OVANA: An Approach to Analyze and Improve the Information Quality of Vulnerability Databases

Philipp Kühn
PEASEC, Technical University of Darmstadt
Darmstadt, Germany
kuehn@peasec.tu-darmstadt.de

Markus Bayer
PEASEC, Technical University of Darmstadt
Darmstadt, Germany
bayer@peasec.tu-darmstadt.de

Marc Wendelborn
PEASEC, Technical University of Darmstadt
Darmstadt, Germany
marc.wendelborn@stud.tu-darmstadt.de

Christian Reuter
PEASEC, Technical University of Darmstadt
Darmstadt, Germany
reuter@peasec.tu-darmstadt.de

ABSTRACT

Vulnerability databases are one of the main information sources for IT security experts. Hence, the quality of their information is of utmost importance for anyone working in this area. Previous work has shown that machine readable information is either missing, incorrect, or inconsistent with other data sources. In this paper, we introduce a system called *Overt Vulnerability source ANAlysis (OVANA)*, which analyzes the information quality of vulnerability databases utilizing state-of-the-art machine learning (ML) and natural language processing (NLP) techniques, searches the free-form description for relevant information missing from structured fields, and updates it accordingly. Our paper exemplifies that on the National Vulnerability Database, showing that OVANA is able to improve the information quality by 51.23% based on the indicators of accuracy, completeness, and uniqueness. Moreover, we present information which should be incorporated into the structured fields to increase the uniqueness of vulnerability entries and improve the discriminability of different vulnerability entries. The identified information from OVANA enables a more targeted vulnerability search and provides guidance for IT security experts in finding relevant information in vulnerability descriptions for severity assessment.

CCS CONCEPTS

• **Security and privacy → Vulnerability management**; • **Computing methodologies →** *Supervised learning*.

KEYWORDS

Security, Information Quality, CVSS, NVD, Deep-Learning

ACM Reference Format:
Philipp Kühn, Markus Bayer, Marc Wendelborn, and Christian Reuter. 2021. OVANA: An Approach to Analyze and Improve the Information Quality of Vulnerability Databases. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3465481.3465744

1 INTRODUCTION

Vulnerabilities in IT systems pose a threat to cyberspace. If ignored or overseen, they can lead to disastrous infrastructure failures of states, organizations, and individuals [28], exemplified by the Not-Petya outbreak in 2017 [27]. This attack was successful even though the vulnerability was patched three months earlier [27]. Hence, it is imperative to find and report vulnerabilities in existing systems, and make information about them accessible to IT service providers and IT administrators as quickly and detailed as possible [21]. Various vulnerability data sources are available online to support IT security experts in their daily workflow, and thus, secure IT infrastructures and systems.

Leading vulnerability databases are the *Common Vulnerabilties and Exposures (CVE)*[1] and *National Vulnerability Database (NVD)*[2], which have accumulated more than 150k and 160k entries, respectively, as of 2021-06-01. The CVE data feed collects not yet publicly known vulnerability information submitted by researchers, organizations, or individuals, while the NVD migrates it into its own data feed. This information is polished, structured, and enhanced with additional features, *e.g.*, the Common Platform Enumeration (CPE) specifier and a severity score using the Common Vulnerability Scoring System (CVSS)[3]. The CVSS severity score is calculated based on the CVSS vector, which is a key-value representation of different parameters and their values, *e.g.*, the parameter *attack vector* and the corresponding values *network*, *adjacent*, *local*, or *physical*.

However, research has shown that information in both databases is sometimes out-of-sync [9] or offers not enough evidence to allow timely reproduction and patching of the described vulnerability [24], what can be interpreted as a sign of poor information quality (IQ). Similar problems have been shown in other areas of IT security, *e.g.*, bug reports [6, 13]. These aspects slow down otherwise urgent action between initial discovery and patch, and create gaps of days, if not weeks or months [23, 31]. The seriousness of this problematic situation increases when one considers the mass of information security experts who face it. Different sources, *e.g.*,

[1]https://cve.mitre.org/
[2]https://nvd.nist.gov
[3]https://www.first.org/cvss/specification-document

vulnerability databases, bug reports, or blog posts, may state different (and possibly even contradictory) facts about a vulnerability, as their knowledge is also based on different sources. To enable security experts to cope with piles of information, it is mandatory to analyze the quality of information available in high-reputation information sources and ideally structure it in a way that machines are able to provide support.

In this paper we analyze the IQ of structured information in the NVD. The NVD is a trusted and tested source of information among IT security experts. Its information is double-checked and research has shown that the CVSS field in the NVD is more credible compared to other databases [18]. However, with more than 30 vulnerabilities disclosed every day, the amount of information seems overwhelming. Therefore, automation to guide and support experts is a promising avenue for research. However, these tools require structured information, *i.e.*, information that allows easy automation of processes. Usually, IQ is considered a subjective metric, depending on the person reviewing it. However, Naumann and Rolker [25] have categorized indicators to measure IQ, some of which can be measured objectively. In this paper we seek to answer two research questions: First, *what are quantifiable criteria for the assessment of IQ in vulnerability databases?* Second, *which distinct information improves the IQ of the NVD?*

For this purpose, we leverage the structured information in the database entries and the free-form entries. Our metric for IQ incorporates the indicators accuracy, completeness, and uniqueness. Therefore, we compare the vulnerability information of all entries to obtain an overall picture rather than focusing on the details of individual entries. To achieve our goal, several challenges need to be solved: (i) a metric must be defined to measure the IQ within the NVD, (ii) relevant information must be extracted from free-form vulnerability descriptions, and (iii) incorrect information must be rectified and missing information must be added (*e.g.*, the CVSS version 3).

These challenges are tackled with our tool OVANA, which leverages the state-of-the-art in NLP, BERT embeddings [8] and the flair-framework [2] to identify relevant information in free-form vulnerability descriptions and automate the process of IQ analysis. To encourage future work on this project, we provide the source code of OVANA[4].

This paper showcases our contributions with OVANA. On the theoretical side, we provide a metric for measuring the IQ within vulnerability databases **(T1)**. Second, we propose a novel, partially explainable, two-step machine learning pipeline capable of predicting security relevant tags and a CVSS score of NVD entries that can facilitate better vulnerability assessment **(T2)**. As practical contribution, the proposed tool identifies information in the NVD entries that can be used to speed up the search, reproducibility, and patching of the vulnerabilities **(P1)**. Further, the proposed pipeline may be used in the early stages of vulnerability disclosure to assist security professionals in finding an appropriate CVSS value and providing users with vulnerability guidance **(P2)**. Lastly, our tool elucidates the actual assignment of CVSS scores by making the prediction explainable **(P3)**.

[4]Published here: https://github.com/PEASEC/OVANA

The rest of this paper is structured as follows: §2 shows related work and outlines the research gap, §3 introduces a preliminary analysis followed by the concept of OVANA. §4 shows the evaluation of the ML models used, additional tweaks of the system, and presents the results of the IQ analysis. §5 discusses contributions and limitations, with a corresponding presentation of future work, while §6 concludes this paper.

2 RELATED WORK

This section presents related work, categorized into report and CVSS analysis, followed by the research gap.

2.1 Report Analysis

Public vulnerability reports have been examined in a variety of contexts. Nguyen and Massacci [26] are the first researchers to conduct a large scale inconsistency analysis. They identified incorrect version numbers in the NVD entries concerning the Google Chrome browser. Based on their results, Dong et al. [9] analyzed inconsistencies in software versions between vulnerability reports in the NVD and CVE. They showed that inconsistencies between NVD entries and corresponding CVE entries are highly prevalent and result in missing information in the NVD. Chaparro et al. [6] analyzed vulnerability reports and found missing steps-to-reproduce and expected behavior of the software. Similar work has been done by Mu et al. [24]. They analyzed the reproducibility of crowd-sourced vulnerability reports and chose a manual approach, by checking, whether vulnerabilities in 368 selected reports were reproducible with the information provided. Only 54.9% of the analyzed reports offered enough information to be reproducible, showing a similar result, *i.e.*, a lack of quality in public vulnerability descriptions. Considering the tremendous amount of work that has to be spent on manual quality analysis, the authors stress the need for an automatic collection of the missing information.

Different possible use cases for vulnerability reports have been analyzed. You et al. [35] studied vulnerability descriptions and automatically generated Proof-of-Concept exploits for the Linux kernel using semantic information in CVE descriptions. Their results show that vulnerable functions, variables, and system calls are critical semantic information for automated exploit generation. Schaberreiter et al. [30] presented a methodology for assessing trust in Cyber Threat Intelligence (CTI) sources based on a closed world assumption. They leveraged different IQ indicators, such as timeliness, to compare different CTI sources. Farhang et al. [14] analyzed Android vulnerabilities and their corresponding security bulletin entries from different vendors and discovered structural differences in the reporting on different vendor platforms. Zhang et al. [36] used information provided by the NVD to predict incoming cyber risks by using information about the affected product and the frequency of occurrence of new vulnerabilities for that product. Fan et al. [13] developed a tool to identify valid bug reports using 33 extracted features from bug reports that can be used in various occasions, *e.g.*, bug bounty programs.

Several papers proposed ways to improve the IQ of vulnerability reports. Guo et al. [15] identified missing information about the vulnerability type and trigger of a vulnerability in NVD entries and completed them using deep learning. Anwar et al. [4] analyzed inconsistencies in the structured fields *publication date*, *vendor*, *product names*, *severity score*, and *vulnerability type* in the NVD entries. They rectified these information by crawling the references of the specific entries and used different ML techniques, followed by an analysis of the rectified data.

2.2 Vulnerability Severity Assessment

The CVSS was introduced in 2003 with the release of version 1 in 2005 and has since been improved, resulting in the current version 3.1, released in 2019. Over the years, it has been criticized for its lack of transparency [12, 33]. Besides these critiques, the CVSS has been analyzed in different dimensions, ranging from comparing the severity perception of IT security experts with the CVSS scores [16], to correlating the CVSS scores with the number of real-world exploitations [3]. Research by Ruohonen [29] showed an average delay of the CVSS severity assessment in the NVD of one week in recent years. Different proposals bridge this gap with ML methods. Elbaz et al. [10] train a ML model based on the free-form NVD descriptions to predict the CVSS vector. Anwar et al. [4], on the other hand, use ML with the structured information of NVD entries to update CVSSv2 scores into v3 scores.

2.3 Research Gap

Vulnerability information sources are mandatory to patch loopholes and consequently secure IT infrastructures. This requires good IQ as timely as possible. Recent work in the field of vulnerability report analysis shows poor report quality [24] and inconsistencies in vulnerability databases [9, 26], which leads to confusion and a delayed patching process. All the works shown either focus on a single IQ indicator or information detail [9, 10, 15], or use solely the structured information to correct information [4]. This urges an automated and universal approach to improve the IQ of official sources such as the NVD, which uses various IQ indicators, corrects erroneous information and predicts empty fields based on all available information, including free-form description. Work that provides such an IQ metric which can be used to compare IQ improvement approaches [4, 9, 10, 15], as well as research that analyzes different indicators of the vulnerability database IQ, is currently still missing. As Elbaz et al. [10] have already indicated, this research needs to be explainable so one understands the decision-making processes of the algorithms.

3 CONCEPT

This section shows a preliminary analysis of the NVD to motivate our research. This analysis evolved into the OVANA architecture, followed by an explanation of how we solve the different challenges.

3.1 Dataset

In the present work, we focus solely on the NVD. Hence, we use all 146,100 NVD vulnerability entries from 1999 until 2020, timestamped on 2020-05-03, which are available at the NVD as JSON feed[5]. From this dataset, we extract 3,000 vulnerability entries distributed over the years 2013 until 2019, which contain a CVSSv3 vector. Their descriptions are manually labeled with the tags in Tab. 1 (details follow) by two IT security experts. These labeled vulnerability descriptions form the basis for training and testing the Named-Entity Recognition (NER) tagger and CVSS score predictors (see §3.3). To reduce manual workload and error-proneness, we only labeled the tags from Tab. 1 without annotating the CVSS values, which can be obtained directly from the NVD database. As an example, we have labeled the tag *attack vector*, but not the associated CVSS values *network*, *adjacent*, *local*, and *physical*.

Table 1: Labels used in descriptions.

Label	Tag	Usage
software name	SN	Dong et al. [9]
software version	SV	Dong et al. [9]
attack vector	AV	CVSS
attack complexity	AC	CVSS
privileges required	PR	CVSS
user interaction	UI	CVSS
scope	S	CVSS
confidentiality impact	CI	CVSS
integrity impact	II	CVSS
availability Impact	AI	CVSS
weakness	W	CWE
vulnerable function	VF	New field
vulnerable path	VP	New field
other	O	

3.2 Preliminary Analysis

To look deeper into the direction of IQ, we performed a preliminary analysis of the information in the NVD (timestamped 2020-05-03). We examined whether the structured information alone is sufficient to identify vulnerabilities. This is an important step in IT security management, as it enables automated detection of vulnerable IT system in an infrastructure. *E.g.*, if the information in vulnerability reports is too broad, one might find many vulnerabilities unrelated to the specific system resulting in a shut down of specific components, causing a business process disruption. For this reason, we implemented a small cluster analysis. It clusters the entries in the NVD dataset `nvd` in one year `y` based on their machine readable information: the fields CPE, Common Weakness Enumeration (CWE), and CVSSv3, which are extracted in `reduce_entry`. The clustering algorithm is given in the following Python excerpt[6].

```
cluster = defaultdict(list)
for v in nvd[y]:
    cluster[reduce_entry(v)].append(v)
```

This method showed large clusters in our dataset, *e.g.*, CVE-2019-2138 to CVE-2019-2165. We further examined our dataset to find the origin of clusters and identified the following problems:

(1) Weakness information are missing (*e.g.*, CVE-2019-5620) or use dummy values (*e.g.*, CVE-1999-0003 or CVE-2019-10732).

[5]https://nvd.nist.gov/vuln/data-feeds

[6]References defaultdict https://docs.python.org/3/library/collections.html?highlight=defaultdict#collections.defaultdict

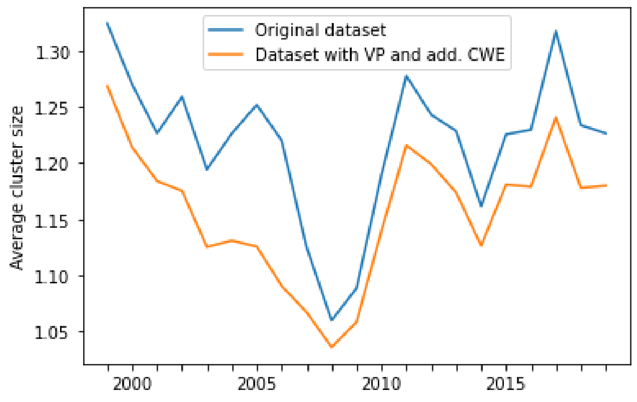

Figure 1: Average cluster sizes per year

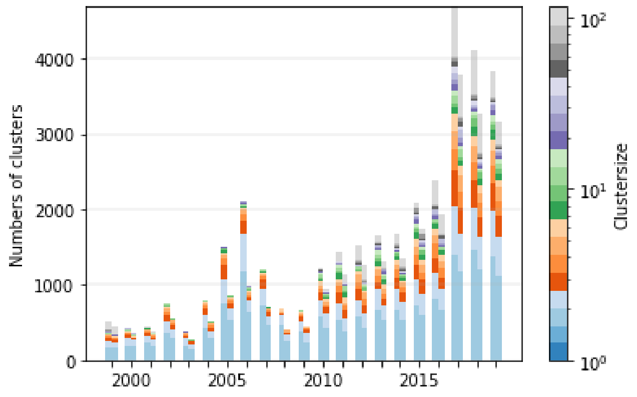

Figure 2: Clusters by year excluding singletons: the first bar of each year shows the clusters of the original data and the second bar the clusters of the enriched data.

(2) The outdated CVSSv2 standard is used (*e.g.*, CVE-2017-18538).
(3) Information in the free-form description is not fully utilized in the structured information (*e.g.*, in CVE-2019-9956 or CVE-2019-1010101).
(4) Some entries miss CPE information (*e.g.*, CVE-2017-0235) and in general the CPE is rarely used at its full potential (*e.g.*, CVE-2019-8010 misses the vulnerable version).
(5) Libraries are accounted as part of the product they are used in, even if there are matching CPE classes and should be standalone products (*e.g.*, in CVE-2019-9232).

Over the course of this paper we focus on problem 2 and 3, while we leave the rest for future work. Given these concrete problems, we wrote a solver that uses regular expressions to identify path information and CWE classes within descriptions of NVD entries. We use the identified information to enrich the corresponding NVD entries to reduce the previously mentioned problem of having too broad information in NVD entries. This simple method reduced the overall average cluster size from ≈ 1.22 to ≈ 1.16, *i.e.*, by ≈ 0.061 (see Fig. 1). Fig. 2 depicts the improvements in cluster sizes of this approach (excluding singletons). The first bar of each year shows the clusters of the original data, while the second bar shows the clusters of the enriched data. The gradient color is based on the clusters size, *i.e.*, clusters in the same size range share the same color. One can see that our approach split clusters into smaller ones (depicted by the decreased average cluster size), while some clusters remain quite large. Based on these results, we looked further into possible solutions with sophisticated methods to solve the problems.

3.3 OVANA Architecture

OVANA is composed of different modules: an IQ module, a NER module with a tagger for each tag, and a CVSS value predictor module with a predictor for each CVSS parameter, while the IQ module is used to benchmark the IQ. The IQ module benchmarks the IQ score of the input data based on the IQ indicators used, while the NER and CVSS modules are used to increase the IQ value of the this data. Specifically, the NER module is able to identify important named entities in the description of NVD entries. The CVSS value predictor module consists of multiple deep-learning models to predict the value of a specific CVSS key based on the identified named entities. The output of the NER module and the CVSS value predictor are then implemented into the vulnerability dataset by either correcting inconsistencies (in case of the CVSS value) or adding identified core information (in case of the vulnerable function (VF) and vulnerable path (VP)). Fig. 3 depicts the pipeline as described.

3.3.1 Information Quality. Measuring the information quality (IQ) of the NVD requires adequate indicators. Many IQ frameworks are available in the literature [1, 11, 25]. They define different IQ indicators for specific contexts, such as social media [1] or online marketing [11], and categorize them into sub-categories, *e.g.*, subject, object, and process criteria [25]. Since we analyze the IQ of the NVD, we have a specific context with specific requirements. We intend to objectively measure the IQ without using information sources other than the NVD. We selected *accuracy* [4, 9, 10, 15, 26] and *completeness* [4] as IQ indicators to enable a comparison of different approaches in future work. Further, we included the indicator *uniqueness* to measure the uniqueness of vulnerability entries. Unique entries give automated tools a better distinguishability. For quantification, we only use structured information that carries information about the described vulnerability, *e.g.*, the CPE information, while we ignore artificially introduced identifiers such as the CVE-ID. We denote the accuracy, completeness, and uniqueness scores with *acc*, *comp*, and *uniq*, respectively. To get the overall IQ, we simply add *acc*, *comp*, and *uniq* together.

Accuracy. Information accuracy has been extensively measured in previous work [4, 9, 26]. However, previous work either ignores the free-form vulnerability description or focuses on extracting specific information, like the software name and software version. Accuracy can be determined by comparing information with other sources, in our case the original with the predicted information. We restrict the accuracy score to a comparison of the original and predicted CVSS score, since comparing different CWE classes is not useful. Eq. 2 returns the accuracy of the entire database, where $cvss : V \longrightarrow [0, 10]$ and $predicted\_cvss : V \longrightarrow [0, 10]$ indicate the current and predicted CVSSv3 scores of a vulnerability, respectively.

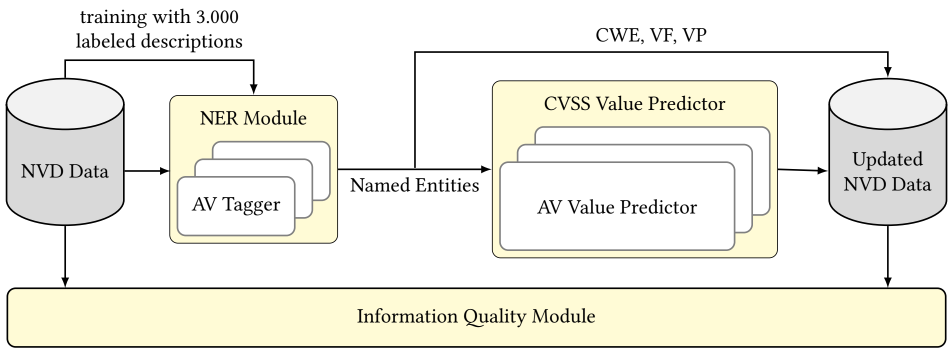

Figure 3: OVANA architecture pipeline

$$acc : D \longrightarrow \mathbb{R}_{\geq 0} \tag{1}$$

$$acc(D) = \frac{\sum_{v \in D} |cvss(v) - predicted\_cvss(v)|}{|D|} \tag{2}$$

It basically calculates the averaged absolute Δ over all original and predicted CVSS scores.

Completeness. The completeness of information can be determined by identifying missing information in a vulnerability entry. As an example, older entries ($\leq$ 2015) sometimes lack CVSSv3 information. The completeness of a single vulnerability is given by $comp : V \longrightarrow [0, |fields|]$, which indicates the number of fields with a value. Eq. 4 shows the completeness formula for the whole range of vulnerabilities in the dataset D. The lower the value of *comp*, the better.

$$comp : V \longrightarrow [0, |fields|] \subseteq \mathbb{R} \tag{3}$$

$$comp(D) = \frac{\sum_{v \in D} c(v)}{|D|} \tag{4}$$

Uniqueness. The uniqueness of vulnerability entries can be identified by comparing entries with each other, *i.e.*, a vulnerability $v \in D$ is unique *iff* $\forall v' \in D \setminus \{v\} : fields_v \neq fields_{v'}$, where D is the set of all vulnerabilities. Since this information alone misses improvements if large clusters are split into smaller ones, we also incorporate cluster sizes in Eq. 6, where $C \subset \mathcal{P}(D)$ is the set of clusters. The smaller the clusters, the smaller the value, indicating a better IQ, with a minimum of 0 if all vulnerabilities are unique.

$$uniq : C \longrightarrow \mathbb{R}_{\geq 0} \tag{5}$$

$$uniq(C) = \frac{\sum_{c \in C} \left(|c|^2\right)}{|C|} - 1 \tag{6}$$

All formulas combined give the IQ of the vulnerability dataset D, where the smallest possible and best IQ value is 0. This answers our first research question.

3.3.2 *Named-Entity Recognition.*

We want to identify named entities in the free-form descriptions, *e.g.*, words, which are related to a specific CVSS parameter, using NER [2, 22].

The labels in this paper are listed in Tab. 1. The former two are equivalent to the labels of Dong et al. [9], but are not in scope of this paper. They may be used in future work to correct possible inconsistent CPE information. The following eight labels are based on the CVSSv3 base metric, which are used to analyze the accuracy of the CVSS and complete incomplete vulnerability entries (see Problem 2 in §3.2). The label *weakness* was identified to potentially increase the accuracy of the CWE field (see Problem 1 in §3.2), while *VF* and *VP* are new fields that increase the *uniq* of entries (see Problem 3 in §3.2).

3.3.3 *CVSS Value Predictors.*

The CVSS score approximates the severity of a vulnerability[7]. It consists of different metrics: the base, temporal, and environmental metric, and of different parameters within these metrics. The NVD covers only the base metric. The other metrics depend on either temporal information that may change over time (temporal metric), or user specific information, like IT infrastructure information (environmental metric). The NVD offers a CVSS base vector, which holds the information used to calculate the severity in a structured way. This vector however, is not always available, yielding the highest severity, or contains data that is inconsistent with the vulnerability description. We try to (i) calculate a CVSSv3 vector for entries, which provide none; and (ii) detect inconsistencies between existing CVSSv3 vectors and the vulnerability description.

For this task, we leverage the NER module to detect named entities in the description associated with the CVSS vector parameter (see Tab. 1). Further, we use a neural network to predict the value of a parameter p based on the words related with that parameter. Here, we train one model for each parameter (see Tab. 1), resulting in eight different models. In this way, we are able to predict the CVSS parameter values for NVD entries with an outdated or non-existent base vector and detect inconsistencies with the description of the original vector. To measure these inconsistencies, we use the *acc* of the IQ module, which calculates the difference of the original CVSS score s_o and the predicted CVSS score s_p, *i.e.*, $|s_o - s_p|$.

In contrast to the machine learning model of Anwar et al. [4], our pipeline is capable of predicting the whole vector, updating old values, correcting incorrect values, and suggesting a vector for newly added entries. In contrast to the work of Elbaz et al. [10], we

[7] https://www.first.org/cvss/specification-document

use deep learning algorithms with state-of-the-art embeddings [8] instead of a linear model with a bag of words approach. The advantage of their model is that no additional labeling of the data is required, which can be beneficial in terms of generalization and variance minimization in machine learning. On the other hand, a more powerful algorithm can reduce the machine learning bias error, as in our case. Despite our more sophisticated model, we can also guarantee some form of explicability since the first part of the pipeline identifies named entities that lead to the prediction.

4 EVALUATION

This section presents the implementation of the ML models, the evaluation of those, and the results of the IQ analysis.

4.1 Architecture and Implementation

The NER module consists of one tagger module for each tag (see Tab. 1). This way, the NER module is able to predict multiple tags for a given word. On top of each CVSS tag (all CVSS tags in Tab. 1), we build an additional classifier that predicts the values of the tagged words.

We base our NER tagger and CVSS value predictor module on deep neural networks. Since neural networks can only work with numbers and vectors and not with natural language, we need to encode words. Context-sensitive word embeddings are the state-of-the-art in this field. We use BERT embeddings [8] and the Flair NLP framework [2], which are stacked together by concatenation. For the NER module, we feed those embeddings into a bidirectional long short-term memory (LSTM) network with a Conditional Random Field (CRF) layer as in the work of Huang et al. [17]. On top of this, the output nodes are fully connected to the LSTM hidden state.

To train a predictor model for a CVSS parameter p, we input all word embeddings, labeled with its tag in the vulnerability description and the corresponding CVSS value given in the NVD entry, and feed them into a gated recurrent unit (GRU) [7]. If a description does not contain information about a CVSS parameter, that description is skipped in the training process. In the end, a fully connected layer connects the hidden state to the output nodes.

4.2 Hyperparameters, Dataset Selection, and Evaluation Metrics

The models of the taggers and classifiers are very similar, so we use similar hyperparameters. Each tagger and classifier is trained using stochastic gradient descent (SGD) with a mini batch size of 32 and cross entropy loss. We use a learning rate of 0.1 with a anneal factor of 0.5 and limit learning to a maximum of 20 epochs. The embeddings are fine-tuned within each task. The language model of the flair embeddings has a dropout rate of 0.25, while the transformer architecture of BERT uses a dropout rate of 0.1. From the recurrent neural network (RNN) layer (LSTM or GRU) to the final layer dropout with a rate of 0.5 is applied. On the development set, we tried several other hyperparameter settings. *E.g.*, we noticed that the higher the number of epochs, the better the results. We monitored the micro F1 score on the development set for 5, 15 and 20 epochs. As a result, the availability impact classifier, for example, had a score of 0.6704, 0.6592, and 0.6536. For the learning rate, we performed the learning rate range test [32] for several taggers and classifiers, of which we identified 0.1 as a reasonable value. We also

Table 2: NER tagger module evaluation.

	Tag	Precision	Recall	F1
Software name	SN	0.8975	0.8226	0.8584
OS: n = 0	O	0.9792	0.9889	0.9841
Weighted Average		0.9706	0.9713	0.9708
Software version	SV	0.8516	0.8965	0.8735
OS: n = 0	O	0.9892	0.9838	0.9865
Weighted Average		0.9762	0.9756	0.9758
User interaction	UI	1.0000	0.8795	0.9359
OS: n = 0	O	0.9994	1.0000	0.9997
Weighted Average		0.9994	0.9994	0.9994
Scope	S	0.3750	1.0000	0.5455
OS: n = 50	O	1.0000	0.9994	0.9997
Weighted Average		0.9998	0.9994	0.9995
Attack complexity	AC	0.6575	0.6005	0.6277
OS: n = 2	O	0.9725	0.9784	0.9754
Weighted Average		0.9521	0.9539	0.9529

tried the ADAM [19] optimizer instead of SGD, but this produced worse results for the taggers. A larger batch size was not possible due to computational restrictions. Encountering a specific type of errors (explained in the next subsection), we also experimented with the tagging scheme. We tried the `iob2` tagging scheme [20], which, however, did not lead to significantly better results.

Furthermore, we noticed a high imbalance in the data for the training tasks. To address this issue, we implemented two versions of oversampling (OS). One is used in the NER module to oversample the training descriptions that contain the specific tag, while the other is used for the CVSS value predictors to oversample the phrases that contain the specific CVSS value. The OS functions are parameterised by $n \in \mathbb{N}_{\geq 1}$, denoting the repetition factor for the specific data instances. The different OS factors are given within the results of the evaluations.

For both evaluation tasks, we split the annotated data 80%, 10%, and 10% into a training-, development-, and testing-set. Unlike many other NER-tasks which split the data sentence-wise, one entry of the data set is seen as a whole NVD description. This way, the neural network can integrate the context of the whole entry without being limited by sentence boundaries. While we kept the same splits for the NER task, we performed a new split for each value prediction task so that the multiple classes of each tag are evenly distributed.

Each evaluation result is presented with the metrics precision, recall, F1 score, and the weighted average.

4.3 NER Module

An extract of the tagger results from the NER module can be found in Tab. 2 (for the full overview, see the appendix). It becomes apparent that the weighted average of all taggers is high. However, the weighted average mostly represents the O tag. While it is important for the performance of a whole tagger, we are also interested in the performance for individual tags. From this perspective, we can observe that the tagger results are mixed. For some cases, *e.g.*, in predicting software name (SN), software version (SV), and user interaction (UI), the tagger results are very good, with a F1 measure of 0.8584, 0.9047, and 0.9359, respectively. However, there are cases where the tagger performs worse, *e.g.*, when tagging scope (S) or attack complexity (AC) with F1 scores of 0.5455 and 0.6136, respectively.

As we inspected these errors, we noticed that some taggers are not able to detect the exact boundaries of tags. For example, see the SV tagger prediction in the description of CVE-1999-0131[8]: *"[..] denial of service in Sendmail 8.7.5 <SV> and <SV> earlier <SV> through <SV> GECOS field [..]"*. While the tagger correctly tags *"8.7.5"*, *"and"*, and *"earlier"* as software version, *"through"* is incorrectly recognized as such. For this reason, we also tested iob2 tagging [20], where tags are supplemented with *beginning (B)* and *inside (I)* tags, which support the CRF layer due to the potentially decreased search space. Unfortunately, this did not lead to better results. However, it should be mentioned that a recall and precision of more than 60% is still far from a poorly learned or random behavior, since extracting the correct words in a large set of words is a difficult task. This is the reason why we introduced OS, because otherwise the classifier would sometimes only predict O-tags, since the interesting tags rarely occur. Moreover, the results suggest that the tags that occur more frequently seem to be better predicted by the taggers. This implicates that more annotated training data leads to better performances of the taggers. The quality measurements are also influenced by incorrect tagger behavior. In contrast to the recognition of random words, as described in the error analysis, errors such as tagging too many words are not particularly serious. In the end, we only expect the NER module to find the correct tags within the descriptions. The exact limits at which the tags appear in the text are helpful, *e.g.*, if we want to make the classifications more explainable.

4.4 CVSS Value Predictors

Each CVSS value predictor is a neural network suitable for multi-class prediction of the values of one extracted CVSS tag. This way, we have, *e.g.*, a specific classifier suitable for the attack vector (AV) tag and able to differentiate between the associated values "network", "adjacent", "local", and "physical".

Similar to the results of the taggers, the classifiers performances are also scattered; an extract of the results is given in Tab. 3 (for the full overview, see §6). With the help of the OS technique, the results for the confidentiality impact (CI) and UI classifiers are good, with a weighted F1 score of 0.7105 and 0.8863, respectively. However, there are also classifier that perform worse, despite OS. *E.g.*, it seems rather difficult for the privileges required (PR) or integrity impact (II) classifier to predict the correct value (weighted F1 scores of 0.6386 and 0.6411, respectively). An excerpt of the results of the classifiers is given in §3 (for a complete overview see §6).

In contrast to the NER tagger results, no obvious error patterns were found. Nevertheless, it can be assumed that the classifier performance is also increased when more training data is available, as the classifiers can then generalize better. *E.g.*, the classifier for II does not produce good results for the class "None". This class is severely underrepresented, with even OS not yielding satisfying results. We assume that a simple OS is not sufficient, since the instances of this class are highly diverse. This problem might require a more sophisticated data augmentation technique or more training.

Table 3: CVSS value classifier evaluation.

	Value	Precision	Recall	F1
Confidentiality impact OS: n = 0	L	0.7059	0.5373	0.6102
	N	0.6400	0.6400	0.6400
	H	0.7395	0.8544	0.7928
Weighted Average		0.7152	0.7179	0.7105
User interaction OS: n = 2 (N)	R	0.8889	1.0000	0.9412
	N	1.0000	0.5000	0.6667
Weighted Average		0.9111	0.9000	0.8863
Privileges required OS: n = 2 (H)	L	0.5714	0.6667	0.6154
	N	1.0000	0.5556	0.7143
	H	0.5000	0.6667	0.5714
Weighted Average		0.6984	0.6296	0.6386
Integrity impact OS: n = 3 (N)	L	0.8056	0.4603	0.5859
	N	0.1429	0.1250	0.1333
	H	0.6396	0.8554	0.7320
Weighted Average		0.6817	0.6558	0.6411

4.5 Information Quality Analysis

With the IQ module, we analyze the dataset using all structured information available, including ones we added (CPE, CWE, CVSS, VF, and VP). Further, we identify named entities in descriptions using the NER module, which are then handed over to the CVSS predictor module. For the CVSS scores, which are mainly used for *acc*, we went for a conservative approach: If we found no information about a CVSS parameter and therefore cannot predict a value, we fall back on the original NVD dataset. If no information is available here either, we used the worst-case assumption implemented by the NVD[9]. We aim at rectifying the information in the NVD based on the free-form description. Hence, we assume the predicted information to be more aligned with the real-world severity [16], which is the reason the *acc* is added to the IQ score of the original dataset. The values for VF and VP are implemented directly into the updated dataset in structured form, since this information is not available beforehand.

Fig. 4 shows the overall IQ with the different indicators in a stacked form over the years 1999 to 2019 (which is further broken down in Fig. 5 and Fig. 6). One can see that the IQ improved drastically after implementing the additional information and resolving clusters, indicated by a lower IQ score. We lowered the average IQ score from ≈ 5.81 to ≈ 2.86, which is an average decrease by 51.2%. Fig. 7 depicts the differences of the IQ on an annual basis.

[8]https://nvd.nist.gov/vuln/detail/CVE-1999-0131

[9]https://nvd.nist.gov/vuln-metrics/cvss

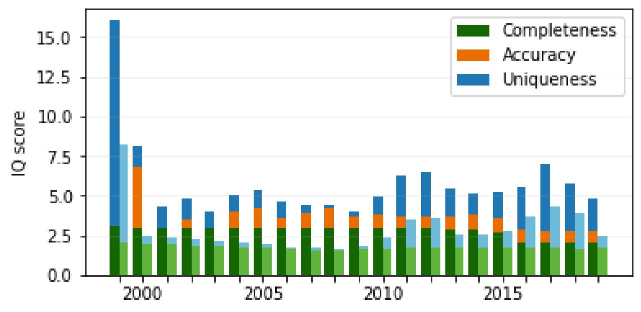

Figure 4: Stacked IQ compared: For each year, the first bar represents the IQ for the original NVD dataset, while the second one represents the IQ of the updated NVD dataset. Note: the lower the score, the better the IQ (see §3.3.1). The depicted indicators are *comp*, *acc*, and *uniq*.

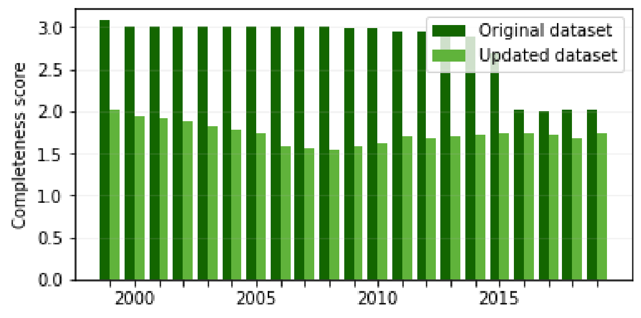

Figure 5: Completeness compared per year. Note: the lower the score, the better the IQ (see §3.3.1).

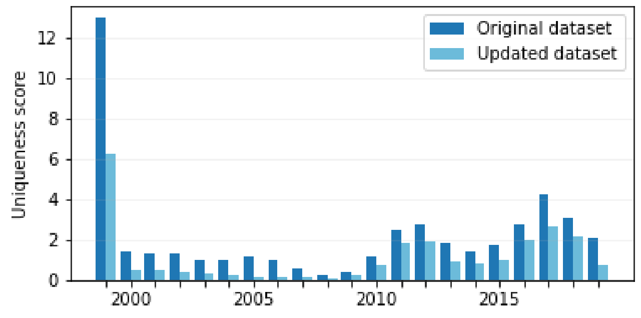

Figure 6: Uniqueness compared per year. Note: the lower the score, the better the IQ (see §3.3.1).

Completeness. In Fig. 5 the original *comp* shows an average of ≈ 2.78. It can be seen to remain at a mean of ≈ 2.97 until (and including) 2015, after which it drops to a mean of ≈ 2.01. The reason for this trend is the introduction of the CVSSv3 in 2015 and its adoption by the NVD at the end of 2015. All NVD entries after 2015-12-20 are published with a CVSSv3 score, which is reflected in the data. Further, the dataset lacks structured information about affected paths and functions, resulting in a base *comp* of 2. In the

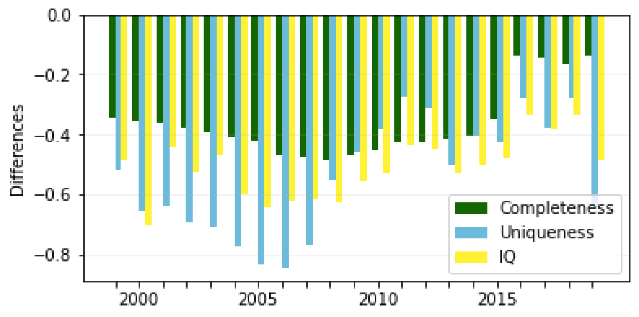

Figure 7: Relative deviations per year of *comp*, *uniq*, and IQ.

updated dataset, *comp* drops to an average of ≈ 1.73, *i.e.*, showing a decrease by 36.31% (see Fig. 7). The improvement can be explained by our conservative approach. This way, each vulnerability is assigned a score, eliminating all missing CVSSv3 scores. Additionally, more vulnerability descriptions include information about VFs or VPs, reflected by the slow decline from 1999 until 2008. The remaining missing fields are VF and VP information and the CPE entries, which, however, are beyond the scope of this paper.

Uniqueness. The *uniq* in Fig. 6 shows a peak in 1999 of the original dataset at ≈ 12.94, but averages out thereafter with a value of ≈ 1.63 excluding 1999 (at ≈ 2.17 including 1999). One can see two peaks in 2012 and 2017 of the original dataset with ≈ 2.76 and ≈ 4.26, respectively. The peak in 1999 arises from a widespread lack of structured information (`cpe`, `cwe`, and `cvss`) and forms a cluster of 117 entries. In 2012, there was one specific cluster with 45 vulnerability entries for `Apple iTunes` with no `cwe` and CVSS specification. These 45 entries are in a range of 123 ids, indicating some sort of batch-add, *i.e.*, all of which were added within a day or two. Similarly, in 2017 there was a cluster for `tcpdump`[10] which contained a very broad CPE specification, the same CWE class and the same CVSS vector for all entries. Due to their consecutive IDs, these vulnerabilities are also probably batch-added. Some of the batch-added entries even have the same description (*e.g.*, the largest cluster in 2011), which is problematic for the NER module. With such descriptions, the NER module is unable to identify distinctive information. The updated dataset shows an average *uniq* of 1.12%, *i.e.*, a decrease by 53.85% (see Fig. 7). Relatively speaking, the original and updated *uniq* look quite similar. This results from the aforementioned problem of the NER module with equal descriptions, which is common in large clusters.

5 DISCUSSION

This section presents the contributions of the research conducted in this paper and discusses its limitations.

5.1 Contributions

The contributions of this paper are are discussed in the following section. We categorized them in theoretical and practical contributions.

[10]https://www.tcpdump.org/

The first theoretical contribution is the **definition of an objective IQ measurement for vulnerability databases**, based on insights in different IQ frameworks [1, 11, 25] **(T1)**. To our best knowledge, no published study introduced such a measurement. Previous work used self-contained measurements bound to their use case [4, 9, 10]. Our approach allows for a comparison of different approaches, independent of the approach itself. The second theoretical contribution is the **proposed machine learning pipeline (T2)**. It consists of a NER tagger (see §3) that is able to identify relevant CVSS information, and CVSS predictors which predict the value of a CVSS parameter based on the identified named entities. While Elbaz et al. [10] use linear models with a bag-of-words approach and need no labeling effort, we use deep learning models with embeddings and manually labeled data. Our pipeline overcomes the limitation of the linear model, but is restricted in the data variety since the training data needs to be manually labeled. Both approaches try to find a way to make the prediction reasonably explainable, which is an important topic in machine learning research and especially important for the CVSS score prediction, since the manual assignment of it lacks transparency.

Our proposed tool is able to **identify additional information in the free-form descriptions and decreases the cluster sizes** within the NVD **(P1)**. This increases the uniqueness of each vulnerability and speeds up the search for vulnerabilities in source code. Moreover, it does allow faster reproducibility of vulnerabilities and consequently faster patches. Previous work [9] used similar schemes to identify named entities, but focused on singular pieces of information, *e.g.*, software name or software version, while our approach captures further information (VP, VF, and CVSS information). One important thing to note here is that more information does not necessarily make everything more secure. This has been discussed by You et al. [35] as they collected semantic information about vulnerabilities on references to automatically generate exploits. Hence, we only include information which was already available in the NVD, just not in a structured form. Our tool is potentially able to **guide security experts and practitioners during CVSS assessment (P2)**. Immediately after a vulnerability is publicly disclosed, usually no information is available besides a free-form description and references, if any [10]. Our proposed tool is able to assist security experts in finding an appropriate CVSS score and help practitioners assess the severity of a vulnerability when no score is available. Further, it provides an unbiased suggestion, *i.e.*, individual parameters are handled separately. The machine learning pipeline consists of a NER tagger, which, among other things, is able to identify the specific CVSS vector fields, and a CVSS value predictor, that classifies the found fields into the appropriate CVSS values. In addition, the pipeline enables a form of interpretability of the CVSS prediction which is helpful when a score already exists but it is not clear on which basis the score was assigned. This leads to the next practical implication. As mentioned earlier, the CVSS has been highly criticized for its non-transparent scoring system (see §2). Our proposal attempts to **increase the transparency of the scoring (P3)**. The tool is split into two layers of machine learning models. Hence, its interim results, *i.e.*, the key words that led to the CVSS prediction, can be presented to the user by highlighting its associated NER tag, CVSS tag, and predicted CVSS value. This can assist the security expert in CVSS rating, by offering some degree of explainability. Enriched with the descriptions of the respective tag and value, the information is also essential for practitioners seeking further information about severity classification.

5.2 Limitations and Future Work

Despite the promising results, our paper has some limitations that could be addressed in future work in this area.

Information Quality. This paper evaluates the IQ based on the indicators of accuracy, completeness, and uniqueness. The selection is based on research conducted in previous work in this field. In order to gain a more universal insight into IQ, additional indicators need to be included into the calculation. Further, we analyzed the IQ without the inclusion of results of analysis done in previous work, if they are not already included in the NVD [4, 9]. Future work could increase the number of indicators used for the IQ assessment and could evaluate the IQ improvements of previous inconsistency analyses. Furthermore, future work could look deeper into the topic of automatic CPEs generation based on vulnerability reports, as there are still problems as indicated in §3.2, and adjust this method to additional vulnerability databases.

Machine Learning. Our results show a high result fluctuation of the NER and CVSS modules. Further, there is a correlation between the quality of the results and the amount of training data available. Since we use manually labeled data, future work could create a crowd sourcing task to gain more training data. Another approach to artificially increase the size of the training data set is to incorporate sophisticated data augmentation techniques [5, 34] to increase the size of this training data set. If only the CVSS score is of interest, it would be possible to learn a regression model that does not require manual labeling because the score can be retrieved from the NVD.

6 CONCLUSION

Securing IT infrastructures is becoming more difficult every year. Vulnerability databases are one of many information sources for IT security experts to publish and learn about new threats. Previous work has shown inconsistencies in vulnerability databases and mixed results in terms of reproducibility, which calls for automated tools to obtain more information about published vulnerabilities and improve the quality of the information available. We answered our first research question, how to assess the IQ of vulnerability databases, by defining an IQ metric based on the indicators of accuracy, completeness, and uniqueness. A preliminary analysis of the NVD revealed large clusters of vulnerabilities sharing the same structured information, but with distinguishable descriptions, containing further information about the vulnerabilities. Hence, we answered our second research question, on which information improves the IQ of the NVD by using a tool called OVANA which analyzes the IQ of the NVD using state-of-the-art NLP and ML. It outlines important words and terms in the free-form descriptions of the vulnerability entries and implements them in the vulnerability dataset. With this technique, we were able to improve the IQ of the NVD by 51.23%. Additionally, our tool is able to guide IT security experts in selecting the correct values for CVSS parameters.

ACKNOWLEDGMENTS

This research work has been funded by the German Federal Ministry for Education and Research (BMBF) in the project CYWARN (13N15407) and the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

REFERENCES

[1] Nitin Agarwal and Yusuf Yiliyasi. 2010. Information quality challenges in social media. *Proceedings of the 2010 International Conference on Information Quality, ICIQ 2010* (2010), 15 pages.

[2] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. 2019. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. In *Proceedings of the 2019 Conference of the North*. Association for Computational Linguistics, Stroudsburg, PA, USA, 54–59. https://doi.org/10.18653/v1/N19-4010

[3] Luca Allodi and Fabio Massacci. 2017. Attack Potential in Impact and Complexity. In *Proceedings of the 12th International Conference on Availability, Reliability and Security* (Reggio Calabria, Italy) *(ARES '17)*. Association for Computing Machinery, New York, NY, USA, Article 32, 6 pages. https://doi.org/10.1145/3098954.3098965

[4] Afsah Anwar, Ahmed Abusnaina, Songqing Chen, Frank Li, and David Mohaisen. 2020. Cleaning the NVD: Comprehensive Quality Assessment, Improvements, and Analyses. *arXiv* (2020), 1–13. arXiv:2006.15074 http://arxiv.org/abs/2006.15074

[5] Markus Bayer, Marc-André Kaufhold, Björn Buchhold, Marcel Keller, Jörg Dallmeyer, and Christian Reuter. 2021. Data Augmentation in Natural Language Processing: A Novel Text Generation Approach for Long and Short Text Classifiers. arXiv:2103.14453 [cs.CL]

[6] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering* Part F1301 (2017), 396–407. https://doi.org/10.1145/3106237.3106285

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs]* (Dec. 2014), 9 pages. http://arxiv.org/abs/1412.3555 arXiv: 1412.3555.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 4171–4186. https://doi.org/10.18653/v1/N19-1423 arXiv:1810.04805

[9] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the Detection of Inconsistencies in Public Security Vulnerability Reports. In *USENIX Security*. USENIX Association, Santa Clara, CA, 869–885. https://github.com/pinkymm/inconsistency{_}detection

[10] Clément Elbaz, Louis Rilling, and Christine Morin. 2020. Fighting N-day vulnerabilities with automated CVSS vector prediction at disclosure. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/3407023.3407038

[11] Martin J. Eppler. 2003. *Managing Information Quality*. Springer Berlin Heidelberg, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24782-1

[12] Doudou Fall and Youki Kadobayashi. 2019. The Common Vulnerability Scoring System vs. Rock Star Vulnerabilities: Why the Discrepancy?. In *Proceedings of the 5th International Conference on Information Systems Security and Privacy*. SCITEPRESS - Science and Technology Publications, 405–411. https://doi.org/10.5220/0007387704050411

[13] Yuanrui Fan, Xin Xia, David Lo, and Ahmed E. Hassan. 2020. Chaff from the Wheat: Characterizing and Determining Valid Bug Reports. *IEEE Transactions on Software Engineering* 46, 5 (2020), 495–525. https://doi.org/10.1109/TSE.2018.2864217

[14] Sadegh Farhang, Mehmet Bahadir Kirdan, Aron Laszka, and Jens Grossklags. 2020. An Empirical Study of Android Security Bulletins in Different Vendors. In *Proceedings of The Web Conference 2020*. ACM, New York, NY, USA, 3063–3069. https://doi.org/10.1145/3366423.3380078

[15] Hao Guo, Zhenchang Xing, and Xiaohong Li. 2020. Predicting Missing Information of Vulnerability Reports. In *Companion Proceedings of the Web Conference 2020*. ACM, New York, NY, USA, 81–82. https://doi.org/10.1145/3366424.3382707

[16] Hannes Holm and Khalid Khan Afridi. 2015. An expert-based investigation of the Common Vulnerability Scoring System. *Computers & Security* 53 (2015), 18–30. https://doi.org/10.1016/j.cose.2015.04.012

[17] Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv:1508.01991 [cs]* (Aug. 2015), 10 pages. http://arxiv.org/abs/1508.01991 arXiv: 1508.01991.

[18] Pontus Johnson, Robert Lagerstrom, Mathias Ekstedt, and Ulrik Franke. 2018. Can the common vulnerability scoring system be trusted? A Bayesian analysis. *IEEE Transactions on Dependable and Secure Computing* 15, 6 (2018), 1002–1015. https://doi.org/10.1109/TDSC.2016.2644614

[19] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[20] Vijay Krishnan and Vignesh Ganapathy. 2005. *Named Entity Recognition*. http://www.stanford.edu/class/cs229/proj2005/KrishnanGanapathy-NamedEntityRecognition.pdf

[21] Philipp Kuehn, Thea Riebe, Lynn Apelt, Max Jansen, and Christian Reuter. 2020. Sharing of Cyber Threat Intelligence between States. *Sicherheit & Frieden* 38, 1 (July 2020), 22–28. https://doi.org/10.5771/0175-274X-2020-1-22 Publisher: Nomos Verlagsgesellschaft mbH & Co. KG.

[22] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2016 - Proceedings of the Conference*. Association for Computational Linguistics, Stroudsburg, PA, USA, 260–270. https://doi.org/10.18653/v1/n16-1030 arXiv:1603.01360

[23] Frank Li and Vern Paxson. 2017. A Large-Scale Empirical Study of Security Patches. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2201–2215. https://doi.org/10.1145/3133956.3134072

[24] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. 2018. Understanding the reproducibility of crowd-reported security vulnerabilities. *Proceedings of the 27th USENIX Security Symposium* (2018), 919–936.

[25] Felix Naumann and Claudia Rolker. 2000. Assessment Methods for Infomration Quality Criteria. In *Fifth Conference on Information Quality (IQ 2000)*. MIT, 148–162. https://doi.org/10.18452/2441

[26] Viet Hung Nguyen and Fabio Massacci. 2013. The (un)reliability of NVD vulnerable versions data. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*. ACM Press, New York, New York, USA, 493. https://doi.org/10.1145/2484313.2484377 arXiv:1302.4133

[27] Nicole Perlroth, Mark Scott, and Sheera Frenkel. 2017. Cyberattack Hits Ukraine Then Spreads Internationally. *The New York Times* (2017), 6 pages. https://www.nytimes.com/2017/06/27/technology/ransomware-hackers.html

[28] Christian Reuter (Ed.). 2019. *Information Technology for Peace and Security*. Springer Fachmedien Wiesbaden, Wiesbaden. https://doi.org/10.1007/978-3-658-25652-4

[29] Jukka Ruohonen. 2019. A look at the time delays in CVSS vulnerability scoring. *Applied Computing and Informatics* 15, 2 (2019), 129–135. https://doi.org/10.1016/j.aci.2017.12.002

[30] Thomas Schaberreiter, Veronika Kupfersberger, Konstantinos Rantos, Arnolnt Spyros, Alexandros Papanikolaou, Christos Ilioudis, and Gerald Quirchmayr. 2019. A Quantitative Evaluation of Trust in the Quality of Cyber Threat Intelligence Sources. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*. ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/3339252.3342112

[31] Muhammad Shahzad, Muhammad Zubair Shafiq, and Alex X. Liu. 2012. A large scale exploratory analysis of software vulnerability life cycles. In *Proceedings - International Conference on Software Engineering*. IEEE, 771–781. https://doi.org/10.1109/ICSE.2012.6227141

[32] Andrew Smith and Miles Osborne. 2006. Using gazetteers in discriminative information extraction. *Proceedings of the Tenth Conference on Computational Natural Language Learning, CoNLL-X* June (2006), 133–140. https://doi.org/10.3115/1596276.1596302

[33] J. M. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick. 2018. Towards Improving CVSS.

[34] Jason Wei and Kai Zou. 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. *arXiv:1901.11196 [cs]* (Aug. 2019), 9 pages. http://arxiv.org/abs/1901.11196 arXiv: 1901.11196.

[35] Wei You, Peiyuan Zong, Kai Chen, Xiao Feng Wang, Xiaojing Liao, Pan Bian, and Bin Liang. 2017. SemFuzz: Semantics-based automatic generation of proof-of-concept exploits. *Proceedings of the ACM Conference on Computer and Communications Security* (2017), 2139–2154. https://doi.org/10.1145/3133956.3134085

[36] Su Zhang, Xinming Ou, and Doina Caragea. 2015. Predicting Cyber Risks through National Vulnerability Database. *Information Security Journal* 24, 4-6 (2015), 194–206. https://doi.org/10.1080/19393555.2015.1111961

APPENDIX

Table 4: Evaluation results of the classifiers of the CVSS value predictor.

	Value	Precision	Recall	F1
Attack Vector	N	0.8357	0.9271	0.8790
OS: n = 10	L	0.5926	0.3636	0.4507
	A	0.8750	0.5385	0.6667
	P	0.3333	0.5000	0.4000
Weighted Average		0.7911	0.8048	0.7891
Attack Complexity	L	0.9174	0.9524	0.9346
	H	0.1667	0.1000	0.1250
Weighted Average		0.8521	0.8783	0.8642
Privileges Required	L	0.5714	0.6667	0.6154
OS: n = 2 (H)	N	1.0000	0.5556	0.7143
	H	0.5000	0.6667	0.5714
Weighted Average		0.6984	0.6296	0.6386
User Interaction	R	0.8889	1.0000	0.9412
OS: n = 2 (N)	N	1.0000	0.5000	0.6667
Weighted Average		0.9111	0.9000	0.8863
Confidentiality Impact	L	0.7059	0.5373	0.6102
OS: n = 0	N	0.6400	0.6400	0.6400
	H	0.7395	0.8544	0.7928
Weighted Average		0.7152	0.7179	0.7105
Integrity Impact	L	0.8056	0.4603	0.5859
OS: n = 3 (N)	N	0.1429	0.1250	0.1333
	H	0.6396	0.8554	0.7320
Weighted Average		0.6817	0.6558	0.6411
Availability Impact	L	0.5000	0.3636	0.4211
OS: n = 0	N	0.7381	0.8158	0.7750
	H	0.8333	0.8716	0.8520
Weighted Average		0.7521	0.7667	0.7567
Scope	U	0.6667	1.0000	0.8000
OS: n = 0	C	1.0000	0.5000	0.6667
Weighted Average		0.8333	0.7500	0.7333

Table 5: Evaluation results of the taggers of the NER modules.

	Tag	Precision	Recall	F1
Attack Vector	AV	0.7610	0.8016	0.7808
OS: n = 0	O	0.9738	0.9669	0.9703
Weighted Average		0.9491	0.9478	0.9483
Attack Complexity	AC	0.6575	0.6005	0.6277
OS: n = 2	O	0.9725	0.9784	0.9754
Weighted Average		0.9521	0.9539	0.9529
Privileges Required	PR	0.7647	0.5123	0.6136
OS: n = 10	O	0.9939	0.9980	0.9959
Weighted Average		0.9910	0.9920	0.9912
User Interaction	UI	1.0000	0.8795	0.9359
OS: n = 0	O	0.9994	1.0000	0.9997
Weighted Average		0.9994	0.9994	0.9994
Confidentiality Impact	CI	0.8115	0.7938	0.8026
OS: n = 0	O	0.9851	0.9867	0.9859
Weighted Average		0.9734	0.9737	0.9735
Integrity Impact	II	0.8179	0.7666	0.7915
OS: n = 0	O	0.9881	0.9912	0.9896
Weighted Average		0.9798	0.9803	0.9800
Availability Impact	AI	0.8134	0.8028	0.8081
OS: n = 0	O	0.9851	0.9861	0.9856
Weighted Average		0.9731	0.9732	0.9732
Scope	SC	0.3750	1.0000	0.5455
OS: n = 50	O	1.0000	0.9994	0.9997
Weighted Average		0.9998	0.9994	0.9995
Software Name	SN	0.8975	0.8226	0.8584
OS: n = 0	O	0.9792	0.9889	0.9841
Weighted Average		0.9706	0.9713	0.9708
Software Version	SV	0.8516	0.8965	0.8735
OS: n = 0	O	0.9892	0.9838	0.9865
Weighted Average		0.9762	0.9756	0.9758
Vulnerable Path	VP	0.8448	0.7091	0.7710
OS: n = 0	O	0.9940	0.9973	0.9957
Weighted Average		0.9910	0.9915	0.9911
Vulnerable Function	VF	0.9107	0.7463	0.8204
OS: n = 0	O	0.9968	0.9991	0.9979
Weighted Average		0.9957	0.9959	0.9957
Weakness	W	0.6968	0.7644	0.7290
OS: n = 2	O	0.9864	0.9809	0.9836
Weighted Average		0.9707	0.9691	0.9698