Detecting and Augmenting Missing Key Aspects in Vulnerability Descriptions

HAO GUO, College of Intelligence and Computing, Tianjin University, China
SEN CHEN*, College of Intelligence and Computing, Tianjin University, China
ZHENCHANG XING, Research School of Computer Science, Australian National University, Australia
XIAOHONG LI, College of Intelligence and Computing, Tianjin University, China
YUDE BAI, College of Intelligence and Computing, Tianjin University, China
JIAMOU SUN, Research School of Computer Science, Australian National University, Australia

Security vulnerabilities have been continually disclosed and documented. For the effective understanding, management, and mitigation of the fast-growing number of vulnerabilities, an important practice in documenting vulnerabilities is to describe the key vulnerability aspects, such as vulnerability type, root cause, affected product, impact, attacker type, and attack vector. In this paper, we first investigate 133,639 vulnerability reports in the Common Vulnerabilities and Exposures (CVE) database over the past 20 years. We find that 56%, 85%, 38%, and 28% of CVEs miss vulnerability type, root cause, attack vector, and attacker type, respectively. By comparing the differences of the latest updated CVE reports across different databases, we observe that 1,476 missing key aspects in 1,320 CVE descriptions were augmented manually in the National Vulnerability Database (NVD), which indicates that the vulnerability database maintainers try to complete the vulnerability descriptions in practice to mitigate such a problem.

To help complete the missing information of key vulnerability aspects and reduce human efforts, we propose a neural network based approach named **PMA** to predict the missing key aspects of a vulnerability based on its known aspects. We systematically explore the design space of the neural network models and empirically identify the most effective model design in the scenario. Our ablation study reveals the prominent correlations among vulnerability aspects when predicting. Trained with historical CVEs, our model achieves 88%, 71%, 61%, and 81% in F1 for predicting the missing vulnerability type, root cause, attacker type, and attack vector of 8,623 "future" CVEs across 3 years, respectively. Furthermore, we validate the predicting performance of key aspect augmentation of CVEs based on the manually augmented CVE data collected from NVD, which confirms the practicality of our approach. We finally highlight that PMA has the ability to reduce human efforts by recommending and augmenting missing key aspects for vulnerability databases, and to facilitate other research works such as severity level prediction of CVEs based on the vulnerability descriptions.

Additional Key Words and Phrases: CVE, Vulnerability description, Data augmentation, Deep neural network

*Sen Chen (senchen@tju.edu.cn) and Xiaohong Li are the corresponding authors. This work has partially been supported by the National Science Foundation of China (No. 62102284, 61872262).

Authors' addresses: Hao Guo, College of Intelligence and Computing, Tianjin University, China; Sen Chen, senchen@tju.edu.cn, College of Intelligence and Computing, Tianjin University, China; Zhenchang Xing, zhenchang.xing@anu.edu.au, Research School of Computer Science, Australian National University, Australia; Xiaohong Li, College of Intelligence and Computing, Tianjin University, China; Yude Bai, College of Intelligence and Computing, Tianjin University, China; Jiamou Sun, Research School of Computer Science, Australian National University, Australia.

1049-331X/2021/1-ART1 $15.00
https://doi.org/10.1145/3498537

ACM Reference Format:
Hao Guo, Sen Chen, Zhenchang Xing, Xiaohong Li, Yude Bai, and Jiamou Sun. 2021. Detecting and Augmenting Missing Key Aspects in Vulnerability Descriptions. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (January 2021), 27 pages. https://doi.org/10.1145/3498537

1 INTRODUCTION

Security vulnerabilities can be exploited to damage system or information confidentiality, integrity, and availability [17]. Significant human efforts have been made to document and manage publicly known vulnerabilities. The core of these efforts is the Common Vulnerabilities and Exposures (CVE) [39]. CVE is a list of entities - each reporting a publicly known vulnerability with a unique identification number, a description, and at least one public reference of the initial announcement of the vulnerability. At the time of this work, over 139,000 vulnerabilities have been recorded in the CVE database. With the fast growth of vulnerabilities, there is also a increasing concern about the information quality of vulnerability descriptions [15, 41].

Remarkably, the description of CVE is one of the most important and informative entities. Typically, a CVE is described by the key vulnerability aspects [29]. Fig. 1 presents the description of the CVE entry "CVE-2005-4676". As highlighted in Fig. 1, a high-quality CVE description should has **six key aspects** of the vulnerability [21, 29], including *vulnerability type* (e.g., buffer overflow), *affected product* (including vendor/version/component information, e.g., Andreas Huggel Exiv2 before 0.9), *root cause* (e.g., does not null terminate strings before calling sscanf), *attacker type* (e.g., remote attacker), *impact* (e.g., cause a denial of service - application crash), and *attack vector* (e.g., via images with crafted IPTC metadata).

Security experts take advantage of these key aspects of CVE descriptions for vulnerability understanding, management, and mitigation of fast-growing number of vulnerabilities. Specifically, CVE descriptions are helpful for understanding and assessing the severity [23], exploitability [6], and many other characteristics (e.g., compromise of system confidentiality, integrity, and availability) [19] of the vulnerabilities. It also infers the related library names of a given CVE with the software composition analysis (SCA) [11, 63]. As vulnerabilities are often documented in multiple databases, such as the CVE [39] curated by "the power of the crowd" and the National Vulnerability Database (NVD) [37] established by the US government (i.e., NIST [44]), people can use CVE descriptions to detecting the inconsistencies between different vulnerability databases [15]. Moreover, CVE descriptions establish and consolidate the traceability links across vulnerabilities, exploits, and patches, such as the CVE-2018-2628[1] in the CVE, the EDB-ID 44553[2] in the ExploitDB, and the patch commit in the GitHub repository. Those traceability links profit for localizing vulnerable functions in the source code [65] and for developing and deploying patches [8, 30]. Most of these description-based studies heavily rely on the information contained in the vulnerability descriptions.

However, one of the biggest challenges of performing vulnerability description-based analysis is that there may not be sufficient information (**incomplete key aspects**) of a large part of CVEs according to our preliminary investigation. For example, missing key aspects in the vulnerability descriptions can affect the prediction of severity level of vulnerabilities (RQ5 in § 4.6) and lead to wrong CVSS scores [17]. Therefore, the main goal is to recommend and augment the missing key aspects of vulnerability descriptions. Besides these description-based studies, vulnerabilities are constantly being discovered and reported, the vulnerability discoverers may not know some critical aspects of the vulnerability when submitting the report. Consequently, one of the goals of

[1]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-2628
[2]https://www.exploit-db.com/exploits/44553

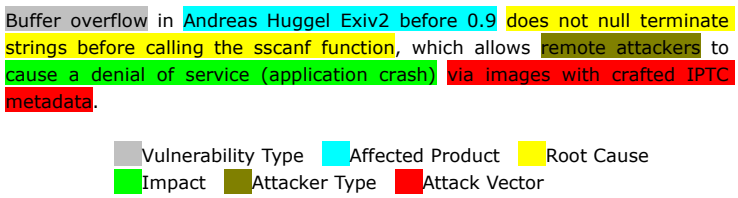
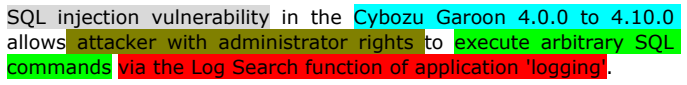
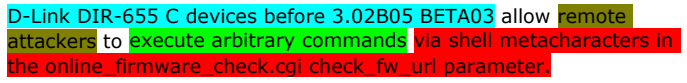


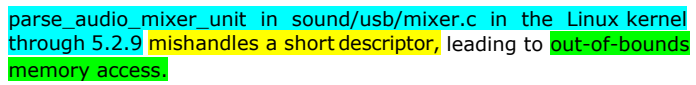
Fig. 1. An example of CVE description



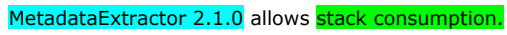
(a) CVE-2019-5934: missing Root Cause



(b) CVE-2019-13561: missing Root Cause and Vulnerability Type



(c) CVE-2019-15117: missing Vulnerability Type



(d) CVE-2019-14262: only have Affected Product and Impact

Fig. 2. Examples of CVEs that miss information

our work is to help these people complete and submit their newly discovered vulnerabilities with unified format.

Considering the mentioned importance of CVE descriptions, **we firstly systematically investigate the information completeness of the CVE descriptions**. We develop a rule-based method[3] to extract the six key aspects from the CVE descriptions. If certain aspect fails to be extracted, we define this aspect as a missing aspect. To develop aspect extraction rules, we first randomly sample 20% CVEs (27,130 in total) per year from January 1999 to August 2020. We manually extract the aspects in the description of these 20% sampled CVEs. Based on such collected aspects, we further develop regular expression patterns for aspect extraction. Then we apply our aspect extraction rules to gain aspects on the rest of CVEs. We consider the extracted CVE aspects are high-quality (>97% accuracy at the 95% confidence level and 5% error margin), by a sampling method [50] on the the full of extracted CVE aspects.

After inspection of the presence or absence of the six key aspects in the description of these 136,639 CVEs, we find that almost all the CVEs (over 99%) have the *affected products*. This conforms to the common sense that the reporters must determine the *affected product* of a new CVE before submitting. However, the other five key aspects can be absent (see in Fig. 2), although the CVE Numbering Authority (CNA) defined quality checks to improve the quality of the vulnerability descriptions before publishing them. For instance, 94% of CVEs define the *impact* on what the attacker gains by exploiting this vulnerability. Furthermore, many CVEs miss the other four aspects (the more technical aspects) - *vulnerability type, root cause, attacker type,* and *attack vector*. Specifically, 28% of CVEs skip *attacker type*, and 38% of CVEs ignore *attack vector*. We also observe that only 58% of CVEs describe either *vulnerability type* or *root cause*, while the rest 42% describe

[3]https://github.com/pmaovr/Predicting-Missing-Aspects-of-Vulnerability-Reports

Table 1. Examples of augmented aspects in NVD

Aspect	CVE-ID	Description type	Descriptions
Vulnerability type	CVE-2006-0795	Modified	***Absolute path traversal vulnerability*** in convert.cgi in Quirex 2.0.2...
		Original	Unspecified vulnerability in convert.cgi in Quirex 2.0.2...
Root cause	CVE-2015-8768	Modified	***click/install.py in click does not require files*** in... which allows remote attackers...
		Original	install.py in click allows remote attackers...
Attacker type	CVE-2010-2206	Modified	Array index error in AcroForm.api... allows ***remote attackers*** to...
		Original	Array index error in AcroForm.api... allows attackers to...
Attack vector	CVE-2010-0189	Modified	... installation of arbitrary programs via ***a crafted name for a download site***.
		Original	... installation of arbitrary programs via unknown vectors.

neither of them. This is unreasonable because a CVE should describe either *vulnerability type* or *root cause* according to the guideline of CVE key details [29].

Hence, **our second task is to automatically recommend and augment the missing aspects** in CVE descriptions. According to our investigation, to improve the completeness of the CVE descriptions, database maintainers try to manually augment more information such as *vulnerability type, root cause, attack vector*, and *attacker type* through different ways, as shown in Table 1 (RQ4 in § 4.5). However, manual augmentation of missing aspects by vulnerability database maintainers will take substantial human efforts in practice. Therefore, automated augmentation of missing aspects for vulnerability descriptions is very practical and useful to reduce human efforts and maintain high-quality vulnerability descriptions.

Unfortunately, rule-based method will not work for this augmentation task. Because the correlations among different vulnerability aspects and their combinations are complex, which is difficult to summarized as a set of explicit rules (see the experiment results of aspect fusion and aspect ablation in § 4.2.3 and § 4.3, respectively). So we adopt a neural network based method named Prediction of Missing Aspect (**PMA**) which can directly learn the intricate relations across different vulnerability aspects from the existing CVE descriptions instead of manual feature engineering. We formulate this task as a multi-class text classification task - predict the label of certain missing aspect of a vulnerability based on its known aspects. To implement it, we first systematically explore the design space of the neural network based classifier, including input text format and representation, model architecture, and network design.

To evaluate the effectiveness of **PMA**, **1)** we build a "historical" dataset of 43,583 CVEs (till September 2016) where each CVE contains at least 4 out of the 6 vulnerability aspects. In the 10-fold cross validations on the historical dataset, the best classifier design achieves the prediction performance 94%, 79%, 89%, and 70% for *vulnerability type, root cause, attacker type*, and *attack vector*, respectively. **2)** We also set ablation experiments to determine the most and least prominent aspect or aspect combinations for predicting a particular missing aspect. Our results show that the *impact* aspect has the greatest impact on the prediction of vulnerability type, while *affected product*, and *vulnerability type* have the greatest impact on the prediction of *root cause, attacker type*, and *attack vector*. At the same time, *root cause* and *attacker type* have least impact on the prediction of other aspects. **3)** To confirm the usefulness of our aspect augmentation method, we build a "future" dataset of 8,623 CVEs (from October 2016 to August 2020). Trained on the historical dataset, our method achieves the prediction performance 88%, 71%, 61%, and 81% for predicting the missing *vulnerability type, root cause, attacker type*, and *attack vector* on the future dataset, respectively. **4)** We finally demonstrate the practicability of PMA (in terms of description augmentation) on the real updated CVEs collected from NVD.

This paper makes the following contributions:

- We are the first to investigate the aspect missing issue of CVEs. We analyze 27,130 CVEs over the past 20 years to develop rules for extracting aspects from their descriptions. We analyze the characteristics and missing severity of six vulnerability aspects.
- We design a machine learning-based approach (**PMA**) for predicting the missing key aspects of CVEs. Our model systematically considers variations in input formats, word embeddings, model architectures, and neural network designs.
- We conduct extensive experiments to demonstrate the effectiveness, predicting performance, usefulness of our approach in terms of vulnerability description augmentation.
- We also conduct experiments to show that our method can improve the performance of vulnerability severity level prediction, and can predict threshold scores more accurately.

Finally, we remark that **PMA** can be used to recommend and further help to complete missing aspects in existing CVEs and help vulnerability discoverers submit more complete CVEs, potentially enabling more research and analysis using vulnerability descriptions. We leave these topics as our future work.

2 DETECTING MISSING KEY ASPECTS

In this section, we introduce the six key aspects for CVE descriptions, discuss how to extract these key aspects, evaluate the quality of the extracted key aspects, and analyze the missing status of different aspects in CVE descriptions.

2.1 Preliminaries of CVE Key Aspects

CVE suggests two description templates [29]: 1) [Vulnerability Type] in [Component] in [Vendor][Product][Version] allows [Attacker Type] to [Impact] via [Attack Vector]; 2) [Component] in [Vendor][Product][Version][Root Cause], which allows [Attacker Type] to [Impact] via [Attack Vector]. These two templates identify six key aspects for describing CVEs, as explained below.

Vulnerability type (Vul-Type) identifies an abstract software weakness for a CVE, which is usually identified as an entry in Common Weakness Enumeration (CWE) [13]. When submitting a new CVE request [14], the reporter must specify the vulnerability type. The request site provides several common candidate software weaknesses for selection (see in Table 3). If the relevant weakness (e.g., PHP Remote File Inclusion (CWE-98)) is not in this list, the reporter can select "Other" or "Unknown", and may optionally mention the weakness in the description.

Root cause is an error in program design, value or condition validation, and system or environment configuration, which results in a CVE. SecurityFocus [51] abstracts the root causes of CVEs into 11 error classes (see in Table 3). But when submitting a new CVE request, specifying *root cause* is not enforced. The reporter may describe the *root cause* in free-form text, as shown in Fig. 1 and Fig. 2.

Affected product refers to [Component] in [Vendor][Product] [Version] information in the CVE description. It identifies software component in certain version(s) of a software product that has been affected by a CVE. As the examples in Fig. 1 and Fig. 2 show, affected components can be source code file, function, or executable. When submitting a new CVE request, the reporter must provide affected product(s) and version(s), and product vendor(s).

Attacker type describes the mechanism by which an attacker may exploit a CVE. The CVE request site provides five mechanisms for selection: authenticated, local, remote, physical, and context dependent. Attacker type is an optional field. That is, the reporters leave this field unspecified or select other. But they may mention attacker type in the CVE description (see Fig. 1 for an example).

Impact indicates what the attacker gains by exploiting this vulnerability. The CVE request site provides four common impacts for selection: code execution, information disclosure, denial of service, escalation of privileges. Impact is also an optional field, which can be left unspecified. But the reporter generally describes the impact in the CVE description (see Fig. 1 and Fig. 2 for examples).

Attack vector describes the method of exploitation, for example, to exploit vulnerability, someone must open a crafted JPEG file. Specifying attack vector is not enforced. It may be mentioned in the CVE description (see Fig. 1 and Fig. 2 for examples). We manually label *attack vector* descriptions into five common types: via field, arguments or parameters, via some crafted data, by executing the script, HTTP protocol correlation, call API.

When submitting a new CVE request, the reporter provides a free-form textual description of the vulnerability, which may or may not cover all the six key aspects. The submission form provides pre-defined options for vulnerability type, attacker type, and impact. But the reporter may select "Other" if the pre-defined options are not appropriate for the reported vulnerability or leave the options unspecified. As such, not all CVEs describe all six aspects (see in Fig. 2).

2.2 Aspect Detection in CVE Descriptions

To understand the missing of the six key aspects in CVE descriptions, we first need to extract these aspects from the description text. To that end, we randomly sample 20% of the CVEs (27,130 CVEs in total) for each year from 1999 to 2020, and manually label the key aspects in the CVE descriptions. We observe that 71% of CVE descriptions follow the suggested templates [29], such as those in Fig. 1 and Fig. 2(a)(b). 29% of CVE descriptions do not follow the suggested templates, such as those in Fig. 2(c)(d). However, even for those non-template-following CVE descriptions, the descriptions of CVE aspects still exhibit similar patterns. Due to the input assistance of the CVE request website, we observe commonly used phrases or their variants for *vulnerability type, attacker type*, and *impact*.

Based on our observation of the aspect-phrase and sentence patterns in CVE descriptions, we develop a set of regular expression patterns to extract the six key aspects from CVE descriptions (see in the GitHub repository: https://github.com/pmaovr/Predicting-Missing-Aspects-of-Vulnerability-Reports). First, based on an advantageous matching technique "Gazetteer" [34, 47], we build a gazetteer (the gazetteer size is 3,685) commonly used for *vulnerability type, root cause, impact, attacker type*, and *attack vector*, as well as a gazetteer for product and vendor names from CVE Details. CVE Details extracts and displays the production, version, and vendor names of the vulnerabilities, which can be easily obtained. We also define sentence-level patterns that represent the common appearance order of different aspects in CVE descriptions.

We adopt Stanford CoreNLP [35] to parse a CVE description and obtain POS tags of this sentence. Some of the hard-to-identify aspects of the vulnerability descriptions (e.g., *root cause*) need to match the phrasal verbs that precede the sentence. Next, we combine gazetteer matching and POS pattern matching to decide the candidates of certain CVE aspects. We also use POS to exclude some aspects that are obviously wrong. For example, if the *affected products* we extract start with a phrasal verb, that aspect is likely to be wrong. Finally, we examine the candidates against the two official description templates and other general sentence-level patterns (see in the GitHub repository) in order to filter out false positive aspect candidates. We write code to determine whether the semantic structure and keywords in the template are present in the vulnerability descriptions, and use these templates to write regular expression code to carry out pattern matching on the vulnerability descriptions. For example, *attacker type, impact,* and *attack vector* often appear together in the form of "allow [attacker type] to [impact] via [attack vector]" or "[attacker type] performs [attack

Table 2. Variations of the extracted aspect descriptions

Aspect	CVE-ID	Descriptions
Vulnerability Type	CVE-2017-11507	cross-site scripting (XSS) vulnerability
	CVE-2018-16481	XSS vulnerability
	CVE-2018-10937	cross site scripting flaw
Root cause	CVE-2008-1419	does not properly handle ...
	CVE-2010-0027	does not properly process ...
	CVE-2015-1992	improperly processes ...
Affected product	CVE-2006-3500	The dynamic linker (dyld) in Apple ...
	CVE-1999-0786	The dynamic linker in Solaris
	CVE-2013-0977	dyld in Apple iOS before 6.1.3 and Apple TV ...
Impact	CVE-2011-4129	obtain sensitive information
	CVE-2005-2436	obtain sensitive data
	CVE-2002-0257	obtain information from other users
Attacker type	CVE-2018-1000634	user with privilege
	CVE-2018-1000084	low privilege user
	CVE-2016-9603	A privileged user/process
Attack vector	CVE-2018-12581	use a crafted database name
	CVE-2019-11768	a specially crafted database name can be used
	CVE-2012-1190	via a crafted database name

vector] in order to [impact]". Meanwhile, "executing the script" can belong to either *attack vector* or *impact* aspect, while it is exactly an *attack vector* when it appears in the sentence "By executing the script ...". Note that, some vulnerability descriptions will describe the name of the vendor at the beginning and the product version number of the *affected product* at the end. We will extract the descriptions of these two parts at the same time to stitch together a complete *affected product*.

2.3 Accuracy of CVE Aspect Extraction

We apply the aspect detection method to the rest 108,500 CVEs. We extract 41,003, 15,129, 92,132, 89,053, 73,134, and 67,188 instances of the *vulnerability type, root cause, affected product, impact, attacker type,* and *attack vector*, respectively. Considering large numbers of instances to examine, we adopt a statistical sampling method [50] to evaluate the accuracy of the extracted aspect instances. Specifically, we sample and examine the minimum number MIN of data instances. MIN is determined by $n_0/(1 + (n_0 - 1)/populationsize)$ where $n_0 = (Z^2 * 0.25)/e^2$, and Z is the confidence level's z-score and e is the error margin. In this work, we consider 5% error margin at 95% confidence level. At this setting, we examine 384 extracted instances for each aspect. One author labels the sampled instances, and the other author validates the results. The two authors discuss to resolve the disagreements. The extraction accuracy is 97%, 96%, 96%, 98%, 99%, and 98% for the *vulnerability type, root cause, affected product, impact, attacker type,* and *attack vector*, respectively. Table 2 shows some examples of the extracted aspect phrases. We can see that our aspect extraction method is flexible to handle the variations of aspect extraction.

2.4 Missing and Distribution of CVE Aspects

Based on the extracted CVE aspects, we analyze the missing of key aspects in CVEs. By observing on different severities of information missing for different aspects, we find that about 43.8% of CVEs

Table 3. Class distribution of CVE aspects

Vulnerability Type			
Cross site scripting (CWE-79)	29.5%	SQL injection (CWE-89)	17.8%
Buffer Overflow (CWE-119)	17.1%	Directory Traversal (CWE-32)	8.9%
Cross-site request forgery (CWE-352)	7.1%	PHP file inclusion (CWE-98)	5.7%
Use-after-free (CWE-416)	3.2%	Integer overflow (CWE-680)	2.6%
Untrusted search path (CWE-426)	1.7%	Format string (CWE-134)	1.6%
CRLF injection (CWE-93)	0.6%	XML External Entity (CWE-661)	0.3%
Others	4.0%		
Root Cause			
Input Validation Error	51.7%	Boundary Condition Error	24.5%
Failure to Handle Exceptional Conditions	11.7%	Design Error	11.0%
Access Validation Error	0.7%	Atomicity Error	0.1%
Race Condition Error	0.1%	Serialization Error	0.1%
Configuration Error	0.1%	Origin Validation Error	0.1%
Environment Error	0.1%		
Attack Vector			
Via field, arguments or parameter	51.7%	Via some crafted data	17.1%
By executing the script	14.0%	HTTP protocol correlation	4.4%
Call API	3.3%	Others	8.0%
Attacker Type			
Remote attacker	72.8%	Local attacker	11.1%
Authenticated user	8.1%	Context-dependent	2.9%
Physically proximate attacker	0.3%	Others	4.7%

describe *vulnerability type*, about 15.2% of CVEs describe *root cause*, about 3% of CVEs describe both *vulnerability type* and *root cause*, 62% of CVEs describe *attack vector*, and 72% of CVEs describe *attacker type*. Meanwhile, almost all (over 99%) CVEs list *affected product*, and about 94% of CVEs present *impact*. We also uncover that about 31% of CVEs miss one aspect, 39% miss two, and 28% miss three or more aspects. Table 3 demonstrates the class distributions of the main extracted aspects, which is obviously imbalanced with a long-tailed distribution. In addition, classes with the frequencies < 0.1 are grouped as Others. Note that, we exclude *affected product* and *impact* because they do not suffer from serious information missing.

3 AUGMENTING MISSING KEY ASPECTS

Motivated by the missing of key aspects in CVE descriptions, we design a neural-network based approach named **PMA** for predicting the missing aspects based on the known aspects in a CVE description. We first give an overview of our approach (§ 3.1) and then describe the design of neural-network classifier (§ 3.2~§ 3.4).

3.1 Approach Overview

We formulate the prediction task as a multi-class classification problem for each aspect. Considering the severity of the information missing (see § 2.3), we predict four aspects: *vulnerability type, root cause, attacker type*, and *attack vector*. Each aspect has a corresponding multi-class classifier, and the class labels for each aspect-specific classifier are summarized in Table 3.

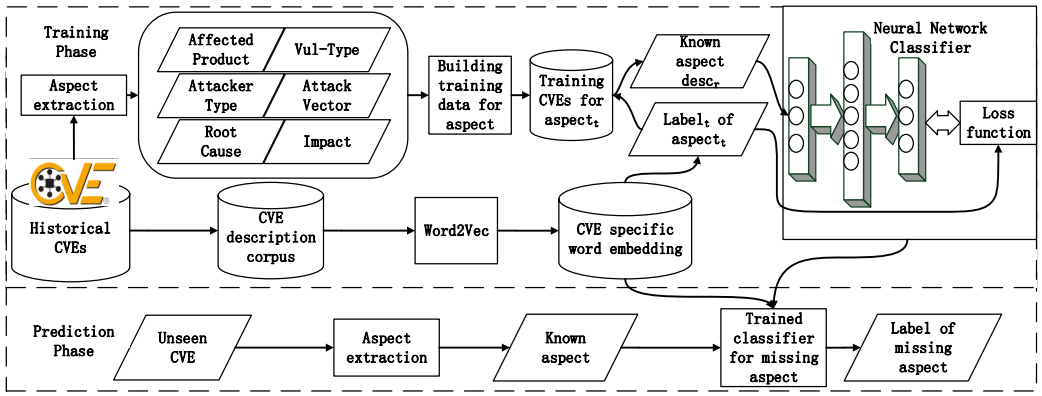


Fig. 3. Overview of **PMA**

As shown in Fig. 3, **PMA** consists of a training phase and a prediction phrase. Training phase uses the historical CVE descriptions to train aspect-specific neural network classifiers. It first uses the aspect extraction method in § 2.2 to extract six CVE aspects from the historical CVE descriptions. Then, we prepare the training data for each aspect t to be predicted. For each CVE that contains the aspect t, a training instance is created with the class label of t (denoted as $label(t)$) as the expected output and the description of rest of aspects $r \in R$ $(1 \leq |R| \leq 5)$ (denoted as $desc(r)$) as the input. From such training data, the neural network classifier is trained to extract syntactic and semantic features from the input aspect descriptions and capture the intrinsic correlations between these input features and the output class label.

At the prediction phrase, given an unseen CVE description, we first extract the CVE aspects present in the description. For each missing aspect, the trained aspect-specific classifier takes as input the aspects present in the description and predicts as output the most likely class label of the missing aspect.

The neural network classifier consists of three layers: an input layer that represents the input text in a vector representation (e.g., word embedding) (§ 3.2); a neural-network feature extractor that extracts syntactic and semantic features from the input text (§ 3.3); and an output classifier that makes the prediction based on the extracted features (§ 3.4). Next, we describe the design of these three layers in details.

3.2 Input Text and Representation

The raw input into classifier is the textual description of CVE aspects, such as those sentence fragments highlighted in Fig. 1 and Fig. 2. In this work, we consider three formats of raw input text: **1)** the sequence of separate aspect descriptions in the original appearance order (denoted as *i-ao*); **2)** the sequence of separate aspect descriptions in a random order (denoted as *i-ar*); **3)** the original CVE description containing all input aspects (denoted as *i-fu*). *i-ar* allows us to investigate the impact of the appearance order of CVE aspects, and *i-fu* allows us to investigate the impact of additional sentence parts in the original CVE description. Additional parts refer mostly to preposition, pronoun and/or determiner that connect separate aspect descriptions into a more complete sentence.

For *i-ao*, we delete the prepositions and connectives between all aspects and the note information in the vulnerability, and only aspects were retained and spliced into a continuous word sequence. The *i-ar* is transformed from *i-ao*, in this input format, the description of vulnerability will be extracted, the order of aspects will be randomly scrambled, and the new word sequence will be spliced. For

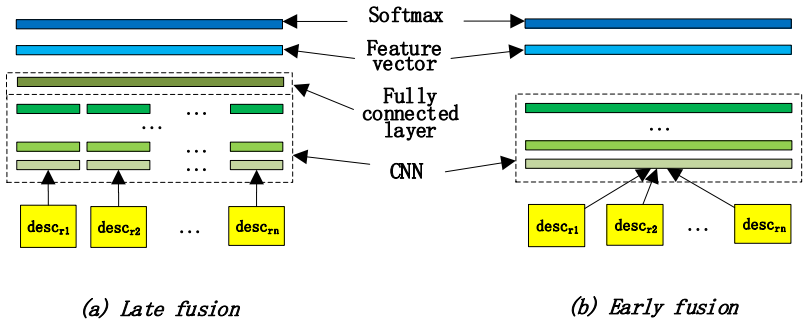


Fig. 4. Two model architectures

i-fu, we use a complete description of vulnerability, and reserve prepositions, connectives, note information, and other information. These word sequences are then converted into word vector sequences.

We use word embeddings to further represent word into vector, which can capture rich syntactic and semantic features of each word in a low-dimensional vector [27, 42, 60]. Both general word embeddings (denoted as we_g) and domain-specific word embeddings (denoted as we_d) are applied in this work. For general word embeddings, we adopt pre-trained word embeddings on the corpus of Google News from the official Word2Vec [20]. For domain-specific word embeddings, two corpora are taken for pre-training: one is from CVE descriptions and another is from the vulnerability report in SecurityFocus [51]. We set the vocabulary size as 50,000 to learn domain-specific word embeddings by using continuous skip-gram model [36] (the Python implementation in Gensim [46]). The output of word embedding is a word dictionary, each of which has a d-dimensional vector. We set d at 300 as in existing studies [19, 23, 56].

The input text is represented into a $N \times d$ matrix $v(w_1) \oplus v(w_2) \oplus ... \oplus v(w_N)$, where N denotes the number of words w_i ($1 \leq i \leq N$), $\oplus$ is vector concatenation and $v(w_i)$ returns the word embedding of the word w_i in the dictionary. We randomly initialize corresponding word vectors to deal with these Out-of-Vocabulary (OOV) words [64]. In addition, we set an input aspect as an empty string if the CVE does not contain this input aspect, in order to keep the model architecture consistent.

3.3 Neural Network Feature Extractors

3.3.1 Model Architecture. As our input consists of separate CVE aspects, we design two model architectures to investigate the effective mechanism for incorporating CVE aspects and capturing their intrinsic correlations: early fusion versus late fusion. As shown in Fig. 4, early fusion architecture first concatenates the input matrix of each aspect into one input matrix, which is fed into a single neural network to extract and fuse features from different aspects. In contrast, late fusion architecture feeds the input matrix of each aspect into a neural network separately and then fuse the output feature vector of the separate networks by a fully connected layer. All neural networks share the same network configuration, but they will learn different weights in different architectures. Both early fusion and late fusion are applicable to the input formats *i-ao* and *i-ar*. But only early fusion is applicable to the input format *i-fu*, because *i-fu* merges the input CVE aspects into a whole sentence.

3.3.2 Backbone Network. We consider two popular neural networks for text classification in the literatures [22, 24, 26]: Convolutional Neural Network (CNN) and Bi-directional Long-Short Term Memory (BiLSTM), which the former is superior when capturing important words and phrases,

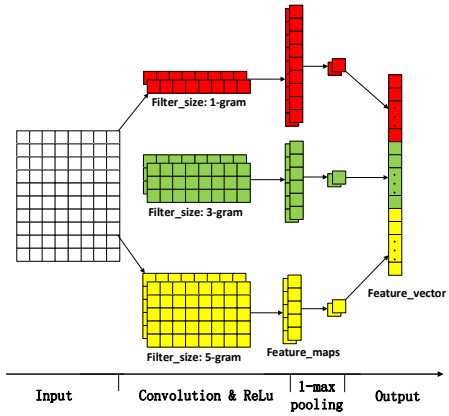


Fig. 5. 1-layer CNN

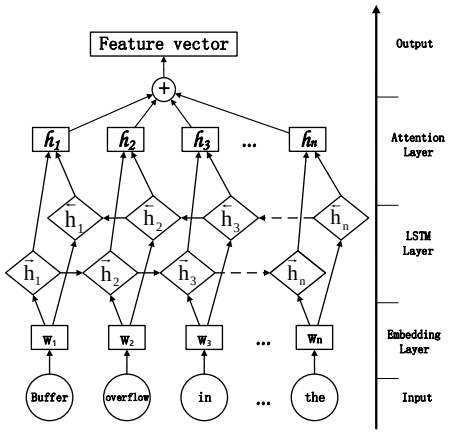


Fig. 6. 1-layer Bi-LSTM with attention

and the later is better at capturing longer-range dependencies in text. Then, we implement six neural network model variants of both of them as follows.

1-layer CNN. CNN is a mature neural network feature extractor. Compared with RNN, the window sliding of CNN has no sequential relationship at all, and different convolution kernels have no interaction before. Therefore, it has a very high degree of parallel freedom. Fig. 5 presents the 1-layer CNN model which mainly has convolution layers and 1-max pooling layers. The convolution layer applies M filters to the input matrix of word embeddings. A filter is a $h \times d$ matrix where h is the window size and d is the word embedding dimension 300. Further, in this neural network nodel, we use three different window sizes h=1, 3, 5, which refers to the number of consecutive words (e.g., n-gram). That is, the filters extract features from 1-grams, 3-grams, and 5-grams, respectively. For each word window, a filter calculates a real value, and then feeds it into the non-linear activation function ReLU [18]: $ReLU(x)$=$max(0, x)$. This filter scans the input word sequence with hyperparameter $stride$=1 (zero-padding at both ends to allow the filter to extract features from the beginning and the end of the input sentence), and generates a feature map of the input sequence length. Next, a 1-max pooling is applied to to obtain the most significant feature from this feature map. Note, we use M=128 filters to learn complementary features from the same word windows. That is, 1-layer CNN neural network model produces a 128-dimensional feature vector for each window size. Finally, The feature vectors of all windows sizes are concatenated into an output feature vector for later classifier.

2-layers CNN. This is a deeper variant of the 1-layer CNN. Both 2-layers CNN and 1-layer CNN have the same first CNN layer. But in 2-layer CNN, the three feature vectors generated by the first CNN layer are directly sent into one more CNN layer (convolution and 1-max-pooling). And this successive CNN layer adopts M (M = 128) filters with the window size h = 3, to reprocess the 128-dimensional feature vector. So, after a series of convolution, ReLu activation and max-pooling, the second CNN layer outputs three 128-dimensional feature vectors similarly and then concatenates them into an output feature vector for the later classifier.

1-layer BiLSTM. Bi-directional Long Short Term Networks, commonly referred to simply as BiLSTM, are a specific type of RNN designed to avoid long-term dependency problems. In this work, we can learn long-term dependency information about vulnerability descriptions. Our proposed 1-layer BiLSTM model includes a forward LSTM $\overrightarrow{lstm_f}$ network that reads the input

from w_1 to w_N, and a backward LSTM $\overleftarrow{lstm}_b$ that reads from w_N to w_1: $\overrightarrow{h}_i = \overrightarrow{lstm}_f(w_i), i \in [1, N]$, $\overleftarrow{h}_i = \overleftarrow{lstm}_b(w_i), i \in [N, 1]$. $\overrightarrow{h}_i$ and $\overleftarrow{h}_i$ are forward and backward hidden vectors for word w_i, respectively. Note that, both forward and backward LSTM have 192 LSTM cells, and other parameters of $\overrightarrow{lstm}_f$ and $\overleftarrow{lstm}_f$ will be learned during model training. We obtain the hidden vector h_i for w_i via concatenating $\overrightarrow{h}_i$ and $\overleftarrow{h}_i$, i.e., $h_i = \overrightarrow{h}_i \oplus \overleftarrow{h}_i$ (h_i is 384-dimensional vector). Thus, we gain BiLSTM output vector h_i of the input word w_i, which encodes both preceding and succeeding sentence context centered around w_i. Finally, we concatenate the last hidden vectors h_1 and h_N into an output feature vector for the classifier.

2-layers BiLSTM. It is a deeper variant of 1-layer BiLSTM. Its first layer is just similar to a 1-layer BiLSTM, while the output vector of the first BiLSTM layer is considered as input into the second BiLSTM layer which also uses 192 LSTM cells for encoding. The last hidden vectors h_1 and h_N by the second BiLSTM layer are concatenated into an output feature vector for the later classifier.

Attention layer for BiLSTM. The attention [59] is a technique that enables models to focus on important information and fully learn, has become a new research hotspot in recent years. For both 1-layers BiLSTM and 2-layers BiLSTM, we can add an attention layer on top of the last BiLSTM layer to weight the importance of words. Therefor, we construct the 1-layer BiLSTM+Attention and 2-layers BiLSTM+Attention. We only give the 1-layer BiLSTM+Attention in Fig. 6 due to space limitation. The new attention layer calculates the word attention weight $\alpha_i = \exp\left((u_i)^\top u_w\right) / \sum_i \exp\left((u_i)^\top u_w\right)$ where $u_i = \tanh\left(W_w h_i + b_w\right)$. Its input is the output h_i of the BiLSTM layer. Specifically, h_i is first fed into a one-layer feed-forward neural network to get u_i as a hidden representation. And W_w and b_w are learnable parameters of the attention layer. Then, the importance of word w_i, also the normalized importance weight α_i, is measured by calculate the similarity of u_i with a word-level context vector u_w (randomly initialized and learned during training) through a softmax function. After getting the weighted sum (output related to both Bi-LSTM layer and attention layer): $\sum_{i=1}^{n} \alpha_i h_i$, we can obtain the final output feature vector for later classifier.

3.4 Predicting Missing Aspects of CVEs

Since the output of our model is a mutually exclusive category and only one category can be selected, the softmax function is used in this paper to calculate the original output value of the network. Given the output feature vector F by each neural-network feature extractor, a softmax classifier predicts the probability distribution $\hat{y}$ over the m class labels of a particular CVE aspect, i.e., $\hat{y} = softmax\left(WF + b\right)$, where $\hat{y}$ is the vector of prediction probabilities over the m class labels, and W and b are the learnable parameters of the classifier. Meanwhile, the all learnable parameters in each neural-network feature extractor and the softmax classifier are trained to minimize the cross-entropy loss between the predicted labels and the ground-truth labels: $L\left(\hat{y}, y\right) = -\sum_{i=1}^{K}\sum_{j=1}^{m} y_{ij} \log\left(\hat{y}_{ij}\right)$, where K denotes the number of training samples. y_{ij} is the ground-truth label of the jth class (1 for ground-truth class, 0 otherwise) for the ith training example, and $\hat{y}_{ij}$ is the predicted probability of the jth class for the ith training example. In addition, the loss gradient is back-propagated to update all learnable parameters of both neural networks and classifier.

4 EXPERIMENTS

In this section, we conduct a series of experiments to investigate and answer to the following five research questions:

- **RQ1: Design of neural network classifier**. How do different input formats, word embeddings, model architectures, and neural network designs affect the prediction performance?

Table 4. Aspect-specific CVE datasets

To-be-predicted aspect Size	Vul-Type 36,103	Root cause 20,813	Attack vector 35,289	Attacker type 43,330
% with vulnerability type	100%	61%	63%	66%
% with root cause	38%	100%	39%	37%
% with attack vector	75%	72%	100%	78%
% with attacker type	86%	86%	92%	100%

- **RQ2: Ablation study**. What is the most or least prominent aspect or aspect combination for predicting a specific aspect?
- **RQ3: Prediction on future CVEs**. How well can our approach predict key aspects of CVEs published after the training CVEs?
- **RQ4: Prediction on updated CVEs in NVD**. How well can our approach predict key aspects of the updated CVEs collected from NVD compared with the manually augmented descriptions?
- **RQ5: Prediction on severity level of vulnerabilities**. How does our approach contribute to other applications based on the descriptions of CVEs (e.g., vulnerability severity level prediction)?

4.1 Experiment Setup

We describe the CVE dataset used in our experiments, our model training setting, and the performance evaluation metrics.

4.1.1 CVE Dataset. CVE list can be downloaded from the official CVE website [39]. In this paper, we download the CVE list that contains 136,639 CVEs from January 1999 to August 2020. We use the aspect extraction method in § 2.2 to extract CVE aspects from these CVEs. Our evaluation in § 2.3 confirms the high accuracy of the extracted CVE aspects.

To evaluate our prediction method, we collect 52,306 CVEs whose descriptions contain at least four aspects. As almost all CVEs contain *affected product* and *impact* aspects (see § 2.3), which means the CVEs in the dataset contain at least two of the other four aspects. The reason is that too little information of the known aspects will have an impact on the prediction of unknown aspects, which is also not conducive to the exploration of the correlation between various aspects of CVEs. We regard one aspect as the "missing" aspect and the rest as "known" aspects for prediction. This guarantees that we have sufficient data to study aspect fusion and ablation. For each to-be-predicted aspect *(vulnerability type, root cause, attacker type,* or *attack vector*), we build an aspect-specific dataset for classifier training and testing from these 52,306 CVEs according to the method in § 3.1. Table 4 summarizes the information of the four aspect-specific datasets. For a specific to-be-predicted aspect (e.g., *vulnerability type*), the other three aspects (e.g., *root cause, attacker type,* or *attack vector*) of the CVEs used as input may also missing, except affected product and impact, which is a real situation of the CVE description.

4.1.2 Model Training. We implement the proposed neural network classifier in TensorFlow [1]. Each To-be-predicted aspect has its own classifier. As discussed in § 3, we have different choices for input format, word embedding, model architecture and network design when implementing a classifier. All the classifiers are trained in the same setting. Specifically, we train each model for 256 iterations with a batch size of 128, set learning rate at 0.001, and use Adam [32] as the optimizer. All experiments run on a NVIDIA Tesla M40 GPU machine, and the video memory size is 24 GB. The CPU is Intel(R) Xeon(R) Gold 5115 CPU and the total memory of the machine is 65 GB.

4.1.3 Evaluation Metrics. The multi-class classification results can be represented in a $m \times m$ confusion matrix M, where m is the number of class labels (see Table 3). We use Precision (Pre), Recall (Re), and F1-score (F1) to evaluate the effectiveness of multi-class classification [53, 55, 57, 62]. Precision for a label L_j of an aspect A represents the proportion of the CVEs whose missing aspect A is correctly predicted as L_j among all CVEs whose missing aspect A is predicted as L_j. Recall for a label L_i of an aspect A is the proportion of the CVEs whose missing aspect A is correctly predicted as L_i compared with the number of ground-truth CVEs whose missing aspect A is actually L_i. F-score is the harmonic average of the precision and recall. The overall performance of a classifier is the weighted average of the evaluation metrics of each class label.

4.2 Design of Neural Network Classifier (RQ1)

4.2.1 Motivation. The design of our neural network classifier considers three input formats (i-ao: separate CVE aspects in the original order appearing in CVE descriptions, i-ar: separate CVE aspects in random order, or i-fu: original CVE descriptions), three word embeddings (Google News [20], SecurityFocus [51], or CVE-specific [39]), two model architecture (early aspect fusion or late aspect fusion), and six specific neural network design (CNN or BiLSTM, 1-layer, or 2-layer, BiLSTM with/without attention layer). We want to investigate the impact of these design options on the prediction performance and identify the most effective design of neural network classifier.

4.2.2 Approach. We conduct four experiments to evaluate the impact of input format, word embedding, model architecture, and network design, respectively. For the experiments on one dimension, we use the most effective options for the other three dimensions. Specifically, **1)** for input format experiments, we use CVE-specific word embeddings, early fusion architecture, and 1-layer CNN. **2)** For word embedding experiments, we use separate CVE aspects in original order, early fusion architecture, and 1-layer CNN. **3)** For model architecture experiments, we use separate CVE aspects in original order, CVE-specific word embeddings, and 1-layer CNN. **4)** For network design experiments, we use separate CVE aspects in original order, CVE-specific word embeddings, and early fusion architecture. Han et al. [23] pointed out in their study of vulnerability severity level prediction task that the window size of 1, 3, and 5 could best capture the information characteristics of CVE vulnerabilities. Therefore, we also adopt the same experimental setting in this RQ. Domain-specific word embeddings use vulnerability descriptions in CVE and SecurityFocus directly as word vector training input, which contains a large number of vocabulary in the field of software security. The vocabulary sizes of CVE and SecurityFocus are about 62,000 and 56,000, respectively. This experiment setting helps to reduce the large number of experiments by the full Cartesian product combination of the design options, and also facilitate the analysis of each design dimension while fixing the other three dimensions. To ensure the reliability of our experiments, we perform 10-fold cross validation in all the experiments. For each fold, we use 80%, 10%, and 10% of data for model training, hyperparameter optimization, and testing, respectively. We conduct Wilcoxon signed-rank test [54] on F1-score between different experiment settings. p-value < 0.05 is considered statistically significant (marked by * in the results tables).

4.2.3 Results. **Input Formats.** As shown in Table 5, the three input formats do not obviously affect the prediction of the four CVE aspects, with only 0.002~0.009 difference in F1 across the three input formats. The only exception is the prediction of *root cause* by separate aspects in random order (i-ar), but the difference in F1 is not very large either. This suggests that the phrase-level information in the CVE aspects alone can support reliable prediction. The presence or absence of the additional information (mostly prepositions, pronouns, determiner) that connect CVE aspects in the original CVE descriptions does not significantly affect the prediction. Furthermore, the

Table 5. Impact of input formats

		Vul-Type	Root Casue	Attack Vector	Attacker Type
Pre	i-ao	0.945	0.779	0.708	0.884
	i-ar	0.943	0.746	0.701	0.882
	i-fu	0.946	0.783	0.703	0.888
Re	i-ao	0.945	0.793	0.716	0.897
	i-ar	0.945	0.770	0.710	0.892
	i-fu	0.946	0.796	0.717	0.899
F1	i-ao	0.943	0.780	0.704	0.885
	i-ar	0.943	0.745	0.699	0.880
	i-fu	0.946	0.788	0.706	0.889

Table 6. Impact of word embeddings

		Vul-Type	Root Cause	Attack Vector	Attacker Type
Pre	CVE	0.946	0.783	0.703	0.888
	SecurityFocus	0.942	0.779	0.701	0.883
	*Google news	0.932	0.759	0.687	0.869
Re	CVE	0.946	0.796	0.717	0.899
	SecurityFocus	0.944	0.792	0.716	0.894
	*Google news	0.935	0.784	0.688	0.885
F1	CVE	0.946	0.788	0.706	0.889
	SecurityFocus	0.942	0.783	0.703	0.883
	*Google news	0.933	0.761	0.687	0.871

prediction is not sensitive to the appearance order of different aspects in the CVE descriptions. Therefore, we use separate aspects in the original appearance order (i-ao) as the default option.

General versus Domain-specific Word Embeddings. Table 6 shows that domain-specific word embeddings (CVE and SecurityFocus) support more accurate prediction in all four aspects than general word embeddings (Google News). The differences are statistically significant in F1. However, the two domain-specific word embeddings have marginal differences. This result can be attributed to two reasons. First, CVE and SecurityFocus use text training word vectors in the field of software security, in which the words contain a lot of professional terms, names of software and its components, etc. Domain-specific word embeddings learn meaningful embeddings for domain-specific terms (e.g., CRLF, XSS, and DoS), which may be regarded as out-of-vocabulary words and not semantically represented by word vectors in general word embeddings. Second, the size of domain-specific vocabularies is around 60,000, and although the general vocabularies are large, they are rarely used in vulnerability descriptions. Domain-specific corpora allow the learning of "purer" word embeddings highly relevant to a particular domain, while the word embeddings learned from general text may embed some unnecessary "noise" irrelevant to the particular domain.

Early Fusion versus Late Fusion. Table 7 shows that early fusion architecture performs better than late fusion architecture (statistically significant for all evaluation metrics). This suggests that using a single network to extract and fuse features directly from all input CVE aspects is much more effective than extracting features from each CVE aspect separately and only fusing the features of

Table 7. Impact of model architectures

		Vul-Type	Root Cause	Attack Vector	Attacker Type
Pre	Early Fusion	0.946	0.783	0.703	0.888
	*Late Fusion	0.921	0.751	0.671	0.844
Re	Early Fusion	0.946	0.796	0.717	0.899
	*Late Fusion	0.927	0.770	0.680	0.872
F1	Early Fusion	0.946	0.788	0.706	0.889
	*Late Fusion	0.923	0.755	0.669	0.850

Table 8. Impact of neural network designs

		Vul-Type	Root Cause	Attack Vector	Attacker Type
Pre	1-L CNN	0.946	0.783	0.703	0.888
	2-L CNN	0.933	0.765	0.673	0.852
	1-L BiLSTM	0.939	0.761	0.682	0.867
	2-L BiLSTM	0.939	0.770	0.688	0.870
	1-L BiLSTM+Attention	0.941	0.769	0.690	0.873
	2-L BiLSTM+Attention	0.943	0.778	0.692	0.876
Re	1-L CNN	0.946	0.796	0.717	0.899
	2-L CNN	0.935	0.775	0.701	0.878
	1-L BiLSTM	0.938	0.778	0.706	0.882
	2-L BiLSTM	0.941	0.780	0.703	0.883
	1-L BiLSTM+Attention	0.943	0.778	0.713	0.887
	2-L BiLSTM+Attention	0.945	0.792	0.714	0.889
F1	1-L CNN	0.946	0.788	0.706	0.889
	2-L CNN	0.932	0.768	0.677	0.859
	1-L BiLSTM	0.938	0.765	0.684	0.871
	2-L BiLSTM	0.940	0.770	0.683	0.874
	1-L BiLSTM+Attention	0.940	0.770	0.692	0.873
	2-L BiLSTM+Attention	0.943	0.778	0.694	0.878

different aspects at the end. In addition, the training time of the early fusion model is shorter than that of the late fusion model, in which the training time of the early fusion model is about 16 hours and the training time of the late fusion model is around 33 hours in our experiments. Therefore, we use early aspect fusion as the default option.

Neural Network Variants. Table 8 presents our experimental results on the six variants of neural network feature extractor. We can see that 1-layer CNN outperforms the other five variants. So we use 1-layer CNN as the baseline to analyze the performance of the other five variants. Compared with 1-layer CNN, 2-layer CNN has worse but statistically non-significant performance for predicting vulnerability type and attacker vector, but has statistically significant worse performance for predicting root cause and attacker type. In addition, the training time of the 1-layer BiLSTM model is the shortest (about 15 hours), followed by the 1-layer CNN model, the training speed of 2-layers CNN is the slowest among the six neural network designs (about 23 hours). This suggests that deeper CNN is less appropriate than 1-layer CNN in our text classification task. The performance of the four BiLSTM networks are very close. Neither deeper BiLSTM nor attention mechanism statistically significantly improve the prediction performance. 1-layer CNN has better performance

Table 9. Ablation results for predicting vulnerability type

Ablated aspect	Root cause	Affected product	Impact	Attacker type	Attack vector
Pre	0.943	0.925	0.821	0.939	0.888
Re	0.943	0.927	0.822	0.941	0.896
F1	0.943	0.925	**0.821**	0.939	**0.890**

Table 10. Ablation results for predicting root cause

Ablated aspect	Vul-Type	Affected product	Impact	Attacker type	Attack vector
Pre	0.740	0.734	0.739	0.781	0.780
Re	0.751	0.741	0.755	0.793	0.795
F1	0.745	0.730	0.736	0.785	0.784

Table 11. Ablation results for predicting attacker type

Ablated aspect	Vul-Type	Root cause	Affected product	Impact	Attack vector
Pre	0.852	0.873	0.850	0.883	0.864
Re	0.876	0.892	0.874	0.895	0.871
F1	0.861	0.878	0.847	0.881	0.863

Table 12. Ablation results for predicting attack vector

Ablated aspect	Vul-Type	Root cause	Affected product	Impact	Attacker type
Pre	0.659	0.696	0.568	0.680	0.670
Re	0.693	0.701	0.601	0.700	0.674
F1	0.665	0.695	**0.572**	0.683	0.669

than 2-layer BiLSTM with attention (the overall best BiLSTM performer). Although the performance differences are not large, the differences in F1 are statistically significant for predicting all four CVE aspects. This result suggests that using CNN to extract important words/phrase features fits better for our text classification task than using LSTM to learn long-range sentence features.

Answer to RQ1. According to our experiments on input formats, word embeddings, model architectures, and network designs, the most effective design of the classifier takes as input separate CVE aspects in original order, uses CVE-specific word embeddings to represent input text, and adopts early-fusion architecture and 1-layer CNN as feature extractor.

4.3 Ablation Study (RQ2)

4.3.1 Motivation. As shown in Table 4, it is unrealistic to assume that all other five CVE aspects are available for predicting a particular aspect. In this RQ, we want to investigate the impact of certain CVE aspects unavailable as input on the accuracy of predicting a particular aspect. This study has two important goals. First, it identifies stronger correlations (if any) among some CVE aspects than others. Second, it identifies the minimum subset of known aspects required for making reliable prediction of a particular aspect. This also help us to understand the practicality of **PMA**.

4.3.2 Approach. For a particular aspect, we conduct five experiments in this section. In each experiment, we ablate one of the other five aspects in the aspect-specific dataset, which produces an ablation dataset without the ablated aspect. For example, for *vulnerability type*, we obtain five datasets without *root cause, affected product, impact, attacker type*, or *attack vector* for the five ablation experiments, respectively. Although the experiments in RQ1/RQ2/RQ3/RQ4 do not assume the availability of all five aspects, the ablation of one aspect in this RQ means that we completely ignore this ablated aspect as known aspect for model input, even the CVEs describe this ablated aspect. We use the most effective classifier design identified in RQ1, and train and test the classifier on each ablation dataset. We also perform 10-fold cross-validation in all these experiments.

4.3.3 Results. Table 9~Table 12 show the results of our ablation study. We compare the performance metrics with those by 1-layer CNN in Table 8. As shown in Table 9, for predicting *vulnerability type*, ablating *impact* results in the most significant drop (12.5%, 0.821 vs. 0.946) in F1, followed by ablating *attack vector* (5.6% drop in F1, 0.890 vs. 0.946). In contrast, ablating *attacker type* and *affected product* have a less significant impact, with 0.7% and 2.1% drop in F1, respectively. Ablating *root cause* almost has no impact on predicting *vulnerability type*. As we discussed in § 2.3, over 94% of CVEs describe *impact* aspect. Therefore, our approach will not actually suffer from the performance degradation due to the unavailability of *impact* as input in practice. Although unavailable attack vector as input aspect affects the prediction of vulnerability type, our approach can still achieve high accuracy (0.89 in F1).

As shown in Table 12, for predicting *attack vector*, ablating affected product has the most significant *impact, resulting* in 13.4% drop in F1 (0.572 vs. 0.706). However, as almost all CVEs describe *affected product*, our approach will not actually suffer from the performance degradation due to the unavailability of *affected product* as input. Ablating the other four aspects results in much smaller drops (about 1.1%~4.1%) in F1. Among these four aspects, *vulnerability type* and *attacker type* have stronger correlations with *attack vector* than *impact* and *root cause*.

As shown in Table 10 and Table 11, there are no prominent aspect for predicting *root cause* and *attacker type* as *impact* for *vulnerability type* and *affected product* for *attack vector*. Specifically, for predicting *root cause*, ablating *vulnerability type*, *affected product*, or *impact* has relatively larger impact (about 5% drop in F1), compared with about 0.4% drop in F1 by ablating *attacker type* and *attack vector*. For predicting *attacker type*, ablating *affected product* results in relative larger drop in F1 (4.2%) than ablating the other four aspects.

Answer to RQ2. Our ablation study shows that omitting *impact* and *affected product* have a larger impact on predicting *vulnerability type* and *attack vector* compared to omitting other aspects (*root cause* and *attacker type*).

4.4 Prediction on Future CVEs (RQ3)

4.4.1 Motivation. Security vulnerabilities are constantly being discovered and reported. The future vulnerabilities may differ from current vulnerabilities. The vulnerability discoverers may not know some critical aspects of the vulnerability when submitting the report. And one of the goals of our work is to help these people submit their newly discovered vulnerabilities and to help the CVE authorities complete their newly submitted vulnerability descriptions. To test whether our work can help submit vulnerability reports, we need to make predictions about future CVEs. In this RQ, we want to investigate if our model trained with historical CVEs can effectively predict the missing aspects of future CVEs.

4.4.2 Approach. We construct a "future" dataset of 8,623 CVEs (from October 2016 to August 2020). 61%, 91%, 47%, and 49% of these CVEs miss *vulnerability type, root cause, attack vector*, and *attacker*

Table 13. Prediction performance on future CVEs

		Vul-Type	Root Cause	Attack Vector	Attacker Type
Pre	1-L CNN	0.891	0.717	0.621	0.820
	2-L CNN	0.881	0.711	0.600	0.802
	1-L BiLSTM	0.880	0.712	0.605	0.809
	2-L BiLSTM	0.857	0.711	0.609	0.805
	1-L BiLSTM+Attention	0.861	0.714	0.610	0.813
	2-L BiLSTM+Attention	0.873	0.715	0.615	0.816
Re	1-L CNN	0.889	0.717	0.623	0.825
	2-L CNN	0.880	0.710	0.603	0.803
	1-L BiLSTM	0.861	0.710	0.610	0.809
	2-L BiLSTM	0.863	0.700	0.611	0.809
	1-L BiLSTM+Attention	0.867	0.715	0.619	0.817
	2-L BiLSTM+Attention	0.870	0.713	0.619	0.818
F1	1-L CNN	0.880	0.714	0.611	0.814
	2-L CNN	0.850	0.702	0.589	0.780
	1-L BiLSTM	0.857	0.705	0.596	0.792
	2-L BiLSTM	0.859	0.702	0.599	0.799
	1-L BiLSTM+Attention	0.862	0.710	0.605	0.815
	2-L BiLSTM+Attention	0.869	0.711	0.607	0.812

type, respectively. 25% of these CVEs miss one aspect, and 75% miss two or more aspects. We train the model with our dataset of historical CVEs (§ 4.1.1), and use the trained model to predict the missing aspects in the future dataset. The experiment setting is different from the training/testing splitting of the historical dataset, because it enforces the time dimension of the CVE data. We use the same network designs and model configurations as in the previous section (§ 4.4.2).

4.4.3 Results. As shown in Table 13, similar to the prediction performance on historical CVEs (see Table 8), the performance of six model variants are close. Overall, 1-L CNN has slightly better performance, which achieves 0.88, 0.71, 0.61, and 0.81 in F1 for predicting *vulnerability type, root cause, attack vector*, and *attacker type*, respectively. Compared with the F1s of 1-L CNN on historical CVEs, the F1s on the "future" CVE drops 6.6%, 7.4%, 9.5%, and 7.5% for *vulnerability type, root cause, attack vector*, and *attacker type*, respectively. These performance drops are acceptable, considering the strict historical-future evaluation setting. It is very likely that new vulnerability information emerges over time, which has never been exposed in the historical CVEs. For relatively stable *vulnerability type* and *attacker type*, the prediction is still highly accurate. However, for *attack vector* which is usually product-specific, the model cannot generalize well based on the information learned from historical CVEs.

Answer to RQ3. From the results, we find that the unseen vulnerability information in the future CVEs moderately degrades the prediction performance for the model trained with historical CVEs. However, the model can still relatively accurately predict the missing *vulnerability type, attacker type*, and *root cause* of future CVEs.

4.5 Predict on Updated CVEs in NVD (RQ4)

4.5.1 Motivation. Many CVEs in NVD database are marked as "MODIFIED", which indicates that the corresponding CVE descriptions have been modified and updated by database maintainers.

By comparing the differences between "MODIFIED" version and the corresponding "ANALYSE" version[4] through our key aspect detection method, we find that there are some missing key aspects in CVEs that were augmented after being manually analyzed by NVD maintainers. We use the augmented key aspects as ground-truth to evaluate the predicting performance of our approach. In this section, we aim to explore whether our model can effectively predict these key aspects of CVEs that were later augmented by NVD in practice.

4.5.2 Approach. We first apply our aspect detection method used in **PMA** to the descriptions of CVEs that are marked as "MODIFIED" and "ANALYSE" in the NVD database. According to the detection results, we obtain 1,320 CVE descriptions in total with 1,476 key aspects that are manually augmented by the NVD maintainers in practice. In detail, there are 421, 279, 317, and 459 missing key aspects for *vulnerability type, root cause, attack vector*, and *attacker type*, respectively that have been augmented (Examples in Table 1) according to the aspect detecting results. Besides the updates on key aspects, many other "MODIFIED" versions have different changes such as augmenting new types of *affected product* for CVE such as CVE-2007-4324 and CVE-2015-8768.[5] We do not take them into the following experiments in this section.

Note that, these two labels (i.e., "MODIFIED" and "ANALYSE") only maintain the latest two versions of CVE descriptions instead of all historical modified description versions, therefore, we cannot access to all historical records of description changes to key aspects, which means many historical revisions of CVE descriptions cannot be inferred by comparing the publicly available "MODIFIED" and "ANALYSIS" versions. Additionally, CVE descriptions will be updated over a long period of time (e.g., over ten years for CVE-2008-0234[6]). We obtain the updating period on average by computing the time difference between "NVD Published Date" and "NVD Last Modified". The average is over one year, and most are between seven months to two years, which means that it will last a long time period to continually update the CVE descriptions in practice.

Unlike the previously trained model used in RQ1, we retrain the model by using the historical dataset but removing the augmented CVEs in the NVD database. After that, we take the previous version of the CVE descriptions that have not been augmented by NVD maintainers as the inputs and use the retained model to predict the missing key aspects for them. Finally, we compare the prediction results with the collected augmented data. Note that, we use the following model configurations: separate CVE aspects in the original order as input, CVE-specific word embeddings, and early-fusion architecture. We also experiment six different network designs: 1-L CNN, 2-L CNN, 1-L BiLSTM, 2-L BiLSTM, 1-L BiLSTM with attention, 2-L BiLSTM with attention.

4.5.3 Results. As shown in Table 14, the experimental results show that the best performance model is still 1-layer CNN, which is consistent with the result in RQ1. Compared with the prediction results of 1-L CNN F1 in RQ1, the results of the key aspects prediction in RQ3 drop by 7.6%, 6.2%, 4.9%, and 14.1%, respectively in the *vulnerability type, root cause, attack vector*, and *attacker type*. For the prediction of *vulnerability type* and *attacker type*, the decline is relatively serious, while for the *root cause* and *attack vector*, the decline is relatively moderate. We further explore the decline reasons by analyzing the real cases. The augmented descriptions by NVD are usually updated with multiple key aspects at once, many descriptions of aspects will be added or fixed in the update, which means that the lack of information is more serious than other data in the historical CVE dataset. Another reason is that the frequency of similar CVEs is relatively lower than that of the other CVEs in the historical dataset, which means that the category distribution of these key

[4]The "MODIFIED" version refers to the latest version of vulnerability description that has been updated, and the corresponding "ANALYSIS" version means the previous version of the current modified version.
[5]https://nvd.nist.gov/vuln/detail/CVE-2007-4324andhttps://nvd.nist.gov/vuln/detail/CVE-2015-8768
[6]https://nvd.nist.gov/vuln/detail/CVE-2008-0234#vulnCurrentDescriptionTitle

Table 14. Prediction performance on augmented CVEs in NVD

		Vul-Type	Root Cause	Attack Vector	Attacker Type
Pre	1-L CNN	0.872	0.735	0.637	0.835
	2-L CNN	0.826	0.702	0.609	0.812
	1-L BiLSTM	0.846	0.713	0.628	0.820
	2-L BiLSTM	0.846	0.711	0.628	0.823
	1-L BiLSTM+Attention	0.856	0.730	0.632	0.828
	2-L BiLSTM+Attention	0.851	0.732	0.630	0.823
Re	1-L CNN	0.876	0.741	0.683	0.694
	2-L CNN	0.845	0.703	0.671	0.673
	1-L BiLSTM	0.840	0.721	0.671	0.682
	2-L BiLSTM	0.843	0.722	0.671	0.687
	1-L BiLSTM+Attention	0.846	0.738	0.675	0.687
	2-L BiLSTM+Attention	0.859	0.738	0.683	0.690
F1	1-L CNN	0.870	0.726	0.657	0.748
	2-L CNN	0.831	0.689	0.635	0.716
	1-L BiLSTM	0.834	0.711	0.646	0.735
	2-L BiLSTM	0.837	0.710	0.646	0.739
	1-L BiLSTM+Attention	0.850	0.722	0.651	0.741
	2-L BiLSTM+Attention	0.851	0.723	0.655	0.743

aspects are different from historical dataset. Taking all of these factors together, the performance is acceptable. Our approach is still fairly effective in predicting the key aspects of NVD that will be added manually.

Answer to RQ4. The performance of our approach is slightly affected by the multiple missing aspects of CVEs in the ground-truth dataset, while the practicability of **PMA** in predicting these four aspects (i.e., *vulnerability type, root cause, attack vector, and attacker type*) are satisfactory in real scenario.

4.6 Prediction on Severity Level of Vulnerabilities (RQ5)

4.6.1 Motivation. Currently, there are many researches analyzing security vulnerabilities based on the descriptions [6, 11, 15, 23], such as predicting vulnerability severity level [23], constructing security knowledge graphs [61]. These studies heavily rely on the information contained in the vulnerability descriptions. To investigate how our approach contributes to such applications based on the descriptions of CVEs, we choose one of applications to demonstrate the usefulness of **PMA** in downstream applications.

CVSS is a specialized system for assessing the severity of vulnerabilities and assigning a score to it. CVSS classifies vulnerabilities into four levels according to their scores: Low (0.1-3.9), Medium (4.0-6.9), High (7.0-8.9), and Critical (9.0-10). Han et al. [23] have done the work of predicting the severity level of vulnerabilities, and they have achieved relatively good results. But their work is poor at predicting *threshold scores* (i.e., scores close to the severity level boundaries), as they mentioned in their paper. For example, vulnerabilities with a severity score of 6.8 (Low) are often predicted to be High (7.0-8.9). The CVSS vulnerability score relies on the basic information of the vulnerability, and its base score metrics for vulnerabilities consist of the following parts: attack vector, attack complexity, privileges required, user interaction, scope, confidentiality impact, integrity impact, and availability impact. In this RQ, we aim to investigate the impact of missing key aspects of

Table 15. Prediction performance on vulnerability severity level

	All data	PMA with all data	Threshold data	PMA with threshold data
Pre	0.801	**0.835**	0.608	**0.657**
Re	0.800	0.837	0.534	0.574
F1	0.801	**0.836**	0.564	**0.617**

vulnerabilities on vulnerability severity level prediction. This study has two purposes. **1)** First, we want to determine the impact (if any) of our work on vulnerability severity level prediction. **2)** Secondly, we want to demonstrate the usefulness of vulnerability information augmentation by our method from in a downstream application.

4.6.2 Approach. We use historical CVE descriptions as the training data set, and 1-layer CNN as the neural network model. First, we use the original CVE description training model to predict severity levels and calculate precision, recall, and F1 score. Next, we use **PMA** to predict the missing aspects of the vulnerability in this data set and complete them into the corresponding descriptions. We use these completed data to retrain the model, predict vulnerability severity levels and evaluate performance. We set the threshold scores to 3.7~4.2, 6.7~7.2, and 8.7~9.2. Then we select 5,000 vulnerability descriptions with threshold scores from the historical data set and extracted key aspects of them as test dataset and reconstruct the training dataset. Among them, 21% of the data with scores between 3.7 and 4.2, 54% of the data with scores between 6.7 and 7.2, and the rest of the data with scores between 8.7 and 9.2. We repeat the previous operation with those datasets.

4.6.3 Results. Table 15 presents the experimental results. It can be seen from the table that after the completion of vulnerability descriptions, the performance of vulnerability severity level prediction has been improved. In predicting the severity levels of all historical CVEs, the use of **PMA** to complete the missing key aspects of the vulnerability led to a 3.5% improvement in F1, and in the experiment that predicted the threshold, it improved by 5.3%. Most of these parts relate to the six key aspects of the vulnerability descriptions. Our work can help to complete and calculate the information required by CVSS, so as to calculate the corresponding vulnerability severity score more accurately and help us classify the vulnerability severity level more accurately, especially for the threshold scores.

Answer to RQ5. Our method (**PMA**) can overall improve the performance of predicting vulnerability severity level, and has a relatively better improvement in the prediction of threshold scores that demand sufficient vulnerability aspect information.

5 DISCUSSION AND THREATS TO VALIDITY

5.1 Discussion

Security vulnerability has been a research focus of many security researchers. This situation has promoted the emergence of many vulnerability databases, such as CVE, NVD. As researchers deepen their understanding of vulnerabilities, the vulnerability reports in these vulnerability databases will also be continuously updated and revised. Since our method converts the augmentation work of key aspects of vulnerabilities into classification work, it aims to predict the labels of key aspects of vulnerabilities. In this case, our method is not a substitute for professional workers, it can locate vulnerability aspects for human analysts and help them fill out new vulnerability reports and update vulnerability reports.

We have proved that **PMA** has good performance in terms of description augmentation, but there are still some limitations. **1)** Each aspect of vulnerabilities has many categories. Among them, there are some extremely uncommon attack methods and root causes that we cannot classify. We summarize these categories as "others", which can only tell people that certain key aspects belong to these categories and cannot be accurately predicted. **2) PMA** cannot match all aspects, because the categories of neural network classifiers are limited. **3)** Note that, our approach cannot be used to replace experts, but to assist and expedite their analysis. Our goal is to recommend missing information to experts. Ultimately, it is up to the experts to decide how to augment and maintain the vulnerability descriptions.

When a security vulnerability is discovered, the impact is usually the first thing to be confirmed, but some aspects (e.g., *root cause* and *attack vector*) are not easy to be discovered. Although security vulnerabilities are constantly updated and missing aspects may be fixed during the update process, their usability is low before sufficient information is provided. A lot of studies rely on a large number of vulnerability descriptions for analysis, such as vulnerability severity level prediction, software security knowledge graph completion based on relational reasoning, which largely rely on the information contained in the vulnerability descriptions. Currently, the vulnerability description often contains insufficient information and may not be sufficient to resolve the problem. Our work on missing vulnerability information may be helpful to these studies that rely on vulnerability descriptions. We have demonstrated that our method can effectively improve the performance of vulnerability severity level prediction, and we believe that our method can help more research based on vulnerability descriptions.

At present, it is a trend to leverage natural language processing (NLP) and deep learning algorithms on the dataset of CVEs. However, there are some challenges to be addressed and some new research directions worth further study. **1)** CVE has a wealth of information resources, including many related databases, such as the descriptions of CWE (Common Weakness Enumeration) [13], and CAPEC (Common Attack Pattern Enumeration and Classification) [9]. These extended databases contain a wealth of resources, such as the skills an attacker needs and how to prevent an attack. If the CVE can be combined with this information in a more fine-grained way, it may be of greater value and is a new research direction in this field. **2)** The databases that record vulnerability information are not only CVE, but also SecurityFocus [51], and IBM X-Force Exchange [28]. There are some inconsistencies in the relevant aspect information recorded in these vulnerability databases. Therefore, it is also a problem that needs to be solved to find out the possible errors in CVE by using the information inconsistencies in these different databases.

With the continuous development of programming languages and software architectures, vulnerabilities are constantly appearing and evolving, so the currently trained predictive model may no longer meet the requirements in the future. Our model also needs to be continuously updated with security vulnerabilities to ensure that it can work robustly under the premise of increasing security vulnerabilities.

5.2 Threats to Validity

Internal validity. The performance of classification model depends largely on the size and quality of training set. In order to make the model have good generalization ability, we extract all CVEs with more than four aspects as data sets. At the same time, our dataset uses the latest CVE data but eliminates the vulnerabilities with the "REJECT" label (e.g., CVE-2010-3885[7]). In addition, the practicability of the method also depends on the structure of the model. We discussed the design of the model in detail in § 4.2.

[7]https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3885

Our approach relies on the patterns of CVE descriptions, in terms of what aspects people describe in the vulnerability reports and how they describe these aspects. This assumption holds in general because there is a common knowledge about important aspects of vulnerabilities and much effort has been made to standardize the description of these aspects [29, 38]. Therefore, patterns exist in historical CVEs, which can be learned by an appropriate machine learning methods.

External validity. Our approach currently uses a rule-based method to extract CVE aspects from CVE descriptions. We obtain aspect extraction rules by observing about 27,000 CVEs from 1999 to 2020, and confirm the extraction accuracy of these rules. However, the development of aspect extraction rules may suffer from human biases and errors, and the developed rules may not cover the emerging CVE description patterns.

6 RELATED WORK

Vulnerability databases have been created to document and analyze publicly known security vulnerabilities. For example, Common Vulnerabilities and Exposures (CVE) [39] and SecurityFocus [51] are two well known vulnerability databases. Common Weakness Enumeration (CWE) abstract common software weaknesses of individual vulnerabilities, which are often referred to as vulnerability type of CVEs. New software vulnerabilities have been regularly discovered and added to the vulnerability database. For example, in about six months from November 2019 to May 2020, 5,685 new CVEs have been added to the CVE database.

The fast-growing number of vulnerabilities demand automatic methods to assist the analysis of newly discovered vulnerabilities. Bozorgi et al. [7] trained a classifier based on various features in vulnerability reports, such as description and time stamp, to predict the exploitability of a vulnerability. Han et al. [23] proposed a CNN-based classifier to predict the severity of CVEs based on only CVE description. Gong et al. [19] developed a multi-task learning method to predict seven vulnerability properties according to the Common Vulnerability Scoring System [17]. Xiao et al. [56] constructed a knowledge graph of CVEs and CWEs, and proposed a graph embedding method to infer the relationships of software vulnerabilities and weaknesses. Binyamini et al. [5] proposed a novel end-to-end automation framework for modeling new attack techniques from textual descriptions of security vulnerabilities. Bhandari et al. [4] used the knowledge representation method to do descriptive logic reasoning and reasoning for the concept of network security state. These works used complete descriptions to carry out information analysis and prediction, and did not take into account different kinds of information in vulnerability descriptions, so it is difficult to perform well in some fine-grained reasoning works. At the same time, these works did not take into account the correlation between various information in the vulnerability descriptions, while **PMA** leverages the correlation across different vulnerability aspects. Besides, Hemberg et al. [25] connected attack tactics, techniques, and patterns with defense weaknesses, vulnerabilities and affected platform configurations, and exploited multiple aspects of vulnerability information. The advantage of this work is that it has a variety of data and supports bidirectional relational path tracing. Anwar et al. [3] cleaned up the NVD data and constructed a more accurate data source. Sultan et al. [2] proposed a knowledge model based on Semantic Web, which provided a formal and semi-automatic method for unifying vulnerability information resources. Different from these works, our work studies the information completeness of vulnerability reports and develops a neural network classifier for predicting the missing key aspects in the vulnerability reports.

Neural networks have been widely adopted for text classification in natural language community [16, 31, 45]. They have also been applied to software text other than vulnerability descriptions, as well as source code. For example, Xu et al. [58] developed a CNN-based siamese network to predict duplicate questions on Stack Overflow. Chen et al. [10] developed a similar Siamese

network architecture but their goal is to support cross-lingual question retrieval. Li et al. [33] developed QDLinker based on word embeddings and CNN for answering programming questions with software documentation. Mou et al. [40] proposed a tree-based CNN to embed source code for classifying programs and detecting code patterns.

Many researches have been done on predicting vulnerable or error-prone components [43, 48], or assessing the security of a system to be attacked [52]. They use various features including software metrics, code churn, developer activity metrics, code structure [12, 49], while our work devote on analyzing vulnerability textual information and the correlations across different aspects of the vulnerability descriptions.

7 CONCLUSION AND FUTURE WORK

This paper studies the information completeness in the vulnerability reports. We examine six key aspects of CVE descriptions and find different severities of information missing for the root cause, vulnerability type, attacker type, and attack vector. We propose a machine learning approach to augment the missing information of these four aspects in the CVE descriptions. Our approach uses a neural network model to extract important features from aspect descriptions and capture intrinsic correlations among different aspects. Our large-scale experiments identify the most effective model design for the prediction task. Our model can be trained effectively using historical CVEs, and the trained model can accurately predict missing information of future CVEs. This could alleviate the information missing issue of the vulnerability reports.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, and Xiaoqiang Zhang. 2016. TensorFlow: A system for large-scale machine learning.

[2] Sultan S Alqahtani and Juergen Rilling. 2019. Semantic Modeling Approach for Software Vulnerabilities Data Sources. In *2019 17th International Conference on Privacy, Security and Trust (PST)*. IEEE, 1–7.

[3] Afsah Anwar, Ahmed Abusnaina, Songqing Chen, Frank Li, and David Mohaisen. 2020. Cleaning the NVD: Comprehensive Quality Assessment, Improvements, and Analyses. *arXiv preprint arXiv:2006.15074* (2020).

[4] P. Bhandari and M. Singh. 2016. Formal Specification of the Framework for NSSA. *Procedia Computer Science* 92 (2016), 23–29.

[5] H. Binyamini, R. Bitton, M. Inokuchi, T. Yagyu, Y. Elovici, and A. Shabtai. 2020. An Automated, End-to-End Framework for Modeling Attacks From Vulnerability Descriptions. (2020).

[6] Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 2010. Beyond Heuristics: Learning to Classify Vulnerabilities and Predict Exploits. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '10)*. 105?114.

[7] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2010. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 105–114.

[8] Brompwnie. 2017. cve-2020-5260. https://github.com/brompwnie/cve-2020-5260/. [Online; accessed 21-January-2017].

[9] CAPEC. 2019. Common Attack Pattern Enumeration and Classification. http://cwe.mitre.org/. [Online; accessed 30-June-2019].

[10] G. Chen, C. Chen, Z. Xing, and B. Xu. 2016. Learning a dual-language vector space for domain-specific cross-lingual question retrieval. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 744–755.

[11] Yang Chen, Andrew E Santosa, Asankhaya Sharma, and David Lo. 2020. Automated Identification of Libraries from Vulnerability Data. In *Proceedings of the 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '20)*. IEEE Press.

[12] Istehad Chowdhury and Mohammad Zulkernine. 2010. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. In *Journal of Systems Architecture*, Vol. 57. 294–313.

[13] CWE. 2019. Common weakness enumeration (CWE). http://capec.mitre.org/. [Online; accessed 30-June-2019].

[14] details. 2019. CveForm. https://cveform.mitre.org/. [Online; accessed 30-June-2019].

[15] 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 *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 869–885.

[16] Ronen Feldman. 2013. Techniques and applications for sentiment analysis. In *Communications of the ACM*, Vol. 56. ACM New York, NY, USA, 82–89.

[17] FIRST. 2019. Common vulnerability scoring system (cvss). https://www.first.org/cvss. [Online; accessed 30-June-2019].

[18] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 315–323.

[19] Xi Gong, Zhenchang Xing, Xiaohong Li, Zhiyong Feng, and Zhuobing Han. 2019. Joint Prediction of Multiple Vulnerability Characteristics Through Multi-Task Learning. In *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 31–40.

[20] Google. 2019. Word2vec. https://code.google.com/archive/p/word2vec/. [Online; accessed 30-June-2019].

[21] Hao Guo, Zhenchang Xing, Sen Chen, Xiaohong Li, Yude Bai, and Hu Zhang. 2021. Key aspects augmentation of vulnerability description based on multiple security databases. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 1020–1025.

[22] Harsha Gurulingappa, Abdul Mateen Rajput, Angus Roberts, Juliane Fluck, Martin Hofmann-Apitius, and Luca Toldo. 2012. Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. In *Journal of Biomedical Informatics*, Vol. 45. 885—892.

[23] Z. Han, X. Li, Z. Xing, H. Liu, and Z. Feng. 2017. Learning to Predict Severity of Software Vulnerability Using Only Vulnerability Description. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 125–136.

[24] A. Hassan and A. Mahmood. 2018. Convolutional Recurrent Deep Learning Model for Sentence Classification. In *IEEE Access*, Vol. 6. 13949–13957.

[25] Erik Hemberg, Jonathan Kelly, Michal Shlapentokh-Rothman, Bryn Reinstadler, Katherine Xu, Nick Rutar, and Una-May O'Reilly. 2020. BRON–Linking Attack Tactics, Techniques, and Patterns with Defensive Weaknesses, Vulnerabilities and Affected Platform Configurations. *arXiv preprint arXiv:2010.00533* (2020).

[26] Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned Language Models for Text Classification. In *CoRR*, Vol. abs/1801.06146. arXiv:1801.06146

[27] Ignacio Iacobacci, Mohammad Taher Pilehvar, and Roberto Navigli. 2016. Embeddings for word sense disambiguation: An evaluation study. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 897–907.

[28] IBM. 2019. IBM X-Force Exchange. https://exchange.xforce.ibmcloud.com/. [Online; accessed 30-June-2019].

[29] MITRE Jonathan Evans. 2020. key details phrasing. http://cveproject.github.io/docs/content/key-details-phrasing.pdf. [Online; accessed February-2020].

[30] kasif dekel. 2017. whatsapp-rce-patched. https://github.com/kasif-dekel/whatsapp-rce-patched/. [Online; accessed 21-January-2017].

[31] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *CoRR*, Vol. abs/1408.5882. arXiv:1408.5882

[32] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

[33] Jing Li, Aixin Sun, and Zhenchang Xing. 2018. Learning to answer programming questions with software documentation through social context embedding. In *Information Sciences*, Vol. 448-449. 36–52.

[34] Bill Yuchen Lin, Frank F Xu, Zhiyi Luo, and Kenny Zhu. 2017. Multi-channel bilstm-crf model for emerging named entity recognition in social media. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*. 160–165.

[35] Christoper Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52Nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

[36] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[37] Corporation MITRE. 2017. National vulnerability database (NVD). https://nvd.nist.gov/. [Online; accessed 21-January-2017].

[38] Corporation MITRE. 2019. Common Attack Pattern Enumeration and Classification Submission. https://cveform.mitre.org. [Online; accessed 30-June-2019].

[39] Corporation MITRE. 2019. Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/. [Online; accessed 30-June-2019].

[40] Lili Mou, Ge Li, Zhi Jin, Lu Zhang, and Tao Wang. 2014. TBCNN: A Tree-Based Convolutional Neural Network for Programming Language Processing. In *CoRR*, Vol. abs/1409.5718. arXiv:1409.5718

[41] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. 2018. Understanding the Reproducibility of Crowd-reported Security Vulnerabilities. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 919–936.
[42] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. 2016. Improving document ranking with dual word embeddings. In *Proceedings of the 25th International Conference Companion on World Wide Web*. 83–84.
[43] Stephan Neuhaus, Thomas Zimmermann, Christian Holler, and Andreas Zeller. 2007. Predicting Vulnerable Software Components. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. 529?540.
[44] NIST. 2017. National Institute of Standards and Technology (NIST). https://www.nist.gov/. [Online; accessed 21-January-2017].
[45] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *CoRR*, Vol. abs/1802.05365. arXiv:1802.05365
[46] Scott Reed and Nando Freitas. 2015. Neural Programmer-Interpreters.
[47] Kashif Riaz. 2010. Rule-based named entity recognition in Urdu. In *Proceedings of the 2010 named entities workshop*. Association for Computational Linguistics, 126–135.
[48] R Scandariato, J Walden, A Hovsepyan, and W Joosen. 2014. Predicting vulnerable software components via text mining. In *IEEE Transactions on Software Engineering*, Vol. 40. 993–1006.
[49] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A Osborne. 2011. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. In *IEEE Transactions on Software Engineering*, Vol. 37. 772–787.
[50] Ravindra Singh and Naurang Mangat. 1996. Elements of Survey Sampling, Vol. 15.
[51] Symantec. 2019. securityFocus. https://www.securityfocus.com/. [Online; accessed 30-June-2019].
[52] Lingyu Wang, Tania Islam, Long Tao, Anoop Singhal, and Sushil Jajodia. 2008. An Attack Graph Based Probabilistic Security Metric. In *Lecture Notes in Computer Science*, Vol. 5094. 283–296.
[53] Song Wang, Taiyue Liu, and Lin Tan. 2016. Automatically learning semantic features for defect prediction. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 297–308.
[54] RF Woolson. 2007. Wilcoxon signed-rank test. In *Wiley encyclopedia of clinical trials*. Wiley Online Library, 1–3.
[55] Xin Xia, David Lo, Sinno Jialin Pan, Nachiappan Nagappan, and Xinyu Wang. 2016. Hydra: Massively compositional model for cross-project defect prediction. In *IEEE Transactions on software Engineering*, Vol. 42. IEEE, 977–998.
[56] Hongbo Xiao, Zhenchang Xing, Xiaohong Li, and Hao Guo. 2019. Embedding and Predicting Software Security Entity Relationships: A Knowledge Graph Based Approach. In *International Conference on Neural Information Processing*. Springer, 50–63.
[57] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 51–62.
[58] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 51–62.
[59] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 1480–1489.
[60] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th international conference on software engineering*. 404–415.
[61] Liu Yuan, Yude Bai, Zhenchang Xing, Sen Chen, Xiaohong Li, and Zhidong Deng. 2021. Predicting entity relations across different security databases by using graph attention network. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 834–843.
[62] Bai Yude, Xing Zhenchang, Li Xiaohong, Feng Zhiyong, and Ma Duoyuan. 2020. Unsuccessful Story about Few Shot Malware Family Classification and Siamese Network to the Rescue. In *2020 IEEE/ACM 42st International Conference on Software Engineering (ICSE '20)*.
[63] Xian Zhan, Lingling Fan, Sen Chen, Feng Wu, Tianming Liu, Xiapu Luo, and Yang Liu. 2021. ATVHUNTER: Reliable Version Detection of Third-Party Libraries for Vulnerability Identification in Android Applications. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1695–1707. https://doi.org/10.1109/ICSE43902.2021.00150
[64] Ye Zhang and Byron C. Wallace. 2015. A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. In *CoRR*, Vol. abs/1510.03820. arXiv:1510.03820
[65] Yaqin Zhou, Shangqing Liu, Jingkai Siow, Xiaoning Du, and Yang Liu. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. In *CoRR*. arXiv:1909.03496