Prediction of Common Vulnerabilities Scores (CVS)

Rimma Shilkina   
 Computer Science Department of College of Engineering  
 University of North Texas Denton, TX, USA

rimmashilkina@my.unt.edu

***Introduction***

According to the National Institute of Standards and Technology “The Common Vulnerability Scoring System (CVSS) is an open framework for communicating the characteristics and severity of software vulnerabilities.”1 The system consists of three types of metrics:

- Base score calculates on CVSS vector where each parameter has its specific weight

- Temporal metrics that change over time due to events external to the vulnerability

- Environmental scores customized to reflect the impact of the vulnerability on an organization

A CVSS score is represented as a vector string, a compressed textual representation of the values used to derive the score. An example of CVSS vector is below:

CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

Based on this vector engineers calculate CVSS base score. For the example above it would be 7.8 which is High severity.

Two common uses of CVSS are calculating severity of vulnerabilities discovered on one's systems and as a factor in prioritization of vulnerability remediation activities, i.e., “patching”.

These vulnerabilities have a unique Common Vulnerabilities and Exposures (CVE) identifier. The list of such identifiers is maintained by MITRE Corporation. Security professionals take CVEs as they are published by MITRE Corporation and link one or more Common Platform Enumerations (CPE) 2 to each CVE. CPE is a structured naming scheme for information technology systems, software, and packages. These CPEs are used to specify which software and versions are vulnerable. NIST also adds other pieces of information, such as a CVSS score.

CVSS score helps prioritize vulnerability remediation. In most cases this score has to be assigned by an engineer who assesses a vulnerability description and related materials (proof-of-concept code, etc.). The issue is that every week an assessment team needs to review 300 - 400 new notifications, which are written by different people from different countries. These people use different programming languages (where applicable) for proof-of-concepts or describe vulnerabilities with different accents (in English, but use different words, sentence structure, etc.). All these variables add complexity to assessments, which results in an average time-lag of 35 days between the first CVE disclosure to correctly assigned CPE. Also, not all CVE-s have CPE attached.

Newest Machine Learning technologies, including Natural Language Processing, could help an engineer with assessment, providing CVSS score faster and more accurate. Security teams in different organizations could use this score to focus only on those vulnerabilities that relevant to their organizations.

The number of identified vulnerabilities significantly increases each year. As example, there were 6,447 vulnerabilities that were identified in 2016, the number roughly doubled to 12,174 in 2019.3

Reducing a time needed to assign a CVSS score to a particular vulnerability could improve timing needed for assessment and prioritization of remediation of this issue over other concurrent activities that security teams need to perform. It also helps vendors to set priorities on developing patches as more widely used software with same criticality of a vulnerability could lead to more serious consequences (e.g., WannaCry - 2017).

In previous project three models were to predict CVSS score: Naïve Bayes Model, XGBoost, and simple Neural Network model. Two models (NBM, XGBoost) gave results, but accuracy scores were not high (~46% for the NBM, and ~56% for the XGBoost). The NN model crashed a free Colab environment, and even after several attempts to tune the model and reduce number of words in a vocabulary, the model did not work.

The goal of this project is to create a deep learning model that could read existing vulnerability notifications, define a way to predict CVSS vulnerability vector based on different keywords that describe vulnerabilities and then use this model to predict CVSS score (numerical) for a new vulnerability that was not in a training set. It will be possible to compare a score assigned by the DL-model with a score assigned by an engineer to see if there is a match.

There are several objectives on the way to achieve the goal:

• effectively clean and preprocess dataset by using standard techniques as well as NLP tools and methods

• find a deep learning model architecture that can work on a free Colab environment, which has memory, GPU, time restrictions.

• achieve accuracy of the best model at least more than 56% which was the best result in the previous project.

Other objectives related to innovation of the Project are using the newest available dataset (May 2022) and applying lemmatization technique in order to consider context of the vulnerability description.

***Related Works***

One of the most complex research papers related to the Project is “Automated Software Vulnerability Assessment with Concept Drift” by Triet Huynh Minh Le, Bushra Sabir, M. Ali Babar was published in 2019.4 The main goal of the research was to consider the concept drift in the prediction of vulnerability score. The Concept Drift in Machine Learning is constant changing of target value under different circumstances that affect the prediction in negative way.5 Regarding CVSS, new vulnerabilities have more new/unknown words at the time of their discovering (like Skype in 2003 when it was launched). Authors used dataset with more than 100,000 records with latest records in 2018. There are two innovations proposed in the research:

- the ML model, built by authors, utilizes both character-level and word-level features,

- time-based version of cross-validation method for model selection and validation which “splits the data by year to embrace the temporal relationship” of software vulnerability.

Multi-class classification and six ML models (Naïve Bayes, Logistic Regression, Support Vector Machine, Random Forest, Extreme Gradient Boosting, and Light Gradient Boosting Machine) were designed by authors. They prove that ensemble models (RF, XGB, and LGBM) perform better than single (first three). The evaluation was based on seven Vulnerability Characteristics - Confidentiality, Integrity, Availability, Access Vector, Access Complexity, Authentication, and Severity.

Another related work is “An Improved Vulnerability Exploitation Prediction Model with Novel Cost Function and Custom Trained Word Vector Embedding” by Mohammad Shamsul Hoque; Jamil, Norziana; Amin, Nowshad; Kwok-Yan, Lam was published in June 2021. The authors believe that CVSS score cannot always determine which vulnerabilities have high risk, they tried to “predict the vulnerabilities that are most likely to get exploited”, i.e., do not considering scores. The research was innovative in two aspects: custom-trained word vector (was created using the “gensim’” model with the “Word2Vec” technique of 5000 words) which can better consider the context and novel cost function (called “co\_efficient\_balance” and addressed the problem of class imbalanced)). The authors combined CVE, CPE, and CWE data from National Vulnerability Dataset, exploit dB and Symantec attack signatures. The dataset includes 146,183 records with such attributes as “cwe”, “cvss”, “references”, “impact”, “summary”, “Published”, “Modified”, etc. from time period 1197 – 2020. The model has high overall performance metrics for accuracy, precision, recall, F1-Score and AUC score with values of 0.92, 0.89, 0.98, 0.94 and 0.97, which according to the paper outperforms other existing models.6

In September 2020, Jay Jacobs, Sasha Romanosky, Idris Adjerid, and Wade Baker published their research “Improving vulnerability remediation through better exploit prediction”. They distinguished “published exploits” as “software code posted publicly and designed to compromise a weakness in another software application” and “exploits in the wild” as “instances of this code observed to be targeted at actual corporate networks”. The authors used NVD dataset as well as MITRE’s CVE list 2009-2018. The dataset contains 75 585 unique CVEs and 209 features from each vulnerability (to add more features each CVE link was scrapped for more information and description). The authors used multiword extraction and manual normalization to get 191 tags encoded as binary features for each vulnerability. Initially three models were built using CVSS scores, published exploits and attributes of the vulnerability. Coverage and efficiency were chosen for evaluation of a simple rules-based approach and ML model. The final, “optimal model”, is full machine learning model. As authors stated, the main finding is that “applying machine learning approaches to existing feature sets strictly dominate heuristic approaches” because the last are limited of data points7.

There are several works published in 2021 that use BERT (Bidirectional Encoder Representations from Transformers). One of them is “CVSS-BERT: Explainable Natural Language Processing to Determine the Severity of a Computer Security Vulnerability from its Description” by Mustafizur R. Shahid; Hervé Debar, and others. Authors built multiple BERT classifiers to get values for each metric of the CVSS base vector. These values then concatenated to get a full vector. From the predicted vector the CVSS score was inferred.

It is obvious that many cybersecurity companies perform research and definitely have advanced deep learning models to predict CVSS scores, but they do not publish their results.

***Methods***

Since the goal of the project is to predict a vulnerability score from 1 to 10 based on a description, the category of machine learning problem is a multilabel text classification.

Conventional neural networks (ANN or feedforward neural network) consider words (input) as independent from each other what is a serious problem for language models. A sequence of words, or context, is important to make a reliable prediction. That is a reason for using Recurrent Neural Networks (RNN) for the text classification problems.

The main differences of RNN from ANN are an ability to remember things about input and then use this information in loops when making a decision, i.e., it considers the current input and also what it has learned from the inputs it received previously.

Diagram

Description automatically generated

Figure 1. RNN vs ANN architecture10

Another difference is that RNN shares parameters across each layer, while ANN have different weight across each node. During the backpropagation process these weights are adjusted. RNN supports backpropagation through time (BPTT) algorithm to determine the gradients. It is slightly different from the traditional backpropagation algorithm. The latter trains itself by calculating errors from its output layer to its input layer and adjust weights, while BPTT sums errors at each time step.

Two problems – exploding gradients and vanishing gradients – are typical for RNN. In the first case, a result is an unstable model, in the second - the model does not learn.

Diagram

Description automatically generated

Figure 2. BPTT. The blue arrows indicate the gradient flow.10

LSTM (Long Short Memory) networks were introduced to solve the vanishing gradient problem. Their advantage is a capability to learn long-term dependencies which makes them good for the text classification problem.

Diagram, schematic

Description automatically generated

Figure 3. Structure of an LSTM cell 11

For this project the LSTM model in Pytorch was used as the model with the highest accuracy among Neural Network Models explored.

***Results***

*Dataset Description*

A National Vulnerability Database created by NIST was used for the project. It is built upon and fully synchronized with the CVE List so that any updates to CVE appear immediately in NVD. [CVE List](https://cve.mitre.org/cve/) was launched by [MITRE](https://cve.mitre.org/about/faqs.html#MITRE_role_in_cve) in 1999, NVD was formed in 2005.

The Project’s dataset contains data from 1999 till today, and it is a combination of twenty zipped json files. A file for 2002 also contains data since 1999. On May 2nd, 2021 the dataset included 185600 records. Each record contains following attributes:

1. CVE ID – consecutive identification number of a vulnerability in the following format "CVE-<year>-<number>”, i.e., CVE-2021-14567
2. Description - free text description of a vulnerability and how it works
3. Impact - it provides CVSS scores in two formats: CVSS v3 and CVSS v2 and contains such sub-attributes as
   * + vectorString – encoded representation of what this vulnerability can do (denial of service, code execution, privileges escalation, etc.), how much effort an attacker needs to exploit a vulnerability (i.e., no coding experience at all, or it should be chained with other vulnerabilities and code should be tailored to specific system), how a vulnerability can be executed (i.e., locally or remotely), how it can impact Confidentiality, Integrity and Availability of a system or information.
     + attackVector – decoded representation on how this vulnerability can be used (i.e., locally or remotely)
     + attackComplexity – regular text description that shows how complex this vulnerability to use (i.e., an attacker needs to know how to program, or an attacker just need to run already developed code)
     + ConfidentialityImpact – this parameter shows how a particular vulnerability can affect confidentiality property (i.e., it can disclose some system parameters, or it can provide access to passwords, or to personal information)
     + availabilityImpact – shows how a vulnerability can affect availability of information or a system (i.e., it can cause denial of service)

and others. The full list with attributes and their explanation/description could be found at  [https://www.first.org/cvss/specification-document.](%20https://www.first.org/cvss/specification-document.%20%20)

For the Project the following parameters for each vulnerability used from NVD CVE database: CVE ID, Description, Impact.

As example, there is an excerpt from a vulnerability record from NVD database (it is in a JSON format) in Appendix A.

*Data preprocessing*

NVD database has an API interface which can be used to download data, but for purpose of this project data is downloaded as a set of zipped json files separated by years, i.e., 2021, 2020. After downloading these files, code unzips them, and reads content of json files into a single list.

For simplicity and convenience, this list is parsed to separate lists: cveID, description, cvss3, vector3, severity, year.

CVSS\_SCORE has range from 0.1 to 10.0 but for the Naïve Bayes models it should be integers, so extracted value was multiplied by 10 and converted to INT type.

Some records have no assigned CVSS v3 score, those are records created before adoption of CVSS v3 score (they use CVSS v2) or these are new records where score is not assigned yet. During the parsing phase these records were assigned CVSS v3 score 0.

After parsing phase, a pandas dataframe was created to store parsed values: CVE\_ID, DESCRIPTION, CVSS3\_BASE, CVSS3\_VECTOR, SEVERITY, YEAR, PROC-DESC. First six of them contains the data from the corresponding lists. Data type of the YEAR attribute was converted to numeric.

A picture containing text

Description automatically generated

Figure 1 – Attributes and their Data Types

PROC\_DESC is to be used later, to store descriptions after the processing phase.

The next step is descriptions preprocessing punctuation removed by using regular expressions, all words converted to lower case, words were replaced by their corresponding lemmas by using Spacy package, stop words removed by checking if they are not in the stop lists.

Since some of records do not have CVSS v3 score assigned, these records cannot be used for training and test purposes, these records were removed from the dataframe which reduced it to 101215 rows.

The next step in the preprocessing stage is splitting the resulting dataset into training and test datasets in proportion 80% (80,972 records) and 20% (20,243) respectively.

*Data Exploration*

Descriptive statistics shows that CVEs in the NVD database are from 1999 to 2022 (current). Every time when code for the Project will be run, numbers could be different as new vulnerabilities added regularly.

Table

Description automatically generated

Figure 2 – Descriptive Statistics (05/02/2022 10.30AM CST)

The plot below shows how vulnerabilities are distributed per year. There is a trend to discover more vulnerabilities next year than previous year. This is due to development of tools for automated discovery, and to the fact that more specialists working on discovery of vulnerabilities now than it was before. Another reason for this increase is that each year new applications are developed and being assessed for vulnerabilities.

Chart, bar chart, histogram

Description automatically generated

Figure 3 – CVE Distribution in 1999 - 2022

One of the characteristics of a text is how often words are used in the text. This graph shows that most common words in all descriptions are allow, attacker vulnerability, via, remote, arbitrary, user, file, execute. The number of unique words is 217530.

Chart, line chart

Description automatically generated

Figure 4 – The Most Frequent Words in the CVE Descriptions

The next graph shows CVSS v3 scores distribution. There are few major groups of vulnerabilities: with score around 51-56 (real CVSS score 5.1-5.6), then near 61-66, 74-80, 88-90, and last group around 98-100.

Chart, histogram

Description automatically generated

Figure 4 – CVSS Scores Distribution

***Deep Learning Model***

Before giving a description of a model for this project, it is necessary to describe previous project attempts to build a DL-model in the following paragraph.

First, a simple conventional Neural Network Model was built. First runs of this model showed that dataset is too big for the free Colab environment. Decision was made to go towards reducing number of unique words. For this, words count procedure was added. This helped to detect some issues with stopwords and description preprocessing step. For the description preprocessing additional stoplist was added to remove words like “via”, “aka”, and other non-relevant words. Also, words with count less than two were removed from the processed descriptions. The procedure was running for 16 hours, and environment stopped at some point due to limitation on time. Next step was to create a corpus of words with count > 1 (to remove less common words from the corpus) and then use this corpus to create a Bag of Words vectors for train and test data. This didn’t help with the model. Few iterations were made to reduce corpus, the last was with count >5 words. NN model still crashed.

For this project dictionary was reduced to about 5000 words and possible reasons for memory consumption were analyzed. The approach to vectorize CVSS descriptions with existing dictionary showed that even with short sentences final vectors were a size of a dictionary (about 5000 integers). Other ways to convert sentences to numbers were researched.

Tokenizer approach looked promising as each encoded sentence would be the length of the original sentence. Tokenizer takes length of a dictionary as an argument and convert sentence in a vector where each word not only encoded by a number, but also by its frequency in the whole set of sentences (descriptions). In order to standardize all inputs to the model, padding function was used to pad to a maximum length of a sentence (processed description) among all sentences (it was calculated in a separate procedure after the step where regular sentences were lemmatized, converted to a lower case, punctuation and stop-words were removed).

Another attempt to make this model run was related to batch-size parameter for models. Batch-size allows to automatically split a train data set into smaller chunks for Colab to process it. That approach worked but accuracy scores were too low (tenth of a percent).

The decision to try transformers model was made at that point. Standard transformer model from Keras was trained. The accuracy of the model was almost 13%. With this model a dictionary was increased to 19,696 words (words with frequency less than 6 are not counted). But changing parameters like number of epochs or layers did not improve the accuracy. With increased number of epochs another limit of Colab was hit – timeout.

After additional research, the decision to use LSTM in Pytorch was made.

This model consists of embedding layer, LSTM layer, and linear output layer with output dimension of 101 because it needs to output a value in range 1 - 100 which corresponds to CVSS 3.0 score range 0.1 - 10.

There are few preprocessing steps for this model. First step is to tokenize the processed CVSS descriptions (they are already converted to lower case, stop words removed and words replaced with its lemmas). Another step is to define a dictionary as a pair of a word and its frequency and remove words where frequency is 5 or less (such word exists in less than .005% of all descriptions). After tokenizing, processed description looks like a list of numbers and can be feed to the model for training.

Table

Description automatically generatedFigure 5 – Tokenized descriptions

With these modifications model started to predict values with accuracy about 48%, and some attempts gave accuracy about 49%.

***Discussion***

At the start of the previous semester’s project Naïve Bayes Classification Model was used as a reference point. This model was able to provide 46.42% accuracy. XGBoost Classifier Model gave final accuracy score at 54.56%. In current project, Neural Network models were explored to improve accuracy scores. Intermediate Transformers Neural Network Model was able to give only 12.9% of accuracy, and its modifications did not improve the situation.

Graphical user interface, text, application, email

Description automatically generated

Figure 6 – Standard Transformers Model results

Then LSTM (based on Pytorch) was introduced to the project. This model provides accuracy of 47.94% and RMSE 14.91.

Table

Description automatically generated with medium confidence

Figure 7 – LSTM model results

Graphical user interface

Description automatically generated with medium confidence

Figure 8 – Accuracy/Loss per Epoch for the LSTM model

This model consists of Embedding layer which receives tokenized and padded descriptions of vulnerabilities and a dictionary as inputs. The length for all sentences is 557 tokens. Dictionary size is 19696 words for this model, words with frequency less than 6 were removed from the dictionary.

LSTM layer gets tokenized descriptions from the Embedding layer and outputs a feature vector (50 weight for current model) with parameters it learned from the data. Linear layer outputs one out of 100 possible values for CVSS score. Dropout layer prevents overfitting of the model with dropout rate 0.2.

Adam optimizer was used for the model and learning rate was set to 0.01.

Increasing number of epochs leads to slight increase in accuracy score, so next step for the project would be tuning the LSTM model including number of epochs, batch size and other to increase accuracy and efficiency of use of virtual Colab environment without hitting its restrictions.

Another possible improvement is to consider use of bigrams, trigrams and quadrigrams for description encoding and tokenization. Using more than one word (unigram) can improve the model as this increased importance of context.

***References***

1. <https://cve.mitre.org>
2. <https://nvd.nist.gov/products/cpe>
3. <https://www.kaseya.com/blog/2020/10/22/national-vulnerability-database-nvd/4>.
4. <https://www.researchgate.net/publication/335499801_Automated_Software_Vulnerability_Assessment_with_Concept_Drift>
5. <https://en.wikipedia.org/wiki/Concept_drift>
6. <https://www.proquest.com/docview/2545187395?pq-origsite=summon&accountid=7113>
7. <https://academic.oup.com/cybersecurity/article/6/1/tyaa015/5905457>
8. <https://ieeexplore.ieee.org/abstract/document/9680155>
9. <https://builtin.com/data-science/recurrent-neural-networks-and-lstm>
10. <https://www.researchgate.net/figure/An-illustration-of-Back-propagation-Through-Time-BPTT-in-an-unfolded-RNN-structure_fig2_336739252>
11. https://towardsdatascience.com/multiclass-text-classification-using-lstm-in-pytorch-eac56baed8df

**Appendix A. Example of a CVE record**

"cve" : {

"data\_type" : "CVE",

"data\_format" : "MITRE",

"data\_version" : "4.0",

"CVE\_data\_meta" : {

"ID" : "CVE-2022-22167",

"ASSIGNER" : "sirt@juniper.net"

},

"problemtype" : {

"problemtype\_data" : [ {

"description" : [ {

"lang" : "en",

"value" : "CWE-863"

} ]

} ]

},

"references" : {

"reference\_data" : [ {

"url" : "https://kb.juniper.net/JSA11265",

"name" : "https://kb.juniper.net/JSA11265",

"refsource" : "CONFIRM",

"tags" : [ "Vendor Advisory" ]

} ]

},

"description" : {

"description\_data" : [ {

"lang" : "en",

"value" : "A traffic classification vulnerability in Juniper Networks Junos OS on the SRX Series Services Gateways may allow an attacker to bypass Juniper Deep Packet Inspection (JDPI) rules and access unauthorized networks or resources, when 'no-syn-check' is enabled on the device. While JDPI correctly classifies out-of-state asymmetric TCP flows as the dynamic-application UNKNOWN, this classification is not provided to the policy module properly and hence traffic continues to use the pre-id-default-policy, which is more permissive, causing the firewall to allow traffic to be forwarded that should have been denied. This issue only occurs when 'set security flow tcp-session no-syn-check' is configured on the device. This issue affects Juniper Networks Junos OS on SRX Series: 18.4 versions prior to 18.4R2-S10, 18.4R3-S10; 19.1 versions prior to 19.1R3-S8; 19.2 versions prior to 19.2R1-S8, 19.2R3-S4; 19.3 versions prior to 19.3R3-S3; 19.4 versions prior to 19.4R3-S5; 20.1 versions prior to 20.1R3-S1; 20.2 versions prior to 20.2R3-S2; 20.3 versions prior to 20.3R3-S1; 20.4 versions prior to 20.4R2-S2, 20.4R3; 21.1 versions prior to 21.1R2-S2, 21.1R3; 21.2 versions prior to 21.2R2. This issue does not affect Juniper Networks Junos OS versions prior to 18.4R1."

} ]

….

},

"impact" : {

"baseMetricV3" : {

"cvssV3" : {

"version" : "3.1",

"vectorString" : "CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H",

"attackVector" : "NETWORK",

"attackComplexity" : "LOW",

"privilegesRequired" : "NONE",

"userInteraction" : "NONE",

"scope" : "UNCHANGED",

"confidentialityImpact" : "HIGH",

"integrityImpact" : "HIGH",

"availabilityImpact" : "HIGH",

"baseScore" : 9.8,

"baseSeverity" : "CRITICAL"

},

"exploitabilityScore" : 3.9,

"impactScore" : 5.9

},

"baseMetricV2" : {

"cvssV2" : {

"version" : "2.0",

"vectorString" : "AV:N/AC:M/Au:N/C:P/I:P/A:P",

"accessVector" : "NETWORK",

"accessComplexity" : "MEDIUM",

"authentication" : "NONE",

"confidentialityImpact" : "PARTIAL",

"integrityImpact" : "PARTIAL",

"availabilityImpact" : "PARTIAL",

"baseScore" : 6.8

},

"severity" : "MEDIUM",

"exploitabilityScore" : 8.6,

"impactScore" : 6.4,

"acInsufInfo" : false,

"obtainAllPrivilege" : false,

"obtainUserPrivilege" : false,

"obtainOtherPrivilege" : false,

"userInteractionRequired" : false

}

},

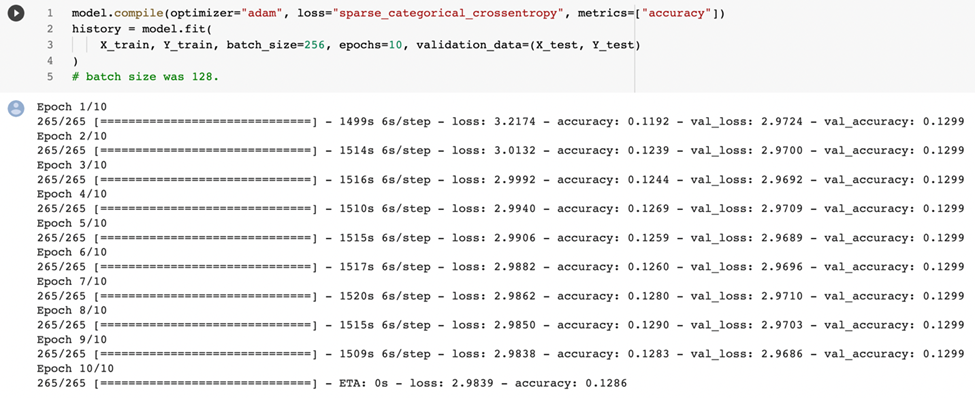
"publishedDate" : "2022-01-19T01:15Z",

"lastModifiedDate" : "2022-01-28T17:37Z"

**Appendix B. Intermediate model – Transformer – architecture and results.**

Graphical user interface, text, application, email

Description automatically generated

****