System Wide Vulnerability and Trust in Multi-Component Communication System Software

Erol Gelenbe *FIEEE*, Mert Nakıp and Miltiadis Siavvas

***Abstract*—In the software rich environment of 6G, systems will be surrounded by edge devices that support distributed software systems which are critical to operations. Such systems may also be subject to frequent updates or uploads of individual software components. Trust in such systems will therefore depend on our ability to rapidly ensure that such software is not vulnerable to cyberattacks or malicious compromises. Thus this paper presents a novel System-Wide Vulnerability Assessment (SWVA) framework based on Machine Learning, that can be frequently activated to assess the vulnerability of interconnected software components over edge systems. The performance of the SWVA framework is illustrated by assessing the vulnerability of 13 versions of a real-world 11 component software system, and comparing the ARNN results against the well-known ML models MLP, KNN, and Lasso. The results show the superior performance of SWVA, offering over 85% median accuracy and good scalability as the number of connected software components increases.**

***Index Terms*—Vulnerability Assessment, Trust in 6G, Interconnected Software, Associated Random Neural Network, Deep Learning**

I. Introduction

SECURE and trustworthy communications are difficult and expensive to build, but the consequences of not meeting such requirements are extremely high [1]. A recent industry position paper [2] entitled "6G orchestration and automation: A system software view", stresses the software dominance in 6G with the term "network as code" and discusses the role of different software components and APIs in 6G.

Since software will be pervasive across 6G systems, there is great need to enhance trust by automatically assessing multi-component software to detect and reduce security vulnerabilities that can arise during software development [3], and also test overall multi-component 6G software systems as new components and APIs are added to the system. While vulnerability assessment of single software components is well developed, the multi-component case is new.

Indeed, in a recent preparatory document for the 2024 6G Summit [4] four of the seven security threats that were listed for 6G are primarily related to software issues, namely AI and machine learning, including algorithmic errors and systemic failures, data privacy and security, autonomous technology governed by multi-component software systems, and the interacting protocols needed to ensure service and resource availability for ultra-reliable low latency communications.

Despite the use of Static Code Analyzers [5], testing techniques and dynamic checks [6], as much as 85% of software applications may contain vulnerabilities and more than 10% may contain security breaches that undermine trust [7]. Conventional software vulnerability assessment is slow and tedious [8], and machine learning can efficiently automate this problem for large scale 6G software, where the vulnerability of each software component can –directly or indirectly– affect the vulnerability of many other components. The availability of a fast and accurate tool for multi-component vulnerability assessment, can greatly *enhance* the trust of users regarding 6G software.

Thus, to address this major challenge, we propose a novel System-Wide Vulnerability Assessment (SWVA) framework based on the Associated Random Neural Network (ARNN) [9], which offers a machine learning approach to vulnerability testing for multi-component systems composed of n separate software components $\mathbf{S} = \{S_1, \ ... \ , S_n\}$, which are interconnected to each other.

A. Prior Research

While sofware vulnerability assessment is well developed for single components, the study of multi-component system vulnerability, which is needed to assess large scale multi-component communicaton software that will be needed for future systems and 6G,

Erol Gelenbe is with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences (PAN), Gliwice, Poland, e-mail: seg@iitis.pl; he is also with Université Côte d'Azur CNRS I3S, Nice, France, and King's College London, UK

Mert Nakıp is with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences (PAN), Gliwice, Poland, email: mnakip@iitis.pl)

Miltiadis Siavvas is with CERTH-ITI, Thermi, Thessaloniki 570 01, Greece; e-mail: siavvasm@iti.gr

has been limited. Indeed, state-of-the-art vulnerability prediction techniques [10] detect the potential presence of vulnerabilities in a software component based only on information retrieved from each component, neglecting critical information about the internal interconnections between components, that affect the system as a whole.

Metrics that assess security flaws for individual software components were examined by several authors including [11], and text mining methods with code tokens are then used as input in training ML models. These tokens are extracted from source code, and tools such as the VulDeePecker system [12] converts each token into a vector to identify vulnerabilities through deep learning. In Devign [13], a graph neural network combines several techniques, and natural language processing (NLP) is used in VulDeBERT for vulnerability prediction [14].

In contrast, the SWVA framework proposed in this paper, assesses the vulnerability of all interconnected components, considering both the data from individual components, together with data about their interconnections. To the best of the authors' knowledge, this is the first systematic approach for a system-wide vulnerability assessment of a software system made up of several interconnected components. Due to its previous high performance for multi-node Botnet detection [9], in this paper we exploit the recurrent Associated RNN (ARNN) architecture for multi-component SWVA, which is a recurrent Random Neural Network (RNN), that was originally invented to mimic the spiking behavior of mammalian brain neurons, and its deep learning algorithm can be used both for feed-forward and recurrent neural networks [15].

B. The System-Wide Vulnerability Assessment (SWVA) Framework

As input, the SWVA method uses:

- The empirically estimated [16] individual vulnerability $0 \leq V_i \leq 1$ of each component S_i, where $V_i = 1$ indicates that S_i is vulnerable, while $V_i = 0$ indicates that it is fully safe, and
- The directed n-node graph representing the connections between components with the binary $n \times n$ adjacency matrix A.

Figure 1 shows how SWVA predicts the updated vulnerability Likelihood Level of each system component S_i, resulting from the inputs V_i, and from A, using a deep learning algorithm. In this paper, we present this approach and illustrate it on a real-world software system consisting of 11 components, with 13 historically different versions, and different possible interconnection matrices A.

The SWVA framework is evaluated and compared against the well-known ML models MLP, KNN, and Lasso, for assessing the vulnerability of 169 distinct instances of a real-world 11 component software system, where each component can have 13 different versions. The results that were obtained have demonstrated the superior performance of SWVA as compared to the other methods, achieveing above 85% median accuracy and good scalability, as detailed in Figure 5.

II. Problem Statement & System Design

The SWVA framework that is illustrated in Figure 1 displays the overall structure of the proposed approach. At the bottom of the right-hand side of the figure, the particular structure of *two opposing neurons X_i and x_i of the ARNN* is shown, as they decide about the vulnerability of component S_i.

The neurons X_i and x_i of the ARNN "defend" two opposing views: X_i whose state value is Q_i "claims" that S_i is vulnerable, while x_i makes the opposite claim through its state q_i. Note that Q_i and q_i are the probabilities that the corresponding neurons are activated.

Both X_i and x_i receive the local vulnerability V_i information as input, but they are connected to all the other pairs X_j and x_j through synaptic weights, using the connection matrix A. The ARNN is trained using the ARNN deep learning algorithm [9] with real data from ground truth data sets about the multi-component software system.

As shown in this figure, we consider each software component S_i, which is part of a system $\mathbf{S}$ comprised of n components, whose *local* (individual) vulnerability V_i is known empirically. The local vulnerability V_i results from the presence of code constructs within S_i that cause vulnerabilities and is predicted via Local Vulnerability Prediction methods that analyze the source code [10].

III. System-Wide Vulnerability Assessment

Since each neuron X_i and x_i in the ARNN node is represented by its internal state Q_i and q_i, these quantities are linked to each other by a non-linear system of $2n$ equations, resulting from the detailed developments of the RNN model [15]:

$$Q_i = min[1, \frac{V_i + \sum_{j=1}^{n} W_{ji}^{+} Q_j}{1 - V_i + \sum_{j=1}^{n} [W_{ij}^{+} + W_{ij}^{-}] + \sum_{j=1}^{n} w_{ji}^{-} q_j}],$$

and

$$q_i = min[1, \frac{1 - V_i + \sum_{j=1}^{n} w_{ji}^{+} q_j}{V_i + \sum_{j=1}^{n} [w_{ij}^{+} + w_{ij}^{-}] + \sum_{j=1}^{n} W_{ji}^{-} Q_j}].$$

Here, W_{ij}^{+} and W_{ij}^{-} are the connection weights from X_i to neurons X_j and x_j, and w_{ij}^{+} and w_{ij}^{-} are the

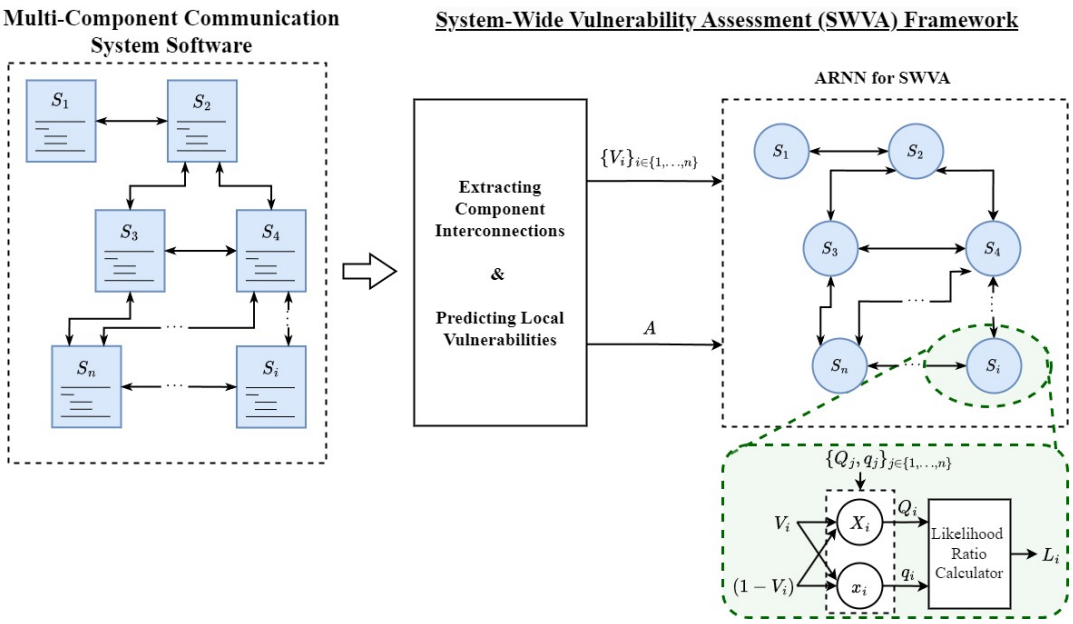

Fig. 1: The SWVA method inputs the vulnerabilities V_i of each of the components, which are extracted from the lexicographic analysis of each software component with existing methods. SWVA also uses the information regarding the inter-component interactions which are reprsented by the adjacency matrix A. It then uses the ARNN to learn from multiple instance and metrics associated with these components. Thus, SWVA learns from prior ground truth that includes real software component vulnerability and interconnection data. Then, for a given, hitherto unseen, real multi-component software system, it inputs the local prior vulnerabilities, and the interconnect structure, to output an accurate prediction of the updated Likelihood Level L_i for each of its components.

connection weights from x_i to neurons x_j and X_j, of node j. These weights are selected via deep learning [9] based on datasets that reflect the vulnerability assessment of examples of real multi-component software systems.

The Likelihood Ratio that component S_i is vulnerable in the presence of its connections with the other components, will be larger than one simply when Q_i is larger than q_i. If a threshold value different from one is used, similarly simple rules based on the Q_i and q_i are obtained, for deciding whether the connected components are vulnerable.

A. Experiments with Real-World Software

In order to evaluate the performance of the proposed SWVA framework, we consider a real-world software system, the GitHubCrawler described in [17], and collect raw vulnerability data using the methods in [10]. During the data collection from GitHubCrawler system, we first identify the interconnections between software components. Then, for each software component in a given version, we utilize a local vulnerability detector to compute inputs and the ground truth. In particular, the GitHubCrawler, has 13 committed versions with 11 interconnected software components, which are independent of the operating system. As shown in Figure 1, the interconnections between all components are extracted using the well-known Java Dependency Viewer by Eclipse.

In order to construct the dataset for the performance evaluation of SWVA, we first used the the Dependency Viewer for each of the 13 versions of the GitHubCrawler system in [17] to obtain the interconnections between components. The interconnection graphs of versions $v8 - v13$ of the GitHubCrawler are shown in Figure 2, yielding the connection matrix A^v of each software version v.

Then, we collect a dataset of the components' local vulnerability predictions, using the vulnerability prediction of the IoTAC Software Security by Design Platform [10], based on text mining as discussed earlier. The approach uses 13 distinct vulnerability detection techniques indexed by z, and is run for all of the 13 distinct versions of the GitHubCrawler.

Thus, for each software component S_i of the GitHubCrawler, the source code of each version v of this is first parsed with each of the 13 different local vulnerability detection techniques, extracting word sequences, yielding the distinct vulnerability score $V_{v,i,z}$ for $z = 1$ to $z = 13$. Thus, from the 13 versions $v1$ to $v13$, and the 13 detection techniques, we obtain 169 different values

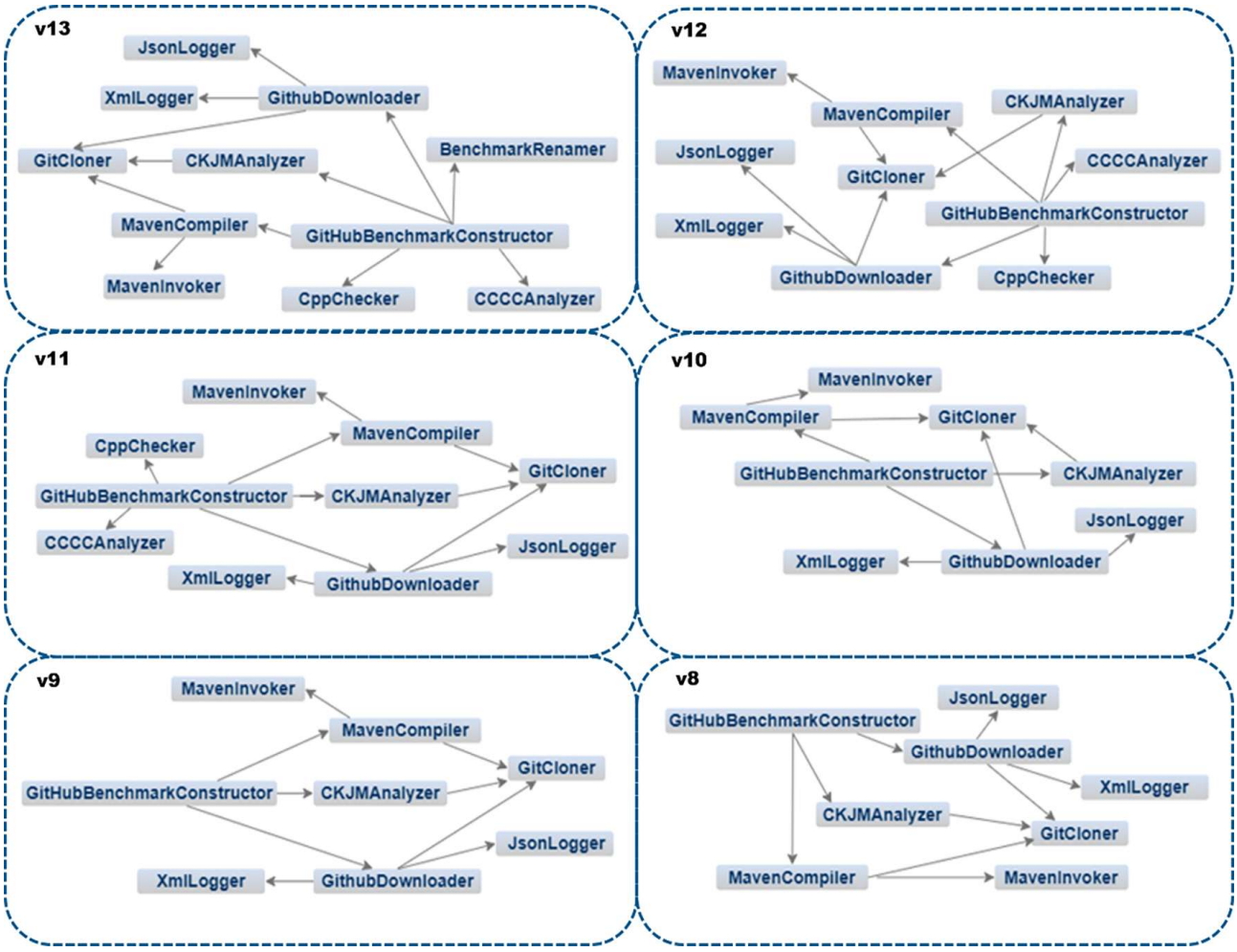

Fig. 2: The graphs with the interconnections between the components of the six latest versions of the GitHubCrawler software system [17], produced by the Dependency Viewer plugin.

$V_{v,i,z}$ for each component i.

Then, the **ground truth value of the Interconnected Vulnerability metric** $V_{v,i,z}^{I,k}$ is derived, according to three different criteria $k = Average,\ Max,\ Max - Std$. These differ from the local vulnerability due to the fact that they include the influence of the communication between neighbouring components in the connection graph represented by the incidence matrix A. Thus $V_{v,i,z}^{I,Average}$ is the average of the vulnerabilities of component i with that of its immediate neighbours, when we consider version v and detection technique z. Similarly, $V_{v,i,z}^{I,Max}$ is the maximum value of the vulnerabilities of the component with that of its immediate interconnected components. Finally, $V_{v,i,z}^{I,Max-Std}$ subtracts, from $V_{v,i,z}^{I,Max}$, the standard deviation of the vulnerability of the component and its neighbours, from the maximum value.

We evaluated the performance of the SWVA framework using the ARNN on the GitHubCrawler system for each of the three ground truth values, and compared the ARNN results against the well-known ML models, MLP, KNN, and Lasso. To this end, since we have 169 samples in total, we perform a 5-fold cross-validation to obtain generalizable results using a small-size dataset, that trains a given model with 80% of the dataset and tests it with the remaining 20%, and iteratively changes the training and test data, so as to provide results that reflect the prediction performance of the model in practice. We trained the ARNN for 100 epochs for each cross-validation fold, and used a threshold to fix the likelihood ratio in each case. The results are shown for just one software version $v13$ in Figure 3 showing that the method appears to work well for all the three different methods for selecting the ground truth.

B. Performance of the SWVA with the Averaged Ground Truth Vulnerability among Communicating Components

We first discuss the resulting performance of the ARNN-based SWVA for different versions of the software system with a variable number of components

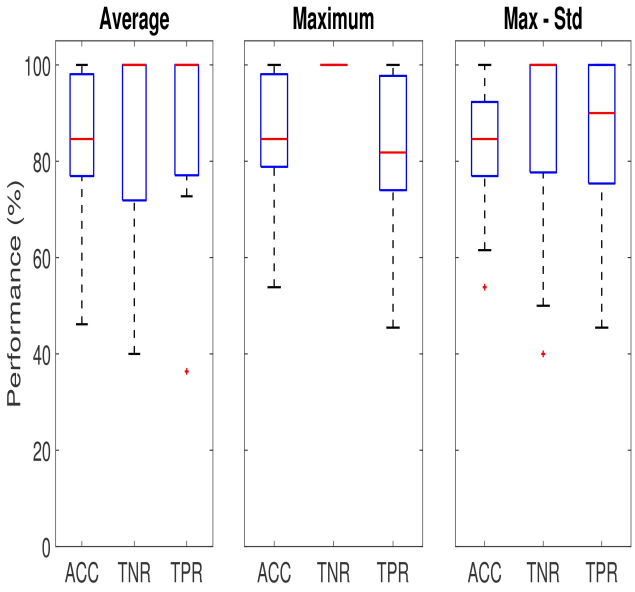

Fig. 3: Performance of the proposed ARNN-based SWVA framework for the latest version (v13) of the GitHubCrawler system using each of the three ground truth calculation methods, namely "Average", "Maximum", and "Maximum-Standard Deviation", with respect to percentage Accuracy, TNR, and TPR metrics.

based on the choice of "Average" as the technique for choosing the ground truth. In Figure 4, we present the percentage accuracy, TPR, and TNR of the ARNN-based SWVA as a boxplot presenting the performance statistics over all components.

We see that the median accuracy is above 85% for all versions except v3 and v4. We also see that the overall accuracy of ARNN-based SWVA tends to be higher when the total number of components n is larger. This comforts us in suggesting that SWVA is the right approach for large systems which are much harder to evaluate by human experts. In addition, our results show that although the median accuracy is high, the lower whisker is often between 45% to 60%.

The results in Figure 4 (middle) show that the median TPR equals 100% for all versions except v1 for which it is around 83%. Despite this high value, there are outliers with low whisker values around $45\% - 50\%$. We observe that the TPR performance is low mainly for components which are connected with only one other component, especially when the Likelihood Ratio ground truth is slightly above 1,while it is just over 1 (between 1 and 1.5) when the ARNN underestimates it, indicating that the given component is not vulnerable.

Next, Figure 4 (bottom) displays the TNR performance of ARNN-based SWVA for each version of the

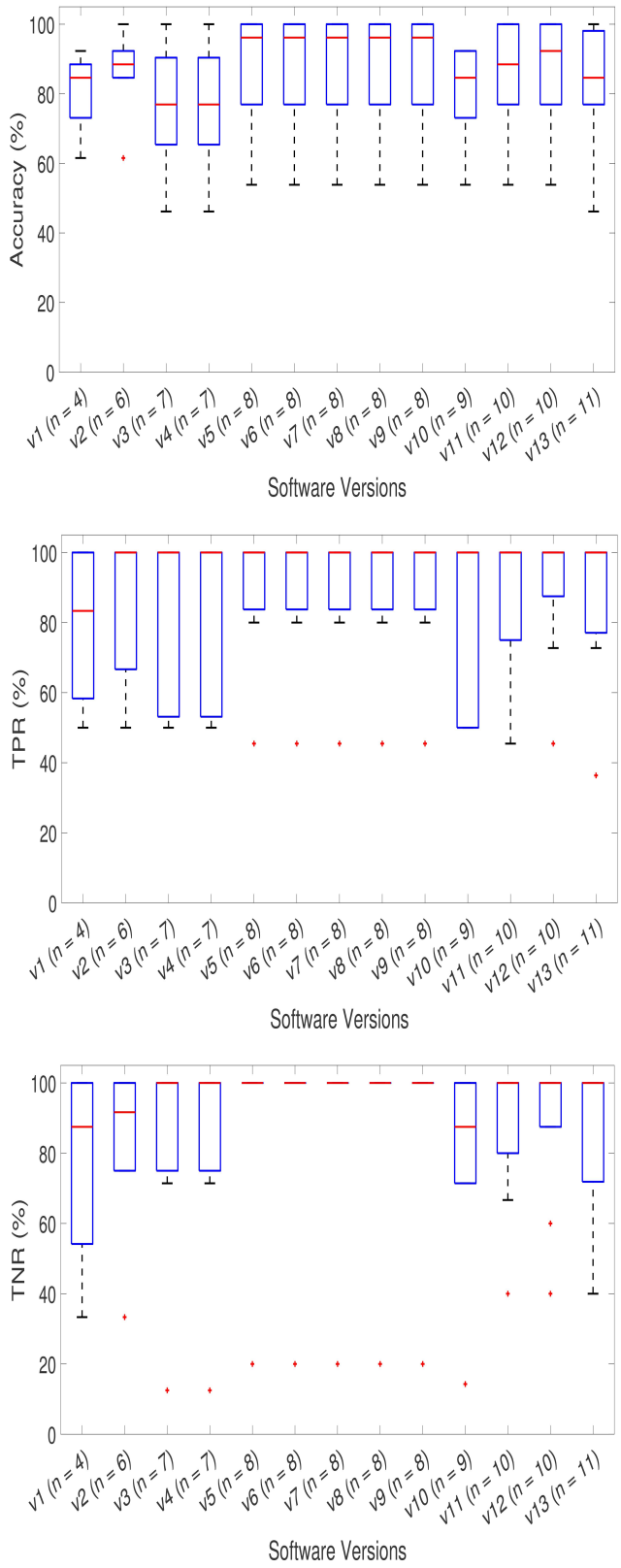

Fig. 4: Boxplot for the percentage Accuracy (top), TPR (middle) and TNR (bottom) performance of the ARNN model for all software versions

considered software. Our TNR results –similar to the TPR– show that the median performance equals 100% for the majority of versions, and it is between 85% to 90% for only v1, v2 and v10. In addition, for each software version, we see that there are one or two components for which the TNR is less than 60%.

We now compare the ARNN for SWVA, with the well-known models MLP, KNN, and Lasso which are implemented as follows:

- **MLP** is comprised of three hidden layers containing n neurons, with a Rectified Linear activation function at each layer, using Keras API in Python to minimize mean squared error.
- **KNN** uses equally weighted 4 neighbours, with 25% of training samples in each fold of cross-validation, implemented using the scikit-learn library in Python.
- **Lasso** is selected to represent the models with the ability of feature selection and used with the regularization coefficient of 0.1, and using the scikit-learn library.

The previously well-known model parameters, which are implementation-dependent, are set to the libraries' default values. Figure 5 compares the ARNN with the MLP, KNN, and Lasso learning models with respect to median Accuracy, TPR, and TNR, showing that the ARNN outperforms other models with 4% to 10% accuracy difference for most software versions, achieving the best accuracy for 8 out of 13 versions. Its performance is comparable to Lasso and KNN for versions v2, v3, v4, v10, and v13. At the bottom left, we see that the ARNN assesses the vulnerabilities significantly better than all other models, while the TPR difference between ARNN and the second-best model is around 15% on average. At the bottom right we see that the ARNN achieves very high TNR. In addition, among these ML, the MLP is the worst-performing model, while the validity of the TNR and KNN predictions significantly increases with the number of components, while that of Lasso and MLP do not change significantly.

C. Evaluating the SWVA Computation Time

Finally, the computation time of ARNN, the core algorithm in the SWVA, is analyzed. Figure 6 displays the average training time (top) and execution time per sample (bottom), with the standard deviation bars, as a function of the number of software components in the different system versions that were tested. Note that we have implemented the ARNN using TensorFlow on Python, and that these measurements were taken using the CPU of a PC with 32 GB of RAM and AMD Ryzen 7 3.70 GHz processor. In addition, the training of the ARNN is performed for 100 epochs with approximately 10 samples at each fold of cross-validation.

We see that while the training time is in the hundreds of seconds, the testing time is under ten milliseconds, so that trust in the 6G software can be frequently established, whenever new software versions or applications are uploaded, or when the interconnections between software components are changed.

IV. Conclusions

In an interconnected system of software components that may be distributed on different hardware units, when some individual components have vulnerabilities, it is difficult to determine how such local vulnerabilities may affect other components. Thus, multiple component software systems may reduce trust in future software intensive communication systems such as 6G.

To improve trust in such systems and automatically test their vulnerability, we have developed the System-Wide Vulnerability Assessment (SWVA) framework based on a specific ARNN model which attempts to infer whether some interconnected component may be affected by the security breaches of other components.

We have evaluated the performance of the SWVA framework for 13 different versions of GitHubCrawler, a real-world software system. We have also compared the results of SWVA with several well-known ML models. Our results indicate that the ARNN-based SWVA successfully assesses the system-wide vulnerability level and outperforms other ML models by achieving over 85% median accuracy. In addition, the ARNN is shown to have a reasonable computation time –with about $200\ s$ of training time and a very low, under $10\ ms$ execution time for an 11-component system. We also provide insight into its scalability by varying the number of software components.

In future work we will address an application environment where new software modules and APIs are being injected into the system, or updated, by a wide community of users. To this effect, we plan to adapt the SWVA framework to systems of interconnected components whose number and connections change dynamically over time, so that the structure of the ARNN, i.e., its neurons and connection weights, may evolve and learn from successive versions of a large software system.

Acknowledgement

This research has been supported by the European Commission Horizon Program DOSS Project, under Grant Agreement No. 101120270.

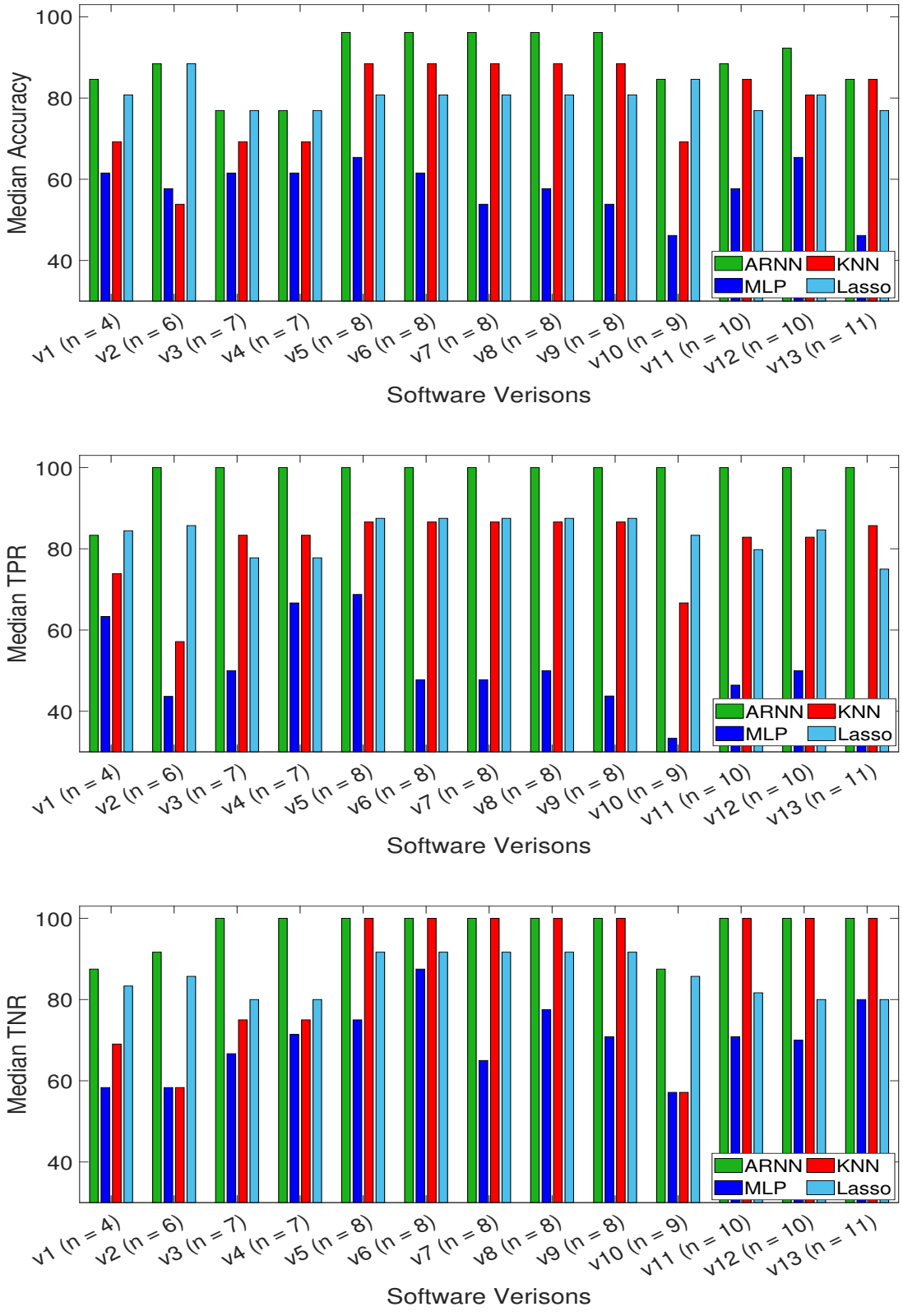

Fig. 5: Comparison of the different machine learning approaches for average Accuracy (top), TPR (middle), and TNR (bottom) for all the versions of the software.

BIOGRPAHIES

EROL GELENBE (IEEE Life Fellow), a Professor in the Institute of Theoretical and Applied Informatics of the Polish Academy of Sciences (IITIS-PAN), Gliwice

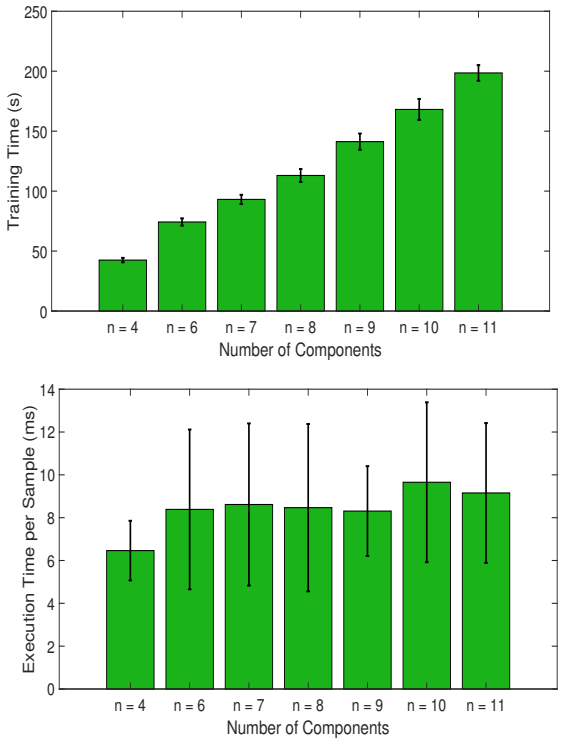

Fig. 6: Training (top) and testing (bottom) times of ARNN for increasing number of components n

Poland, is also a Visiting Professor at King's College, London, UK, and a Researcher at the CNRS I3S Laboratory, University of Nice Côte d'Azur. He previously held the Dennis Gabor Chair at Imperial College, London. His research addresses quantitative methods in the design and evaluation of computer systems and networks, including Cybersecurity, Sustainability and Quality of Service. He received the ACM-SIGMETRICS Life-Time Achievement Award (2008), and was elected a Fellow of several National Academies and professional societies.

MERT NAKIP (IEEE Student Member) is an Assistant Professor in the in the Institute of Theoretical and Applied Informatics of the Polish Academy of Sciences (IITIS-PAN), iGliwice Poland. He holds the Bachelor's and Master's Degree in Computer Engineering from Yaşar University,Bornova, Izmir, Turkey, and received the PhD from the Institute of Theoretical and Applied Informatics of the Polish Academy of Sciences in 2024. His research addresses Machine Learning based methods in Computer Science and Cybersecurity.

MILTIADIS SIAVVAS is a Post-doctoral Researcher at the Centre for Research and Technology Hellas (CERTH), in Thermi, Thessaloniki, Greece. He holds the Diploma in Electrical and Computer Engineering from the Aristotle University of Thessaloniki (2016), and the PhD in Electrical and Electronic Engineering, Imperial College London (2019). His research interests include the Cybersecurity, Sustainability and Efficiency of Large Scale Software Systems, and has been active in several European Union Horizon Research Projects.

REFERENCES

[1] "Cisco 2019 Annual Report," [online], 2019, available: https://www.cisco.com/c/dam/en_us/about/annual-report/cisco-annual-report-2019.pdf [Accessed: 2020-08-05].

[2] V. Raisanen and C. Vulkan, "6G orchestration and automation: A system software view," May 2024), url = https://onestore.nokia.com/asset/213955.

[3] J. Walden, J. Stuckman, and R. Scandariato, "Predicting vulnerable components: Software metrics vs text mining," in *2014 IEEE 25th international symposium on software reliability engineering*, 2014, pp. 23–33.

[4] "6G Network Dangers: 7 Tips for Mitigating Future Concerns." [Online]. Available: https://www.6gworld.com/blog/6g-network-dangers-7-tips-for-mitigating-future-concerns/

[5] "SonarQube," [online], 2024, available: https://www.sonarqube.org/ [Accessed: 2020-08-03].

[6] Y. Pang, X. Xue, and H. Wang, "Predicting vulnerable software components through deep neural network," in *Proceedings of the 2017 International Conference on Deep Learning Technologies*, 2017, pp. 6–10.

[7] Veracode, "State of software security Volume 9," Tech. Rep., 2018.

[8] K. A. Jackson and B. T. Bennett, "Locating SQL injection vulnerabilities in java byte code using natural language techniques," in *SoutheastCon 2018*, 2018, pp. 1–5.

[9] E. Gelenbe and M. Nakip, "Iot network cybersecurity assessment with the associated random neural network," *IEEE Access*, vol. 11, pp. 85 501–85 512, 2023.

[10] I. Kalouptsoglou, M. Siavvas, D. Kehagias, A. Chatzigeorgiou, and A. Ampatzoglou, "Examining the capacity of text mining and software metrics in vulnerability prediction," *Entropy*, vol. 24, no. 5, p. 651, 2022.

[11] M. Zagane, M. K. Abdi, and M. Alenezi, "Deep learning for software vulnerabilities detection using code metrics," *IEEE Access*, vol. 8, 2020.

[12] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, and Y. Zhong, "Vuldeepecker: A deep learning-based system for vulnerability detection," *arXiv preprint arXiv:1801.01681*, 2018.

[13] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," *arXiv preprint arXiv:1909.03496*, 2019.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[15] E. Gelenbe, "Learning in the recurrent random neural network," *Neural Computation*, vol. 5, no. 1, pp. 154–164, 1993.

[16] M. Zhang, X. d. C. de Carnavalet, L. Wang, and A. Ragab, "Large-scale empirical study of important features indicative of discovered vulnerabilities to assess application security," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 9, pp. 2315–2330, 2019.

[17] M. Siavvas. [Online]. Available: https://gitlab.seis.iti.gr/sdk4ed-wiki/wiki-home/-/wikis/GitHub-Crawler