Iranian (Iranica) Journal of Energy & Environment
Journal Homepage: www.ijee.net
IJEE an official peer review journal of Babol Noshirvani University of Technology, ISSN:2079-2115

Secure Environment via Prediction of Software Vulnerabilities-Severity

E. S. Aghaee Meybodi, M. Ghasemzadeh*

Computer Engineering Group, Engineering Campus, Yazd University, Yazd, Iran

P A P E R I N F O

Paper history:
Received 02 March 2019
Accepted in revised form 24 June 2019

Keywords:
Machine Learning
Pattern Recognition
Prediction
Vulnerability Severity

A B S T R A C T

Prediction of software vulnerabilities-severity is of particular importance. Its most important application is that managers can first deal with the most dangerous vulnerabilities when they have limited resources. This research shows how we can use the former patterns of software vulnerabilities-severity along with machine learning methods to predict the vulnerabilities severity of that software in the future. In this regard, we used the SVM, Decision Trees (DT), Random Forests (RF), K Nearest Neighbors (KNN), bagging and AdaBoost algorithms along with the already reported vulnerabilities of Google Android applications, Apple Safari and the Flash Player. The experimental results showed that the Bagging algorithm can predict Google Android vulnerability with accuracy of 78.21% and f1-measure equal to 77%, the vulnerability of the Flash Player software with accuracy of 82.37% and f1-measure equal to 87.73% and predict the vulnerability severity of the Apple Safari with accuracy of 70.58% and f1-measure equal to 70%. The novelty of this research is introduction of a new method for prediction of software vulnerabilities severity.

***doi**: 10.5829/ijee.2019.10.02.14*

INTRODUCTION

Vulnerability is of great importance in many contexts such as deep learning, software and live migration of virtual machines [1-2]. In fact, vulnerability is a bug, flaw, weakness or exposure of an application, system, device or service that could lead to a failure of confidentiality, integrity or availability. The exploitation of software vulnerabilities creates significant security risks in the host computer system. Software vulnerability is the most important item that represents the level of software risk [3]. Given the restriction of human and financial resources, software manufacturers can prioritize vulnerabilities based on their criticality. For less damage to the system, the vulnerabilities that are more critical to be prioritized for repair.

So far, a lot of research has been conducted on the forecasting of software vulnerabilities and several models have been developed [4-15]. Due to limited financial and human resources and increasing the number of vulnerabilities, priority is given to predict and amendment of vulnerabilities [16]. However, severity prediction of vulnerabilities is also important because the same number of vulnerabilities might cause different grades of damage because of their different access spaces and occurrence methods. According to the above, it seems that paying less attention to the severity of the vulnerability is a problem that is presented in all predictive models. For this reason, in this article, we focussed on predicting severity rather than the number of vulnerabilities.

Because software risk is associated with vulnerability severity, we further study a security risk metric based on the severity prediction of software vulnerability. In order to investigate the ability of machine learning algorithms in predicting the severity of the software, three examples of highly vulnerable software are selected and the vulnerability of each software is extracted from NVD database [17]. Then, we provide the data of each software to the basic and ensemble machine learning algorithms and examine the results.

BACKGROUNDS AND RELATED WORKS

In recent years, many researchers have been working on anticipating vulnerabilities and providing a lot of predictive models. Here are some examples of these models:

Rahimi et al. [4] reviewed the correlation between vulnerabilities with complexity and code quality. They defined two metrics to represent code complexity (CC) and code quality (CQ), respectively. They have modelled the relationship between number of vulnerabilities and the two factors via a random walk.

Riccardo et al. [5] used some ideas from defect prediction and also used the security-oriented static analysis tool to identify potential vulnerabilities in source code. Based on text

* Corresponding Author Email: m.ghasemzadeh@yazd.ac.ir (Mohammad Ghasemzadeh)

mining of the source code, they presented an approach to predict whether a software component is vulnerable.

In 2010, Nguyen and Tran [12] used machine learning techniques and introduced a prediction model using the correlation graph based on the relationship between software components such as classes, variables, functions. Their method is based on two different versions of JavaScript engine Mozilla Firefox (JSE) have confirmed that the average accuracy and recall for the first version were 60 and 68% respectively. The second version was 60% for accuracy and 61% for recall.

Shin et al. [6-9] Commented on the relationship between software features and vulnerabilities, and concluded that three complexity features, code churn, and developer activity metrics could be useful in predicting vulnerabilities.

Rescorla [10] proposeds an exponential model to approximate the relation between time and the number of vulnerabilities. This perhaps is the first time-series-based VDM and it is inspired by some reliability models such as G-O model and NHPP model.

Alhazmi and Malaiya [3] proposed a time-based model for a cumulative number of vulnerabilities found after studying the vulnerability data of two operating systems, Windows 98 and Windows NT. The model has been proved good fit on a range of operating systems vulnerability data. Later they propose an alternative model based on the effort that leads into finding vulnerabilities. The effort-based model also makes a good performance on the two operating systems

According to research, vulnerability prediction models are allocated into two groups:

(1) **Code-attributes-based models** These models focus on discovering the relationship between code attributes and vulnerabilities. Nguyen and Tran [12] concentrated on the source code dependencies of the software and developed the prediction model using machine learning methods. Rahimi and Zargham [4] extracted complexity features and code quality from the source code, and developed a vulnerability-detection model to forecast vulnerability detection. Shin et al. [6-9] used software benchmarks, code churn and developer activity metrics as forecasters and assumed logistic regression to find vulnerabilities.

(2) **Time-series-based models** Basically, the target of these models is to describe the relationship between time and the accumulated number of vulnerabilities and to predict the total number of vulnerabilities until a certain time point. To achieve this goal, various kinds of models are proposed, such as Alhazmi's AML model [3], Rescorla's exponential model [10], Poisson's logarithmic model [14], Anderson's thermodynamic model [15].

Current researches focus on the recognition of vulnerable components and the prediction of the number of vulnerabilities. However, the qualitative analysis of software to determine whether it is vulnerable is too rough and has a limited effect on the software security evaluation. Also, taking the same severity for all vulnerabilities cannot accurately reflect the software security level. We need to find a new benchmark to replace the amount of vulnerabilities as targets.

In our opinion, different vulnerabilities may have different levels of severity. Therefore, a metric is needed to evaluate the seriousness of each vulnerability. Then, there comes the definition of “Common Vulnerability Scoring System (CVSS)”.

The Common Vulnerability Scoring System (CVSS) is a way to achieve the main features of a vulnerability and generate a numerical score that reflects its severity. Numeric scores can be converted into qualitative displays such as low, medium, high, and critical and help users to properly evaluate and prioritize their vulnerability management processes. Severity vulnerability scores are calculated based on a formula and depend on several criteria that clearly indicate the exploitation and the impact of exploitation. CVSS scores range from 0 to 10 with 0.1 increments and vulnerabilities with severity of 10 are the most critical vulnerabilities. CVSS criteria for severely scaling vulnerabilities are divided into three groups:

Basic metrics measure the inherent and essential characteristics of a vulnerability that does not change over time or in different environments. Temporary metric, measure the characteristics that change over time, but do not change in user environments. Environmental metric measure those vulnerabilities that are relevant and unique to a specific user.

There are six basic metrics that record the most important vulnerabilities: accessibility vector, access complexity, identity authentication, privacy impact, impact integrity, and impact of availability. The scoring process first calculates benchmarks based on the fundamental equation, which gives a score of 0 to 10 and creates a vector. If desired, the base score can be more complete by assigning the values to the time and environment criteria.

Software application vendors are providing CVSS base scores and vectors to their customers. This helps them properly communicate the severity of vulnerabilities in their products and helps their customers effectively manage their IT risk.

METHODOLOGIES

The algorithms that we used in our experiments were: the SVM, Decision Trees (DT), Random Forests (RF), K Nearest Neighbors (KNN), bagging and AdaBoost algorithms [18]. Support Vector Machines (SVM) are used for classification in machine learning. SVM is the support vector of the classification or classifier algorithm and is recognized as one of the best techniques for categorizing and detecting outlier, and unlike clustering algorithms in the learning category, it is monitored and has two phases of training and testing. The support vector machine was originally constructed for the separation and categorization of linear separable data, but was later extended for nonlinear mode. In fact, SVM needs to use different kernels to separate nonlinear data. To do this, it does not work in two-dimensional space, but data is mapped to a more dimensional space so that it can be linearly segmented in this new space. In fact, the main idea behind the backup engineer is to draw up cloud-based screens in space to optimize the operation of different data model types. Finds the super-page with the largest margin of isolation, and the closest educational data to the superconductor is called the support vectors. The purpose of KNN algorithm is to use a database in which the data points are separated into K separate classes to predict the classification of a new sample point.

It has been used the concept of Bootstrap Aggregating to make different estimates. In principle, Bagging (Bootstrap Aggregation) can be used to evaluate the accuracy of the estimates used in data mining methods through sampling by replacing educational data. In this technique, it is assumed that the educational series is representative of the community under study, and a variety of realized states of society can be simulated from this data set. Therefore, using the re-sampling method will be achieved by employing a variety of different data sets. When a new sample is entered into each of the class divisions, a majority agreement is used to identify the class to be desired.

Random forest is a tree-based technique that employs a large number of decision trees that are made up of random sets of properties. Unlike a simple decision tree, the random forest method is highly untranslatable, but it's generally well-functioning operation has made it a well-known algorithm. Random Forests is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the classes output by individual trees [18]. AdaBoost or adaptive boosting is a well-known method for boosting. AdaBoost produces a very accurate classification rule by combining moderately inaccurate weak classifiers.

The main idea of this paper is to investigate the ability of machine learning algorithms to forcast the vulnerability severity. The number of data to run the machine learning algorithms should be high enough otherwise the training is not done well. For this purpose, we use three software with high vulnerability in this article. Table 1 shows the names of these software and the number of registered vulnerabilities from 1999 until the end of 2016. The experimental data comes from NVD [17].

For each of this three software, we extract the recorded vulnerability from 1999 to 2016 and create a feature vector. To create a feature vector, CVE identifier, release date, and severity of the vulnerability are used. The severity of the vulnerabilities registered in NVD is based on the CVSS standard which is continuous numbers for 0 to 10 with intervals of 0.1.

Finally, we only consider the Sorted Vulnerability severity as the feature vector and the time series data. We create a feature vector based on the number of time delay. The number of time delays that used for different algorithms and software varies; if the number of time delays used is high, it causes overfitting. If we have too many features, the learning hypothesis may fit the training set very well (with cost function $J(\theta)\approx 0$) but fail to generalize to new examples. This means overfitting. So the number of delays used should be appropriate.

For each experiment, among these samples, in accordance with the usual training procedure, two-thirds of the samples are selected as training data and one-third of the samples are taken as test data; that is, the ratio of the data from the training to the test is from 70 to 30.

Random forest and decision tree are implemented using WEKA and the rest of the algorithms are implemented with MATLAB software. Due to its importance in this research, we have implemented the bagging algorithm with both of the software. Sampling method with replacement has been used in the method of bagging and random forest. It is obvious that the random selection with replacement leads to an overlap between the different education collections and the number of sample collections being different. Also in the Bagging method and the random forest, at each step after training, the classifiers are cast for each sample of the test data. The result of the vote indicates that the severity of the next vulnerability is in which class.

In the implementation of the SVM algorithm, the RBF kernel function is used. The two parameters for this algorithm must be set. The j48 algorithm is the decision tree algorithm implemented with WEKA software. The algorithms of RUSBoost, AdaboostM2, and Subspace are AdaBoost algorithms. The discrete and interpolation method varies for different software applications. In the following, the breakdown and trivialization of the three Google Android applications, Flash player and Apple Safari can be summarized in Tables 2 to 5.

TABLE 1. Properties of tree Software with high vulnerability

Product	Vendor	Type	Number of vulnerability
Flash player	Adobe	Application	995
Android	Google	OS	942
Safari	Apple	Application	860

TABLE 2. Labeling method and number of samples in the 4-class of Google Android software

Class name	Intervals	Number of training data	Number of test data	Total data
Low	0-6.6	172	49	221
Medium	6.7-8.8	155	55	210
High	8.9-9.3	230	96	326
Critical	9.4-10	97	81	178
Total		654	281	935

TABLE 3. Labeling method and number of samples in the 3-class of Google Android software

Class name	Intervals	Number of training data	Number of test data	Total data
Medium	0-7.1	243	66	309
High	7.2-9.3	314	133	447
Critical	9.4-10	97	81	178
Total		654	280	934

TABLE 4. Labeling method and number of samples in the 3-class of flash player software

Class name	Intervals	Number of training data	Number of test data	Total data
Medium	0-9	106	46	152
High	9.1-9.5	175	66	241
Critical	9.6-10	411	185	596
Total		692	297	989

TABLE 5. Labeling method and number of samples in the 3-class of apple safari software

Class name	Intervals	Number of training data	Number of test data	Total data
Medium	0-5.4	188	83	271
High	5.5-7.2	191	99	291
Critical	7.3-10	217	73	290
Total		596	255	851

RESULTS

We evaluated the algorithm performance by means of four indicators:

• Accuracy is the percentage of correct results.

$$\text{Accuracy} = \frac{TP+\text{TN}}{TP+FP+TN+FN} \tag{1}$$

• Precision is the probability that a (randomly selected) retrieved document is relevant.

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2}$$

• Recall is the probability that a (randomly selected) relevant document is retrieved in a search

$$\text{Recall} = \frac{TP}{TP+FN} \tag{3}$$

• F1-measure is often introduced as a harmonic mean of precision and recall.

$$f_1 = 2.\frac{Precision\ .Recall}{Precision+Recall} \tag{4}$$

Since F1-measure considers both precision and recall, we select F1-measure as the main evaluation metric. The goal is to maximize this metric and compare the results by this metric. In the following, we will describe the results obtained from the implementation of machine learning algorithms for Google Android applications, Flash Player and Apple Safari.

Results of prediction for Google Android

Table 6 shows the results of the accuracy of prediction in Google Android. Since the number of data in classes are unbalanced and the number of classes is more than two; the accuracy metric is not a complete metric for comparison. Therefore, we have used other metrics such as F1-measure which results are presented in Table 7.

As indicated in Table 6, in 4-class mode, the highest accuracy devoted to Bagging and RUSBoost algorithms which have 68.93 and 68.68%, respectively. In 3-class mode, the highest values for the Bagging and SVM algorithms are 78.21%. In 3- class mode results were better than 4-class. According to summary of results in Table 7, the best F1-measure in 4-class mode belongs to the RUSBoost and Begging algorithms with values of 65% and 65.31%. In 3-class mode, the best results belong to beginning algorithms and SVM that the F1-measure is approximately 76.98%. In this experiment, the Bagging classifier performs better and the best result is when the number of classifiers is 3.

In general, the severity of the next Android vulnerability is predictable with accuracy 78.21% and F1-measure about 77% with the Bagging algorithm that uses the combination of the decision tree classifiers.

Results of prediction for flash player

Table 8 shows the results of the accuracy of prediction for flash player. As can be seen in this table, the highest accuracy values associated with bagging random forest which is approximately 82.49%. As can be seen in Table 9, subspace and bagging algorithms have a higher average F1-measure and their values are 73.56% and 73.87%, respectively. So, in general, the Bagging algorithm with an accuracy of 82.37% and an F1-measure of approximately 73.87%, are capable of predicting the severity of the software vulnerability for the flash player.

TABLE 6. Accuracy results for google android

Number of class	4-class		3-class	
Classification algorithm	**Train accuracy**	**Test accuracy**	**Train accuracy**	**Test accuracy**
RUSBoost	69.47	**68.68**	74.66	73.21
Bagging	72.63	**68.93**	**82.52**	**78.21**
AdaBoostM2	74.65	67.62	75.42	75.1
KNN	65.70	65.60	70.78	70.57
Subspace	84.58	65.48	78.32	76.16
SVM	69.31	67.5	**80.67**	**78.21**
J48	68.70	65.48	75.45	71.15
Random forest	84.89	65.48	78.78	75.8
Bagging(WEKA)	67.78	67.14	77.5	72.37

TABLE 7. Precision, recall and f1-measure results for google android

Number of class	**3-class**			**4-class**		
Classification algorithm	**Precision**	**Recall**	**F1-measure**	**Precision**	**Recall**	**F1-measure**
RUSBoost	71.38	73.53	72.17	64.88	65.25	**65**
Bagging	77.57	76.33	**76.96**	65.80	64.92	**65.31**
AdaBoostM2	73.92	72.27	72.96	64.22	63.55	63.83
KNN	69.85	71.93	69.83	63.36	62.1	62.61
Subspace	75.42	73.69	74.43	62.76	62.1	62.35
SVM	77.80	76.33	**76.98**	63.98	63.65	64.51

TABLE 8. Accuracy results for flash player

Number of class	**3-class**	
Classification algorithm	**Train accuracy**	**Test accuracy**
RUSBoost	80.66	80.47
Bagging	**82.83**	**82.37**
AdaBoostM2	86.13	81
KNN	85.84	81.48
Subspace	81.53	81.15
SVM	**84.13**	**82.49**
J48	83.38	81.82
Random forest	**83.29**	**82.49**
Bagging(WEKA)	82.52	**82.1**

TABLE 9. Precision, recall and f1-measure results for flash player

Number of class	**3-class**		
Classification algorithm	**Precision**	**Recall**	**F1-measure**
RUSBoost	72.61	71.25	72.56
Bagging	77.88	72.54	**73.87**
AdaBoostM2	75.78	77.65	72.28
KNN	77.62	71.21	72.78
Subspace	78.76	70.91	**73.56**
SVM	80.12	69.89	72.97

TABLE 10. Accuracy results for apple safari

Number of class	**3-class**	
Classification algorithm	**Train accuracy**	**Test accuracy**
RUSBoost	74.29	68.5
Bagging	**80.87**	**70.58**
AdaBoostM2	80.94	67.19
KNN	76.3	67.19
Subspace	**70.64**	**70.2**
SVM	**75.46**	**70.70**
J48	69.73	69.53
Random forest	**75.46**	**70.31**
Bagging(WEKA)	**72.86**	**70.70**

TABLE 11. Precision, recall and f1-measure results for apple safari

Number of class	**3-class**		
Classification algorithm	**Precision**	**Recall**	**F1-measure**
RUSBoost	67.52	67.53	67.26
Bagging	69.42	69.84	**69.45**
AdaBoostM2	66.85	66.83	66.61
KNN	69.15	66.62	66.43
Subspace	69	69.17	**69.1**
SVM	71.15	70.34	**70.32**

Results of prediction for apple safari

The accuracy' results of the algorithms on the Apple Safari software are shown in Table 10. As can be seen in this table, the highest accuracy values associated with SVM, random forest, Bagging, and subspace which are approximately 70.58%.

As shown in Table 11, subspace, bagging, and SVM algorithms have a higher mean F1-measure and have a value of 69.1, 69.45 and 70.32%, respectively. So, in general, the Bagging algorithm and SVM with an accuracy of approximately 70.58% and F1-measure of approximately 70% can be very good in predicting the severity vulnerability of the Apple Safari.

CONCLUSIONS

Software vulnerability is the most important item that represents the level of software risk. Different vulnerabilities may have different levels of severity. Current researches focus on the recognition of vulnerable components and the prediction of the number of vulnerabilities. However, the qualitative analysis of software to determine whether it is vulnerable is too rough and has a limited effect on the software security evaluation; Also, taking the same severity for all vulnerabilities cannot accurately reflect the software security level. For this reason, In this paper we applied machine learning methods to forecast the vulnerabilities severity. Bagging which uses the combination of decision trees could forecast the severity of the future vulnerability of software with higher accuracy than the other applied algorithms. Using the result of this research, software vendors can predict the vulnerability severity of their products.

REFERENCES

1. Yuan, Xiaoyong, et al. "Adversarial Examples: Attacks and Defenses for Deep Learning." IEEE Transactions on Neural Network,, 2019, pp. 1–20.
2. M. Tajamolian, and M. Ghasemzadeh. "A Versioning Approach to VM Live Migration." International Journal of Engineering, Transactions B: Applications, vol. 31, no. 11, 2018, pp. 1838–1845.
3. O. H. Alhazmi and Y. K. Malaiya, 'Prediction capabilities of vulnerability discovery models', in RAMS '06. Annual Reliability and Maintainability Symposium, 2006. 2006, pp. 86–91.
4. S. Rahimi and M. Zargham, 'Vulnerability Scrying Method for Software Vulnerability Discovery Prediction Without a Vulnerability Database', IEEE Transactions on Reliability, vol. 62, no. 2, pp. 395–407, 2013.
5. R. Scandariato, J. Walden, A. Hovsepyan, and W. Joosen, 'Predicting vulnerable software components via text mining', IEEE Transactions on Software Engineering, 2014.
6. Y. Shin and L. Williams, 'An empirical model to predict security vulnerabilities using code complexity metrics', Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement - ESEM '08, 2008
7. Y. Shin and L. Williams, 'Is Complexity Really the Enemy of Software Security',, Proc. the 4th ACM Workshop on Quality of Protection, Alexandria, Virginia, USA, Oct. 2008.
8. Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, 'Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities', IEEE Transactions on Software Engineering, 2011.
9. Y. Shin and L. Williams, 'Can traditional fault prediction models be used for vulnerability prediction?', Empirical Software Engineering, 2013.
10. E. Rescorla, 'Is finding security holes a good idea?', IEEE Security and Privacy. 2005
11. R. Scandariato and J. Walden, 'Predicting vulnerable classes in an Android application', Proceedings of the 4th international workshop on Security measurements and metrics - MetriSec '12, 2012.
12. V. H. Nguyen and L. M. S. Tran, 'Predicting vulnerable software components with dependency graphs', Proceedings of the 6th International Workshop on Security Measurements and Metrics - MetriSec '10, New York, USA: ACM Press. pp. 3–10, 2010.
13. C. Nie, X. Zhao, K. Chen, and Z. Han, 'An software vulnerability number prediction model based on micro-parameters', Jisuanji Yanjiu yu Fazhan/Computer Research and Development, 2011.
14. J. D. Musa and K. Okumoto, 'A logarithmic poisson execution time model for software reliability measurement', in Proceedings of the 7th international conference on Software engineering, 1984, pp. 81–87.
15. R. Anderson, 'Security in Open versus Closed Systems - The Dance of Boltzmann, Coase and Moore', vol. 4, no. 15, pp. 121–127, 2002.
16. Geng, Jinkun, Daren Ye, and Ping Luo. 2015. "Forecasting Severity of Software Vulnerability Using Grey Model GM(1,1)." In 2015 IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 344–48.
17. U.S Department of commerce, "NVD. National Vulnerability Database." [Online]. Available at: https://nvd.nist.gov/. [Accessed: 106-May-2019].
18. Ian H Witten, and Frank Eibe. Data Mining: Practical Machine Learning Tools with Java Implementations. 1999.

Persian Abstract

DOI: 10.5829/ijee.2019.10.02.14

چکیده

پیش‌بینی شدتِ آسیب‌پذیری نرم‌افزار اهمیت خاصی دارد. مهم‌ترین کاربرد آن، این است که مدیران با توجه به زمان و دیگر منابع محدودی که در اختیار دارند، می‌توانند ابتدا با آسیب‌پذیرترین موارد مقابله کنند. این تحقیق نشان می‌دهد که چگونه می‌توان از الگوهای سابقِ شدت آسیب‌پذیری نرم‌افزارها و به‌کارگیری روش‌های یادگیری ماشین، برای پیش‌بینیِ آسیب‌پذیری نرم‌افزارها در آینده بهره برد. در این راستا، عملکرد الگوریتم‌های درختان تصمیم، ماشین بردار پشتیبان، جنگل‌های تصادفی، نزدیک‌ترین همسایگان، بگینگ و آدابوست را در کنار آسیب‌پذیری‌های گزارش‌شده برای برنامه‌های کاربردی اندروید گوگل، سافاری اپل و فلش پلیرمورد بررسی قرار دادیم. نتایج نشان دادند که الگوریتم بگینگ می‌تواند آسیب‌پذیری گوگل اندروید را با دقت ۷۸/۲۱ درصد و ملاک f1 برابر ۷۷ درصد، آسیب‌پذیری نرم‌افزار فلش پلیر را با دقت ۸۲/۳۷ درصد و ملاک f1 برابر با ۸۷/۷۳ درصد پیش‌بینی کند. همچنین شدت آسیب‌پذیری اپل سافاری را با دقت ۷۰/۵۸ درصد و ملاک f1 برابر با ۷۰ درصد پیش‌بینی کند. نوآوری این تحقیق متکی بر معرفی یک روش جدید برای پیش‌بینی شدت آسیب‌پذیری نرم‌افزار است .