The Random Neural Network as a Bonding Model for Software Vulnerability Prediction*

Katarzyna Filus[1], Miltiadis Siavvas[2], Joanna Domańska[1], and Erol Gelenbe[1]

[1] Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Baltycka 5, 44-100 Gliwice, PL
{kfilus,joanna,seg}@iitis.pl
[2] Information Technologies Institute, Centre for Research & Technology Hellas, 6th km Harilaou - Thermi, Thessaloniki 57001, GR
{siavvasm}@iti.gr

Abstract. Software vulnerability prediction is an important and active area of research where new methods are needed to build accurate and efficient tools that can identify security issues. Thus we propose an approach based on mixed features that combines text mining features and the features generated using a Static Code Analyzer. We use a Random Neural Network as a bonding model that combines the text analysis that is carried out on software using a Convolutional Neural Network, and the outputs of Static Code Analysis. The proposed approach was evaluated on commonly used datasets and led to 97% training accuracy, and 93%-94% testing accuracy, with a 1% reduction in false positives with respect to previously published results on similar data sets.

Keywords: Random Neural Networks · Software Vulnerability Prediction · Machine Learning · Convolutional Neural Networks · Text Mining

1 Introduction

The cost of building secure software is high but the results of not meeting security requirements may be much more severe. Cyberattacks occur on a daily basis and threaten the security and privacy of valuable and sensitive information. Indeed, the Cisco Consumer Privacy Survey shows that a significant majority of 84% of respondents care about data privacy [10]. Thus security was stated to be "foundational" and a "top IT priority" in the Cisco 2019 Annual Report [9].
Security of software is mostly based on carefully written source code, and software vulnerabilities are the consequence of defects that are hard to find [89]. Most vulnerabilities are caused by common programming mistakes introduced by programmers in the early stages of the Software

* This research was funded by the European Commission (EC) through the EU H2020 IoTAC Research and Innovation Action under Grant Agreement ID: 952684, and through the EU H2020 SDK4ED Research and Innovation Action under Grant Agreement ID: 780572. The EC's financial support does not constitute an endorsement of this paper, which reflects the views only of the authors.

Development Life Cycle (SDLC) [79]. Programmers often lack the security expertise and their testing resources are limited. Hence, Static Code Analyzers based on software security standards, e.g. SonarQube [80] and Veracode [87] are often used in the software production cycle. It has also become common to use binary classifiers to prioritize testing efforts [69]. Despite the fact that modern programming languages focus on security and that many software security guidelines are available, vulnerabilities are still rife. Referring to Veracode's software security report [86], more than 85 % of all applications scanned using the Veracode's application security platform between April 1, 2017 and March 31, 2018, had at least one software vulnerability. More than 13 % of the applications contained at least one critical flaw. Approximately 85.7 % of .NET applications, 87.5 % of Java applications and 92 % of C++ applications contain at least one vulnerable component.
Unfortunately, code analysis in terms of security is time-consuming and expensive [50], and thus researchers should commit their efforts to create accurate and efficient Vulnerability Predictors based on new approaches. In recent years, results based on heterogeneous features suggest that it is a good approach to follow [15,99], while the application of deep learning to Vulnerability Prediction is increasing [12, 62, 69], and it would also be beneficial to establish guiding principles regarding the representation and utilization of software components with such models [59]. The currently used methodologies presented in Section 2 offer a panorama in this respect.

1.1 Scope of the Present Work

The aim of our work is to create a software vulnerability prediction model based on Random Neural Networks (RNN) and Convolutional Neural Networks (CNN) that uses both text mining features and metrics generated from a Static Code Analyzer.
Over the years, RNNs have found application to video compression [41], medical image segmentation [25], the search for buried explosive devices [35], vehicle classification [46], in the field of virtual reality [33, 52], in augmented reality [34, 48], and network attack detection and mitigation [4, 17, 39, 65, 70].
Other areas where RNNs have been successful include deep learning [98], smart network management [14, 31, 36, 37, 40], emergency management and cyberphysical systems [42, 43], the dynamic management of Cloud and Fog services [21, 22, 90], the use of machine learning in smart search [75, 76] and network routing including the use of Software Defined Networks [5, 19, 20, 29, 30, 38, 91].
The CNN has also been successfully used in many fields [1,58], including for Magnetic Resonance Image reconstruction [94], automatic road segmentation [57], music generation [93] and relation extraction from plain text [45].
This wide usage and success of both the RNN and the CNN in a variety of applications justifies their use for Software Vulnerability Prediction in this paper, where text data processing and dimensionality reduction is carried out with a small CNN, and the RNN is used as a model that

bonds both parts of the analysis. We utilize transfer learning: feature maps generated by a hidden layer of the CNN are provided as input to the RNN model and additional features that are obtained via metrics generated from a static analysis tool are used to improve the identification of vulnerabilities and the reduction of false positives.
Section 2 describes the background and work regarding Software Vulnerability Prediction. It also describes the approaches used in this field. Section 3 makes a brief introduction to the Random Neural Network. Section 4 presents the methodology used based on machine learning, and the dataset that was used for training the network. In section 5 we describe our experimental results, while Section 6 concludes the article and discusses future work.

2 Software Vulnerability Prediction

Security plays a crucial role in modern information systems, and has become a primary concern in programming language design and implementation [71]. Modern programming languages focus on safety and reliability, because weaknesses in language models can be exploited by attackers, and compilers are used as control mechanisms on program execution.
To enhance software security, languages such as OCaml, Java, and C# use static analysis and dynamic checks [71]. RUST is a system programming language recommended for safety critical domains, which assumes memory-safety and thread-safety [13]. Additionally, software-oriented organizations such as the Open Web Application Security Project (OWASP) [66], SANS Institute [72] and the Computer Emergency Response Team Coordination Center (CERT/CC) [7] create standards concerning software security and publish guidelines, e.g. OWASP Secure Coding Practices Guide [67], on how to create secure software components.
Many critical software vulnerabilities are grouped into rankings such as OWASP Top10 [68] and CWE/25 [11]. To produce secure code, developers focus on static security tests [86] that help eliminate vulnerabilities in the coding stage of the Software Development Life Cycle (SDLC), offered by some of the Integrated Development Environments (IDEs) themselves, e.g. Visual Studio (C/C++) [88], IntelliJ IDEA (Java) [49] and Eclipse (Java) [16]. Dedicated applications that focus on static analysis have also been created, e.g. SonarQube [80] and Veracode [87].
Two types of software analysis are generally used for vulnerability detection: static and dynamic analysis, each with testing methodologies that differ significantly from each other. Static analysis is usually applied in the early stages of the SDLC. It can be based on Text Mining, Software Metrics, or security-related Automated Static Analysis (ASA) alerts [79]. It is successful in detecting leaks of private data, unauthorised access to resources, permission misuse, intent injection, clone detection, code verification, cryptography implementation issues and test generation [3]. Dynamic testing is much more time consuming; it needs an executable version of the software, and is therefore used later in the life cycle. It is perceived to be more complex than static analysis [3] because it not

only requires an executable version of the software, but also requires additional resources and the ability to simulate user behaviour.
Dynamic testing can be divided into Fuzz testing [54], Concolic Testing [92] and Search Based Testing [73]. It is important to highlight that none of these methods should be treated as being superior. They offer different testing methodologies and environments. Some vulnerabilities are easier to find using static analysis, and for some of them one must apply dynamic analysis, as they simply cannot be found before the program is actually executed [86]. It is generally a good practice to use both of them, because using a single testing technique is not sufficient to identify all vulnerabilities and address all the problems that may be encountered [85].
Software metrics can be used to discriminate between vulnerable and non-vulnerable software components [2]. In [61] complexity metrics are used, while in [8] metrics named complexity, coupling and cohesion have been used as early indicators of vulnerabilities. In [78] the Complexity, Code Churn and Developer Activity metrics are used to indicate the potential presence of vulnerabilities.
ASA alerts which apply Singular Value Decomposition, have been used to identify fault-prone software components [77]. In [23], ASA alerts have been used for early identification of vulnerability-prone and attack-prone software components, and this approach can be used for prioritizing redesign, as well as for verification and validation efforts.
Text Mining for Software Vulnerability Prediction has also generated much interest. In [63] software components were represented as a list of extracted "includes" and "function calls" with information about whether a file incorporates them. In [51] function calls have been retrieved from Abstract Syntax Trees. As a result, a single component can be represented as a binary vector of features. Some approaches from Natural Language Processing (NLP) have been applied as well. In [59] the software components were represented as a series of terms. Using this approach, the order of the terms is also taken into consideration. Alternatively, the components can be treated as a Bag-Of-Words (i.e. a set of tokens or unit terms) with associated frequencies [74] or using term-weighing called the Term Frequency-Inverse Gravity Moment [56]. As an alternative to a high-level representation of the program, in [50] cleaned java bytecode lines were treated as a set of words, together with n-grams, a popular NLP technique. Using this method, bigger groups of terms are also treated as unit terms. In [69], an approach that combines n-grams and statistical feature selection is presented.
In some works, mixed features are used to build Vulnerability Prediction models. In [100] Text Mining features are used along with Software Metrics. In [99] traditional software metrics are used along with Developer Metrics, Software Property Metrics, Popularity Metrics and Security Metrics which are alerts generated by a static code analyzer. In [15], besides the Complexity Metrics (Software Metrics), special Vulnerability Metrics such as Dependency Metrics, Pointer Metrics and Control Structures are used.
Other works have compared different approaches to vulnerability prediction. In [89] and [81] the approaches based on Software Metrics and Text

Mining are compared. In a paper [51] three approaches have been discussed, namely: using the presence of includes and function calls (text mining based on regex expressions and Abstract Syntax Trees), traditional text mining using tokens with corresponding frequencies of occurrence, and software metrics. The results in [51] suggest that the performance of models based only on software metrics is insufficient. On the contrary, the results presented in [81] suggest that from a practical perspective, the performance of models based on software metrics is comparable to text mining techniques.
Apart from the form of the training features, many different prediction models have been used in this field: Support Vector Machines (SVM) [63], Random Forests [50, 56, 74, 81, 89], Bayesian Networks [78], Linear Discriminant Analysis [78], Decision Trees [56, 99], Boosted Trees [99], Linear Regression [99], Naïve Bayes Classifier [74], K-Nearest Neighbours [56], Artificial Neural Networks [6, 99] with Deep Learning in particular [12, 59, 62, 69].
This paper proposes an approach based on mixed features, i.e. the text mining features and the features generated using a Static Code Analyzer. To build our model we use Convolutional Neural Networks and, novel in this field, Random Neural Networks.

3 Random Neural Networks

The Random Neural Network (RNN) is a specific type of Artificial Neural Network introduced in [26, 27], whose purpose was to mimic the spiking behaviour of neurons in the mammalian brain. A gradient based learning algorithm for the RNN was introduced for both feedforward and recurrent (feedback) networks [24]. The RNN has been considerably extended to develop the theory of G-Networks [18, 28] in queueing theory and stochastic processes, and from a machine learning perspective it has also resulted in deep learning algorithms [44].
In the RNN, each neuron's internal state is represented by a non-negative integer, and information is exchanged between neurons using positive and negative spikes which play opposite roles: a positive spike received by a neuron will increase its internal state by 1 representing excitation, while the arrival of a negative spike will reduce its state by 1 (provided its state is non-zero) representing inhibition. If a neuron's potential is strictly positive, we will say that it is "excited" and it is then able to "fire" or send out spikes at exponentially distributed intervals, while if it potential is zero the neuron is quiescent and it cannot send spikes to other neurons or to entities that are external to the network. When a neuron fires, its internal state drops by one for each outgoing spike.
RNNs have been shown to exhibit a great classification power and are able to outperform other traditional methods in many different areas [47, 95–97]. They have been successfully used in the fields of vehicle classification [46], medical image segmentation [25], inserting 3-D images in moving virtual reality scenes [33], augmented reality simulation of transportation systems [48], determining the state of servers in a dynamic network Cloud environment [96] and attack detection [4, 64, 70]. Many

different models of RNNs have been introduced: multiple signal class random neural networks [32], the Dense Random Neural Network [4], the Spiking Random Neural Network Function Approximator [95].
In this paper we use a modification of the RNN's training process with a Weight Restriction that reduces the number of computationally-demanding operations by considering only the derivatives of positive weights during the Gradient Descent algorithm. Since in this case the sum of the outgoing excitatory and inhibitory weights of a neuron to another neuron is fixed, the change of the excitatory weights will imply an identical but opposite change in the inhibitory weights. Also, as an alternative to random weight initialization [24, 46] or some other computationally-demanding approaches [24, 83, 84], we use a "neutral" initialization that sets the weights to such values which set the initial excitation probability of each neuron in the network to 0.5, when the input values to the network are also neutral.

4 Methodology

In our approach, we propose a system that combines text mining features with metrics obtained from a Static Code Analyzer. To predict the labels of the examined software components we utilize Artificial Neural Networks, namely Random Neural Networks and Convolutional Neural Networks. We used 70 % of the data for training and 30 % for testing. To eliminate the imbalance of the data, we down-sampled the majority class (i.e. the non vulnerable class) for the training set and left the original number of samples in the testing dataset. We select five best features out of all the metrics generated by the Static Code Analyzer using the χ^2 ranking technique. Additionally, we performed feature analysis using a visualization method called the $t-distributed$ stochastic neighbour embedding (t-SNE) which is a non-linear algorithm used to decrease the dimensionality of data [55]. It is a good choice for examining whether a local structure exists in high-dimensional data, and as a means to visualise it in a two-dimensional space [60].

4.1 Dataset

We have used a dataset introduced in [59] for our experiments that consists of 61,638 source code components, of which 43,913 are not vulnerable and 17,725 are vulnerable. The software elements are written in C/C++. Two types of vulnerabilities are considered in this dataset: CWE-119 - buffer error vulnerabilities (10,440 components) and CWE-399 - resource management error vulnerabilities (7,285 components), and we only consider CWE-399 vulnerabilities.
To achieve the text mining features, we follow a method described in [74]. This approach is based on the assumption that the programming language can be treated as an natural language, so that common text mining techniques can be applied to represent the information that it contains. Using the Bag-Of-Words techniques, the elements can be represented as tokens with associated frequencies. To obtain the desired metrics, we

utilize a static code analyzer named SonarQube [80]. The tool allows to extract a broad number of metrics, including size (e.g. lines of code) and complexity. The number of instances in the classes and the whole dataset has been presented in Table 1.

Table 1: The number of instances in non-vulnerable and vulnerable classes of the examined data

Category	**Instances**	
non-vulnerable	815	1499
vulnerable	684	

4.2 Proposed system

The predictive part of the proposed system is composed of two parts: the first one is used for dimensionality reduction and extracting the information from text features, the second one is a bonding model for the extracted feature maps and metrics generated from a static analysis tool. The first part is built on a small CNN neural network. The text mining model has been built using the Keras functional API [53], and the second part is a computationally-efficient RNN. The workflow of the system is presented in Figure 1.
The feature maps we take from the CNN part are 1-dimensional with 10 features. To enhance the result,s we use additional features generated using SonarQube tool [80]. The best features are chosen with a popular ranking technique - χ^2, and we take the five best features to estimate how the feature diverges from the expected distribution [82].

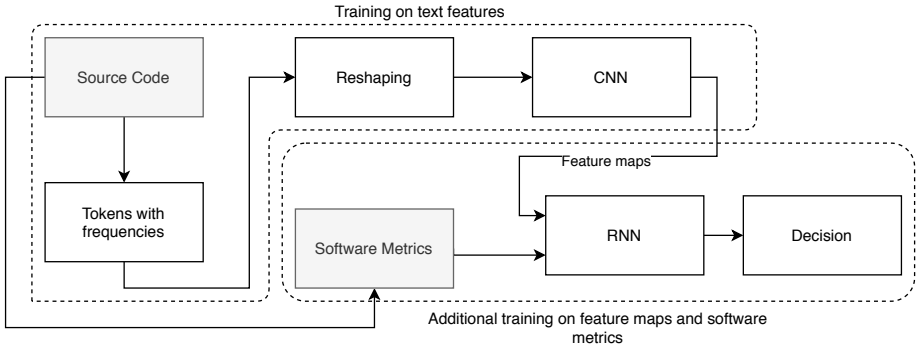
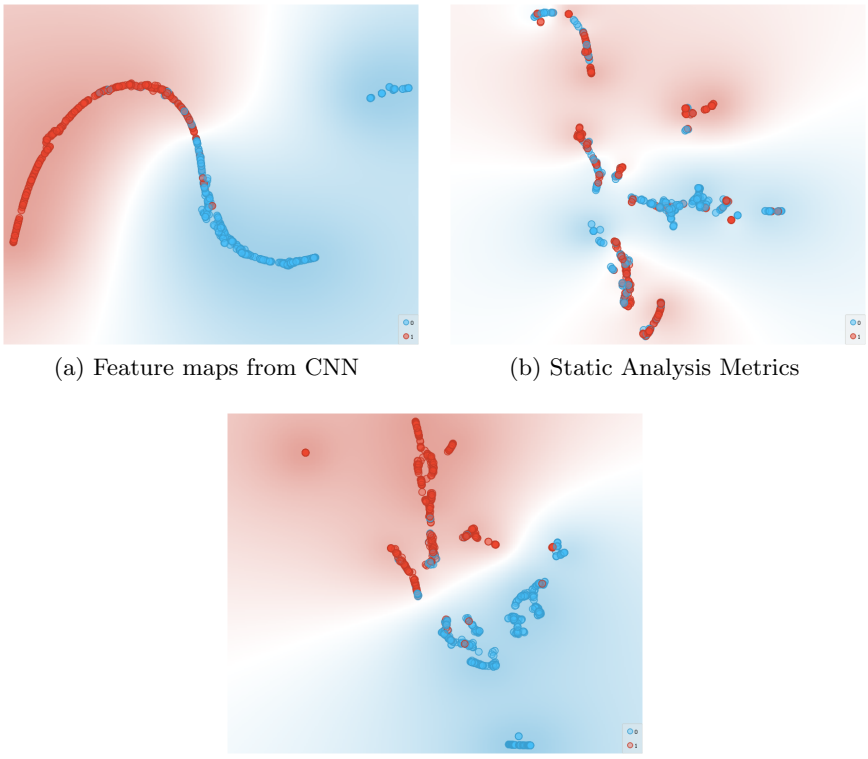


Fig. 1: The workflow of a software vulnerability prediction system based on both text mining features and metrics generated from a static analysis tool using Convolutional Neural Networks and Random Neural Networks

5 Experimental Results

We first examine the generated dataset using the t-SNE algorithm as presented in Figure 2. It is clearly visible that the algorithm can find a local structure in high-dimensional data in both the feature maps generated using a CNN and the software metrics (all the metrics have been taken under consideration). Although it is harder to divide features generated in the Static Analysis tool, a local structure exists there and it can suggest that additional information can be extracted by adding these features to the overall analysis. Using both sets of features, an almost linear separation of vulnerable and non-vulnerable instances can be achieved.



(a) Feature maps from CNN

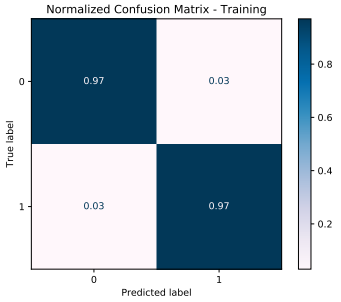
(b) Static Analysis Metrics

(c) Feature maps from CNN + Static Analysis Metrics

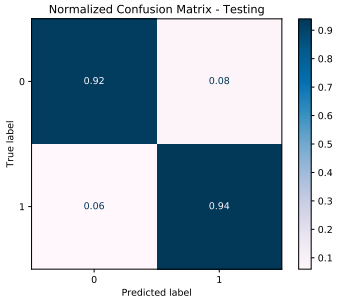
Fig. 2: t-SNE visualization of vulnerable and not vulnerable classes on the basis of different types of features

The best model created during the experiments for text mining, and the hybrid model both achieved 97 % training accuracy, while they achieved 93% and 94 % testing accuracy, respectively, for both the CNN model

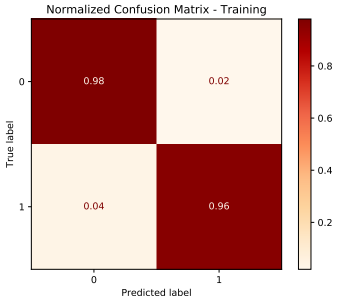
based on text mining and the hybrid model using the RNN. The results for the text mining model and the hybrid model are shown in the form of normalized confusion matrices in Figure 3.
Although the Hybrid model achieves 1 % lower recall or True Positive Rate for the training set, the value is this same for the testing set for both the text mining model and the hybrid model. Using a model that utilizes both feature maps obtained using CNN and metrics obtained from SonarQube static code analyzer, we managed to decrease the False Positive Rate for both training and testing data by 1%. We notice that the model is good at generalizing and despite the fact that data in the testing set is distributed differently than in the training set (since we used down-sampling to balance the vulnerable and non-vulnerable instances) it achieves satisfactory results in terms of Recall and False Positive Rate.



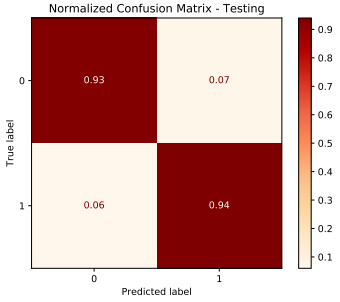
(a) Training - text mining model



(b) Testing - text mining model



(c) Training - joined model



(d) Testing - joined model

Fig. 3: Normalized confusion matrices presenting the best results achieved by the text mining model and the hybrid model

6 Conclusions and Future Work

In this paper, we have exploited the successful usage in a wide variety of applications of two neural network models with their respective learning algorithms, namely the RNN and the CNN, to address the challenging issue of Software Vulnerability Prediction. In our proposed approach, text data processing and dimensionality reduction is accomplished by a small CNN, while an RNN operates as a bonding model for both parts of the analysis. Feature maps obtained from a CNN hidden layer are given to the RNN as input, while additional features obtained from metrics generated by static analysis, help to improve the correct determination of vulnerabilities and to reduce the false alarms.

The analysis performed using the t-SNE algorithm have shown that high-dimensional features used for training the neural networks contain a local structure that can be used for Software Vulnerability Prediction. Although this local structure obtained using features from a Static Code Analyzer is not as marked as the features obtained from the text mining, it is reasonable to assume that the information contained in these features can help improve the approaches based on text mining. It also appears that the biggest challenge in many Software Vulnerability Prediction techniques and Static Analysis tools is a significant number of false positives. Our experiments show that by using text features as well as metrics generated by a static code analyzer, false positives can be reduced.

To fully evaluate the proposed method, more tests need to be performed. More complex neural network architectures may also provide us with even better results. Such models should be tested on larger datasets containing different types of vulnerabilities, possibly leading to a general Vulnerability Prediction approach. As an alternative, the use of multiple predictors that address different types of vulnerabilities, rather than a tool that predicts the occurrence of vulnerabilities in general, could be implemented.

In future work we will also consider a more exhaustive set of statistical features that are generated by a static analyzer, with features extracted by different feature selection methods. Different Static Code Analyzers could also be used to identify the best features, which can then be compared and evaluated with respect to their usefulness in building Software Vulnerability Prediction tools.

References

1. Ajit, A., Acharya, K., Samanta, A.: A review of convolutional neural networks. In: 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India. pp. 1–5. IEEEXpress (2020). https://doi.org/10.1109/ic-ETITE47903.2020.049
2. Alves, H., Fonseca, B., Antunes, N.: Software metrics and security vulnerabilities: dataset and exploratory study. In: 2016 12th European Dependable Computing Conference (EDCC). pp. 37–44 (2016)

3. Amin, A., Eldessouki, A., Magdy, M.T., Abdeen, N., Hindy, H., Hegazy, I.: Androshield: Automated android applications vulnerability detection, a hybrid static and dynamic analysis approach. Information **10**(10), 326 (2019)
4. Brun, O., Yin, Y., Gelenbe, E.: Deep learning with dense random neural network for detecting attacks against IoT-connected home environments. Procedia Computer Science **134**, 458 – 463 (2018)
5. Brun, O., Wang, L., Gelenbe, E.: Big data for autonomic intercontinental overlays. IEEE Journal on Selected Areas in Communications **34**(3), 575–583 (2016)
6. Catal, C., Akbulut, A., Ekenoglu, E., Alemdaroglu, M.: Development of a software vulnerability prediction web service based on artificial neural networks. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 59–67. Springer (2017)
7. Computer Emergency Response Team Coordination Center. [online], available: https://www.kb.cert.org/vuls/ [Accessed: 2020-08-05]
8. Chowdhury, I., Zulkernine, M.: Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. Journal of Systems Architecture **57**(3), 294–313 (2011)
9. Cisco 2019 Annual Report. [online] (Cisco 2019), available: https://www.cisco.com/c/dam/en_us/about/annual-report/cisco-annual-report-2019.pdf [Accessed: 2020-08-05]
10. Cisco Cybersecurity Series 2019. Consumer Privacy Survey. [online] (Cisco 2019), available: https://www.cisco.com/c/dam/en_us/about/annual-report/cisco-annual-report-2019.pdf [Accessed: 2020-08-05]
11. 2019 CWE Top 25 Most Dangerous Software Errors. [online], available: https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html [Accessed: 2020-08-05]
12. Dam, H.K., Tran, T., Pham, T., Ng, S.W., Grundy, J., Ghose, A.: Automatic feature learning for vulnerability prediction. arXiv preprint arXiv:1708.02368 (2017)
13. Ding, Y., Duan, R., Li, L., Cheng, Y., Zhang, Y., Chen, T., Wei, T., Wang, H.: Poster: Rust sgx sdk: Towards memory safety in intel sgx enclave. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2491–2493 (2017)
14. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **1**(2), 223–259 (2006)
15. Du, X., Chen, B., Li, Y., Guo, J., Zhou, Y., Liu, Y., Jiang, Y.: Leopard: Identifying vulnerable code for vulnerability assessment through program metrics. In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). pp. 60–71 (2019)
16. Enabling Open Innovation & Collaboration — The Eclipse Foundation. [online], available: https://www.eclipse.org/ [Accessed: 2020-08-05]
17. Evmorfos, S., Vlachodimitropoulos, G., Bakalos, N., Gelenbe, E.: Neural network architectures for the detection of syn

flood attacks in iot systems. In: Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments. pp. 1–4. No. 69, ACM; https://doi.org/10.1145/3389189.3398000 (2020)
18. Fourneau, J.M., Gelenbe, E.: G-networks with adders. Future Internet **9**(3), 34 (2017)
19. Francois, F., Gelenbe, E.: Optimizing secure sdn-enabled inter-data centre overlay networks through cognitive routing. In: 2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). pp. 283–288. IEEE (2016)
20. Francois, F., Gelenbe, E.: Towards a cognitive routing engine for software defined networks. In: 2016 IEEE International Conference on Communications (ICC). pp. 1–6. IEEE (2016)
21. Frohlich, P., Gelenbe, E.: Optimal fog services placement in sdn iot network using random neural networks and cognitive network map. In: Fröhlich, P., and E. Gelenbe, The 19th International Conference on Artificial Intelligence and Soft Computing, Zakopane, PL. Springer (2020)
22. Frohlich, P., Gelenbe, E., Nowak, M.P.: Smart sdn management of fog services. In: GIOTS 2020: Global IoT Summit 2020, IEEE Communications Society, 1-5 June 2020, Dubin, Ireland. TechRxiv (2020)
23. Gegick, M., Williams, L.: Toward the use of automated static analysis alerts for early identification of vulnerability-and attack-prone components. In: Second International Conference on Internet Monitoring and Protection (ICIMP 2007). pp. 18–18. IEEE (2007)
24. Gelenbe, E.: Learning in the recurrent random neural network. Neural Computation **5**, 154–164 (1993)
25. Gelenbe, E., Feng, Y., Krishnan, K.R.R.: Neural network methods for volumetric magnetic resonance imaging of the human brain. Proceedings of the IEEE **84**(10), 1488–1496 (1996)
26. Gelenbe, E.: Random neural networks with negative and positive signals and product form solution. Neural computation **1**(4), 502–510 (1989)
27. Gelenbe, E.: Stability of the random neural network model. Neural computation **2**(2), 239–247 (1990)
28. Gelenbe, E.: G-networks with signals and batch removal. Probability in the Engineering and Informational Sciences **7**(3), 335–342 (1993)
29. Gelenbe, E.: Steps toward self-aware networks. Communications of the ACM **52**(7), 66–75 (2009)
30. Gelenbe, E.: Machine learning for network routing. In: 2020 9th Mediterranean Conference on Embedded Computing (MECO). pp. 1–1. IEEE (2020)
31. Gelenbe, E., Domanska, J., Frohlich, P., Nowak, M., Nowak, S.: Self-aware networks that optimize security, qos and energy. Proceedings of the IEEE, accepted for publication **108**(7) (2020)
32. Gelenbe, E., Fourneau, J.M.: Random neural networks with multiple classes of signals. Neural computation **11**(4), 953–963 (1999)

33. Gelenbe, E., Hussain, K., Kaptan, V.: Simulating autonomous agents in augmented reality. Journal of Systems and Software **74**(2), 255–268 (2005)
34. Gelenbe, E., Hussain, K., Kaptan, V.: Simulating autonomous agents in augmented reality. Journal of Systems and Software **74**(3), 255–268 (2005)
35. Gelenbe, E., Koçak, T.: Area-based results for mine detection. IEEE transactions on geoscience and remote sensing **38**(1), 12–24 (2000)
36. Gelenbe, E., Lent, R., Nunez, A.: Self-aware networks and qos. Proceedings of the IEEE **92**(9), 1478–1489 (2004)
37. Gelenbe, E., Lent, R., Xu, Z.: Measurement and performance of a cognitive packet network. Computer Networks **37**(6), 691–701 (2001)
38. Gelenbe, E., Liu, P., Laine, J.: Genetic algorithms for route discovery. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **36**(6), 1247–1254 (2006)
39. Gelenbe, E., Loukas, G.: A self-aware approach to denial of service defence. Computer Networks **51**(5), 1299–1314 (2007)
40. Gelenbe, E., Montuori, A., Nunez, A., Lent, R., Xu, Z.: Experiments with qos driven learning packet networks. Internet Process Coordination pp. 215–233 (2020)
41. Gelenbe, E., Sungur, M., Cramer, C.: Learning random networks for compression of still and moving images. JPL: A Decade of Neural Networks; Practical Applications and Prospects pp. 171–189 (1994)
42. Gelenbe, E., Wu, F.J.: Large scale simulation for human evacuation and rescue. Computers & Mathematics with Applications **64**(12), 3869–3880 (2012)
43. Gelenbe, E., Wu, F.J.: Future research on cyber-physical emergency management systems. Future Internet **5**(3), 336–354 (2013)
44. Gelenbe, E., Yin, Y.: Deep learning with dense random neural networks. In: International Conference on Man–Machine Interactions. pp. 3–18. Springer, Cham (2017)
45. He, Z., Chen, W., Li, Z., Zhang, W., Shao, H., Zhang, M.: Syntax-aware entity representations for neural relation extraction. Artificial Intelligence **275**, 602–617 (2019)
46. Hussain, K., Moussa, G.: On-road vehicle classification based on random neural network and bag-of-visual words. Probability in the Engineering and Informational Sciences **30**(3), 403–412 (2016)
47. Hussain, K., Yousef, M., Gelenbe, E.: Accurate and energy-efficient classification with spiking random neural network. Probability in the Engineering and Informational Sciences p. 1–11 (2019)
48. Hussain, K.F., Radwan, E., Moussa, G.S.: Augmented reality experiment: Drivers' behavior at an unsignalized intersection. IEEE Transactions on Intelligent Transportation Systems **14**(2), 608–617 (2013)
49. IntelliJ IDEA: The Java IDE for Professional Developers by JetBrains. [online], available: https://www.jetbrains.com/idea/ [Accessed: 2020-08-05]
50. Jackson, K.A., Bennett, B.T.: Locating SQL injection vulnerabilities in java byte code using natural language techniques. In: SoutheastCon 2018. pp. 1–5 (2018)

51. Jimenez, M., Papadakis, M., Le Traon, Y.: Vulnerability prediction models: A case study on the linux kernel. In: 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM). pp. 1–10 (2016)
52. Kaptan, V., Gelenbe, E.: Fusing terrain and goals: agent control in urban environments. In: Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006. vol. 6242, p. 624208. International Society for Optics and Photonics (2006)
53. Keras Functional API. [online], available: https://keras.io/guides/functional_api/ [Accessed: 2020-08-06]
54. Klees, G., Ruef, A., Cooper, B., Wei, S., Hicks, M.: Evaluating fuzz testing. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 2123–2138 (2018)
55. Kobak, D., Berens, P.: The art of using t-SNE for single-cell transcriptomics. Nature communications **10**(1), 1–14 (2019)
56. Kudjo, P.K., Chen, J., Zhou, M., Mensah, S., Huang, R.: Improving the accuracy of vulnerability report classification using term frequency-inverse gravity moment. In: 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS). pp. 248–259 (2019)
57. Lan, M., Zhang, Y., Zhang, L., Du, B.: Global context based automatic road segmentation via dilated convolutional neural network. Information Sciences **535**, 156 – 171 (2020)
58. Li, Z., Yang, W., Peng, S., Liu, F.: A survey of convolutional neural networks: Analysis, applications, and prospects (2020)
59. Li, Z., Zou, D., Xu, S., Ou, X., Jin, H., Wang, S., Deng, Z., Zhong, Y.: Vuldeepecker: A deep learning-based system for vulnerability detection. arXiv preprint arXiv:1801.01681 (2018)
60. Liu, J., Li, Q., Yang, H., Han, Y., Jiang, S., Chen, W.: Sequence fault diagnosis for pemfc water management subsystem using deep learning with t-sne. IEEE Access **7**, 92009–92019 (2019)
61. Moshtari, S., Sami, A., Azimi, M.: Using complexity metrics to improve software security. Computer Fraud & Security **2013**(5), 8–17 (2013)
62. Nafi, K.W., Roy, B., Roy, C.K., Schneider, K.A.: A universal cross language software similarity detector for open source software categorization. Journal of Systems and Software **162**, 110491 (2020)
63. Neuhaus, S., Zimmermann, T., Holler, C., Zeller, A.: Predicting vulnerable software components. In: Proceedings of the 14th ACM conference on Computer and communications security. pp. 529–540 (2007)
64. Öke, G., Loukas, G.: A denial of service detector based on maximum likelihood detection and the random neural network. The Computer Journal **50**(6), 717–727 (2007)
65. Oke, G., Loukas, G., Gelenbe, E.: Detecting denial of service attacks with bayesian classifiers and the random neural network. In: 2007 IEEE International Fuzzy Systems Conference. pp. 1–6. IEEE (2007)
66. Open Web Application Security Project (OWASP). [online], available: https://owasp.org/ [Accessed: 2020-08-05]

67. OWASP Secure Coding Practices Quick Reference Guide. [online], available: https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v1.pdf [Accessed: 2020-08-05]
68. OWASP Top Ten. [online], available: https://owasp.org/www-project-top-ten/ [Accessed: 2020-08-05]
69. Pang, Y., Xue, X., Wang, H.: Predicting vulnerable software components through deep neural network. In: Proceedings of the 2017 International Conference on Deep Learning Technologies. pp. 6–10 (2017)
70. Saeed, A., Ahmadinia, A., Javed, A., Larijani, H.: Intelligent intrusion detection in low-power IoTs. ACM Transactions on Internet Technology (TOIT) **16**(4), 1–25 (2016)
71. Salka, C.: Programming languages and systems security. IEEE security & privacy **3**(3), 80–83 (2005)
72. Information Security Training - SANS Cyber Security Certifications & Research. [online], available: https://www.sans.org/ [Accessed: 2020-08-05]
73. Scalabrino, S., Grano, G., Di Nucci, D., Oliveto, R., De Lucia, A.: Search-based testing of procedural programs: Iterative single-target or multi-target approach? In: International Symposium on Search Based Software Engineering. pp. 64–79 (2016)
74. Scandariato, R., Walden, J., Hovsepyan, A., Joosen, W.: Predicting vulnerable software components via text mining. IEEE Transactions on Software Engineering **40**(10), 993–1006 (2014)
75. Serrano, W., Gelenbe, E.: Deep learning clusters in the cognitive packet network. Neurocomputing **396**, 406–428 (2020)
76. Serrano, W., Gelenbe, E., Yin, Y.: The random neural network with deep learning clusters in smart search. Neurocomputing **396**, 394–405 (2020)
77. Sherriff, M., Heckman, S.S., Lake, M., Williams, L.: Identifying fault-prone files using static analysis alerts through singular value decomposition. In: Proceedings of the 2007 conference of the center for advanced studies on Collaborative research. pp. 276–279 (2007)
78. Shin, Y., Meneely, A., Williams, L., Osborne, J.A.: Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. IEEE transactions on software engineering **37**(6), 772–787 (2010)
79. Siavvas, M., Gelenbe, E., Kehagias, D., Tzovaras, D.: Static analysis-based approaches for secure software development. In: International ISCIS Security Workshop. pp. 142–157 (2018)
80. SonarQube. [online], available: https://www.sonarqube.org/ [Accessed: 2020-08-03]
81. Tang, Y., Zhao, F., Yang, Y., Lu, H., Zhou, Y., Xu, B.: Predicting vulnerable components via text mining or software metrics? an effort-aware perspective. In: 2015 IEEE International Conference on Software Quality, Reliability and Security. pp. 27–36 (2015)
82. Thaseen, I.S., Kumar, C.A.: Intrusion detection model using fusion of chi-square feature selection and multi class svm. Journal of King Saud University-Computer and Information Sciences **29**(4), 462–472 (2017)

83. Timotheou, S.: A novel weight initialization method for the random neural network. Neurocomputing **73**(1), 160 – 168 (2009)
84. Timotheou, S.: The random neural network: A survey. The Computer Journal **53**(3), 251–267 (2010)
85. Veracode: State of software security. Tech. rep. (2016)
86. Veracode: State of software security Volume 9. Tech. rep. (2018)
87. Veracode. [online], available: https://www.veracode.com/ [Accessed: 2020-08-05]
88. Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio. [online], available: https://visualstudio.microsoft.com/ [Accessed: 2020-08-05]
89. Walden, J., Stuckman, J., Scandariato, R.: Predicting vulnerable components: Software metrics vs text mining. In: 2014 IEEE 25th international symposium on software reliability engineering. pp. 23–33 (2014)
90. Wang, L., Brun, O., Gelenbe, E.: Adaptive workload distribution for local and remote clouds. In: 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 003984–003988. IEEE (2016)
91. Wang, L., Gelenbe, E.: Real-time traffic over the cognitive packet network. In: International Conference on Computer Networks. pp. 3–21. Springer (2016)
92. Wang, X., Sun, J., Chen, Z., Zhang, P., Wang, J., Lin, Y.: Towards optimal concolic testing. In: Proceedings of the 40th International Conference on Software Engineering. pp. 291–302 (2018)
93. Wu, J., Liu, X., Hu, X., Zhu, J.: Popmnet: Generating structured pop music melodies using neural networks. Artificial Intelligence p. 103303 (2020)
94. Wu, Y., Ma, Y., Liu, J., Du, J., Xing, L.: Self-attention convolutional neural network for improved mr image reconstruction. Information Sciences **490**, 317–328 (2019)
95. Yin, Y., Gelenbe, E.: A classifier based on spiking random neural network function approximator (2018)
96. Yin, Y., Wang, L., Gelenbe, E.: Multi-layer neural networks for quality of service oriented server-state classification in cloud servers. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 1623–1627. IEEE (2017)
97. Yin, Y.: Deep learning with the random neural network and its applications. ArXiv **abs/1810.08653** (2018)
98. Yin, Y., Gelenbe, E.: Deep learning in multi-layer architectures of dense nuclei. arXiv preprint arXiv:1609.07160 (2016)
99. Zhang, M., de Carnavalet, X.d.C., Wang, L., Ragab, A.: Large-scale empirical study of important features indicative of discovered vulnerabilities to assess application security. IEEE Transactions on Information Forensics and Security **14**(9), 2315–2330 (2019)
100. Zhang, Y., Lo, D., Xia, X., Xu, B., Sun, J., Li, S.: Combining software metrics and text features for vulnerable file prediction. In: 2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS). pp. 40–49 (2015)