial neural network is calculated.
4. The difference between the expected G and real network output.
5. Artificial neural network weights are adjusted to minimize the error.

The most important stage of training is the stage of forming a training data sample. The correctness of the formed training sample directly affects not only the efficiency of the neural network, but also its key features such as the ability to generalize and learnability.

To optimize the learning process, there are strategies such as determining support vectors and identifying principal components. However, existing approaches do not take into account the influence of the order of examples in the training sample on the final learning outcome [12].

It should also be noted the need to ensure a "dense" distribution of values in the training sample in the zone of the threshold value of the software safety characteristic to minimize the probability of making an incorrect decision in the conditions of a slight deviation of the analyzed characteristic from the safe value. Practical studies have shown that in order to achieve the required quality of training, it is sufficient that the size of the training set N satisfies the following relation:

$$N = O(W/\varepsilon), \quad (10)$$

where W – total number of free parameters (synaptic weights) of the network, ε – allowable classification error accuracy, $O(\ )$ – order of value enclosed in brackets. In accordance with the ratio (8): W_{min} = 37; W_{max} = 2758.

Based on relation (10), it is possible to calculate the dependence of the number of elements in the training sample on the required accuracy of work. On Fig. 4 a graph of the dependence of the number of elements in the training sample on the required accuracy of work is presented, under conditions: $O()$ = 10, $1\% \le \varepsilon \le 23\%$, W_{min} = 37, W_{midl}=*1000* W_{max} = 2758.

It should be noted that in order to reduce the number of training examples, it is necessary to choose the optimal values G and ε. As can be seen from the graphs in Fig. 4, for the above example, with $\varepsilon = 9\%$ results take the maximum value.

Based on the above, it follows that the developed teaching method should provide:

– the possibility of scaling;
– minimizing a sufficient number of training examples;
– maximum accuracy in solving the problem of software security classification.

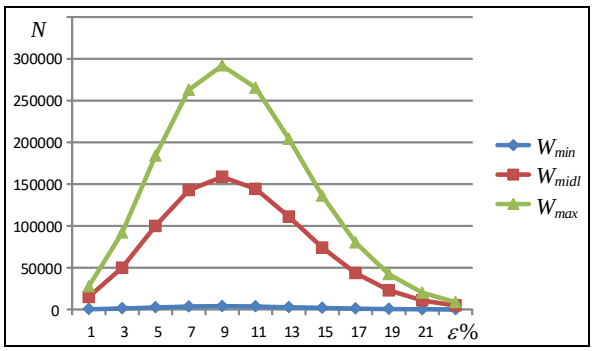


Fig. 4. Graphs of the dependence of the number of elements in the training sample on the required accuracy of work

The most important stage of the artificial neural network training method is the stage of generating a set of training examples. The key properties of the artificial neural network depend on the quality of this set. At the same time, it is necessary that the set of training examples correspond to the principles of emphasis and uniformity in the presentation of safety classes.

This generation method includes three levels of generation: generation of a training sample, generation of a training example, and generation of a specific value of a security characteristic. Lots of teaching examples M_x is a set of pairs V_i, Y_i.

$$M_x = \{(V_1, Y_1), \ldots, (V_i, Y_i)\}, \quad i = 1, 2, \ldots N, \quad (11)$$

where V_i – source data vector, a Y_i – known beforehand result of the work of an artificial neural network for V_i.

At the level of generating a set of examples, the general requirements for the training sample are taken into account. First of all, determining its scope, ensuring that all levels of control are equally represented, and also controlling the equal presentation of positive and negative examples. The generation of the value of a particular training example is determined by the function F(), whose arguments are: the maximum value of the characteristic, the number of the generated characteristic, the level of control, and a variable indicating the positiveness of the example.

The output of the example generation is a vector, the first four elements of which indicate the level of security control, the last element is the required result, and the remaining elements are the values of the software characteristics. The *Fgen*(*Si*,*Cnt*) function is responsible for generating the characteristic values. The function is formed on the basis of the binomial distribution law, which allows you to focus the attention of the artificial neural network on the region of the transition value of the characteristic:

$$F_{gen}(Vi)) = P(Y_{spec} \le Vi) = \\ = \sum_{k=0}^{\lfloor Vi \rfloor} C_n^k p^k q^{n-k}, \quad Vi \in \{0;1\}. \quad (12)$$

In addition, the feasibility of generating the correct value of the characteristic is determined using the accounting matrix - M, which signals the level of security control. If the value of the matrix element is 1, then the characteristic must be generated correctly, and the value 0 indicates that the value of this characteristic is not taken into account for this control level and this characteristic can be neglected in this example.

The matrix taking into account the characteristics is compiled in accordance with the requirements of regulatory documents. This example is based on the requirements of a non-profit organization MITRE Corporation. To test the effectiveness and feasibility of using the improved method for generating training examples, experimental studies were carried out using the principles of uniform distribution of training examples over the entire set of options, and also focusing on the improved method. The number of training examples in the sample was taken equal to 4000.

The result of the generation is a matrix of training examples *Mx* containing *N* training examples and *N* results. The matrix determines the values of the two resulting neurons for each training example. As a result, two sets containing 4000 training examples were obtained. The results of comparative studies of the improved generation method are presented in Table 2.

Table 2 – **Comparative studies of an improved method for generating a set of training examples**

	MxGen	**Gen_teach**
Number of elements	4000	4000
Positive		
Number of correct decisions	640	991
% correct decisions	64	99,1
Negative		
Number of correct decisions	3240	3987
% correct decisions	81,8	99,8

As can be seen from the values of the experimental results, the accuracy of classification and decision making increased by 1.6 times for positive elements in the sample and by 1.2 times for negative elements in the sample.

5. Investigation of the effectiveness of the decision support method for software security

To study the effectiveness of the proposed method for supporting decision making on software security, software was developed using the built-in libraries of the Phyton programming language [13]. This software made it possible to simulate the processes of functioning of an artificial neural network with training on relevant examples. A number of test values were generated. 1000 sets of security indicators with a predetermined result, as well as 500 - not corresponding. The values of this set were fed to the input of the trained artificial neural network, and the result obtained was compared with the known one. According to the results of the experiment, the results obtained were more than 96% consistent with the expected. The results of the experiment are presented in the form of graphs of ROC-curves of the distribution of the obtained results by solutions [14] in Fig. 5. To conduct a comparative analysis of the developed method, classifiers based on discriminant and cluster analysis are taken as reference solutions. Fig. 6 shows graphs of the ROC distribution curves of the results obtained using discriminant (Fig. 6, a) and cluster (Fig. 6, b) analysis.

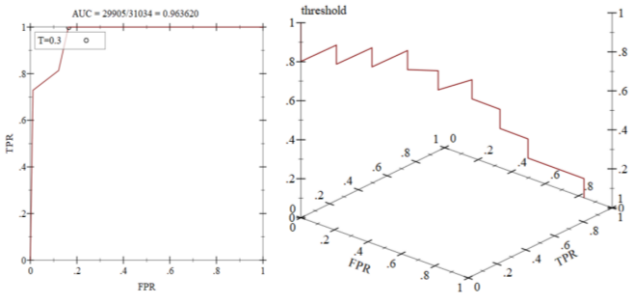
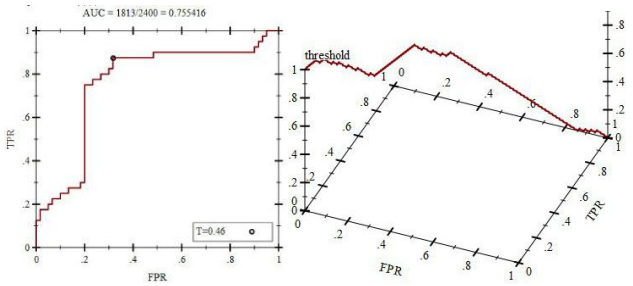
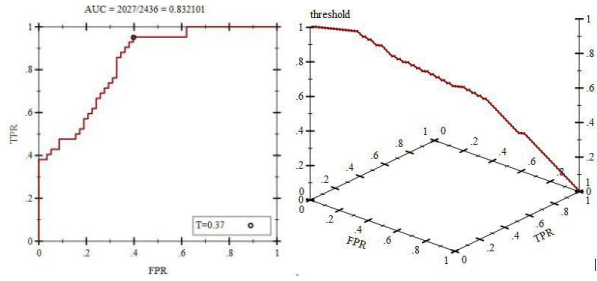


Fig. 5. Plots of ROC-curves for the distribution of the results obtained by solutions



a



b

Fig. 6. Graphs of ROC-curves of distribution of the obtained results using discriminant (a) and cluster (b) analysis

The analysis of the classification ROC curve based on the fuzzy cluster classifier (Fig. 6, b) was performed only for two classes, since the procedure for constructing an ROC curve for a larger number is difficult. The results of the analysis of the quality of the functioning of the fuzzy cluster classifier confirmed the hypothesis about the effectiveness of the developed method for supporting decision-making on software security up to 1.2 times.

Conclusions

1. A method has been developed to support decision-making on software security. A distinctive feature of the method is the synthesis of an improved method for generating a training sample in the process of training an artificial neural network. This made it possible to increase the efficiency of the software security decision support method up to 1.2 times.

2. In the course of the study, a model for the formation of input data vectors was developed. In accordance with this model, for the formation of input data, a set of signs of potential vulnerabilities and undeclared software capabilities is formed in accordance with the data PVS-Studio Analysis Results.

3. It was proposed to take a multilayer perceptron as a basis for the design of the neural network architecture for solving the problem of supporting decision-making about software security.

4. Artificial neural network training method that is different the way of generating the learning sample has been improved. This generation method included three levels of generation: generation of a training sample, generation of a training example, and generation of a specific value of a security characteristic. This made it possible to increase the accuracy of classification and decision making by 1.6 times for positive elements in the sample and by 1.2 times for negative elements in the sample.

5. Using ROC-analysis procedures, the effectiveness of the method for supporting decision-making on software security was carried out. The results of the experiment confirmed the hypothesis about the effectiveness of the developed method for supporting decision-making on software security up to 1.2 times compared to methods based on the provisions of discriminant and cluster analysis.

REFERENCES

1. Semenov, S., Zhang, L., Cao, W., Bulba, S. & Davydov, V. (2021), “Development of a fuzzy GERT-model for investigating common software vulnerabilities”, *EEJET*, Vol. 6(2, 114), pp. 6–18, DOI: https://doi.org/10.15587/1729-4061.2021.243715
2. Semenov, S., Liqiang, Z., Weiling, C. & Davydov, V. (2021), “Development a mathematical model for the software security testing first stage”, *EEJET*, Vol. 3(2, 111), pp. 24–34, DOI: https://doi.org/10.15587/1729-4061.2021.233417
3. (2021), *Introduction Welcome to the OWASP Top 10 – 2021*, available to: https://owasp.org/Top10/
4. (2015), Dan Sullivan Six criteria for procuring security analytics software, available to: https://www.techtarget.com/searchsecurity/feature/Six-criteria-for-procuring-security-analytics-software
5. (2022), CWE List Version 4.1, available to: https://cwe.mitre.org/data/
6. Schilling, A., Maier, A., Gerum, R., Metzner, C. & Krauss, P. (2021), “Quantifying the separability of data classes in neural networks”, *Neural Networks*, Vol. 139, pp. 278-293, DOI: https://doi.org/10.1016/j.neunet.2021.03.035.
7. Gelvih, M. (2021), *CWE Top 25 2021*, available to: https://pvs-studio.com/en/blog/posts/0869/
8. Zhao, Z., Xu, Sh., B. Ho, Kang, Kabir, M., Liu, Yu. & Wasinger, R. (2015),”Investigation and improvement of multi-layer perceptron neural networks for credit scoring”, *Expert Systems with Applications*, Vol. 42, Is. 7, pp. 3508-3516, DOI: https://doi.org/10.1016/j.eswa.2014.12.006
9. Zaki, P.W. (2019), “A Novel Sigmoid Function Approximation Suitable for Neural Networks on FPGA”, 2019 15th Int. Computer Engineering Conference (ICENCO), pp. 95-99, DOI: https://doi.org/10.1109/ICENCO48310.2019.9027479.
10. Kleyko, D., Rosato, A., Frady, E.P., Panella, M. & Friedrich, T. (2012), *Sommer Perceptron Theory for Predicting the Accuracy of Neural Networks*, available to: https://arxiv.org/pdf/2012.07881.pdf
11. Dzerzhinsky, R.I., Trifonov, M.D. & Ledovskaya, E.V. (2021), “The Support Vectors and Random Forest Methods Analysis in the Forecasting Customer Churn Problem in Banking Services”, *Data Science and Intelligent Systems*, CoMeSySo, Lecture Notes in Networks and Systems, vol 231, Springer, Cham, available to: https://doi.org/10.1007/978-3-030-90321-3_26

12. (2021), Introduction to Python Programming (Beginner's Guide), available to: https://www.analyticsvidhya.com/blog/2021/05/introduction-to-python-programming-beginners-guide/
13. Gavrylenko, S., Chelak, V., Semenov, S. & Chelak E. (2019), "Development of anomalous computer behavior detection method based on probabilistic", *Security in cervatury, the social internet space in context values and hazards*, Slupsk-Kharkiv, pp. 237-258.

Надійшла (received) 23.11.2021
Прийнята до друку (accepted for publication) 26.01.2022

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Ліцян Чжан – викладач коледжу комп'ютерних наук, Типовий університет Нейцзяна, Нейцзян, Китай;
Zhang Liqiang – teacher, College of Computer Science, Neijiang Normal University, Neijiang, China.
e-mail: zhangiq@njtc.edu.cn; ORCID ID: https://orcid.org/0000- 0003-1278-2209.

Мірошніченко Наталія Миколаївна – кандидат технічних наук, доцент кафедри "Обчислювальна техніка та програмування", Національний технічний університет "Харківський політехнічний інститут", Харків, Україна;
Nataliia Miroshnichenko – Candidate of Technical Sciences, Associate Professor of Computer Engineering and Programming Department, National Technical University "Kharkiv Polytechnic Institute", Kharkiv, Ukraine;
e-mail: natnikdr@gmail.com; ORCID ID: https://orcid.org/0000 0003-4329-7126.

Розробка методу підтримки прийняття рішення про безпеку програмного забезпечення

Чжан Ліцян, Н. М. Мірошніченко

Анотація. Визначено актуальність питання підвищення точності результатів прийняття рішення щодо процесу тестування безпеки програмного забезпечення. Проведено аналіз методів підтримки прийняття рішення. Визначено необхідність і можливість підвищення точності результатів прийняття рішення при дослідженні вразливостей програмного забезпечення в умовах нечіткості вхідних і проміжних даних. З цією метою на основі математичного апарату нечіткої логіки розроблено метод підтримки прийняття рішення про безпеку програмного забезпечення. Відмінною особливістю даного методу є синтез удосконаленого способу генерації навчальної вибірки у процес навчання штучної нейронної мережі. В рамках моделювання виконані наступні етапи дослідження. Для математичної формалізації процесу прийняття рішення та визначення вхідних даних розроблено модель формування векторів вхідних даних. Відповідно до даної моделі для формування вхідних даних формується безліч ознак потенційних вразливостей та недекларованих можливостей ПЗ відповідно до даних PVS-Studio Analysis Results. Для підвищення точності класифікації оброблюваних даних удосконалено метод навчання штучної нейронної мережі, що відрізняється способом генерації вибірки, що навчається. Даний спосіб генерації включив три рівні генерації: генерація навчальної вибірки, генерація навчального прикладу і генерація конкретного значення характеристики безпеки. Це дозволило підвищити точність класифікації та прийняття рішення у 1,6 рази для позитивних елементів у вибірці та у 1,2 рази для негативних елементів у вибірці. Для підтвердження ефективності розробки методу підтримки прийняття рішення щодо безпеки програмного забезпечення проведено ROC-аналіз з виконанням відповідних процедур. Результати експерименту підтвердили гіпотезу про ефективність розробленого методу підтримки прийняття рішення про безпеку ПЗ до 1,2 рази порівняно з методами, в основі яких використовуються положення дискримінантного та кластерного аналізу.

Ключові слова: вразливість програмного забезпечення; тестування безпеки; підтримка прийняття рішення; нечітка логіка; кіберзагроза.

Разработка метода поддержки принятия решения о безопасности программного обеспечения

Чжан Лицян, Н. Н. Мирошниченко

Аннотация. Определена актуальность вопроса повышения точности результатов принятия решения процесса тестирования безопасности программного обеспечения. Проведен анализ методов поддержки принятия решения. Определены необходимость и возможность повышения точности результатов принятия решения при исследовании уязвимостей программного обеспечения в условиях нечеткости входных и промежуточных данных. С этой целью на основе математического аппарата нечеткой логики разработан метод поддержки принятия решения о безопасности программного обеспечения. Отличительной особенностью данного метода является синтез усовершенствованного способа генерации обучающей выборки в процесс обучения искусственной нейронной сети. В рамках моделирования выполнены следующие этапы исследования. Для математической формализации процесса принятия решения и определения входных данных разработана модель формирования векторов входных данных. В соответствии с данной моделью для формирования входных данных формируется множество признаков потенциальных уязвимостей и недекларируемых возможностей ПО в соответствии с данными PVS-Studio Analysis Results. Для повышения точности классификации обрабатываемых данных усовершенствован метод обучения искусственной нейронной сети, отличающийся способом генерации обучающейся выборки. Данный способ генерации включил три уровня генерации: генерация обучающей выборки, генерация учебного примера и генерация конкретного значения характеристики безопасности. Это позволило повысить точность классификации и принятия решения в 1,6 раз для положительных элементов в выборке и в 1,2 раза для отрицательных элементов в выборке. Для подтверждения эффективности разработки метода поддержки принятия решения безопасности программного обеспечения проведен ROC-анализ с выполнением соответствующих процедур. Результаты эксперимента подтвердили гипотезу об эффективности разработанного метода поддержки принятия решения о безопасности ПО в 1,2 раза по сравнению с методами, в основе которых используются положения дискриминантного и кластерного анализа.

Ключевые слова: уязвимость программного обеспечения; тестирование безопасности; поддержка принятия решения; нечеткая логика; киберугроза.