Summarizing Vulnerabilities' Descriptions to Support Experts during Vulnerability Assessment Activities

Ernesto Rosario Russo[a,c], Andrea Di Sorbo[b], Corrado A. Visaggio[b], Gerardo Canfora[b]

[a]*Department of Informatics, University of Bari Aldo Moro, Italy*
[b]*Department of Engineering, University of Sannio, Italy*
[c]*Exprivia S.p.A., Sede Legale Via A. Olivetti 11, 70056 Molfetta (BA) - Italy*

Abstract

Vulnerabilities affecting software and systems have to be promptly fixed, to prevent violations to integrity, availability and confidentiality policies of targeted organizations. Once a vulnerability is discovered, it is published on the Common Vulnerabilities and Exposures (CVE) database, freely available on the web. However, vulnerabilities are described using natural language, which makes them hard to be automatically interpreted by machines. As a consequence, vulnerability assessment activities tend to be time-consuming and imprecise, as the assessors must manually read the majority of the vulnerabilities concerning the perimeter to be protected, to make a decision on which vulnerabilities have the highest priority for patching. In this paper we present CVErizer, an approach able to automatically generate summaries of daily posted vulnerabilities and categorize them according to a taxonomy modeled for industry. We empirically assess the classification capabilities of the approach on a set of 3369 pre-labeled CVE records and perform an end-to-end evaluation of CVErizer summaries involving 15 cybersecurity master students and 4 professional security experts. Our study demonstrates the high performance of the proposed approach in correctly extracting and classifying information from CVE descriptions. Summaries are also considered highly useful for helping analysts during the vulnerability assessment processes.

Keywords: Natural Language Processing, Software Security, Summarization, Software Maintenance

1. Introduction

In the context of Information Technology, the term "vulnerability" indicates a defect that can allow the violation of a security policy in a system. According to the definition provided by the MITRE corporation, a vulnerability is *"a mistake in software that can be directly used by a hacker to gain access to a system or network"* [1]. Thus, a vulnerability could allow an attacker to: execute arbitrary commands, obtain open access to private data, impersonate any other user, throw a DoS attack [2], or any other action which can violate privacy and security policies and subvert the related controls. For example, recent infamous ransomware WannaCry[1] and NotPetya[2] both exploit the Microsoft Windows SMB Server Remote Code Execution Vulnerability (CVE-2017-0145).

Consequently, it is important to timely apply the latest security patch to an operating system, a web server, or any kind of software layer, to mitigate the risk for a system of being exposed to potential attacks. Indeed, the presence of vulnerabilities is the main cause of security attacks [3]. In addition, modern software systems often rely on third-party libraries to provide specific functionalities. Such libraries are in constant evolution: newer versions with fixed defects, patched vulnerabilities, and enhanced features, are progressively released. Obviously, the immediate migration (*e.g.,* update to a newer version) of a vulnerable dependency is strongly recommended to avoid the exposure of the dependent application to malicious attacks. However, a recent study [4] demonstrated that updates of third-party library dependencies are not regularly practiced by developers (especially to fix vulnerabilities), since they are often unaware of vulnerable dependencies, highlighting the need of strategies aimed at improving developers personal perception of third-party security updates.

In this context, "Common Vulnerabilities and Exposures" (CVE)[3] is an industry standard of common names for publicly known information security vulnerabilities, and has been widely adopted by organizations to provide easier interoperability and better security coverage. More specifically, CVE is a list of information security vulnerabilities and exposures that aims to provide common names for publicly known problems. The use of CVE identifiers ensures confidence among parties when they have to discuss or share information about a unique software or firmware vulnerability. In general, when a new vulnerability is discovered and patched, it is posted on the CVE database.

Email addresses: `ernestorosario.russo@uniba.it`, `ernesto.russo@exprivia.com` (Ernesto Rosario Russo), `disorbo@unisannio.it` (Andrea Di Sorbo), `visaggio@unisannio.it` (Corrado A. Visaggio), `canfora@unisannio.it` (Gerardo Canfora)

[1]`https://www.symantec.com/security_response/writeup.jsp?docid=2017-051310-3522-99`

[2]`https://www.us-cert.gov/ncas/alerts/TA17-181A`

[3]`https://cve.mitre.org/`

A common problem with CVE resides in its syntactic description format. Since CVE descriptions are written in natural language text, they are hard to be automatically interpreted by machines. Moreover, different organizations may perceive the severity of a particular vulnerability differently, and, consequently, they may also prioritize its mitigation differently [5]. Thus, security teams are required to:

1. Constantly monitor CVE databases.
2. Read CVE descriptions to acquire semantic information about the vulnerabilities (*e.g.*, which product would be affected by the vulnerability, what kind of damage the vulnerability would generate, which is the attack vector) [6]
3. Evaluate the risks derived from each specific vulnerability.
4. Perform software maintenance operations to minimize these risks [7].

As discussed by Guo and Wang in [6], CVE descriptions present a quite regular syntax. In our work, we are interested in exploiting this regularity of vulnerability descriptions, for allowing companies to automatically collect the relevant information that will be used by analysts to make decisions about vulnerability prioritization and mitigation. Pragmatically speaking, our work is aimed at automating the step 2 of the process discussed above.

Every year, thousands of vulnerabilities are discovered. Companies' security teams need to check their own environment against the vulnerabilities daily reported, and prioritize mitigation of such vulnerabilities on the basis of the associated risk. However, when curating vulnerability information is not a main line of business, companies prefer to find trusted software vulnerability management partners whose function is to perform this work [8]. State-of-the-art tools used by vulnerability assessors, as asset inventories and vulnerability scanners, currently associate all the CVEs to the software they affect [9]. The main limitation of these tools is that they do not provide any semantic information about such vulnerabilities (*e.g.*, the origin of a vulnerability and the undesired effects which it may produce on the system). Without manually analysing the description of each collected vulnerability, it is not possible to know which of these vulnerabilities allows for a specific *kind of attack*, leverages a specific *attack vector*, causes a *specific damage*.

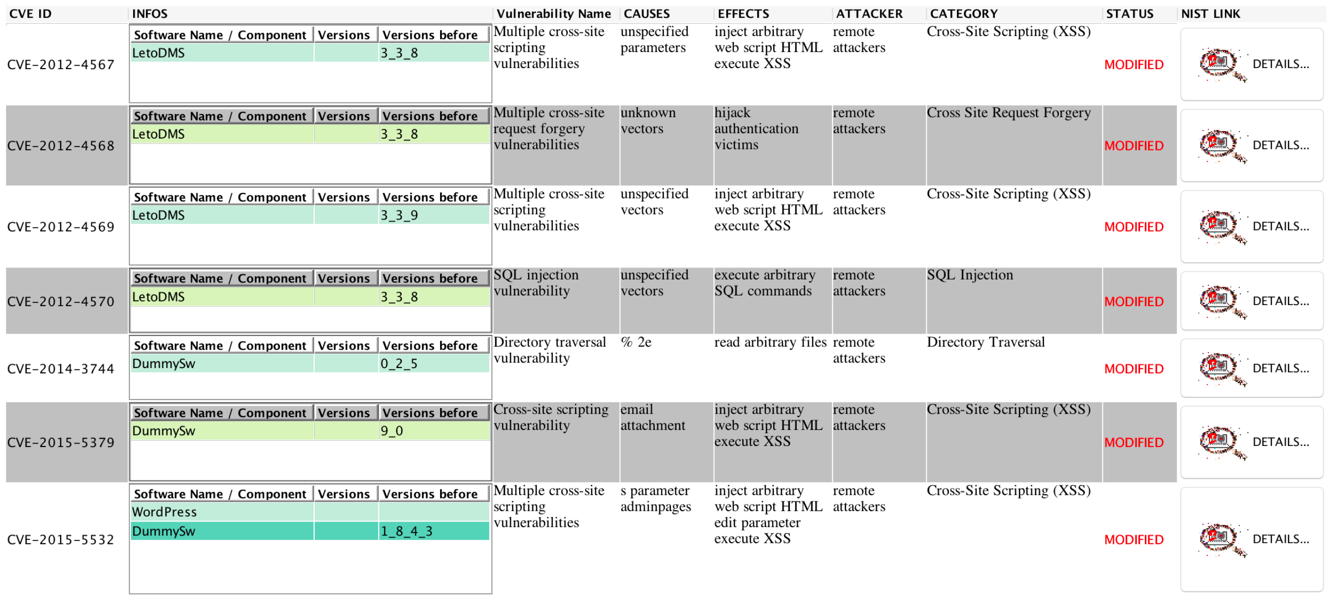
This has a direct impact on the costs and the efficacy of the vulnerability assessment.

Paper contributions. In this paper we present an approach and a tool, called `CVErizer`, able to automatically extract semantic information from natural language descriptions of CVEs. Since our goal is to allow companies to timely react to recently published CVEs, we decided to focus on CVE descriptions, as they would be the only data available that security experts would likely analyze at the time of a CVE publication. On the other hand, when the information from other sources (*e.g.,* CVSS) is available, CVE summaries could provide orthogonal details (*e.g.,* effects and causes) to the ones available from other sources. Indeed, as reported by the NVD website[4], CVE entries are updated several times and CVSS, CPE or CWE data are added during the vulnerability life-cycle. For this reason, when first published, the CVE only comprises the identifier, the description, some references (*i.e.,* list of URLs and other information, such as vendor advisory numbers) and the date the entry was created.

In particular, our approach is able to mine the following information: (i) the software affected by the vulnerability, (ii) the specific versions of the software interested, (iii) the type of attacker which could exploit the vulnerability (e.g., remote or local attacker), (iv) the origin of the vulnerability (e.g., an unchecked POST request), (v) the consequences (e.g., obtain sensitive information), and (vi) the specific category of the vulnerability according to a taxonomy modeled for industry (detailed in Table 3).

Categorization of vulnerabilities is a precondition for in depth vulnerability analysis [10] and, recently, some researchers have made preliminary exploration on automatic vulnerabilities categorization [11, 12, 13, 14, 15]. Differently from these previous works, which are mainly aimed at developing taxonomies or approaches with the explicit purposes of precisely categorizing and assigning a severity score to each item in the context of a vulnerability database, our approach is more industry oriented, since its goal is to (i) automatically analyze daily published CVEs and (ii) present the extracted information in the form of CVE summaries to the security teams, in order to (iii) minimize their reaction time and (iv) allow developers fixing critical exposures in the shortest possible time. Indeed, the criticality of a vulnerability is based on the assessment of the (i) vulnerability's potential impact on a system, (ii) the attack vector, (iii) mitigating factors, and (iv) whether an exploit exists for the vulnerability and is being actively exploited, prior to the release of a patch [8]. `CVErizer`, through Natural Language Processing abilities, aims to summarize these kinds of information for daily reported vulnerabilities, to help security teams and vulnerability assessors prioritizing security issues. Since different companies could assign a different patching priority to a specific vulnerability, according to the diverse companies' strategies, rather then designing a fine-grained list of categories of vulnerabilities and discussing which categories have to be prior patched, the present work is mainly aimed at proposing a technique for automatically extracting information that could support security experts in deciding such priorities by their own. For instance, security engineers would be interested at first remedying to remotely exploitable vulnerabilities rather than locally exploitable ones, or vulnerability assessors could be more concerned about vulnerabilities of a specific class (e.g., *SQL Injection*). With summaries provided by `CVErizer`, these (and similar) kinds of information would be at their fingertips, minimizing the time that would be otherwise spent in mining these data from the collected CVEs. Our approach is highly extensible, since it provides an easy way to retrieve further information by simply adding extraction rules in a xml file. Closing

[4]`https://nvd.nist.gov/vuln`



CVE ID	INFOS			Vulnerability Name	CAUSES	EFFECTS	ATTACKER	CATEGORY	STATUS	NIST LINK
	Software Name / Component	Versions	Versions before							
CVE-2012-4567	LetoDMS		3_3_8	Multiple cross-site scripting vulnerabilities	unspecified parameters	inject arbitrary web script HTML execute XSS	remote attackers	Cross-Site Scripting (XSS)	MODIFIED	DETAILS...
CVE-2012-4568	LetoDMS		3_3_8	Multiple cross-site request forgery vulnerabilities	unknown vectors	hijack authentication victims	remote attackers	Cross Site Request Forgery	MODIFIED	DETAILS...
CVE-2012-4569	LetoDMS		3_3_9	Multiple cross-site scripting vulnerabilities	unspecified vectors	inject arbitrary web script HTML execute XSS	remote attackers	Cross-Site Scripting (XSS)	MODIFIED	DETAILS...
CVE-2012-4570	LetoDMS		3_3_8	SQL injection vulnerability	unspecified vectors	execute arbitrary SQL commands	remote attackers	SQL Injection	MODIFIED	DETAILS...
CVE-2014-3744	DummySw		0_2_5	Directory traversal vulnerability	% 2e	read arbitrary files	remote attackers	Directory Traversal	MODIFIED	DETAILS...
CVE-2015-5379	DummySw		9_0	Cross-site scripting vulnerability	email attachment	inject arbitrary web script HTML execute XSS	remote attackers	Cross-Site Scripting (XSS)	MODIFIED	DETAILS...
CVE-2015-5532	WordPress			Multiple cross-site scripting vulnerabilities	s parameter adminpages	inject arbitrary web script HTML edit parameter execute XSS	remote attackers	Cross-Site Scripting (XSS)	MODIFIED	DETAILS...
	DummySw		1_8_4_3							

Figure 1: CVErizer output example

up, the approach we propose generates extractive summaries of CVEs, not available in the current systems collecting CVEs. CVErizer identifies within a CVE description (written in natural and unstructured text), relevant pieces of information, structure them in a format that allows a wider spectrum of search and analysis than a natural text description allows. The goal of CVERizer is to produce extractive summaries of available CVEs, i.e. published on official CVE's repositories, while it does not deal with the problem of CVE's disclosure which is delayed by companies. An example application for CVErizer is in the field of risk evaluation. Risk assessment requires to define the impact of each vulnerability found within a system. This evaluation must be done manually by the assessor who has to read the complete list of CVEs affecting the system. CVErizer could allow to semi-automatically compute the risk, once the list of CVEs is built, if each kind of *effects* in the CVErizer summary is associated to an impact level through a risk model. Indeed, previous research [16] demonstrated that common risk models, as the CVSS base metric, are not enough to assess the impact of the vulnerability. Moreover, the CVSS score calculation can be somewhat subjective and two users may be having different CVSS scores for the same vulnerability [17].

We assessed the tool's classification capabilities on a set of 3369 CVE records manually classified by industrial subjects, according to a taxonomy of vulnerabilities categories modeled for Cybaze S.p.A.[5]. Moreover, an end-to-end evaluation of the `CVErizer`'s summaries has been conducted involving 15 master's students of a software security academic course at University of Molise[6] and 4 professional security experts employed at Cybaze. The results of the evaluation demonstrate (i) the high accuracy of the tool in classifying vulnerabilities, as well as (ii) the high performance in correctly extracting relevant information that could help security analysts improving the risk assessment process (*i.e.,* having automatically extracted information, analysts can take decisions in less time). In addition, study participants considered summaries generated through `CVErizer` (i) useful for supporting security experts during vulnerability assessment processes, (ii) easy to read and understand, and (iii) providing the right amount of information to comprehend vulnerabilities. Finally, we discuss the results of an industrial evaluation of CVErizer's capabilities when used to automatically extract salient information from vulnerabilities affecting two competing hypervisor products.

We make publicly available a replication package[7] with (i) results achieved through the different classification algorithms (due to non-disclosure-agreement with our industrial partner, we are unauthorized to share raw data concerning this part), (ii) complete results and raw data concerning our controlled experiment (see Section 3.2), and (iii) the prototypical implementation of CVErizer.

Paper structure: Section 2 presents the approach and techniques we used. Section 3 reports the dataset and the evaluation methods we employed. Section 4 presents and discusses the results of the study. Section 5 reports the results of an industrial case study in which our tool has been used to assess vulnerabilities affecting hypervisors' systems. Section 6 deals with the threats that could affect the validity of our work. Section 7 illustrates the related literature and Section 8 concludes the paper outlining future research directions.

2. Approach

To provide support to security assessment activities, we propose a lightweight and extensible approach for summarizing CVE's content, which is mainly composed by two steps: (i) information extraction, and (ii) classification of vulnerabilities.

[5]`https://cybaze.it/`

[6]`http://dipbioter.unimol.it/english/degree-programmes/second-cycle-degree/master-in-software-system-security/`

[7]`https://github.com/jssrp2018/CVErizer`

In particular, our approach aims at automatically mining useful information from texts contained in vulnerabilities' descriptions. Indeed, although there are several resources (*e.g.*, CPE[8] or CWE[9]) that can be used for gaining different information about vulnerabilities, significant amounts of key information remain only in unstructured text [18].

In the following, we provide a detailed description of the method. Figure 1 illustrates an example output of CVErizer. Each row represents a different CVE, while in each column the information types extracted by CVErizer from the original plaintext of the CVE (detailed in Table 1) are shown. The "Status" column indicates whether the CVE has been recently modified or if it is a new added vulnerability. By clicking on the `Details` button, users will be redirected to the National Vulnerability Database webpage describing the specific vulnerability.

2.1. Information Extraction

Previous research [18] has shown that (i) the name of the software affected by the vulnerability (*i.e.*, *Software name* in Table 1) , (ii) the versions of the Software affected by the vulnerability (*i.e.*, *Versions* and *Previous versions* in Table 1), (iii) the kind of attacker who could exploit the vulnerability (*i.e.*, *Attacker* in Table 1), (iv) the attack mechanism used by an attacker for causing the vulnerability to be exploited (*i.e.*, *Cause* in Table 1), (v) the consequences of the vulnerability (*i.e.*, *Effect* in Table 1) and the (vi) vulnerability name (which is available only for some records) are some of the most relevant information pieces usually contained in a *CVE* description.

Through a manual inspection of 130 common vulnerabilities, we observed that their descriptions present some recurrent linguistic patterns which could be exploited to automatically recognize the aforementioned information. For instance, considering the following description:

"The ironic-api service in OpenStack Ironic before 4.2.5 (Liberty) and 5.x before 5.1.2 (Mitaka) allows remote attackers to obtain sensitive information about a registered node by leveraging knowledge of the MAC address of a network card belonging to that node and sending a crafted POST request to the v1/drivers/$DRIVER.NAME/vendor.passthruresource."

We notice that the plain text matches the following structure:

"in **[Software name]** before **[Version]** and before **[Previous Version]** allows **[Attacker]** to **[Effect]** by **[Cause]**"

Table 1 shows the relevant data related to each of the aforementioned *"Information Classes"* extracted from the example description.

We argue that this (and other) recurrent structure(s) may be exploited to automatically recognize the salient information present in CVE descriptions. Specifically, to accomplish this task, we leverage an approach previously adopted for the recognition of text fragments useful for performing software maintenance and evolution activities, within discussion among developers [19, 20]. This approach is based on the identification of Natural Language Patterns within sentences contained in the target texts, leveraging the Stanford Typed Dependencies (SD) representation [21] of these sentences. The SD parser is able to represent the grammatical frame of a sentence as a list of triples. Each triple describes the grammatical relation that exists between two words: the governor and the dependent.

Table 1: Information in CVE descriptions.

Information Class	Example
Software name	in **OpenStack Ironic** before 4_2_5
Versions	and **5_x** before 5_1_2
Previous version	before **5_1_2**
Attacker	allows **remote attackers** to
Cause	by ... and **sending** a crafted **POST request**
Effect	to **obtain sensitive information** about

This technique, previously used in the context of developers' discussions, has been profitably adapted to our needs. In particular, to support the process of NL patterns' formulation and make the information extraction task more flexible to further improvements, we provided the ability to formalize the identified patterns within an XML file.

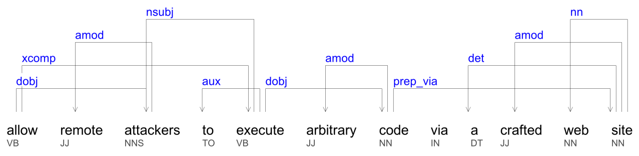


Figure 2: Stanford Typed Dependencies representation

Figure 2, shows the process applied to define each heuristic. We first analyze a sentence containing one identified recurrent pattern (*e.g.*, *"allow remote attackers to execute arbitrary code via a crafted web site"* in Figure 1) and build its SD representation. For instance, the example sentence shown in Figure 2 presents the following SD representation:

- *"execute"* represents the clausal complement of the main verb *"allow"* (*i.e.*, xcomp);
- *"code"* is the direct object of the verb *"execute"* (*i.e.*, dobj);
- "arbitrary" is an adjectival modifier of *"code"* (*i.e.*, amod);
- connected with "code" we find a preposition introduced by *"via"* (*i.e.*, prep_via).

Based on this representation we can formalize the heuristic as shown in Listing 1. A similar formalism has been adopted for defining heuristics aimed at identifying privacy leaks in social network posts [22]. The definition of a NLP heuristic consists in the definition of a rule able to recognize a particular path in the Stanford Typed Dependencies tree of a generic sentence. For example, the heuristic shown in Listing 1, aims to identify the *Effect* information by capturing the pattern *"allow [someone] to [do something] via..."*.

[8]`https://nvd.nist.gov/products/cpe`
[9]`https://cwe.mitre.org/`

Table 2: Number of heuristics for each Information class

Information Class	Heuristics
Software name	32
Versions	7
Previous version	5
Attacker	8
Cause	24
Effect	26
Vulnerability name	28
TOTAL	130

Listing 1: Example of a heuristic's definition

```
<NLP_heuristic>
  <type>xcomp/dobj/amod/prep_via</type>
  <text>Effect: {xcomp.dependent}[1]{amod.dependent}[3]{
      amod.governor}[3]</text>
  <conditions>
    <condition>xcomp.governor="allow␣allows"</condition>
    <condition>xcomp.dependent=dobj.governor</condition>
    <condition>dobj.dependent=amod.governor</condition>
    <condition>amod.governor=prep_via.governor</
        condition>
  </conditions>
  <sentence_class>EFFECT</sentence_class>
</NLP_heuristic>
```

The "conditions" represent the core part of the heuristic (*i.e.*, they describe the path to be searched in the SD representation of the text composing the description). The "type" tag is used to indicate the typed dependencies which have to be analyzed, while the "text" tag is used to dynamically compose the text to extract by specifying the term (grammar relation followed by governor/dependent) and the row of the condition involved. Finally the "sentence_class" is used to discriminate from the different *Information Classes.* The heuristic described in Listing 1 extracts the text *"Effect: execute arbitrary code"* from the sentence shown in Figure 2 .

We defined a set of heuristics for each of the defined information classes. Table 2 illustrates the number of heuristics implemented for each class. More specifically, an author of the paper used 130 vulnerability descriptions (not included in the subsequent experiments) as input to identify common natural language patterns, and, consequently, heuristics aimed at automatically extracting the required information through the process illustrated above. The identification process terminated when the set of defined heuristics succeed in correctly mining the required information for all the 130 considered vulnerabilities. On average, we defined for each Information Class 27 heuristics, with the exception of *Version*, *Previous Version* and *Attacker* classes (for which we defined 7, 5 and 8 heuristics, respectively).

Analyzing the SD representation of the CVE descriptions, our approach exploits the defined heuristics to extract most of the relevant information contained in such descriptions.

2.2. Classification of Vulnerabilities

As a result of a collaboration with Cybaze S.p.A., a renowned cyber security company, we identified a lightweight taxonomy for categorizing the different types of vulnerabilities. It is important to highlight that detailed categorization information for many vulnerabilities is already accessible via further resources, as CWE. However, CWE is a collection of weaknesses having a highly tangled structure at various levels of abstraction which is quite difficult to use for stakeholders in the software development community [23]. For this reason, according to the aim of providing a quick overview of vulnerabilities' semantic-related information, we designed a set of self-explained categories. The taxonomy comprises 10 categories, which covers a wide range of vulnerabilities. Such categories were identified through an open card sorting procedure [24], carried out by the company's security experts. All the identified categories are described in Table 3. For example, the CVE Details website, a widely used resource for collecting information about vulnerabilities [25], proposes a list of 13 conceptual categories for vulnerabilities. All the CVE Details categories are covered by our taxonomy, with the exception of two vulnerability classes that mainly affect web applications: CSRF[10], and HTTP Response Splitting[11] (as shown in Table 4). Overall, only about 2% of all the vulnerabilities collected by the CVE details website fall in these two categories.

According to our taxonomy, the example vulnerability (illustrated in Section 2.1) belongs to the *Information Disclosure and/or Arbitrary File Read* category, since this vulnerability might allow remote attackers to *"obtain sensitive information"*, as stated in its description.

To automatically classify vulnerabilities according to our taxonomy, we trained a set of machine learning (ML) techniques. This section illustrates the features and the methodology we used to train the different ML classifiers, while Section 3 describes the data used as training and test set (below we refer them as T_1 and T_2). More formally, given a training set of vulnerabilities' descriptions T_1 and a test set of vulnerabilities' descriptions T_2, we automatically classify the vulnerabilities in T_2, by performing the following steps:

1. *Text Features*: in this step, we use all the descriptions contained in T_1 and T_2 as base information to build a textual corpus (indexing the text). We preprocessed the textual content by (i) applying stop-words removal and stemming (through Lucene Porter stemmer[12]) [26], and (ii) removing some of the information that may cause bias in the classification. In particular, we analyzed the descriptions through our information extractor (see Section 2.1) and removed from the texts the information belonging to the (i) *Software name*, (ii) *Versions* and (iii) *Previous versions* classes. Moreover, we discarded all the words containing numbers and special characters. The output of this step is a Term-by-Documents matrix M where each column represents a description and each row represents a term contained in the given description. Thus, each entry $M_{i,j}$ of

[10]https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

[11]https://www.owasp.org/index.php/HTTP_Response_Splitting

[12]https://lucene.apache.org/core/6_3_0/analyzers-common/org/apache/lucene/analysis/en/PorterStemFilter.html

the matrix represents the weight (or importance) of the i-th term contained in the j-th description. Similarly to the work by Khazei *et al.* [27], we weighted words using the tf-idf (term frequency - inverse document frequency). This weight is a statistical measure used to evaluate the importance of a word in a document, in a collection of documents. Given $f_{i,j}$, the frequency for word i in a document j, D_{all}, the number of overall documents in a collection, and D_i, the amount of documents containing the word i, each word i in a document j is weighted according to the following formula:

$$w_{i,j} = f_{i,j} * \log \frac{D_{all}}{D_i}$$

2. *Split training and test features:* in this step we split the matrix M (the output of the previous step) in two sub-matrices $M_{training}$ and M_{test}. Specifically, $M_{training}$ represents the matrix containing the descriptions (*i.e.*, the corresponding columns in M) of T_1 and M_{test} contains the descriptions of T_2.

3. *Oracle building*: this step builds the oracle to allow ML techniques to train from $M_{training}$ and predict on M_{test}. As reported in Section 3.1, all the CVEs considered in our study have been already assigned by professional developers and analysts to one of the categories described in Table 3. Thus, in this step, each description in T_1 and T_2 is merely associated with the previously assigned label.

4. *Classification:* This step automatically classifies sentences relying on the classified descriptions contained in $M_{training}$ and M_{test}. In particular, we experimented (using the Weka tool [28]) different machine learning algorithms: J48, Random Forest, BayesNet, Simple Logistic, the standard probabilistic naive Bayes classifier. The choice of the classifiers for our study is motivated by the effectiveness of these algorithms when categorizing text [29, 30, 31, 32].

3. Study Design

The *goal* of this study is to investigate the correctness and usefulness of CVE summaries generated by `CVErizer`. The *quality focus* concerns the ability of developers and security engineers to collect salient information contained in CVE descriptions, when supported by our summarization tool. The *perspective* is that of researchers interested in evaluating the effectiveness of automated approaches for CVE summarization when applied in a real working scenario. We therefore designed our study to answer the following research question:

- **RQ1: To what extent is `CVErizer` accurate in classifying CVE descriptions?** Our first goal is to evaluate the CVE classification capabilities achieved by our tool.

- **RQ2: To what extent does `CVErizer` help security engineers better manage information contained in CVE descriptions?** Our aim is to verify whether `CVErizer` represents a valid support for developers interested in analyzing CVEs. Thus, stemming from RQ2, we derive two research subquestions that need to be answered to qualitatively and quantitatively assess the practical usefulness of our summarizer.

  – **RQ2-a: How do CVE summaries generated by `CVErizer` impact time required by security ana-**

Table 3: Categories of Vulnerabilities

Category	Description
Authentication bypass or Improper Authorization	An exploitation of this issue might allow an attacker to bypass the required authentication. Or the application does not perform properly the authentication check, when an user attempts to access a resource without the necessary permissions.
Cross-Site Scripting or HTML Injection	An exploitation of this issue might allow an attacker to execute arbitrary script code in the web browser of the site visitor and steal his cookie-based authentication credentials.
Denial Of Service (DoS)	An exploitation of this issue might allow an attacker to crash the affected application, denying any further access.
Directory Traversal	An exploitation of this issue might allow an attacker to gain read access to arbitrary file content on the affected system.
Local File Include, Remote File Include and Arbitrary File Upload	An exploitation of this issue might allow an attacker to include arbitrary remote files containing malicious code. The code could then be executed on the affected system with the webserver process privileges.
Information Disclosure and/or Arbitrary File Read	An exploitation of this issue might allow an attacker to get access to arbitrary files on the affected system.
Buffer/Stack/Heap/Integer Overflow, Format String and Off-by-One	Input data are copied to an insufficiently sized memory buffer. An exploitation of this issue might allow an attacker to execute arbitrary code in the context of the affected application or cause denial of service conditions.
Remote Code Execution	An exploitation of this issue might allow an attacker to execute arbitrary code within the context of the affected application, potentially allowing an unauthorized access or a privilege escalation.
SQL Injection	The vulnerable application does not properly sanitize user supplied input data before using them in a SQL query. An exploitation of this issue might allow an attacker to compromise, access and modify data on the affected system with the database user process privileges.
Unspecified Vulnerability	A successful exploitation of this issue might allow an attacker to affect confidentiality or integrity or availability or all of them.This class comprises vulnerabilities that could be not assigned to any of the other categories.

Table 4: CVE Details Categories: mapping with CVErizer's categories

CVE details Category	CVErizer Category
DoS	Denial of Service (DoS)
Code Execution	Remote Code Execution
Overflow	Buffer/Stack/Heap/Integer Overflow, Format String and Off-by-One
Memory Corruption	Buffer/Stack/Heap/Integer Overflow, Format String and Off-by-One
SQL Injection	SQL Injection
XSS	Cross-Site Scripting or HTML Injection
Directory Traversal	Directory Traversal
HTTP Response Splitting	Not Covered
Bypass Something	Authentication bypass or Improper Authorization
Gain Information	Information Disclosure and/or Arbitrary File Read
Gain Privileges	Authentication bypass or Improper Authorization
CSRF	Not Covered
File Inclusion	Local File Include, Remote File Include and Arbitrary File Upload

Table 5: Dataset

Category	CVEs
Authentication bypass or Improper Authorization	183
Cross-Site Scripting or HTML Injection	544
Denial Of Service (DoS)	520
Directory Traversal	133
Local File Include, Remote File Include and Arbitrary File Upload	153
Information Disclosure and/or Arbitrary File Read	388
Buffer/Stack/Heap/ Integer Overflow, Format String and Off-by-One	386
Remote Code Execution	257
SQL Injection	276
Unspecified Vulnerability	529
TOTAL	3369

Table 6: Survey Questions

	Question	Options
Q1	How do you judge the usefulness and comprehensibility of the provided summaries of CVE descriptions?	Very Low, Low, Medium, High, Very High
Q2	How difficult is to analyze the provided summaries of CVE descriptions?	Very Low, Low, Medium, High, Very High
Q3	How difficult is to analyze CVE descriptions without the proposed summaries?	Very Low, Low, Medium, High, Very High
Q4	Proportionally, how much time can you save by analyzing provided summaries instead of reading original CVE descriptions?	0%, 25%, 50%, 75%, 100%
Q5	Considering only the content of the summaries of CVE descriptions and not the way they are presented, do you think that in the reports?	A. Are not missing any information B. Are missing some information C. Are missing some very important information
Q5.1	If some important information is missing, can you specify which kinds of information are missed?	
Q6	Considering only the content of the summaries of CVE and not the way they are presented, do you think that the reports?	A. Contain no unnecessary information B. Contain some unnecessary information C. Contain a lot of unnecessary information
Q6.1	If summaries contain some unnecessary information, can you specify which kinds of information are useless?	
Q7	Considering only the way the CVE summaries are presented, and not their contents, do you think that the reports?	A. Are easy to read and understand B. Are somewhat readable and understandable C. Are hard to read and understand
Q8	Do you have any suggestion to improve the summaries and make them more understandable?	
Q9	In conclusion: are the summaries useful for understanding software vulnerabilities and preventing from them?	Very Low, Low, Medium, High, Very High

lysts to analyse CVE descriptions?

– **RQ2-b How do developers consider the summaries generated by** `CVErizer` **in terms of correctness, content adequacy, conciseness, and expressiveness?**

3.1. Context

The context of our study is represented by a set of 3369 vulnerabilities previously labeled according to the proposed taxonomy (see Table 3). Each vulnerability has been downloaded from the NIST National Vulnerability Database and categorized by security experts, employed at our industrial partner. Table 5 illustrates the amount of vulnerabilities assigned to each category. In particular, *"Cross Site Scripting or HTML Injection"* and *"Directory Traversal"* are the most and least populous categories of our dataset, respectively.

Listing 2: Example of an Association rule to identify "Authentication bypass or Improper Authorization" vulnerabilities

```
/*
 * VULN_01 = "Authentication Bypass or Improper Authorization"
 */

if effects matches
        "^.*bypass.*(restriction|permission).*\textdollar" OR "^.*gain.*
              privilege.*\textdollar"

OR if causes matches
        "^.*missing.*access.*(restriction|permission|validation).*\textdollar"
              OR "^.*not.*(restrict|check|enforce).*(access|permission|
              validation|use|restriction).*\textdollar" OR "^.*not.*implement
              .*(access|restriction|permission|validation).*control.*\
              textdollar"
```

3.2. Analysis Method

To answer our RQ1 we experimented with the different machine learning techniques (described in Section 2.2) and compared the performance obtained by each technique, relying on widely adopted metrics of Information Retrieval: Precision, Recall and F-Measure [33]. We trained the machine learning algorithms through a training set (T_1) containing 500 vulnerabilities extracted from our dataset (see Section 3.1), while the remaining vulnerabilities (*i.e.*, 2869 samples) were used as the test set (T_2). In particular, to populate T_1 we selected 50 different, randomly chosen samples for each category. In addition, we manually analyzed the vulnerabilities' descriptions in T_1, to define a set of association rules (*i.e.*, regular expressions). able to automatically assign a vulnerability to one of the categories described in Table 3. More specifically, each association rule checks if some of the *causes* or *effects* extracted by the tool match particular regular expressions associated with the vulnerability category, as showed in Listing 2. A total of 27 association rules have been defined and used to classify the vulnerabilities in T_2. Thus, Precision, Recall and F-measure achieved through association rules have been compared with the results obtained by the various machine learning algorithms.

To answer RQ2 we performed an experiment involving students of the cybersecurity course at Computer Science MSc (University of Molise) and asking them to complete a survey to evaluate: (i) the *practical usefulness* of `CVErizer` and how it can facilitate the vulnerabilities analysis process (RQ2-a), and (ii) the *quality* of the generated summaries along to 4 widely adopted dimensions [34, 35, 36] (RQ2-b):

1. **Correctness**: which assesses the accuracy of the information extracted by `CVErizer`.

2. **Content adequacy**: which assesses whether the generated summaries encompass all the salient information to support developers in the vulnerabilities' analysis process.

3. **Conciseness**: to evaluate whether output summaries contain some superfluous information.

4. **Expressiveness**: which measures the readability and understandability of the generated summaries.

It is worth highlighting that, during the aforementioned cybersecurity course all the participants in our study were introduced to penetration testing and vulnerability assessment activities: at the time of the experiment, all the students were already familiar with CVEs, and the ways of using CVE descriptions for exploiting (or preventing from) security flaws. Moreover, the CVE framework was already used by the surveyed students in some training activities during the course.

We randomly sampled 10 CVE records from our dataset and assigned to each of the sampled CVEs a unique identifier (id, with $1 \leq id \leq 10$). We separated the participants in two groups Group 1 (composed by 7 participants) and Group 2 (comprising 8 participants) and split the experiment in two steps (A and B). Each study participant has been randomly assigned to one of the two groups.

During step A of the experiment, each participant of Group 1 analyzed CVE descriptions having $1 \leq id \leq 5$, to manually extract relevant information (see Section 2.1) and classify them according to the categories showed in Table 3. In the same time, participants of Group 2 analyzed CVE descriptions having $6 \leq id \leq 10$ in the same way. Each participant has also been asked to report the time taken for extracting the required information from each description. Before starting with the information extraction task, an example summary related to a CVE (different from the ones considered in the study) was illustrated to experiment participants.

During step B of the experiment, participants of Group 1 analyzed the summaries produced by `CVErizer` for CVE descriptions having $6 \leq id \leq 10$ with the aim of validating their content, while subjects in Group 2 evaluated the correctness of information contained in summaries of CVE descriptions having $1 \leq id \leq 5$. As in the previous step, for each CVE summary we required to report the time spent for validating it. At the end of this phase, all the participants were required to answer the questions of our survey (see Table 6). The correctness of information contained in all the automatically-generated CVEs summaries submitted to students, were first manually checked by two authors of the work.

The time for the entire experiment was 60 minutes: (i) 30 minutes for step A; (ii) 20 minutes for step B; (iii) 10 minutes for answering the questionnaire.

Moreover, we designed a Feigenbaum test [37], to better understand the extent to which automatically generated CVE summaries through `CVErizer` can be distinguished from manually generated ones. The Feigenbaum Test (FT) can be regarded as a generalized Turing test in which an intelligent system in a professional domain should behave as a human expert, and the behavior can not be *distinguished* from human experts, when judged by human experts in the same domain [38]. To this aim, we reused some of the experimental material used in our experiment with students. Specifically, from each students' group we selected the subject belonging to that group that provided the best responses in the step A of the experiment (*i.e.*, Subject 5 for Group 1 and Subject 8 for Group 2) and built a dataset containing 10 CVE summaries, some of them were human-generated, while the remaining were machine-generated. In particular, for 5 out of 10 CVEs (*i.e.*, *CVE-2016-2108, CVE-2016-2560, CVE-2016-4072, CVE-2016-6289* and *CVE-2016-7128*) we considered the vulnerability summaries generated by the selected human subjects, while for the remaining 5 CVEs (*i.e.*, *CVE-2016-0777, CVE-2015-1927, CVE-2015-5174, CVE-2015-5352* and *CVE-2015-6658*) we considered the vulnerability summaries generated by `CVErizer`. These CVE summaries (along with the vulnerabilities' original descriptions) were provided to 4 security experts (*i.e.*, 1 lab director and 3 security analysts) employed at Cybaze S.p.A., who were asked to (i) separately indicate whether each CVE summary was generated either automatically or manually, and (ii) fill the same questionnaire provided to students. It is worth to highlight that the surveyed security experts were unaware of the amounts of manually-generated and machine-generated summaries. Due to the limited availability of professionals, and the need of obtaining evaluations on the widest possible spectrum of summaries, we designed the experiment so that it could be finalized in one hour. In particular, we scheduled a time slot of 5 minutes for analyzing each CVE summary, and a time slot of 10 minutes for completing the survey.

We assessed the proportions of human-machine decisions for both human-generated and machine-generated summaries. Moreover, for statistically evaluating the reliability of agreement between the human-machine judgments given by the surveyed experts, we computed the Fleiss' kappa coefficient [39].

3.3. Research Method

To assess the practical usefulness of `CVErizer` and answer RQ2-a, we asked the survey participants to report the time required for each analyzed description, for performing the information extraction (step A of the experiment), and validation (step B of the experiment) processes. We also asked to express their opinion on the speed-up introduced by the summaries in analyzing CVEs (Q4 in Table 6). To qualitatively complement these data, we asked the participants to judge the usefulness and comprehensibility of the provided summaries (Q1 in Table 6), and to express their opinion on difficulties encountered when analyzing vulnerabilities' descriptions, with (Q2 in Table 6) and without (Q3 in Table 6) the proposed summaries.

To answer RQ2-b we asked the participants to manually validate the accuracy of data (step B of the experiment) contained in each summary (*i.e.*, at least 7 humans validated each summary). Moreover participants have also been invited to provide their opinions on: (i) the content adequacy (Q5 in Table 6), (ii) the conciseness (Q6 in Table 6), and (iii) the expressiveness (Q7 in Table 6) of the provided summaries. We, finally, enrich these data by asking participants: (i) to provide suggestions for improving summaries (Q8 in Table 6), and (ii) to provide a general judgment on the usefulness of the summarization approach in a real working context (Q9 in Table 6). Survey participants were explicitly instructed to use survey open questions for reporting (i) missing (Q5.1), or (ii) useless (Q6.1) information contained in CVErizer summaries, and (iii) general feedback about understandability (Q8) of such summaries.

4. Evaluation

We illustrate the results obtained in our experiments, to answer the research questions stated in Section 3.

Table 7: Classification Results

Class	J48			BayesNet			NaiveBayes			Simple Logistic			RandomForest			Association Rules		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Authentication bypass or Improper Authorization	0.50	0.71	0.59	0.46	**0.79**	0.58	0.51	0.74	0.61	0.41	0.74	0.53	0.50	0.78	0.61	**0.65**	0.61	**0.63**
Cross-Site Scripting or HTML Injection	0.96	0.83	0.89	0.96	**0.94**	**0.95**	0.96	0.92	0.94	**0.97**	0.83	0.89	0.95	0.93	0.94	**0.97**	0.86	0.91
Denial of Service (DoS)	0.57	**0.90**	0.70	0.86	0.88	**0.87**	0.85	0.83	0.84	0.65	**0.90**	0.76	0.78	0.86	0.82	**0.87**	0.78	0.82
Directory Traversal	**0.73**	0.70	0.71	0.65	**0.82**	0.72	0.63	0.76	0.69	0.72	0.76	**0.74**	0.69	0.76	0.72	0.62	0.66	0.64
Local File Include. Remote File Include and Arbitrary File Upload	0.78	0.79	0.78	0.80	0.78	**0.79**	0.66	0.78	0.71	0.55	0.80	0.65	0.70	**0.85**	0.76	**0.88**	0.64	0.74
Information Disclosure and/or Arbitrary File Read	0.68	0.57	0.62	0.79	0.62	0.69	0.80	0.64	**0.71**	0.78	0.63	0.70	0.78	**0.65**	**0.71**	**0.84**	0.39	0.53
Buffer/Stack/Heap/ Integer Overflow. Format String and Off-by-One	0.85	0.88	**0.86**	0.76	0.87	0.81	0.71	0.87	0.78	0.85	0.87	**0.86**	0.75	**0.90**	0.82	**0.86**	0.87	**0.86**
Remote Code Execution	0.66	**0.77**	**0.71**	0.68	0.58	0.63	0.58	0.56	0.57	0.65	0.69	0.67	0.73	0.46	0.56	**0.77**	0.53	0.63
SQL Injection	0.75	0.92	0.83	0.92	0.91	**0.92**	0.93	0.89	0.91	0.83	0.91	0.87	0.88	**0.93**	0.91	**0.97**	0.87	**0.92**
Unspecified Vulnerability	0.84	0.29	0.43	**0.88**	0.79	**0.83**	**0.88**	0.78	**0.83**	0.85	0.44	0.58	0.86	0.68	0.76	0.77	**0.83**	0.80
Weighted Avg.	0.76	0.72	0.71	**0.83**	**0.81**	**0.81**	0.81	0.80	0.80	0.78	0.75	0.74	0.80	0.79	0.79	0.82	0.70	0.76

Table 8: Confusion Matrix related to BayesNet Algorithm

a	b	c	d	e	f	g	h	i	j	← *classified as*
105	0	0	0	0	16	1	7	2	2	a = AuthenticationBypass
2	464	4	2	3	6	0	5	6	2	b = CrossSiteScripting
6	0	412	0	0	1	16	4	0	31	c = DenialOfService
5	1	0	68	4	4	1	0	0	0	d = DirectoryTraversal
0	0	0	14	80	3	0	6	0	0	e = FileInclude
66	7	12	18	7	208	5	5	2	8	f = InformationDisclosure
2	2	18	0	0	6	291	17	0	0	g = Overflow
3	2	6	0	4	4	56	120	7	5	h = RemoteCodeExecution
1	5	0	2	2	5	0	4	205	2	i = SQLInjection
39	1	30	1	0	12	12	8	0	376	j = UnspecifiedVulnerability

Table 9: Classification Results with 10 fold cross-validation

Class	J48			BayesNet			NaiveBayes			Simple Logistic			RandomForest		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Authentication bypass or Improper Authorization	0.63	0.65	0.64	0.63	0.79	0.70	0.59	0.83	0.69	0.71	0.67	0.69	0.76	0.69	0.72
Cross-Site Scripting or HTML Injection	0.95	0.94	0.95	0.96	0.95	0.95	0.97	0.95	0.96	0.94	0.94	0.94	0.94	0.95	0.94
Denial of Service (DoS)	0.87	0.89	0.88	0.85	0.88	0.86	0.88	0.84	0.86	0.87	0.85	0.86	0.83	0.91	0.87
Directory Traversal	0.76	0.82	0.79	0.69	0.85	0.76	0.71	0.84	0.77	0.77	0.83	0.80	0.78	0.79	0.79
Local File Include. Remote File Include and Arbitrary File Upload	0.80	0.75	0.77	0.84	0.78	0.81	0.85	0.78	0.81	0.84	0.79	0.82	0.82	0.77	0.80
Information Disclosure and/or Arbitrary File Read	0.71	0.75	0.73	0.75	0.74	0.74	0.75	0.73	0.74	0.72	0.77	0.75	0.73	0.80	0.76
Buffer/Stack/Heap/ Integer Overflow. Format String and Off-by-One	0.86	0.87	0.86	0.87	0.84	0.85	0.82	0.87	0.85	0.86	0.88	0.87	0.84	0.85	0.85
Remote Code Execution	0.73	0.77	0.75	0.78	0.69	0.73	0.74	0.70	0.72	0.76	0.75	0.76	0.82	0.72	0.77
SQL Injection	0.93	0.92	0.93	0.94	0.90	0.92	0.94	0.90	0.92	0.97	0.92	0.94	0.93	0.92	0.92
Unspecified Vulnerability	0.88	0.81	0.84	0.84	0.81	0.83	0.88	0.80	0.84	0.83	0.82	0.83	0.88	0.82	0.85
Weighted Avg.	**0.84**	**0.84**	**0.84**	**0.84**	**0.84**	**0.84**	**0.84**	**0.83**	**0.84**	**0.84**	**0.84**	**0.84**	**0.85**	**0.85**	**0.85**

4.1. RQ1 results

Table 7 shows, for each class of our taxonomy, the Precision (P), Recall (R) and F-measure (F1) results obtained through the different machine learning algorithms (see Section 2), as well as the classification performance achieved through the defined regular expressions (*i.e.*, Association Rules). Specifically, the BayesNet classifier results the best performing classifier with an overall precision of 83%, a recall of 81% and a F-measure of 81%. This specific algorithm achieves the best F-measure results in 5 out of 10 categories (*i.e.*, *"Cross Site Scripting or HTML Injection", "Denial of Service", "Local File Include, Remote File Include and Arbitrary File Upload"*, "SQL Injection" and *"Unspecified Vulnerability"*) and three of these categories (*i.e.*, *"Cross Site Scripting or HTML Injection", "Denial of Service"* and *"Unspecified Vulnerability"*) are the most densely populated ones in our dataset.

On the other hand, the J48 Classifier achieved the worst classification results with a general precision of 76%, a recall of 72% and an F-measure of 71%. Indeed, we observe that the J48 machine learning algorithm obtained the worst F1 measure results in the three most densely populated categories of our dataset, along with the *"SQL Injection"* class.

In Table 8 the confusion matrix related to the best performing ML algorithm (*i.e.*, BayesNet) is shown. By observing this matrix, we can notice that the main inaccuracies derive from misclassifications of vulnerabilities belonging to the (i) *"Denial Of Service"* (*i.e.*, 31 samples are erroneously classified as belonging to the *"Unspecified Vulnerability"* class), (ii) *"Information Disclosure"* (*i.e.*, 66 samples are erroneously classified as belonging to the *"Authentication Bypass"* class), (iii) *"Remote Code Execution"* (*i.e.*, 56 samples are erroneously identified as belonging to the *"Overflow"* class), and (iv) *"Unspecified Vulnerability"* (*i.e.*, 39 samples are erroneously classified as belonging to the *"Authentication Bypass"* class and 30 samples are wrongly identified as belonging to the *"Denial of Service"* class) categories.

Looking deeper into recurrent classification errors, we realize that (i) many of the items in the *"Denial of Service"* class and wrongly identified as belonging to the *"Unspecified Vulnerability"* category are positioned by the algorithm in the *"Unspecified Vulnerability"* category probably due to the fact that they report *"unspecified vulnerability … affect availability"* in the descriptions. Moreover, (ii) in many of the items in the *"Information Disclosure"* class and wrongly identified as belonging to the *"Authentication Bypass"* class, keywords as *"bypass security mechanism"* or *"bypass security restriction"* appearing in their descriptions could lead the ML model to assign the wrong category. Furthermore, (iii) many of the items actually belonging to the *"Remote Code Execution"* category and classified as belonging to *"Overflow"*, contain the keywords *"Buffer Error"* in their description. Finally, as previously mentioned, (iv) vulnerabilities of the *"Unspecified Vulnerability"* are sometimes positioned by the ML algorithm in the *"Authentication Bypass"* (when the descriptions contain keywords like *"gain privilege"*) or in the *"Denial of Service"* (when the descriptions contain keywords like *"cause denial of service"*) categories.

Whilst we achieved promising performance, further research is needed to limit inaccuracies in the results, as erroneous categorizations could lead experts to underestimate (or overrate) risks that could arise from specific vulnerabilities (*e.g.*, it may be erroneously assigned a low-level priority to a vulnerability actually belonging to the *"Denial of Service"* category but wrongly classified as *"Unspecified Vulnerability"*, even if the assessor could be interested in first mitigating unavailability flaws of the system).

The classification through association rules (Assn. Rules) produced very promising results with a better F1-measure (*i.e.*, 76%) than two (*i.e.*, J48 and SimpleLogistic) out of five machine learning classifiers. In particular, association rules obtained a quite similar precision (82% related to Assn. Rules and 83% related to BayesNet) with the best performing classification algorithm (*i.e.*, BayesNet) but a lower recall. This emerges from the fact that association rules approach obtains the best precision results (outperforming all the ML techniques) in 8 out of 10 categories. Thus, we argue that the definition of more rules (*i.e.*, actually 27 total rules have been defined), could improve the overall recall (and, consequently, the F1-measure) of this approach. For instance, the recall results achieved by this approach in the *"Information Disclosure and/or Arbitrary File Read"* and *"Remote Code Execution"* classes are very poor and the definition of new rules for these two categories may improve these outcomes (and, consequently, the overall performance).

To mitigate concerns related to overfitting and asymmetric sampling, we repeated the classification experiment by applying 10-fold cross-validation. Table 9 reports the results of this experiment, in which we can observe improvements in the F1-measure ranging from 0.03 (*i.e.*, BayesNet) to 0.13 (*i.e.*, J48) when compared with the previously obtained results. Moreover, in this new setting the best performing classification algorithm resulted RandomForest, probably due to the fact that more precise decision trees may be inferred when considering higher numbers of points in the training set (*i.e.*, in each run of the 10-fold cross-validation the training set was composed by 90% of items in the overall dataset).

Summary RQ1: *Among all the experimented classifiers, the BayesNet algorithm achieves the best classification results in terms of both precision (83%), recall (81%) and F-measure (81%). Very promising results are also obtained by the defined association rules, that in 8 out of 10 categories achieve the best precision results.*

4.2. RQ2 results

In this section, we report the results concerning our RQ2.

4.2.1. RQ2-a results

All the subjects involved in our experiment considered CVE summaries time-saving. In particular, 66.7% of survey respondents (10 out of 15) replied that CVErizer allowed them to save at least 75% of the time that they would otherwise have spent on manually analyzing vulnerabilities. Even if 2 subjects answered that the saved time was even 100%, these answers should not be considered, as a 100% time reduction would be acceptable only

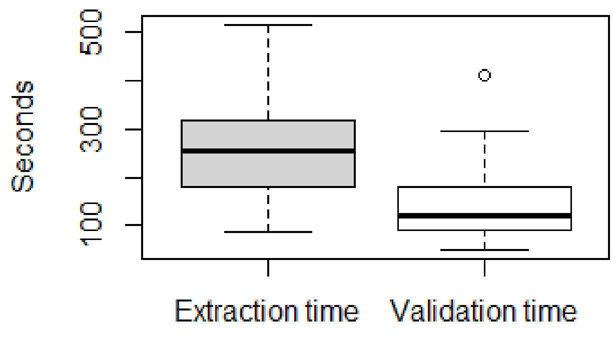


Figure 3: Tasks' Completion Times

in the case in which the vulnerability assessment and prioritization processes were fully automated. The remaining 3 subjects (20%) affirmed that CVE summaries saved 50% of their time. These results are related to the time saving capability of summaries perceived by the participants. Since this measure is fairly subjective, a more quantitative evaluation is required. In order to avoid a biased estimation of these results, we compared the information extraction times declared by subjects in step A of the experiment with the validation times reported by the participants in step B of the experiment (see Section 3.2). Figure 3 shows the distributions of times taken by the participants for (i) extracting data from each vulnerability's description (*i.e.*, step A of the experiment), and (ii) validating each summary (*i.e.*, step B of the experiment). Specifically, the difference between the two distributions is statistically significant (the Mann-Whitney U test [40] returned a $p-value << 0.0001$) with a large effect size (d = 0.76), measured through the Cliff's delta [41]. These results suggest that, when supported by our summarization approach, participants spent on average 50% less time for analyzing vulnerabilities. It is worth noticing that, when performing vulnerability assessment activities (especially for third parties), security experts are required to extract salient information from vulnerabilities descriptions, in order to (i) profile security flaws that could affect company's assets, and, consequently, (ii) prioritize the mitigation of defects that could have consequences on the IT infrastructure. Thus, a 50% time reduction represents just a lower bound, as in a real working scenario we expect that vulnerability assessors would be not required to validate each single report.

From a qualitative point of view, all the subjects considered the provided summaries highly/enough useful and comprehensible (see Q1 in Table 10). In particular, 60% of participants (9 out of 15) considered the summaries highly (or very highly) useful and comprehensible. Only 2 participants out of 15 (13.3%) replied that analyzing vulnerabilities with the provided summaries is hard (see Q2 in Table 10), while the same activity without the summaries is considered challenging or very challenging by 46.7% (7 out of 15) of the survey respondents (see Q3 in Table 10).

Summary RQ2-a: *Summaries of vulnerabilities' descriptions help to save about a half of the time required for manually analyzing vulnerabilities. CVErizer summaries are considered useful and comprehensible.*

Table 10: Aggregated answers

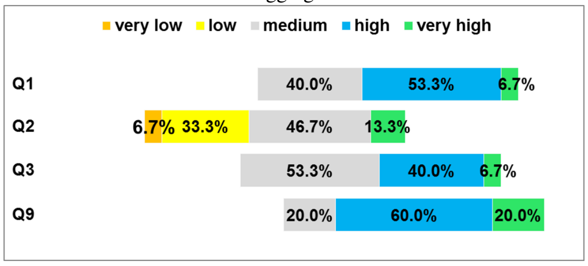


Table 11: Step A and B outputs

	Step A		**Step B**	
Information class	humans' output ≡ tool's output	humans' output ≠ tool's output	labeled as correct	labeled as incorrect
Software name	88.0%	12.0%	89.3%	10.7%
Version	74.7%	25.3%	94.7%	5.3%
Previous version	92.0%	8.0%	98.7%	1.3%
Vulnerability name	82.7%	17.3%	97.3%	2.7%
Attacker	90.7%	9.3%	94.7%	5.3%
Cause	30.7%	69.3%	62.7%	37.3%
Effect	66.7%	33.3%	88.0%	12.0%
Vulnerability category	74.4%	25.3%	62.7%	37.3%
Average	**75.0%**	**25.0%**	**86.0%**	**14.0%**

4.2.2. *RQ2-b results*

Table 11 reports for each information class (each row) the average percentage of times that (i) participants used to extract the same information the tool extracted, (ii) subjects extracted an information differing from tool's output, (iii) participants used to label the data reported by the CVErizer summaries as correct, and (iv) subjects labeled information contained in summaries as incorrect. On average, 86% of data extracted by our tool has been considered correct by the participants and in 75% of cases the information reported in summaries coincides with the information extracted by humans. According to study participants, the tool exhibits an high accuracy (88% and up) in extracting (i) software name, (ii) version, (iii) previous version, (iv) vulnerability name, (v) attacker, and (vi) effect information, while it achieves lower performance (less than 65%) in mining cause and category information (step B of the experiment). In particular, in 69.3% of cases subjects extracted a cause differing from the cause contained in the summary.

However, comparing the results obtained in the two steps of the experiment, we can notice some discrepancies between the outcomes of Step A (which consists in the manual extraction of the information related to CVEs) and the achievements of Step B (which consists in the validation of information extracted by our tool). In particular, for *Cause* and *Effect* information classes, while in Step A the percentages of times that subjects reported an information differing from the one extracted by the tool are considerable (*i.e.*, 69.3% and 33.3%, respectively), in Step B participants labeled as incorrect the information extracted by the tool fewer times (*i.e.*, 37.3% and 12.0%,

Table 12: Examples of CVEs used in our experiment

CVE	Description
CVE-2015-6658	*Cross-site scripting (XSS) vulnerability in the Autocomplete system in Drupal 6_x before 6_37 and 7_x before 7_39 allows remote attackers to inject arbitrary web script or HTML via a crafted URL, related to uploading files.*
CVE-2016-6289	*Integer overflow in the virtual_file_ex function in path-0 in PHP before 5_5_38, 5_6_x before 5_6_24, and 7_x before 7_0_9 allows remote attackers to cause a denial of service stack-based buffer overflow) or possibly have unspecified other impact via a crafted extract operation on a ZIP archive.*
CVE-2016-7128	*The exif_process_IFD_in_TIFF function in path-0 in PHP before 5_6_25 and 7_x before 7_0_10 mishandles the case of a thumbnail offset that exceeds the file size, which allows remote attackers to obtain sensitive information from process memory via a crafted TIFF image.*

respectively). Thus, we perform a qualitative analysis of the results achieved in Step A of the experiment, by investigating more in depth the differences between manually extracted information and the tool's outputs. Such qualitative analysis highlights the tool's usefulness in correctly extracting the right amount of information needed by security teams or vulnerability assessors to have a quick overview of semantic information related to a generic vulnerability. In Table 12 some of the CVEs (with the related descriptions) discussed in our analysis are reported. In particular, `CVErizer` helps to avoid human errors made in extracting valuable information from CVEs. For instance, for `CVE-2016-7128` (illustrated in Table 12), the tool correctly indicated *"crafted TIFF image"* as the main *cause* of the vulnerability, while 4 (out of 7) participants wrongly reported *"The exif_process_IFD_in_TIFF function in path-0"* which represents the function and the path involved by the vulnerability, but not the *cause*, and 2 (out of 7) participants have reported *"Process memory via a crafted TIFF image in the exif_process_IFD_in_TIFF function in path-0"* which is a partially wrong information because, along with the *cause*, it also includes the function and the path information, which is not required. Another example is related to `CVE-2015-6658` (detailed in Table 12), for which the tool correctly indicated *"crafted URL"* as the main *cause* of the vulnerability, while participants reported four different misleading information. In particular, (i) subject 11 reported *"Inject arbitrary"*, which does not represent the *cause* of the vulnerability, (ii) subject 12 was not able to identify any *cause* of the vulnerability, (iii) subject 15 reported *"Autocomplete system"*, which is a Drupal's module and not the origin of the vulnerability, while (iv) subject 10 reported *"Exploiting crafted URL relating to uploading files"*, which is semantically correct, but it contains words not included in the CVE description.

The main goal of our summarizer is to present a quick overview of the information related to unstructured texts (*i.e.*, vulnerabilities descriptions) to experts, by providing the right amount of information useful for supporting humans in the decision making processes. During step A of the experiment, we observe that in most cases subjects reported more information than needed. For example, by analyzing the information reported by the participants for `CVE-2016-6289` (see Table 12), we can notice that 4 subjects out of 7 wrongly included the *cause* of the vulnerability (*e.g.*, *"crafted extract operation"*) and the *attacker* (*e.g.*, *"remote attackers"*) information, while extracting the vulnerability's effects (*e.g.*, *"It allows remote attackers to cause a denial of service or possibly have unspecified other impact via a crafted extract operation on a ZIP archive"*), instead of just indicating *"cause denial service, have unspecified other impact"*, as correctly reported by our tool. A similar example occurred for `CVE-2016-7128` (shown Table 12), in which `CVErizer` correctly extracted *"obtain sensitive information"* as the *effect* of the vulnerability, while subjects 3,4,6 and 7 wrongly included the cause (*e.g.*, *"crafted TIFF image"*) and subjects 3 and 6 also included the attacker (*e.g.*, *"remote attackers"*) information for describing the *effects* of the vulnerability.

Table 13: Raw data of the questionnaire concerning the evaluation of summaries.

Content adequacy

Response category	Ratings
A) Are not missing any information.	8 (53.3%)
B) Are missing some information but the missing information is not necessary to have an overview of software vulnerabilities	7 (46.7)%
C) Are missing some very important information	0 (0%)

Conciseness

Response category	Ratings
A) Contain no unnecessary information	15 (100%)
B) Contain some unnecessary information	0 (0%)
C) Contain a lot of unnecessary information	0 (0%)

Expressiveness

Response category	Ratings
A) Are easy to read and understand	11 (73.3%)
B) Are somewhat readable and understandable	4 (26.7)%
C) Are hard to read and understand	0 (0%)

Nevertheless, as reported in Table 13, none of the subjects stated that summaries lack very important information to have an overview of software vulnerabilities. Specifically, 8 subjects out of 15 (53.3%) considered the summaries' content to be adequate, while the remaining (46.7%) declared that, even if there were a possible information loss, our summaries still provide a complete overview of vulnerabilities. In particular, 4 out of 15 of the survey respondents complained about the fact that some of the summaries were missing the vulnerability name information. Moreover, all the survey participants affirmed that summaries did not contain unnecessary information (see Table 13) and 73.3% (11 out of 15) of them considered the summaries easy to read and understand (see Table 13).

In conclusion, 80% (12 out of 15) of participants perceived CVErizer summaries as highly useful for understanding software vulnerabilities and preventing from them (see Q9 in Table 10).

Concerning the Feigenbaum test, in Table 14 the results of the human-machine decisions provided by the surveyed security experts (see Section 3.2) are shown. For summaries generated by our tool (*i.e.*, *CVE-2016-0777, CVE-2015-1927, CVE-2015-5174, CVE-2015-5352* and *CVE-2015-6658*) only in 15% (*i.e.*, 3 out of 20) of the cases the expert decisions fell on the *"Human"* category; for manually-generated summaries in 70%

Table 14: Human-machine decisions as provided by security experts.

CVE	Source	SubjectA	SubjectB	SubjectC	SubjectD
CVE-2016-0777	**CVErizer**	CVErizer	CVErizer	CVErizer	CVErizer
CVE-2015-1927	**CVErizer**	CVErizer	CVErizer	CVErizer	Human
CVE-2016-2108	**Human**	CVErizer	CVErizer	Human	CVErizer
CVE-2016-2560	**Human**	Human	Human	CVErizer	CVErizer
CVE-2016-4072	**Human**	CVErizer	CVErizer	CVErizer	CVErizer
CVE 2015-5174	**CVErizer**	CVErizer	CVErizer	CVErizer	Human
CVE-2015-5352	**CVErizer**	CVErizer	CVErizer	CVErizer	CVErizer
CVE-2016-6289	**Human**	CVErizer	Human	Human	Human
CVE-2015-6658	**CVErizer**	CVErizer	Human	CVErizer	CVErizer
CVE-2016-7128	**Human**	CVErizer	CVErizer	CVErizer	CVErizer

(*i.e.*, 14 out of 20) of the cases the security experts wrongly identified the summaries as generated by the tool. This suggests that the summaries provided by our tool are very similar to the ones provided by humans. Indeed, when surveyed experts were asked why they often selected the CVErizer option, they reported that most of the information entailed in the summaries contained the exact words that were in the descriptions; this led them to mostly choose this option. In addition, we report that appraisal agreements between raters, when they are asked to identify the information source, mostly occur by chance, as the Fleiss' kappa test returned $k = 0.092$.

As discussed in Section 3.2 we also asked the professional experts participating in our study to answer the questions of the survey (reported in Table 6). All these experts considered the summaries provided by our tool highly (2 out of 4) or very highly (the remaining two subjects) useful (Q1), and most of them reported that the difficulty of analyzing CVEs through `CVErizer` (Q2) is low (50%) or very low (25%). Interestingly, as students, also all the surveyed professional security experts believe that such summaries allow to save at least 50% time otherwise spent in analyzing original CVE descriptions (Q3). However, only one subject stated that summaries are not missing any information (Q5), while the remaining 3 subjects reported that, even if the summaries are lacking some information (*e.g.*, impacts on confidentiality, integrity and availability), it is not relevant for having an overview of the vulnerability. 75% of subjects answered that the summaries contain no unnecessary information (Q6) and all of them considered the summaries easy to read and understand (Q7). In conclusion, 75% of subjects regarded the summaries as highly useful for preventing from vulnerabilities.

Summary RQ2-b: *According to the experiment participants, the summaries generated by CVErizer are reasonably correct, adequate, concise, and expressive. Moreover, they are also considered useful to prevent from software vulnerabilities.*

5. Case Study: Assessing Vulnerabilities of Hypervisors

To estimate how CVErizer could satisfy the needs of security experts in an industrial scenario and assess the tool's usefulness when adopted in specific contexts, in this section we report the results of a case study conducted at our industrial partner and aimed at evaluating CVErizer for assessing vulnerabilities that affect hypervisor systems.

Hypervisors are widely used for malware analysis activities [42]. To keep the malware under analysis in a controlled environment and avoid infections of the host environments, it is of primary importance mitigating hypervisor vulnerabilities [43]. Thus, cyber security companies performing malware analysis activities should constantly monitor the security flaws affecting hypervisor technologies, to avoid threats that may compromise their IT infrastructure.

In the context of an industrial case study conducted at Cybaze S.p.A. and aimed at evaluating vulnerabilities that affect two competing hypervisor products, (i) *VMware ESXi*[13], and (ii) *XEN*[14], `CVErizer` has been assessed as enabling technology for a simplified, rapid and intuitive assessment of vulnerabilities occurring on such tools. In particular, it has been evaluated the extent to which `CVErizer` could provide correct vulnerabilities-related information to security experts, for the specific context of hypervisor products.

In practice, for conducting the evaluation, we provided a modified version of `CVErizer` (*i.e.*, able to summarize customized subsets of CVEs) to Davide, a professional penetration tester employed at Cybaze. Moreover, all the required information for modifying/refining XML heuristics has been provided to Davide. To assess `CVErizer` effectiveness and evaluate whether modifications/refinements were required for adapting the original heuristics' set to the specific context of hypervisors, Davide (i) queried the MITRE database, for collecting all the vulnerabilities affecting either *VMWare ESXi* or *XEN*, (ii) sampled a total of 24 CVEs (*i.e.*, 12 vulnerabilities spanning from 2016 to 2018 for each hypervisor), (iii) analyzed the heuristics' set, with the aim of applying changes when it was necessary, (iv) launched our summarizer using the 24 sampled CVEs as input, (v) manually checked the correctness of the automatically extracted information from each CVE, and (vi) reported, for each *information class* and each considered hypervisor, the amount of cases in which the tool extracted correct information from unstructured descriptions.

The results of the assessment performed by Davide, are reported in Figure 4. Our tool, was able to correctly extract the *software name, versions* and *vulnerability category* information in more than 90% of the cases for both considered hypervisors, while we observe that for the *cause, effect, vulnerability name* and *attackers* categories, lower levels of accuracy are achieved. More in-depth, while for the *cause* information class, we notice that results obtained for both hypervisors are similar, when considering the outcomes related to the *effect, vulnerability name* and *attacker* classes, we report a higher accuracy ($\simeq$ 85%, 100% and $\simeq$ 70%, respectively) achieved for the *XEN* hypervisor. These dissimilarities in the results, are manly due to the way in which the vulnerabilities are described (*i.e.*, vulnerabilities reported for *XEN* comply with a more stable template, while for *VMWare ESXi* inaccuracies on the *cause, effect, vul-*

[13]`https://www.vmware.com/products/esxi-and-esx.html`

[14]`https://www.cl.cam.ac.uk/research/srg/netos/projects/archive/xen/`

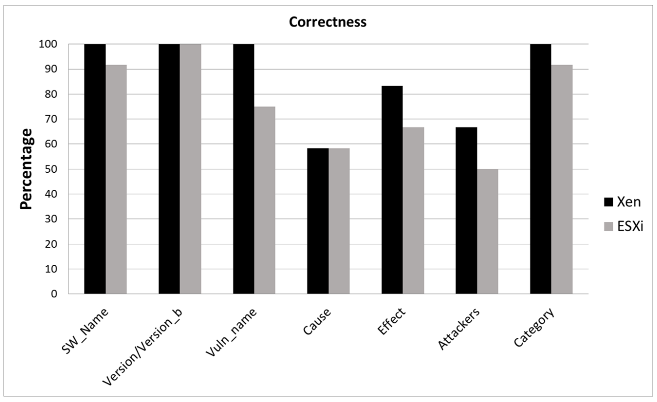


Figure 4: Percentages of cases in which the tool extracted correct information.

nerabiliy name and *attacker* classes mainly occur for vulnerabilities reported in the year 2018), and, according to Davide's opinion, the majority of inaccuracies could be fixed by further refining the heuristics' set. From this last result, we can observe how automated approaches aimed at mining information from unstructured sources are sensitive to mutations in writing styles. Since writing styles may change at any moment, a way to easily adapt the approach to such changes is surely desirable.

After the experiment, we asked Davide to provide general feedback about his experience with `CVErizer`. In particular, Davide feels that `CVErizer` *"... if well integrated in the companies internal processes, would really be a great help for the system operators for both the resolution of vulnerabilities and to make systemists aware about what every single vulnerability could cause to the entire IT infrastructure ... "*. During the assessment, Davide also refined some heuristics to adapt the tool to the specific context of hypervisors; *"... in general the tool was already working correctly; I just added two or three heuristics (related to the identification of specific causes and attacker types), and refined some existing NLP heuristics to improve the tool's output"*. In particular, Davide reported that *"... the XML grammar used to define heuristics is very easy to use"*. For this reason, Davide found the heuristics set *"very simple to expand/maintain"*. Finally, Davide provided us ideas for improving the tool: *"A reminder system with notifications (e.g., when new CVEs related to specific assets are discovered) could improve the reaction time of the system engineers"*. Moreover, according to his opinion *"... it would be also useful to provide a functionality for system engineers to require new heuristics for treating specific CVE cases or extracting particular information"*.

6. Threats to Validity

Internal validity. These kinds of threats concern any confounding factor that could influence our results. We evaluated `CVErizer`'s classification capabilities on a set of 3369 vulnerabilities preliminary labeled by industrial subjects for different strategic purposes. This might be a potential threat to validity, since we are not totally aware of the manual coding process followed by these subjects. In addition, the quality of generated summaries has been assessed through data provided by human subjects, hence it could be biased. Moreover, during the experiment, the extraction (step A) and validation (step B) times have been manually reported by the participants and this could lead to inaccuracy. To mitigate this issue, we provided detailed timing instructions to participants (*i.e.*, the starting and stopping of the chronometer are event triggered, we explained to participants how to recognize these events).

Misunderstanding of some survey questions could have some impact on the conclusions that can be drawn. To alleviate this issue, before starting to answer the questionnaire, we carefully explained each question to participants, and the information that we expect to gain from it. Furthermore, learning and fatigue effects could have affected the results obtained in the two tasks performed by study participants, as CVEs in each task were annotated by students of the same group following the same order. To alleviate these issues, we fixed reasonable time limits (30 mins and 20 mins for step A and B, respectively) for each task.

External validity. These kinds of threats are related to the generalizability of our findings. Our study is exploratory and a larger scale validation is necessary. In particular, the results obtained in our controlled experiment may be specific to the 10 vulnerabilities we have selected. To counteract this issue, the selected vulnerabilities (i) belong to different categories of our taxonomy, and (ii) have descriptions which vary in terms of size, lexicon and contents. Moreover, the majority of subjects involved in our experiment students (see Section 3.2). This could be a potential threat to validity because professional and experienced security analysts could perceive (or consider) summaries in a different way. To counteract this issue we additionally surveyed professional security experts, and presented the results of an industrial case study in which CVErizer has been used to assess the vulnerabilities affecting hypervisor products. In addition, we carried out our study on students attending an advanced course on software security. This course assumes that such students have basic knowledge about (i) software and protocols vulnerabilities, and (ii) how to perform vulnerability assessment. According to previous work [44], using students as subjects in software engineering experiments is perfectly reasonable when the study involves basic programming and comprehension skills. Nevertheless, further studies involving a larger set of experimental materials and professional subjects are needed, in order to generalize our results.

7. Related Work

This section discusses related work about text summarization and text mining in vulnerability databases.

7.1. Text Summarization

According to Radef *et al.* [45], a summary can be defined as *"a text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually, significantly less than that"*. The existing approaches to text summarization are extractive or abstractive. The extractive summarization consists of extracting parts of the input document and combines them to form a shorter summary [46, 47]. While extractive models can not use sentences which do not occur in the text, abstractive summarization consists of producing a brief text that resumes the source document, using new sentences.

In the software engineering domain, summarization techniques have been employed for helping developers and software engineers during different tasks. Specifically, summarization may result particularly useful for assisting developers during decision making processes [48]. Indeed, software artifacts, such as requirement documents or bug reports, may contain excessive information and automated summarization approaches can help developers to find the information necessary for the task [49]. For example, several research efforts have been devoted in summarizing bug reports. Rastkar *et al.* [50] employed an existing technique [51], originally conceived for emails and conversations summarization, to produce summaries of bug reports. Subsequent studies [52, 53, 54, 55] in bug report summarization complemented the Rastkar *et al.*'s work and used it as a benchmark for developing or improving summarization systems [48].

Other studies presented approaches for generating summaries (i) to allow developers better understanding the reasons behind specific code changes [56, 57], (ii) to help developers digesting user feedback contained app reviews [36, 58, 59], or (iii) to recover traceability links between emails and source code [60]. In this paper, we make use of some techniques previously employed for locating salient information within unstructured texts [36, 20] and evaluate the quality of summaries on content adequacy, conciseness and expressiveness standards, as it happened in several previous works [35, 36].

7.2. Text mining in vulnerability databases

Several studies [61, 62, 63, 64, 65, 66, 27] use text mining techniques on CVEs for retrieving textual information from this repository, in order to perform a variety of tasks.

Bozorgi *et al.* [61] were the first researchers that focused on textual information hidden in vulnerabilities databases. Actually, their work aims to define a more reliable risk score for vulnerability than the expert evaluation. This is the first difference with our paper, while the second difference refers to the approach used. Bozorgi and colleagues employ common algorithms of machine learning, while we propose a new method for summarizing CVEs.

Le and Loh [62] used Natural Language Processing to support the automatic derivation of a Vulnerable Property Relation Graph (VPRG) model based on properties extracted from text-based vulnerability descriptions. A VPRG graph is not a summary of the CVE, which is the purpose of CVErizer, but it models the process of vulnerability exploitation. Moreover authors leverage existing tools for extracting information from the text, while we developed a text summarizer specifically tailored for extracting certain categories of information from a CVE. Simmons *et al.* [63] propose an issue resolution system (IRS) to detect and extract information from external vulnerability repositories and internal log files to assist with classifying and disseminating defenses. Mokhov *et al.* [64] used natural language processing (NLP) and artificial intelligence (AI) methods for finding segments of code affected by specific vulnerabilities. This problem is apparently different from the problem faced by our paper, because authors treat the code as a text and mine it with classical n-gram, smoothing techniques (add-, Witten-Bell, MLE, etc.) and machine learning combined with dynamic programming. Scandariato *et al.* [67] studied how to predict the vulnerable components of software, based on text mining applied to the components' source code. Glanz *et al.* [65] proposed to enrich the National Vulnerability Database (NVD) with the version ranges of each vulnerable software component along with the version numbers through a rule-based approach able to automatically extract this information from the informal vulnerability description. The authors use an approach based on entity extraction. We obtain a summary much richer than the one produced by this solution, and we developed a new approach of information extraction instead of merely using available tools. Toloudis *et al.* [66] employed Text Mining technique on vulnerabilities' descriptions for automatically assessing their severity. In a similar effort, Khazaei *et al.* [27] showed the potential of text mining for predicting the score of CVSS (Common Vulnerability Scoring System, i.e., a quantitative metric which returns a value as severity of a vulnerability). Although they mine CVE's description as well as CVErizer, their aim is not to extract a summary of the CVE. Additionally, they determine such score by classifying the CVEs with machine learning rather than extract relevant information from the decriptions.

Differently from these previous works, we propose an approach for extracting and summarizing useful information from CVE descriptions, to support security engineers and developers in making decisions. To the best of authors' knowledge, no prior work has proposed highly-extensible approaches for summarizing this kind of artifacts and assessed the quality of the produced summaries through an end-to-end validation involving external subjects.

8. Conclusions

The presence of vulnerabilities in software systems and protocols is the main cause of security attacks. Vulnerabilities may be directly used by attackers to violate integrity, availability and confidentiality policies of targeted organizations. Common Vulnerabilities and Exposures (CVE) Database is a list of common names for publicly known vulnerabilities, which is used to provide information about patched vulnerabilities to security experts. However, the descriptions of these vulnerabilities are mostly written in natural language. Since for a specific version of a specific software might exist tens of vulnerabilities, it is challenging and time-consuming for a security analyst to manually check all these vulnerabilities. To cope with this issue, in this paper we presented CVErizer a tool able to (i) automatically classify software vulnerabilities with high accuracy (*i.e.*, 81%), and (ii) assist vulnerability assessors in quickly understanding software vulnerabilities, by providing concise and reasonably accurate summaries of vulnerabilities' descriptions. We evaluated the classification capabilities of our tool (RQ1), as well as the usefulness and correctness of summaries (RQ2) by conducting a controlled experiments involving 15 MSc cyber-security students. Summaries generated by CVErizer substantially reduce the time required for manually analyzing vulnerabilities' descriptions, and are considered highly useful during vulnerability assessment processes. High usefulness of CVErizer's summaries has been confirmed by 4 professional security experts who found machine-generated summaries very similar to the manually-generated ones. Moreover, we presented the results of an industrial case study aimed at evaluating CVErizer when mining relevant information from vulnerabilities related to two competing hypervisor products.

As future work, we are investigating the use of topic modeling techniques (*e.g.*, LDA) to improve the performance and cope with the limitations that affect our approach. We are also exploring the possibility to associate the summary extracted by the CVEs to the code of vulnerability exploitation which is provided attached to the CVE description. This could allow to associate attack strings of code to the type of attack, and improve Information Security appliances, like firewall and SIEM. Moreover, we plan to enrich CVErizer summaries with remediation information, to provide, at the same time, an overview of the vulnerability and the exact steps to prevent from it.

Acknowledgments

We gratefully thank Cybaze S.p.A. and its employees who supported us during the definition and validation steps of our approach. We also thank all the students and practitioners involved in our experiments.

References

[1] MITRE Corporation, Terminology, `https://cve.mitre.org/about/terminology.html`, [Online; accessed 23-November-2016].

[2] W. A. Arbaugh, W. L. Fithen, J. McHugh, Windows of vulnerability: a case study analysis, Computer 33 (12) (2000) 52–59. `doi:10.1109/2.889093`.

[3] C. Joshi, U. K. Singh, K. Tarey, A review on taxonomies of attacks and vulnerability in computer and network system, International Journal of Advanced Research in Computer Science and Software Engineering 5 (1) (2015) 742–747.

[4] R. G. Kula, D. M. German, A. Ouni, T. Ishio, K. Inoue, Do developers update their library dependencies?, Empirical Softw. Engg. 23 (1) (2018) 384–417. `doi:10.1007/s10664-017-9521-5`.
URL `https://doi.org/10.1007/s10664-017-9521-5`

[5] Y. Lai, P. Hsia, Using the vulnerability information of computer systems to improve the network security, Computer Communications 30 (9) (2007) 2032–2047. `doi:10.1016/j.comcom.2007.03.007`.

[6] M. Guo, J. A. Wang, An ontology-based approach to model common vulnerabilities and exposures in information security, in: ASEE Southest Section Conference, 2009.

[7] S. Beattie, S. Arnold, C. Cowan, P. Wagle, C. Wright, A. Shostack, Timing the application of security patches for optimal uptime, in: Proceedings of the 16th USENIX Conference on System Administration, LISA '02, USENIX Association, Berkeley, CA, USA, 2002, pp. 233–242.
URL `http://dl.acm.org/citation.cfm?id=1050517.1050546`

[8] V. Smyth, Software vulnerability management: how intelligence helps reduce the risk, Network Security 2017 (3) (2017) 10 – 12. `doi:https://doi.org/10.1016/S1353-4858(17)30027-2`.
URL `http://www.sciencedirect.com/science/article/pii/S1353485817300272`

[9] B. Arief, M. A. B. Adzmi, T. Gross, Understanding cybercrime from its stakeholders' perspectives: Part 1–attackers, IEEE Security Privacy 13 (1) (2015) 71–76. `doi:10.1109/MSP.2015.19`.

[10] Q. Liu, Y. Zhang, Y. Kong, Q. Wu, Improving vrss-based vulnerability prioritization using analytic hierarchy process, J. Syst. Softw. 85 (8) (2012) 1699–1708. `doi:10.1016/j.jss.2012.03.057`.
URL `http://dx.doi.org/10.1016/j.jss.2012.03.057`

[11] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, C. Zhai, Have things changed now?: An empirical study of bug characteristics in modern open source software, in: Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, ASID '06, ACM, New York, NY, USA, 2006, pp. 25–33. `doi:10.1145/1181309.1181314`.
URL `http://doi.acm.org/10.1145/1181309.1181314`

[12] J. A. Wang, H. Wang, M. Guo, L. Zhou, J. Camargo, Ranking attacks based on vulnerability analysis, in: System Sciences (HICSS), 2010 43rd Hawaii International Conference on, 2010, pp. 1–10. `doi:10.1109/HICSS.2010.313`.

[13] Z. Chen, Y. Zhang, Z. Chen, A categorization framework for common computer vulnerabilities and exposures, Comput. J. 53 (5) (2010) 551–580. `doi:10.1093/comjnl/bxp040`.
URL `http://dx.doi.org/10.1093/comjnl/bxp040`

[14] J. A. Wang, M. Guo, H. Wang, M. Xia, L. Zhou, Ontology-based security assessment for software products, in: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies, CSIIRW '09, ACM, New York, NY, USA, 2009, pp. 15:1–15:4. `doi:10.1145/1558607.1558625`.
URL `http://doi.acm.org/10.1145/1558607.1558625`

[15] T. Wen, Y. Zhang, Q. Wu, G. Yang, ASVC: an automatic security vulnerability categorization framework based on novel features of vulnerability data, JCM 10 (2) (2015) 107–116. `doi:10.12720/jcm.10.2.107-116`.
URL `http://dx.doi.org/10.12720/jcm.10.2.107-116`

[16] L. Allodi, F. Massacci, Comparing vulnerability severity and exploits using case-control studies, ACM Trans. Inf. Syst. Secur. 17 (1) (2014) 1:1–1:20. `doi:10.1145/2630069`.

[17] A. Khazaei, M. Ghasemzadeh, V. Derhami, An automatic method for CVSS score prediction using vulnerabilities description, Journal of Intelligent and Fuzzy Systems 30 (1) (2016) 89–96. `doi:10.3233/IFS-151733`.

[18] A. Joshi, R. Lal, T. Finin, A. Joshi, Extracting cybersecurity related linked data from text, in: Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on, IEEE, 2013, pp. 252–259.

[19] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, H. C. Gall, Development emails content analyzer: Intention mining in developer discussions (t), in: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), ASE '15, IEEE Computer Society, Washington, DC, USA, 2015, pp. 12–23. `doi:10.1109/ASE.2015.12`.
URL `http://dx.doi.org/10.1109/ASE.2015.12`

[20] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, H. Gall, Deca: Development emails content analyzer, in: Proceedings of the 38th International Conference on Software Engineering Companion, ICSE '16, ACM, New York, NY, USA, 2016, pp. 641–644. `doi:10.1145/2889160.2889170`.
URL `http://doi.acm.org/10.1145/2889160.2889170`

[21] M.-C. De Marneffe, B. MacCartney, C. D. Manning, et al., Generating typed dependency parses from phrase structure parses, in: Proceedings of LREC, Vol. 6, Genoa, 2006, pp. 449–454.

[22] G. Canfora, A. Di Sorbo, E. Emanuele, S. Forootani, C. A. Visaggio, A nlp-based solution to prevent from privacy leaks in social network posts, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018, 2018, pp. 36:1–36:6. `doi:10.1145/3230833.3230845`.

[23] Y. Wu, I. Bojanova, Y. Yesha, They know your weaknesses–do you?: Reintroducing common weakness enumeration, CrossTalk (2015) 45.

[24] D. Spencer, Card sorting: Designing usable categories, Rosenfeld Media, 2009.

[25] A. Gkortzis, S. Rizou, D. Spinellis, An empirical analysis of vulnerabilities in virtualization technologies, in: 2016 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2016,

Luxembourg, December 12-15, 2016, 2016, pp. 533–538. doi:10.1109/CloudCom.2016.0093.

[26] Y. Zhou, Y. Tong, R. Gu, H. Gall, Combining text mining and data mining for bug report classification, in: Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution, ICSME '14, IEEE Computer Society, Washington, DC, USA, 2014, pp. 311–320. doi:10.1109/ICSME.2014.53.
URL http://dx.doi.org/10.1109/ICSME.2014.53

[27] A. Khazaei, M. Ghasemzadeh, V. Derhami, An automatic method for CVSS score prediction using vulnerabilities description, Journal of Intelligent and Fuzzy Systems 30 (1) (2016) 89–96. doi:10.3233/IFS-151733.
URL http://dx.doi.org/10.3233/IFS-151733

[28] I. H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[29] L. Villarroel, G. Bavota, B. Russo, R. Oliveto, M. Di Penta, Release planning of mobile apps based on user reviews, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, ACM, New York, NY, USA, 2016, pp. 14–24. doi:10.1145/2884781.2884818.
URL http://doi.acm.org/10.1145/2884781.2884818

[30] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, H. C. Gall, How can i improve my app? classifying user reviews for software maintenance and evolution, in: Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), ICSME '15, IEEE Computer Society, Washington, DC, USA, 2015, pp. 281–290. doi:10.1109/ICSM.2015.7332474.
URL http://dx.doi.org/10.1109/ICSM.2015.7332474

[31] E. Guzman, M. El-Haliby, B. Bruegge, Ensemble methods for app review classification: An approach for software evolution (n), in: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, pp. 771–776. doi:10.1109/ASE.2015.88.

[32] F. Sebastiani, Machine learning in automated text categorization, ACM Comput. Surv. 34 (1) (2002) 1–47. doi:10.1145/505282.505283.
URL http://doi.acm.org/10.1145/505282.505283

[33] R. A. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[34] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, K. Vijay-Shanker, Automatic generation of natural language summaries for java classes, in: 2013 21st International Conference on Program Comprehension (ICPC), 2013, pp. 23–32. doi:10.1109/ICPC.2013.6613830.

[35] S. Panichella, A. Panichella, M. Beller, A. Zaidman, H. C. Gall, The impact of test case summaries on bug fixing performance: An empirical investigation, in: Proceedings of the 38th International Conference on Software Engineering, ICSE '16, ACM, New York, NY, USA, 2016, pp. 547–558. doi:10.1145/2884781.2884847.
URL http://doi.acm.org/10.1145/2884781.2884847

[36] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, H. C. Gall, What would users change in my app? summarizing app reviews for recommending software changes, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, ACM, New York, NY, USA, 2016, pp. 499–510. doi:10.1145/2950290.2950299.
URL http://doi.acm.org/10.1145/2950290.2950299

[37] E. A. Feigenbaum, Some challenges and grand challenges for computational intelligence, J. ACM 50 (1) (2003) 32–40. doi:10.1145/602382.602400.
URL http://doi.acm.org/10.1145/602382.602400

[38] Q. Wang, T. Luo, D. Wang, Can machine generate traditional chinese poetry? a feigenbaum test, in: C.-L. Liu, A. Hussain, B. Luo, K. C. Tan, Y. Zeng, Z. Zhang (Eds.), Advances in Brain Inspired Cognitive Systems, Springer International Publishing, Cham, 2016, pp. 34–46.

[39] J. L. Fleiss, Measuring nominal scale agreement among many raters., Psychological bulletin 76 (5) (1971) 378.

[40] H. B. Mann, D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, Ann. Math. Statist. 18 (1) (1947) 50–60. doi:10.1214/aoms/1177730491.

[41] R. J. Grissom, J. J. Kim, Effect sizes for research: A broad practical approach, 2nd Edition, Lawrence Earlbaum Associates, 2005.

[42] A. Bulazel, B. Yener, A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web, in: Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium, ROOTS, ACM, New York, NY, USA, 2017, pp. 2:1–2:21. doi:10.1145/3150376.3150378.

[43] A. Thongthua, S. Ngamsuriyaroj, Assessment of hypervisor vulnerabilities, in: 2016 International Conference on Cloud Computing Research and Innovations (ICCCRI), 2016, pp. 71–77. doi:10.1109/ICCCRI.2016.19.

[44] D. G. Feitelson, Using students as experimental subjects in software engineering research - A review and discussion of the evidence, CoRR abs/1512.08409.
URL http://arxiv.org/abs/1512.08409

[45] D. R. Radev, E. H. Hovy, K. McKeown, Introduction to the special issue on summarization, Computational Linguistics 28 (2002) 399–408.

[46] K. Filippova, Y. Altun, Overcoming the lack of parallel data in sentence compression., in: EMNLP, ACL, 2013, pp. 1481–1491.
URL http://dblp.uni-trier.de/db/conf/emnlp/emnlp2013.html#FilippovaA13

[47] J. L. Neto, A. A. Freitas, C. A. A. Kaestner, Automatic text summarization using a machine learning approach, in: G. Bittencourt, G. L. Ramalho (Eds.), Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 205–215.

[48] N. Nazar, Y. Hu, H. Jiang, Summarizing software artifacts: A literature review, J. Comput. Sci. Technol. 31 (5) (2016) 883–909. doi:10.1007/s11390-016-1671-1.

[49] E. Lloret, M. Palomar, Text summarisation in progress: a literature review, Artif. Intell. Rev. 37 (1) (2012) 1–41. doi:10.1007/s10462-011-9216-z.

[50] S. Rastkar, G. C. Murphy, G. Murray, Summarizing software artifacts: a case study of bug reports, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010, 2010, pp. 505–514. doi:10.1145/1806799.1806872.

[51] G. Murray, G. Carenini, Summarizing spoken and written conversations, in: 2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL, 2008, pp. 773–782.

[52] R. Lotufo, Z. Malik, K. Czarnecki, Modelling the 'hurried' bug report reading process to summarize bug reports, Empirical Software Engineering 20 (2) (2015) 516–548. doi:10.1007/s10664-014-9311-2.

[53] H. Jiang, J. Zhang, H. Ma, N. Nazar, Z. Ren, Mining authorship characteristics in bug repositories, SCIENCE CHINA Information Sciences 60 (1) (2017) 12107. doi:10.1007/s11432-014-0372-y.

[54] S. Yeasmin, C. K. Roy, K. A. Schneider, Interactive visualization of bug reports using topic evolution and extractive summaries, in: 30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014, 2014, pp. 421–425. doi:10.1109/ICSME.2014.66.

[55] S. Mani, R. Catherine, V. S. Sinha, A. Dubey, AUSUM: approach for unsupervised bug report summarization, in: 20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012, 2012, p. 11. doi:10.1145/2393596.2393607.

[56] S. Rastkar, G. C. Murphy, Why did this code change?, in: 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013, 2013, pp. 1193–1196. doi:10.1109/ICSE.2013.6606676.

[57] L. F. Cortes-Coy, M. L. Vásquez, J. Aponte, D. Poshyvanyk, On automatically generating commit messages via summarization of source code changes, in: 14th IEEE International Working Conference on Source Code Analysis and Manipulation, SCAM 2014, Victoria, BC, Canada, September 28-29, 2014, 2014, pp. 275–284. doi:10.1109/SCAM.2014.14.

[58] S. Scalabrino, G. Bavota, B. Russo, M. D. Penta, R. Oliveto, Listening to the crowd for the release planning of mobile apps, IEEE Trans. Software Eng. 45 (1) (2019) 68–86. doi:10.1109/TSE.2017.2759112.

[59] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. C. Gall, F. Ferrucci, A. D. Lucia, Recommending and localizing change requests for mobile apps based on user reviews, in: Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017, 2017, pp. 106–117. doi:10.1109/ICSE.2017.18.

[60] J. Aponte, A. Marcus, Improving traceability link recovery methods through software artifact summarization, in: TEFSE'11, Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering, May 23, 2011, Waikiki, Honolulu, HI, USA, 2011, pp. 46–49. doi:10.1145/1987856.1987867.

[61] M. Bozorgi, L. K. Saul, S. Savage, G. M. Voelker, 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, ACM, New York, NY, USA, 2010, pp. 105–114. doi:10.1145/1835804.1835821.
URL http://doi.acm.org/10.1145/1835804.1835821

[62] H. T. Le, P. K. K. Loh, Using natural language tool to assist vprg automated extraction from textual vulnerability description, in: 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications, 2011, pp. 586–592. doi:10.1109/WAINA.2011.56.

[63] C. B. Simmons, S. Shiva, V. Phan, V. Shandilya, L. Simmons, Irs: An issue resolution system for cyber attack classification and management, in: Proceedings of the International Conference on Security and Management (SAM), The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012, p. 1.

[64] S. A. Mokhov, J. Paquet, M. Debbabi, The Use of NLP Techniques in Static Code Analysis to Detect Weaknesses and Vulnerabilities, Springer International Publishing, Cham, 2014, pp. 326–332. doi:10.1007/978-3-319-06483-3_33.
URL http://dx.doi.org/10.1007/978-3-319-06483-3_33

[65] L. Glanz, S. Schmidt, S. Wollny, B. Hermann, A vulnerability's lifetime: Enhancing version information in cve databases, in: Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business, i-KNOW '15, ACM, New York, NY, USA, 2015, pp. 28:1–28:4. doi:10.1145/2809563.2809612.
URL http://doi.acm.org/10.1145/2809563.2809612

[66] D. Toloudis, G. Spanos, L. Angelis, Associating the Severity of Vulnerabilities with their Description, Springer International Publishing, Cham, 2016, pp. 231–242. doi:10.1007/978-3-319-39564-7_22.
URL http://dx.doi.org/10.1007/978-3-319-39564-7_22

[67] R. Scandariato, J. Walden, A. Hovsepyan, W. Joosen, Predicting vulnerable software components via text mining, IEEE Transactions on Software Engineering 40 (10) (2014) 993–1006.