Forecasting the Number of Bugs and Vulnerabilities in Software Components using Neural Network Models

– Technical report –

Ovidiu Cosma[1], Petrică Pop[1], Cosmin Sabo[1] and Laura Cosma[2]

Abstract The frequency of cyber attacks has been rising rapidly lately, which is a major concern. Because each attack exploits one or more vulnerabilities in the software components that make up the targeted system, the number of vulnerabilities is an indication of the level of security and trust that these components provide. In addition to vulnerabilities, the security of a component can also be affected by software bugs, as they can turn into weaknesses, which if exploited can become vulnerabilities. This paper presents a comparison of several types of neural networks for forecasting the number of software bugs and vulnerabilities that will be discovered for a software component in certain timeframe, in terms of accuracy, trainability and stability to configuration parameters.

Keywords: Security, Software vulnerabilities, Forecasting, Neural networks.

1 Introduction

The frequency of cyber-attacks has been rising rapidly lately, which is a major concern. Because each attack exploits one or more vulnerabilities of the software components that make up the targeted system, their number of vulnerabilities is an indication of the system level of trust. In addition to vulnerabilities, the security of a component can also be affected by software bugs, as they can turn into weaknesses, which if exploited can become vulnerabilities. To forecast the cyber-attacks, first, we need to know the trends of the vulnerabilities and bugs that can involve other weakness.

This paper presents a comparison of several types of Neural Networks (NN) for forecasting the number of software bugs and vulnerabilities that will be discovered for a software component in certain timeframe, in terms of accuracy, trainability and

Technical University of Cluj Napoca, North University Centre of Baia Mare, e-mail: [1]{ovidiu.cosma, petrica.pop, cosmin.sabo}@cunbm.utcluj.ro, [2]laura.ov.cosma@student.utcluj.ro

stability to configuration parameters. The experimental part of this paper covers the Ubuntu Operating System and the Robot Operating System (ROS) which represents a set of software libraries and tools needed to build a robot application. The most complete repository that offers detailed information about vulnerabilities and weaknesses is the National Vulnerability Databse (NVD), which is the U.S. government repository of standards-based vulnerability management data, represented using the Security Content Automation Protocol (SCAP). This data enables automation of vulnerability management, security measurement, and compliance [11]. NVD includes databases of security checklists, security related software flaws, misconfigurations and product names.

Regarding ROS vulnerabilities, the number of resources is very limited, but ROS is running under Ubuntu OS, and the most of its vulnerabilities affect ROS. The project ROSIN [12], [13] identified more then 200 bugs associated with ROS, that cover a reliable time interval, and these have been used to train our NN models.

This paper is organized in six chapters, as follows: Section 2 presents a literature review regarding vulnerabilities forecasting, Section 3 describes the neural network models we have compared in our experiments, Section 4 presents the data collection used for training our models, Section 5 presents the experimental results, and Section 6 presents conclusions and new research directions.

2 Literature Review

Several methods have been proposed to forecast the ICT system vulnerabilities. The proposed methods can be classified into three classes: time series analysis-based models, artificial intelligence models and statistical based models. Next, we will review some of the most methods described in the literature.

The most important time series models introduced to forecast the vulnerabilities have been considered by Gencer and Basciftci [2] who used the Auto Regressive Moving Average (ARIMA) model and deep learning methods in the case of android operating system. Pokhrel et al. [4] described a vulnerability analytic prediction model of Desktop Operating System based on linear and non-linear approaches using time series analysis. Roumani et al. [5] used an exponential smoothing model for vulnerability analysis and prediction.

Some of the recent models for forecasting the vulnerabilities have Long Short-Term Memory (LSTM) cells in their composition and are trained by Gradient Descent and Back Propagation (BP-GD) algorithms. A comparative analysis of several types of deep neural networks was described by Kaushik et al. [3]. The conclusion drawn in [3] was that the Convolutional Neural Networks (CNN), Multilayer Perceptron Models (MLP), Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) models perform well for one step forecasting and less satisfactory for multiple steps forecasting.

Rahimi and Zargham [7] described a novel paradigm for vulnerability discovery prediction based on code properties, called vulnerability scrying. Their proposed

method extracts code complexity and quality properties from a source code and then uses a stochastic model to forecast vulverabilities. Williams et al. [8] described an integrated data mining framework that depicts automatically how the vulnerabilities evolve over time and detect the evolution of a particular vulnerability. In addition, their described framework has a predictive component that may be used to predict vulnerabilities or to approximate future appearance probabilities of vulnerability groups.

For comprehensive reviews on forecasting the ICT vulnerabilities, we recommend the papers: Roumani et al. [5], Yasasin et al. [6] and more recently Cosma et al. [1], who presented a comparative study of the most important and promising methods for forecasting the ICT systems vulnerabilities.

3 Neural Network Models

We used three Types of Neural Network Models for forecasting the number of bugs and vulnerabilities of software components: Long Short Term Memory (LSTM), MultiLayer Perceptron (MLP) and Convolutional Neural Network (CNN). Their general architecture has been adjusted based on preliminary experiments. In order to deliver good results, the models must have sufficient complexity to assure proper learning capacity, but unfortunately complex models are difficult to train, and they quickly enter in over-training. The LSTM model has been equipped with a dropout layer to mitigate the overtraining phenomenon.

All the models have been trained with the Gradient Descent - Back Propagation algorithm. Because there is little data available for training the models, we used the k-fold cross-validation technique. The model parameters and the GD-BP algorithm parameters have been fine-tuned based on grid search.

4 Data Collection

In this section we present the information sources we have used in our experiments. The collected data was used for training our NN forecasting models, as is going to be shown in the next section. We retrieved information regarding the Ubuntu operating system vulnerabilities from the National Vulnerability Database (NVD), and information regarding the Robot Operating System (ROS) bugs from the Robust-Rosin repository [12]. The structure of our vulnerabilities data collection is shown in Figure 2. It was defined based on the NVD data structure, transformed in a JSON format, with some modifications to enable easier identification of relevant information.

The information related to the ROS operating system bugs are taken from the ROSIN project [13] and its public GitHub repository [12]. The repository contains software bugs information in Yet Another Markup Language (YAML) [14] format. In

order to be used in our experiments, we scanned the files of the project, we extracted all its YAML files and transformed the content to a JSON format consistent with our data collection.

Finally we collected 3038 Common Vulnerabilities and Exposures (CVE) regarding Ubuntu operating system, which are related to different versions and cover a wide time period, and 220 ROS operating system related bugs.

5 Experimental results

In our experiments, we have built two sets of forecasting models: one for the Ubuntu operating system vulnerabilities, and the other one for the Robot Operating System (ROS) bugs. The models were developed in Python language, using the Keras deep learning API [9]. Each set was built to performed four types of forecasts: the number of bugs / vulnerabilities in the next month, and the average number of monthly bugs / vulnerabilities in the next 2, 3 and 6 months. Each type of forecast was performed using three different neural network models: LSTM, MLP and CNN. Each model was built, trained, tested and validated multiple times in a grid-search algorithm, in order to fine-tune its parameters.

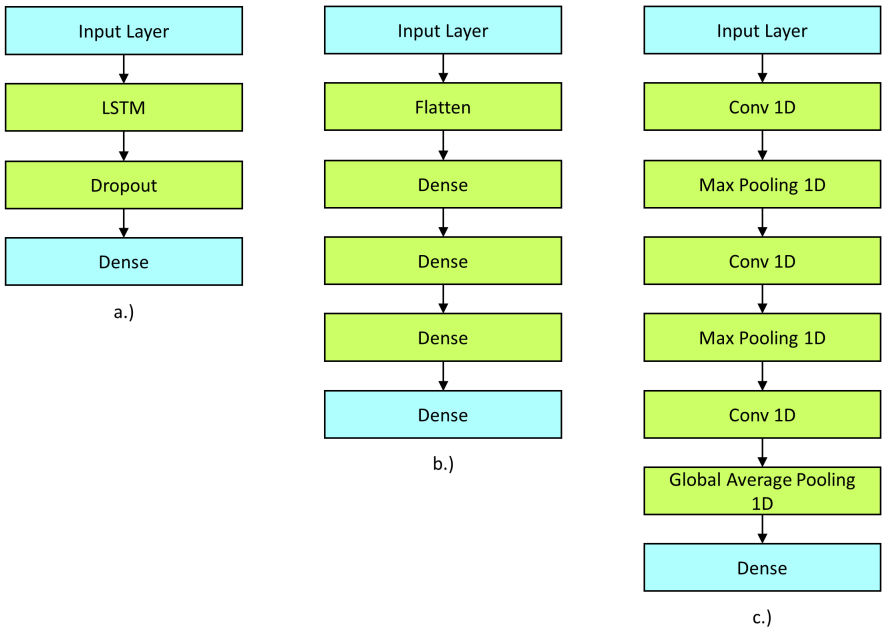


Fig. 1: Model architecture: a. Long Short Term Memory, b. Multilayer Perceptron, c. Convolutional Neural Network

The training data for the models in the Ubuntu set was taken from the National Vulnerability Database [11], and from the Ubuntu version history [10]. The input features are the monthly vulnerabilities and the age of the last version (in months), or the averages, depending on the type of forecast.

The training data for the models in the ROS set was taken from the ROBUST-ROSIN data set [12]. This data set contains both vulnerabilities and warnings. There are 4 levels of warnings, labeled as *"not-a-bug"*, *"bad-smell"*, *"bad-style"* and *"warning"*. The bugs are labeled by severity as *"minor-issue"* and *"error"*. We associated severity scores from 0 to 3 to the 4 warning levels, based on which we calculated cumulative warning levels for each month. The input features for the models in the ROS set are the number of monthly bugs and the warning level, or the averages, depending on the type of forecast. The input features for the 6 months average number fo monthly bugs forecasting models are presented in Figure 3. It can be seen that there is a correlation between the warning level and the number of bugs that will be discovered in the next period.

For increasing the forecasts accuracy, all the input data was normalized using Z-score normalization, in order to bring the average to 0 and the standard deviation to 1.

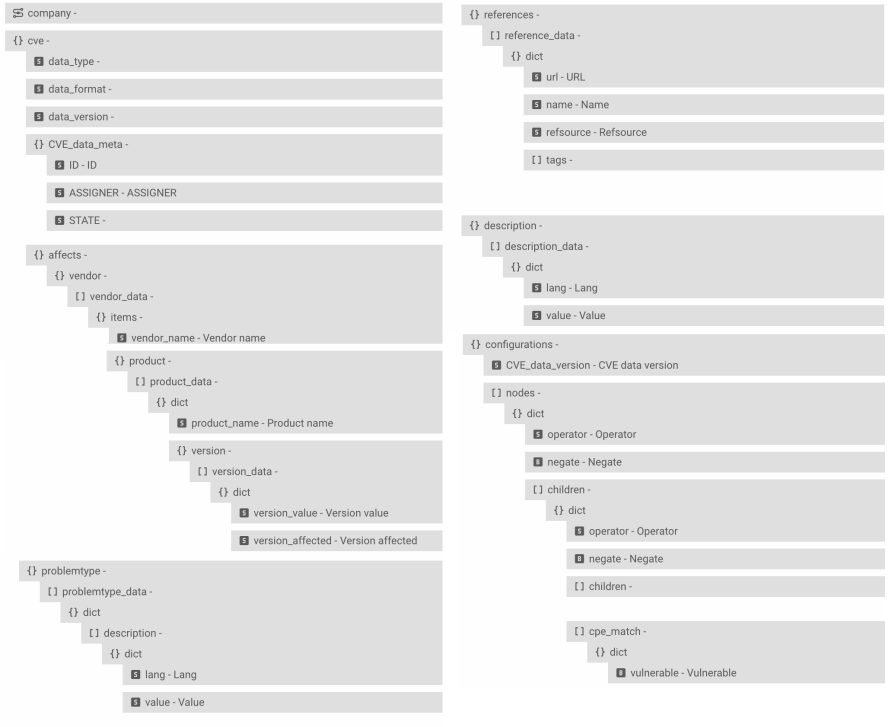


Fig. 2: ROS average number of vulnerabilities per month and average warning level

The results given by the best forecasting models in the Ubuntu set are shown in Figures 4 - 7. The forecasts cover a period of about 200 months, (from 2005-10 to 2022-04). The forecasting accuracy expressed by the Mean Absolute Error (MAE) computed for the entire period is situated between 3.53 for the 1 month model, and 1.98 for the 6 months average model, which is good, having in mind that the actual value ranges are $[0, 133]$ and $[0, 80]$ respectively.

A total of 24 forecasting models were built and tested: 12 in the Ubuntu set, and 12 in the ROS set. A comparison of their performance in terms of accuracy is shown in Figure 8. The MLP model consistently gave the best results for all the forecast types performed by the ROS set models. In the case of the Ubuntu set models, the best results were given by the CNN model for the 2, 3 and 6 months forecasts, and by the MLP model for the 1 month forecast. The LSTM model took the third place in each of the cases.

A complete model specification (CMS) is composed of the model architecture and all its configuration parameters, including the ones referring to the GD-BP algorithm.

For each CMS we biult, trained and evaluated several models. The overall accuracy of each model is expressed by the Mean Absolute Error (MAE) of the forecasts performed for the entire data set.

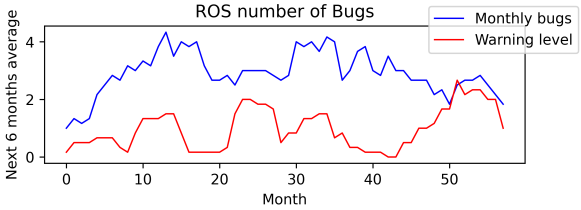


Fig. 3: ROS average number of vulnerabilities per month and average warning level

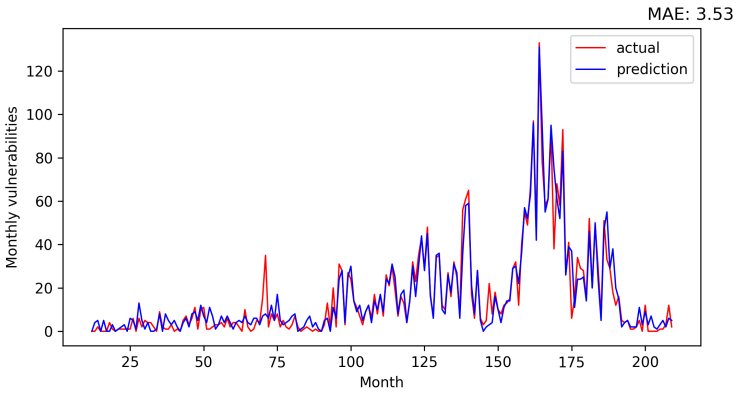


Fig. 4: Ubuntu monthly vulnerabilities forecasting

Thus, a Model Accuracy Result Set (MARS) was determined for each CMS, which allows us to make a comparison of the models in terms of trainability and in terms of sensitivity to configuration parameters.

All the models were trained using the Gradient Descent – Back Propagation (GD-BP) algorithm, which can be easily trapped in a local minimum. If there were no local minimums, the GD-BP algorithm would find the best solution every time. But this is not the case with real problems. The GD-BP algorithm usually ends in a local minimum, which depends on its parameters and the initialization of the model.

A certain Model Architecture (MA) can be considered easy to train, if when it is trained several times for the same CMS, similar results are obtained.

Thus, the MARS STandard Deviation (MARS-STD) calculated for all the models having the same CMS is an indication of the model architecture trainability.

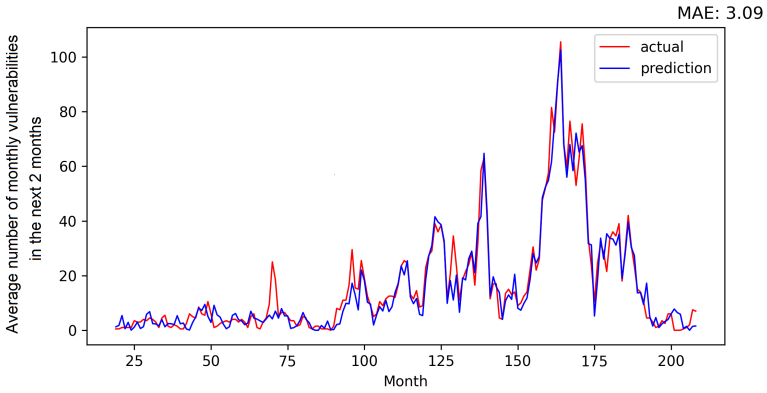


Fig. 5: Ubuntu vulnerabilities, 2 months average forecasting

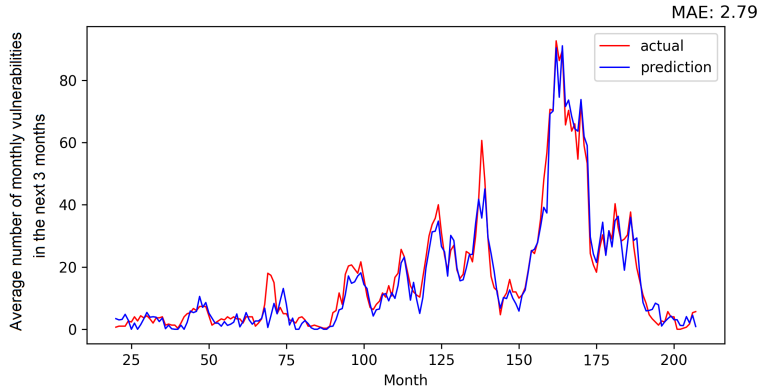


Fig. 6: Ubuntu vulnerabilities, 3 months average forecasting

A comparison between the model architectures in terms of trainability is presented in Figure 9. The plots represent the average MARS-STD for each model architecture in each model set. For easy comparison, the actual values were scaled to the $[0, 100]$ range. The plots in Figure 9 show that the LSTM models have the best trainability. This is no surprise, because they were specially designed to successfully handle long sequences of data. The MLP models are situated at the other extreme. They are hard to train, because the chosen architecture suffers of the vanishing gradient problem, because it has multiple intermediate layers.

For evaluating the model architectures sensibility to the configuration parameters, we calculated the AVerage of each MARS (AV-MARS) first, and then the standard deviation of the AV-MARS for each model architecture in each model set. The results scaled to the range $[0, 100]$ are presented in Figure 10. The CNN models seem to have the greatest sensitivity to the configuration parameters (with three exceptions). This sensitivity is not necessary a bad property, but it can be an indication that

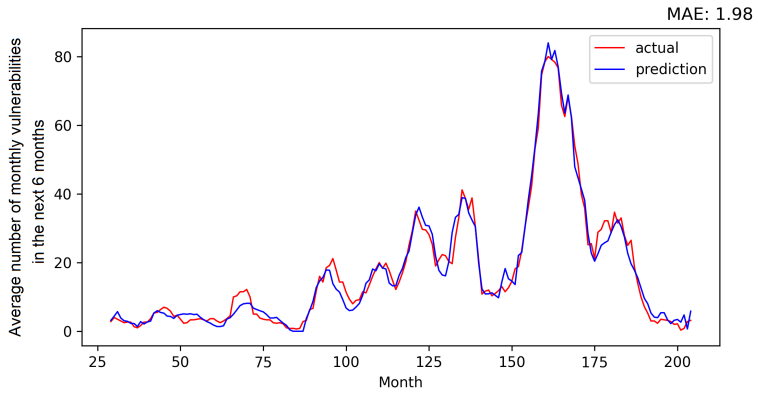
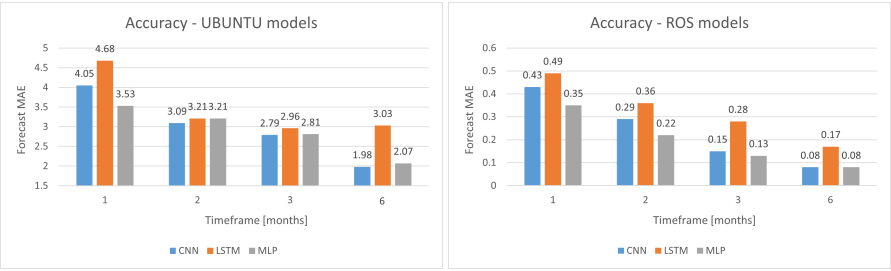
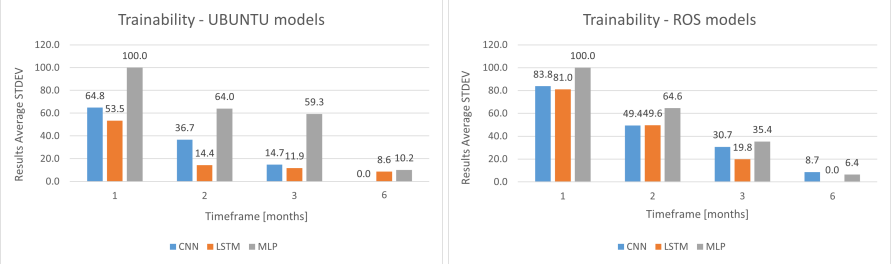


Fig. 7: Ubuntu vulnerabilities, 6 months average forecasting



(a) Ubuntu models (b) ROS models

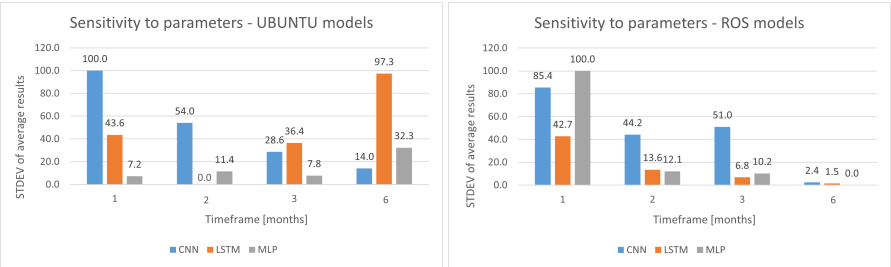
Fig. 8: Accuracy



(a) Ubuntu models (b) ROS models

Fig. 9: Trainability

the fine-tuning of the configuration parameters might be more difficult. The least sensitive to configuration parameters seems to be the MLP architecture.



(a) Ubuntu models (b) ROS models

Fig. 10: Sensitivity to configuration parameters

6 Conclusions

In this paper, we presented a comparison of several types of neural networks for forecasting the number of software bugs and vulnerabilities that will be discovered for a software component over a period of time, in terms of accuracy, trainability and stability to configuration parameters.

By analyzing the experimental results, the folowing conclusions can be drawn: In terms of forecasting accuracy, the best results were given by the CNN models in the case of vulnerabilities, and by the MLP models in the case of software bugs. In terms of trainability, the best results were given by the LSTM models, and in terms of stability to configuration parameters, the MLP models showed the best results. In

our next research we will add new models to our study, and will develop a genetic algorithm for improving the trainability of the MLP models which are worst from this perspective.

Acknowledgements This work was supported by the project BIECO (www.bieco.org) that received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 952702.

References

1. Cosma, O., Macelaru, M., Pop, P.C., Sabo, C., Zelina, I.: A Comparative Study of the Most Important Methods for Forecasting the ICT Systems Vulnerabilities. In: International Conference on Advanced Information Networking and Applications, 224-233, Springer, Cham (2021)
2. Gencer, K., Basciftci, F.: Time series forecast modeling of vulnerabilities in the android operating system using ARIMA and deep learning methods, Sustainable Computing In: Informatics and Systems, 30, 100515, (2021)
3. Kaushik, R., Shikhar Jain, Siddhant Jain, Tirtharaj Dash: Performance evaluation of deep neural networks for forecasting time-series with multiple structural breaks and high volatility. In: CAAI Transactions on Intelligence Technology, 1–16 (2021)
4. Pokhrel, N.R., Rodrigo, H., Tsokos, C.P.: Cybersecurity: Time Series Predictive Modeling of Vulnerabilities of Desktop Operating System Using Linear and Non-Linear Approach. In: Journal of Information Security, 8, 362-382, (2017)
5. Roumani Y., Nwankpa J.K., Roumani Y.F.: Time series modeling of vulnerabilities. In: Computers & Security, 51, 32-40 (2015)
6. Yasasin, E., Prester, J., Wagner, G., Schryen, G.: Forecasting IT security vulnerabilities – An empirical analysis. In: Computers & Security, 88, 101610 (2020)
7. Rahimi, S., Zargham, M.: Vulnerability Scrying Method for SoftwareVulnerability Discovery Prediction Without a Vulnerability Database. In: IEEE TRANSACTIONS ON RELIABILITY, 62(2), 395-407, (2013)
8. Williams, M.A., Barranco, R.C., Naim, S.M., Dey, S., Hossain, M.S, Akbar, M.: A vulnerability analysis and prediction framework. In: Computers & Security, 92, 101751 (2020)
9. Keras. https://keras.io/
10. Canonical: UBUNTU releases. http://releases.ubuntu.com/
11. National Institute of Standards and Technology: National Vulnerability Database. https://nvd.nist.gov/
12. robust-rosin: ROBUST ROS Bug Study. https://github.com/robust-rosin/robust
13. ROS-Industrial Quality-Assured Robot Software Components https://www.rosin-project.eu/
14. YAML.org: Yet Another Markup Language (YAML) 1.0.