TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 09/12 pp270–280
DOI: 10.26599/TST.2019.9010003
Volume 25, Number 2, April 2020

Software Vulnerabilities Overview: A Descriptive Study

Mario Calín Sánchez, Juan Manuel Carrillo de Gea*, José Luis Fernández-Alemán, Jesús Garcerán, and Ambrosio Toval

Abstract: Computer security is a matter of great interest. In the last decade there have been numerous cases of cybercrime based on the exploitation of software vulnerabilities. This fact has generated a great social concern and a greater importance of computer security as a discipline. In this work, the most important vulnerabilities of recent years are identified, classified, and categorized individually. A measure of the impact of each vulnerability is used to carry out this classification, considering the number of products affected by each vulnerability, as well as its severity. In addition, the categories of vulnerabilities that have the greatest presence are identified. Based on the results obtained in this study, we can understand the consequences of the most common vulnerabilities, which software products are affected, how to counteract these vulnerabilities, and what their current trend is.

Key words: descriptive study; software security; software vulnerabilities; vulnerability databases

1 Introduction

Computers in general, and the Internet in particular, have a great social importance nowadays; the network of networks allows us to live interconnected in a relatively easy way. Currently, 50% of the world's population uses the Internet, that is, more than 3.77 billion people are connected online[1]. From the point of view of companies, this technological globalization means both advantages and disadvantages. There has been a longstanding consensus on the idea that being connected to the Internet has its risks[2]; when the company's data is no longer confined under full control, the likelihood of a third party being able to violate that data increases[3]. Despite the security measures currently used in any Internet service, information systems are frequently exposed to different threats and potential damages[4,5].

A *software vulnerability* is a flaw in the system that allows an attacker to breach the security measures implemented[6]. Errors and bugs are not new in the software world; a large and complex software system could contain a large number of bugs[7], and security bugs can sometimes be exploited by malicious users to produce damage or obtain benefits. Security problems due to software vulnerabilities can become particularly worrisome in the case of compromising information of a private nature, such as, for example, information related to a patient's health history[8–10].

In order to fight against software vulnerabilities, Internet databases have been created that record all these vulnerabilities to inform companies and programmers. A *vulnerability database* is a platform that stores, maintains, and disseminates information about vulnerabilities discovered in real computer systems[11]. These databases allow for security measurement and vulnerability management. In addition, these data can be listed and each vulnerability can be saved with a unique identifier, which in turn facilitates the sharing of information on those vulnerabilities.

Currently, one of the most extensive vulnerability databases is the National Vulnerability Database (NVD) from the U.S. government (https://nvd.nist.gov/).

• Mario Calín-Sánchez, Juan Manuel Carrillo de Gea, José Luis Fernández-Alemán, Jesús Garcerán, and Ambrosio Toval are with the Department of Informatics and Systems, Faculty of Computer Science, University of Murcia, Murcia 30100, Spain. E-mail: {mario.calin, jmcdg1, aleman, jesus.garceran, atoval}@um.es.

∗ To whom correspondence should be addressed.

Manuscript received: 2018-12-12; revised: 2019-02-23; accepted: 2019-03-11

This governmental repository stores the vulnerability management data. The body in charge of this database is the National Institute of Standards and Technology (NIST). This database provides the data according to the Security Content Automation Protocol (SCAP) specifications. SCAP is a set of NIST specifications for expressing and manipulating information related to failures and configurations in a standardized way[12]. SCAP has a number of components on which NVD relies, as in the case of the Common Vulnerabilities and Exposures (CVE) vulnerability dictionary. The CVE has related Common Vulnerability Scoring System (CVSS) scores that indicate the severity of a vulnerability[13].

CVE is the vulnerability dictionary that contains all the vulnerabilities with their respective identification (https://cve.mitre.org/about/). All these vulnerabilities are grouped into categories. Common Weaknesses Enumeration (CWE) offers that categorization and the required functionality to provide the security industry with a list of types of weaknesses. The agency responsible for the management of both CVE and CWE is MITRE Corporation, a non-profit company that offers information technology support to the United States Government.

This article is structured as follows: Section 2 introduces different studies related to the identification and categorization of software vulnerabilities. Section 3 explains in detail the methodology followed to carry out this work about software vulnerabilities, and presents the tool that was developed to gather the data to perform the study. Section 5 shows the results and information obtained after analyzing the data. Section 6 discusses the results of this work. Finally, Section 7 presents our conclusion and future work.

2 Related Work

Li et al.[14] classified vulnerabilities according to the complexity of their identification, repair, and exploitation. The vulnerabilities are thus divided into those that are (1) easy to identify and exploit (Bohr-Vulnerability, BOV); (2) complex to identify and exploit (Non-aging-related Mandel Vulnerability, NMV); (3) exploited by attackers to degrade performance (Aging-Related Vulnerability, ARV); and (4) not classified in any of the other three categories (Unknown Vulnerability, UNK). Once the study was done, the results indicated that the most common type of vulnerabilities is NMV. In addition, the study concludes that the time to repair a vulnerability according to its type is 66.8 days in BOV, 70.5 in NMV, and 60.6 in ARV.

Venter et al.[15] focused on performing a categorization of vulnerabilities following this process: (1) acquiring data sources (for example, the CVE list); (2) data preprocessing to add important information and eliminate vulnerabilities that are not of interest; (3) making a data storage using a Self-Organized Map (SOM); and (4) inspecting and labeling the clusters in order to categorize the CVE database. In this work, vulnerabilities are divided into seven categories: buffer overflow, Denial of Service (DoS), scripting metacharacters, privilege escalation, data corruption, information gathering, and configuration vulnerabilities. The results obtained show that out of a total of 167 vulnerabilities, the most common type was DoS (61) and the least common was the buffer overflow (4).

Alhazmi et al.[16] carried out measurements of the security of the systems according to their density of vulnerability. This metric is the number of vulnerabilities multiplied by code size, and it helps, for example, a provider to know when a product should be placed on the market. This work studies this metric in the five most used operating systems in 2005. In Windows XP, it showed a lower density compared to Windows 95 and Windows 98 because it was more recent and not so many vulnerabilities were discovered so far. This article concludes by stating the benefits of this metric and the possibility to expand it with additional data.

Alqahtani et al.[17] used a research methodology focused on a unified ontological representation that processes vulnerabilities and project information. The objective is to establish a bidirectional relationship between the vulnerability databases and traditional software repositories. This study focuses on high-level security vulnerabilities, creating its own ontology to relate products with vulnerabilities, and vulnerabilities with a series of properties such as date, weakness, author, source, etc. The product has a series of properties such as library, application, and operating system. With this ontological representation, the authors[17] expect to make their analysis accessible to developers.

Cruz et al.[18] established a categorization of existing vulnerabilities in the context of cloud computing. The

objective of this work is to study existing security research on cloud computing to analyze the state of the art and identify future directions in this field. Finally, security companies also release periodic reports on computer security threats and general information about the latest news in the security world. For example, McAfee[19] highlighted at the end of 2016 the increase of threats in Mac OS and mobile devices, the increase of malware based on macros, or the decrease of phishing URLs within the web.

In addition to these research articles, there are different classifications available on the Internet. The CVE Details database makes a series of interesting classifications on its website. For example, the top 50 products by total number of distinct vulnerabilities are reported (https://www.cvedetails.com/top-50-products.php). The same top 50 ranking is also presented but concerning vendors instead of products (https:// www.cvedetails.com/top-50-vendors.php). In other databases, such as NVD, the percentage of vulnerabilities of each type found in their database is shown in a chart (https://nvd.nist.gov/vuln/visualizations/cwe-over-time). In addition, they offered a study of how the frequency of the type of vulnerabilities has changed with time, based on the CWE vulnerability classification.

3 Method

The method proposed in this work offers an approach for the study of software vulnerabilities. It is focused on the number of software products affected by each vulnerability in the NVD database (the *presence* metric), and the measure of *severity* of those vulnerabilities. Both parameters are related by means of a formula with which a metric called *impact* is calculated. Each year is investigated individually to later unify the results and conduct a general study.

3.1 Data collection

We relied on the NVD database website to obtain data, which provides a series of XML files that contain information about the emergence of software vulnerabilities following the SCAP specifications. Vulnerabilities from the year 2015 to 2017 were studied in order to analyze the current situation on this regard.

The information used to carry out this study is as follows:

• **entry id** (Identifier). It is a unique vulnerability identifier per file.

• **cvss:score** (Severity). Each vulnerability has a degree of severity from 0 to 10 that is given according to the CVSS metric. Within this metric, the classification of a vulnerability is *mild* if the value is between 0.0–3.9, *medium* if the value is between 4.0–6.9, *high* if the value is between 7.0–8.9, and *critical* if the value is between 9.0–10.0[13].

• **vuln:cwe** (Category). Categories are used to group the vulnerabilities. These categories are defined according to the SCAP component called CWE and there are about 1000 different categories of vulnerability types.

3.2 Obtaining the annual impact

We initially address each year separately. For each year, a process to obtain the impact of a vulnerability is carried out. In the context of this study, the impact is defined as a relation between the number of software products that are affected by a vulnerability in a specific year, and the severity that is assigned to that vulnerability. For example, if two vulnerabilities affect 100 different systems and one has a severity of 5 and another one has a severity of 7, the vulnerability with a severity of 7 will have more impact than the other. The process described below is followed to calculate the impact:

• The software products that are affected by each of the vulnerabilities are identified, and their severity is obtained.

• The percentage of software products that are affected by a vulnerability with respect to the total number of products of that year is obtained.

$$\text{Presence}_{\text{vul}} = \frac{\text{Products}_{\text{vul}}}{\text{TotalProducts}}.$$

Once the frequency of appearance is obtained (i.e., the *presence*), the *severity* of the vulnerability is also considered to compute the *impact* of the vulnerability. In this way, a value is obtained that allows us to relate presence and severity giving a similar importance to both values.

$$\text{Impact}_{\text{vul}} = \text{Presence}_{\text{vul}} \times \text{Severity}_{\text{vul}}.$$

3.3 Obtaining the annual category

The vulnerabilities are split up into categories once the presence of each vulnerability is obtained. Despite the large amount of information available in NVD, there are many vulnerabilities that do not have a category within this data source. Part of the desired information can thus be missing. Even so, all existing categories are studied and the most repeated ones are obtained.

4 Data Collection Tool

A tool was developed using the Java programming language to work with the XML files, allowing for the extraction and presentation of data in the CSV format, which is compatible with spreadsheet solutions such as Libreoffice Calc and Microsoft Excel. The developed tool is also a contribution of this work, and it is open source, so it is possible to download, use, and modify it through a repository (https://github.com/mariocalin/nistAnalysis).

This tool is developer-oriented; neither graphical user interface nor command-line program is therefore provided. In order to use the tool, the repository must be downloaded or cloned, and imported into your preferred Java Integrated Development Environment (IDE). Then, the tool can be run, and its parameters can be changed as desired.

4.1 Source code description

The complete Java documentation of the tool can be found in the *doc* folder inside the mentioned repository. In this section, however, the most important aspects of the tool in terms of development are described.

Firstly, the package *src/nist/functions* is presented. There are three important files in this package:

- *INistDataAnalysis.java*: Interface that defines the operations a NIST data analyzer must have. This interface is created to provide parsers to different file formats (e.g., XML or JSON).
- *INistDataResult.java*: Interface that defines the elements a *NistDataResult* must have. Every *NistDataAnalyzer* must create *NistDataResults*.
- *XMLNistParser.java*: It implements *INistData Analysis*. This parser processes the XML Data Feed. It will parse the XML into the tool model.

Secondly, the package *src/nist/model* includes the three main concepts that can be found in the model:

- **Entry** represents a vulnerability entry.
- **Category** represents a CWE vulnerability category.
- **Result** represents a summary of a year in terms of vulnerabilities. It offers two different types of *Result*: (1) per categories, and (2) per entries.

Thirdly, in the package *src/nist/utils*, there is only one class. It contains some utilities in terms of readability.

Finally, the *Main.java* file is the entry point for the tool and contains the main function to be run with the IDE.

4.2 Usage of the tool

A usage example of the data collection tool is shown in Fig. 1.

A typical execution flow of the tool is illustrated in Fig. 2. As shown in the diagram, an XML file is injected into the *XMLNistParser*, which processes the file and creates a *Result* that includes yearly data, either focused on categories of vulnerabilities or vulnerabilities.

4.3 XML data feeds

In order to run the tool and get results, it is mandatory to download the data feeds that NIST provides in their official website (https://nvd.nist.gov/vuln/data-feeds), and place them into the corresponding folder with the same exact name that is initially defined. At the moment, only XML feeds are allowed and the folder is *XML Data*.

4.3.1 XML data feed structure

Each XML file contains multiple elements of the type *entry*. This element represents a vulnerability registered in the database. There are some elements appended as children of each *entry*; among other info elements (e.g., *vuln:cve-id* or *vuln:published-datetime*), the most interesting elements are as follows.

- **vuln:vulnerable-configuration**. This element refers to the configuration in which the vulnerability

```
public static void main(String[] args) throws Exception {
  // Creates an analyzer instance with the year to analyze
  INistDataAnalysis analyzer = new
       XMLNistParser(XMLNistParser.XMLFiles.FULL_YEAR_2017);

  // Creates a result
  Result result = analyzer.createResult();

  // Prints the entries result to a CSV file or String
  result.entriesResult().toCSV("entries-2017.csv", true);
  result.entriesResult().toString();

  // Prints the categories result to a CSV file or String
  result.categoriesResult().toCSV("categories-2017.csv",
       true);
  result.categoriesResult().toString();

  System.out.println("END OF PROGRAM");
}
```

Fig. 1 A usage example of the data collection tool.

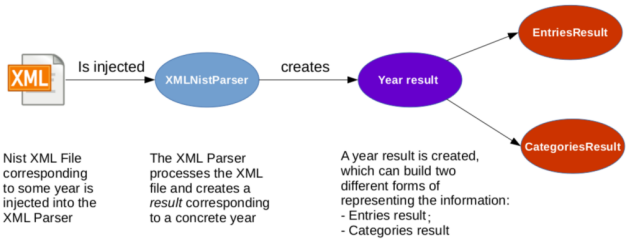


Fig. 2 Data collection tool diagram.

was found. NIST provides a list of configurations where they test the vulnerabilities.

• **vuln:vulnerable-software-list**. It contains the specific software products that are affected by the vulnerability.

• **vuln:cvss**. It includes the CVSS metrics of the vulnerability. Among them, it is of special interest the score given by the *cvss:score* element.

• **vuln:cwe**. This element refers to the category of the vulnerability in the CVE dictionary.

4.3.2 Parsing mechanism

The parsing mechanism implemented to process the XML files is described below:

(1) The *XMLNistParser* receives an XML file path that corresponds to the NIST XML year data feed. It loads the file and, by using the Java libraries of *javax.xml.parsers* and *org.w3c.dom*, it creates a *Result* object containing the information needed in a object oriented way.

(2) With the *Result* object, it can be chosen whether to print the entries results or the categories results (via CSV or console). The only difference between them is the way of representing the information:

• *EntryResult* is focused on vulnerability entries; it shows information about the entry code, the entry CVSS score, and the products affected by the vulnerability.

• *CategoryResult* is focused on CWE categories; it shows information about the total number of vulnerability entries that a category has, and its average vulnerability CVSS score.

5 Results

The next step is to present the results. As it has been defined in the method, the results are extracted on an annual basis (see Section 5.1) and after that, the global results are shown (see Section 5.2).

Some vulnerabilities have different identifiers but they can be analyzed together for the sake of simplicity. Since both their causes and their consequences are similar and have little differences between them, we have grouped these vulnerabilities into one.

5.1 Annual analysis

Firstly, Fig. 3a shows the results regarding the impact of vulnerabilities in 2015. The vulnerability with the greatest impact (CVE-2015-1290) is buffer overflow in the Google Chrome browser versions ranging from 0.1 through 43, which allow attackers to obtain operating system privileges.

The second, third, and fourth vulnerabilities with the greatest impact this year (CVE-2015-0569/70/71) are also buffer overflow, but this time in the Linux kernel, in versions 3.x and 4.x. These vulnerabilities allow attackers to obtain privileges through an application ran on the mentioned platform.

The fifth (CVE-2015-0573) vulnerability also affects the Linux 3.x kernel through a driver that is used in the Qualcomm Innovation Center (QuIC), allowing guest users to obtain operating system privileges or produce a DoS.

Figure 3b shows a breakdown by categories of the vulnerabilities with greater presence detected in 2015. Among them, there is a category (CWE-119) that refers to *Improper Restriction of Operations within the Bounds of a Memory Buffer* that has been registered 1073 times, followed by CWE-79 (i.e., *Improper Neutralization of Input During Web Page Generation* ("*Cross-site Scripting*")) with 773 records. The third most important category is CWE-200, which refers to *Information Exposure*, with a record of 690 times.

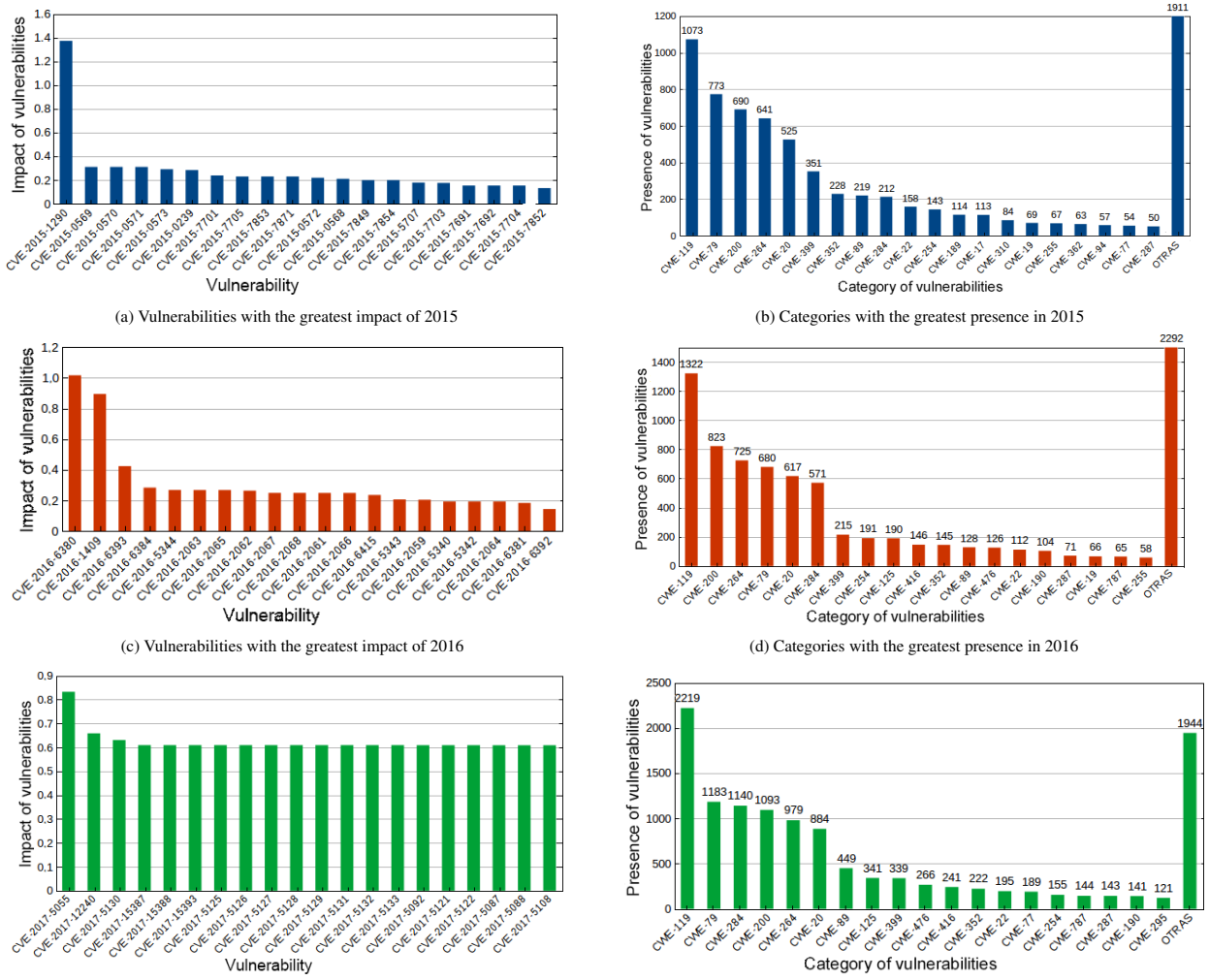
It should be noted that there are 1911 vulnerabilities not associated with a specific category whose average severity is 6.68, which represent 24% of the total number of vulnerabilities registered in 2015.

Secondly, the results of the year 2016 are presented in Fig. 3c. As shown in the chart, the vulnerability with the greatest impact (CVE-2016-6380), which refers to obtaining sensitive information or causing DoS, occurs in the DNS forwarder in Cisco IOS 12.0 through 12.4 and 15.0 through 15.6 and IOS XE 3.1 through 3.15.

The vulnerability with the second highest impact is CVE-2016-1409, referring to DoS. It affects the Neighbor Discovery (ND) protocol implementation in the IPv6 stack in Cisco IOS XE 2.1 through 3.17S, IOS XR 2.0.0 through 5.3.2. The occurrence of this vulnerability reaches 9795 cases, but a severity score of just 5 (i.e., *medium*) is assigned.

The third vulnerability with the greatest impact is CVE-2016-6393, referring to inadequate management of system resources. It mainly affects CISCO DNS forwarders from 4.1 through 7.2.

With respect to the categories, as shown in Fig. 3d, the most repeated category of vulnerabilities is again CWE-119, with 1322 cases out of 9431 records. The next category with the highest representativeness, with 823 cases, is CWE-200. The third category in this ranking is CWE-264, which refers to *Permissions, Privileges, and Access Controls*, with a total of 725



(a) Vulnerabilities with the greatest impact of 2015

(b) Categories with the greatest presence in 2015

(c) Vulnerabilities with the greatest impact of 2016

(d) Categories with the greatest presence in 2016

(e) Vulnerabilities with the greatest impact of 2017

(f) Categories with the greatest presence in 2017

Fig. 3 Impact of vulnerabilities and presence of categories (annual breakdown).

records.

A significant percentage of vulnerabilities are not associated with a category, with a total of 2292 cases and an average severity of 6.3. This represents 24.3% of the total registered vulnerabilities.

Finally, the results of the year 2017 are shown in Fig. 3e. The vulnerability with the greatest impact is CVE-2017-5055, which refers to out-of-bounds reads, and allows the attacker to access information from unauthorized memory areas or cause a failure.

The second vulnerability with the greatest impact is CVE-2017-12240, referring to a buffer overflow condition in the DHCP relay subsystem of Cisco IOS 12.2 through 15.6 and Cisco IOS XE Software. It could allow an attacker to execute arbitrary code and gain full control of the system, and also perform a DoS attack.

The third vulnerability with the greatest impact is CVE-2017-5130, which refers to an integer overflow that may allow an attacker to potentially exploit heap corruption. It affects Google Chrome versions prior to 62.0.3202.62, and the *libxml2* library before version 2.9.5.

With respect to the vulnerability categories, which are shown in Fig. 3f, the most repeated category is again CWE-119 with 2219 cases out of 14 027. The next category with greatest presence is CWE-79. This category has 1183 cases, followed by the 1140 cases of the CWE-284 category, which refers to *Improper Access Control*.

The number of vulnerabilities not associated with specific categories is 1944, with an average severity of 5.42. This amount represents 13.8% of total vulnerabilities.

5.2 Interannual analysis

The total grouped results of this study are presented

below. Figure 4a shows the 20 vulnerabilities with the greatest impact of the three years under study, and Fig. 4b shows a breakdown of the categories with the largest number of vulnerabilities detected during that time.

Our results indicate that the vulnerability with the greatest impact in the period of time analyzed was CVE-2015-1290, referring to buffer overflow. The second and third vulnerabilities with the greatest impact occurred in 2016 (i.e., CVE-2016-6380 and CVE-2016-1409). The next vulnerability is the one with the greatest impact of the year 2017, CVE-2017-12240. Of the 20 vulnerabilities with the greatest general impact, 17 of them are from the year 2017, while there are only one of the year 2015 and two of the year 2016.

With respect to the categories, CWE-119 is the most repeated category throughout the different years. It includes 4614 vulnerabilities, which represents 14.68% of the total. Secondly, the CWE-79 category has 2636 vulnerabilities or 8.39%. The category CWE-200 has 2606 vulnerabilities, being thus the third with the largest global presence, 8.29%. Out of a total of 31 426 vulnerabilities, 6147 of them (or 19.5%) do not specify category.

6 Discussion

Once the global data of the three years were presented, some conclusions can be outlined about the most relevant vulnerabilities according to the metrics proposed in this study, its type and its category.

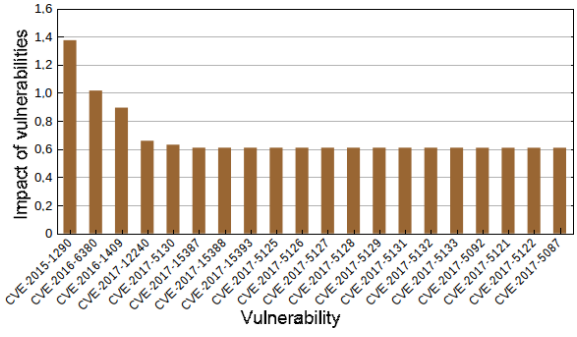
The vulnerabilities with the greatest impact identified in this work have as a consequence the DoS, and are of the utmost importance for software products. This type of vulnerability causes a service or resource to be inaccessible to legitimate users.

Another consequence of the vulnerabilities that are among those with the greatest impact is the escalation of privileges. Vulnerabilities of this nature are among the most serious that exist today; there are, however, fewer cases of this type of vulnerabilities. When privilege management is inadequate or fails, an attacker can compromise the security of the software by unauthorized appropriation of permissions (e.g., reading, modification) on files and directories that could contain sensitive information.

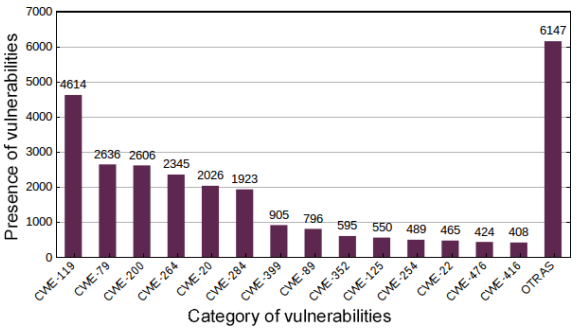
Inferential statistical analysis has been performed to formally check whether the probability that an observed difference between the impact of vulnerabilities in different years has happened by chance. The statistical software package IBM SPSS Statistics (https://www.ibm.com/products/spss-statistics) version 20 was used to carry out the data analysis.

The assumptions about the data that are entailed by statistical tests must be taken into consideration to apply the correct technique. In this regard, parametric tests require the variables coming from a normal distribution; when this requirement is not satisfied, a non-parametric test is recommended. In addition, the number of groups is a key factor to decide upon the technique. Typically, the one-way ANalysis Of VAriance (ANOVA) and the Kruskal-Wallis test (parametric and non-parametric techniques, respectively) are used to test for differences among at least three groups. Indeed, since our analysis encompasses three years, either the ANOVA or the Kruskal-Wallis test should to be used.

When applying the Kruskal-Wallis test to compare the medians of the impact of the vulnerabilities between the years 2015 ($M = 0.003\,147\,10$), 2016 ($M = 0.002\,572\,54$), and 2017 ($M = 0.004\,772\,00$), statistically significant differences were observed ($\chi^2(2) = 2689.536$, $p < 0.001$). In the post-hoc contrasts, it can be seen that in 2017 the impact of the vulnerabilities was greater than in 2016 ($p < 0.001$) and 2015 ($p < 0.001$). Statistically significant differences were also found between the year 2015 and 2016 ($p < 0.001$) that show a greater impact of the



(a) Vulnerabilities with the greatest impact



(b) Categories with the greatest presence

Fig. 4 Impact of vulnerabilities and presence of categories (combined).

vulnerabilities in 2015.

With respect to the categories, it is noteworthy that despite the fact that the CWE organism has defined 125 categories, the 14 most common categories represent 66.13% of the total number of vulnerabilities. The rest of the categories represent only 14.31% of the total vulnerabilities. Likewise, in the three years under study (i.e., 2015, 2016, and 2017), 19.56% of the vulnerabilities have not been associated with a category. Figure 5 shows all these details graphically.

Within these categories, buffer overflow (CWE-119) is by far the most common problem, with almost twice as many cases as the second largest in the ranking. A buffer overflow is a read or write to a memory location outside the buffer limit[20]. This category, which represents 14.68% of the vulnerabilities, provides an indication of the type of weaknesses the attackers are taking advantage of, as well as where the most significant problems are in terms of security of the main software products. The typical consequences of vulnerabilities in this category are usually running unauthorized code, modifying or reading memory, DoS, and consuming resources, among others (http://cwe.mitre.org/data/definitions/119.html).

The categories placed next in the list also have a considerable presence. CWE-79 corresponds to failure to neutralize user input that is used as a web page that is served to others; CWE-200 refers to disclosure of information to someone who does not have authorization to have access to the information; finally, CWE-264 is described as weaknesses related to access control (i.e., permissions, privileges, and other security features).

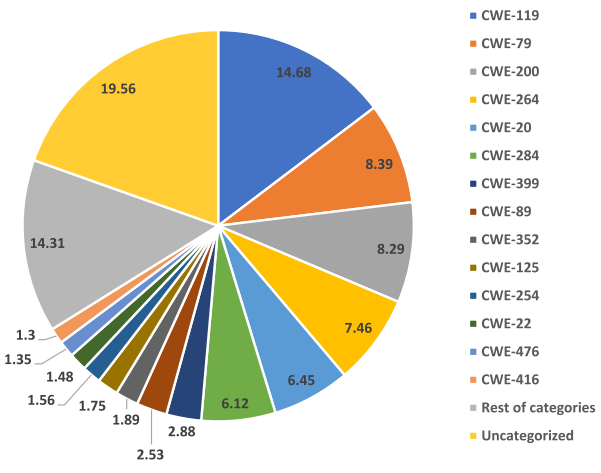


Fig. 5 Percentage of presence of each category.

The four categories mentioned above (i.e., CWE-119, CWE-79, CWE-200, and CWE-264) represent 38.82% of the total. This is an important detail, since it could have been initially thought that the amount of information about categories would be too extensive to be studied in this way. However, in the end the categories of vulnerabilities are constantly repeated. Therefore, attackers frequently use the same strategies against different software. In other words, a large number of weaknesses are common to different software products.

To provide more information about the categories of vulnerabilities that stand out throughout the three-year period, Table 1 shows a relationship between these categories and the affected programming languages, paradigms, technologies, and platforms.

The C, C++, and assembler languages are, despite their age, the most affected programming languages by the CWE-119 category, which has the highest number of registered vulnerabilities. In addition, another of the categories with the most vulnerabilities (CWE-200) makes explicit mention of an information exhibition in a mobile environment. This is in line with the drastic increase in recorded attacks to the security of mobile devices. According to Nokia[21], more than 100 million mobile devices were infected by malware in 2016, including smartphones, laptops, and a wide range of Internet of Things devices.

In summary, buffer overflow is currently the most common vulnerability category; on the other hand, the main consequence of vulnerabilities is DoS.

7 Conclusion and Further Work

This work can be of help to inform users, researchers, and security practitioners about the vulnerabilities with the greatest impact in recent years, and the software that is affected by them. This information can be useful, for example, to apply the corresponding security

Table 1 Environments affected by the main categories of vulnerabilities.

Category	Programming language	Paradigm	Technology	Platform
CWE-119	C, C++, and assembler	*Independent*	*Independent*	*Independent*
CWE-79	*Independent*	Web-based	Web technology	*Independent*
CWE-200	*Independent*	*Independent*	*Independent*	Mobile environment
CWE-264	*Independent*	*Independent*	*Independent*	*Independent*

patches[22]. We believe that it is crucial to have a grasp on the most relevant vulnerabilities nowadays to protect ourselves against them. In addition, the study can serve as a guide to foresee the evolution of vulnerabilities in the coming years, as well as identify the most common categories of vulnerabilities.

The results of this study indicate that the vulnerabilities with the greatest impact are usually found in free and open source software. The most repeated software products in the ranking are different versions and products of CISCO (IOS, XE, etc.), versions 3.x and 4.x of the Linux Kernel, and Mozilla software (Firefox and Thunderbird). To a lesser extent, there is also presence of software products from NTP, ImageMagick, Moodle, Tryton, Django, etc.

It is worth mentioning the notable presence of vulnerabilities in the Linux kernel and Apple's Mac OS X operating system, despite their good reputation in terms of security features. In this sense, the information of the NVD database contains more vulnerabilities and with greater impact of the Linux Kernel and Apple's Mac OS X operating system than those of Microsoft's Windows operating system in the three years under study. According to Ref. [23], 384 vulnerabilities were detected in Mac OS X, 77 in the Linux kernel, and 53 in Microsoft Windows 10 in 2015. This fact may be motivated by the greater or lesser willingness to make public a vulnerability detected in the system, which the free and open source software community seems to do more frequently than Microsoft[24].

As shown in Section 6, buffer overflow is the most common software vulnerability. This vulnerability causes problems ranging from a DoS to the total appropriation of the control of the application by the attacker. It is mainly a problem of low-level languages, such as C or C++, while higher level languages such as Java or Visual Basic prohibit direct access to memory and avoid this problem. For this reason, when possible, it is better to not allow users to work with low-level code, and work only with high-level code. It is also recommended that developers replace insecure functions such as `strcpy`, `strcat`, and so on[20].

Among other solutions, there is a series of well-known countermeasures to protect against buffer overflow[25]:

• *Dynamic linking of secure libraries*. These libraries replace unsafe functions with other functions with the same purpose that incorporate measures that protect against this attack. An example of this countermeasure is the library *libsafe* (https://directory.fsf.org/wiki/Libsafe).

• *Compiler tools*. The compilers insert instructions that allow verifying the integrity of the stack, as well as eliminate the conditions that an attacker needs to perform a buffer overflow attack. The best known solutions are *StackShield* (http://www.angelfire.com/sk/stackshield/info.html) and *StackGuard*[26].

Our future work includes the monitoring of vulnerabilities that can compromise the main operating systems for mobile devices (i.e., iOS and Android). Owing to the strong presence of this type of devices in our daily life[27], we consider this topic of the utmost importance. Another possible line of future work is the study of the variation of the consequences of the vulnerabilities with the greatest impact. This kind of study would analyze the situation of a vulnerability or category of vulnerabilities in a given year, and compare it to the situation in the previous years. As presented in Section 5, DoS attacks in 2017 are not as frequent as in 2015 and 2016. This suggests that we could be witnessing a change of trend, which could be caused by a greater ability or interest of the cybercriminals to attack other software weaknesses that were less common in the past[28]. In this sense, the study of the evolution over time of vulnerabilities in software products is another interesting line of work for researchers in this field.

Acknowledgment

This research was part of the BIZDEVOPS-GLOBAL-UMU project (No. RTI2018-098309-B-C33) supported by the Spanish Ministry of Economy and Competitiveness and the European Fund for Regional Development (ERDF).

References

[1] We Are Social and Hootsuite, Digital in 2017: Global overview, https://wearesocial.com/special-reports/digital-in-2017-global-overview, 2017.

[2] S. Lichtenstein, Internet risks for companies, *Comput. Secur.*, vol. 17, no. 2, pp. 143–150, 1998.

[3] M. P. Qi, J. Chen, and Y. Chen, A secure biometrics-based authentication key exchange protocol for multi-server TMIS using ECC, *Comput. Methods Programs Biomed.*, vol. 164, pp. 101–109, 2018.

[4] M. Jouini, L. B. A. Rabai, and A. B. Aissa, Classification of security threats in information systems, *Proced. Comput. Sci.*, vol. 32, pp. 489–496, 2014.

[5] A. N. Navaz, M. A. Serhani, N. Al-Qirim, and M. Gergely, Towards an efficient and energy-aware mobile big health

data architecture, *Comput. Methods Programs Biomed.*, vol. 166, pp. 137–154, 2018.

[6] O. Alhazmi, Y. Malaiya, and I. Ray, Security vulnerabilities in software systems: A quantitative perspective, in *Proc. 19th Ann. IFIP WG 11.3 Working Conf. on Data and Applications Security XIX*, Storrs, CT, USA, 2005, pp. 281–294.

[7] J. T. Gong and H. Y. Zhang, BugMap: A topographic map of bugs, in *Proc. 9th Joint Meeting on Foundations of Software Engineering*, Saint Petersburg, Russia, 2013, pp. 647–650.

[8] J. L. Fernández-Alemán, I. C. Señor, P. Á. O. Lozoya, and A. Toval, Security and privacy in electronic health records: A systematic literature review, *J. Biomed. Inform.*, vol. 46, no. 3, pp. 541–562, 2013.

[9] I. C. Señor, J. L. Fernández-Alemán, and A. Toval, Are personal health records safe? A review of free web-accessible personal health record privacy policies, *J. Med. Internet Res.*, vol. 14, p. e114, 2012.

[10] C. T. Li, D. H. Shih, and C. C. Wang, Cloud-assisted mutual authentication and privacy preservation protocol for telecare medical information systems, *Comput. Methods Programs Biomed.*, vol. 157, pp. 191–203, 2018.

[11] Y. H. Gu and P. Li, Design and research on vulnerability database, in *Proc. 3rd Int. Conf. on Information and Computing*, Wuxi, China, 2010, pp. 209–212.

[12] C. Schmidt, Technical introduction to SCAP, https://www.energy.gov/sites/prod/files/cioprod/documents/Technical_Introduction_to_SCAP_-_Charles_Schmidt.pdf, 2010.

[13] P. Mell, K. Scarfone, and S. Romanosky, Common vulnerability scoring system, *IEEE Secur. Privacy*, vol. 4, no. 6, pp. 85–89, 2006.

[14] X. D. Li, X. L. Chang, J. A. Board, and K. S. Trivedi, A novel approach for software vulnerability classification, in *Proc. 2017 Ann. Reliability and Maintainability Symp.*, Orlando, FL, USA, 2017.

[15] H. Venter, J. H. P. Eloff, and Y. L. Li, Standardising vulnerability categories, *Comput. Secur.*, vol. 27, nos. 3&4, pp. 71–83, 2008.

[16] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, Measuring, analyzing and predicting security vulnerabilities in software systems, *Comput. Secur.*, vol. 26, no. 3, pp. 219–228, 2007.

[17] S. S. Alqahtani, E. E. Eghan, and J. Rilling, Tracing known security vulnerabilities in software repositories—A semantic web enabled modeling approach, *Sci. Comput. Programming*, vol. 121, pp. 153–175, 2016.

[18] Z. B. Cruz, J. L. Fernández-Alemán, and A. Toval, Security in cloud computing: A mapping study, *Comput. Sci. Inform. Syst.*, vol. 12, no. 1, pp. 161–184, 2015.

[19] McAfee, McAfee labs threats report, https://www.mcafee.com/enterprise/en-us/assets/reports/rpquarterly-threats-mar-2016.pdf, 2016.

[20] J. C. Foster, V. Osipov, N. Bhalla, N. Heinen, and D. Aitel, *Buffer Overflow Attacks*. Syngress Publishing, 2005.

[21] Nokia, Android & iOS infections rose by 400%. Windows Infections declined, https://nokiapoweruser.com/nokia-malware-report-smartphones-infections-rose-nearly-400-percent-2016/, 2016.

[22] A. V. Uzunov, E. B. Fernandez, and K. Falkner, Assessing and improving the quality of security methodologies for distributed systems, *J. Softw.: Evol. Process*, vol. 30, no. 11, p. e1980, 2018.

[23] C. Manes, 2015's MVPs-the most vulnerable players, https://techtalk.gfi.com/2015s-mvps-the-most-vulnerable-players/, 2016.

[24] N. Metha and B. Leonard, Disclosing vulnerabilities to protect users, https://security.googleblog.com/2016/10/disclosing-vulnerabilities-to-protect.html, 2016.

[25] W. L. Du, Chapter 4: Buffer overflow attack, *Computer Security: A Hands-on Approach*, Syngress Publishing, 2017.

[26] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang, StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks, in *Proc. 7th Conf. on USENIX Security Symp.*, San Antonio, TX, USA, 1998, p. 5.

[27] Newzoo, Newzoo global mobile market report 2018—Light version, https://newzoo.com/insights/trend-reports/newzoo-global-mobile-market-report-2018-light-version/, 2018.

[28] F. Mercaldo, A. Di Sorbo, C. A. Visaggio, A. Cimitile, and F. Martinelli, An exploratory study on the evolution of android malware quality, *J. Softw.: Evol. Process*, vol. 30, no. 11, p. e1978, 2018.

Mario Calín Sánchez received the BS and MS degrees from University of Murcia, Murcia, Spain, in 2016 and 2018, respectively. He worked as a researcher in the software sustainability area for the Software Engineering Research Group of the University of Murcia, and now he is currently a software developer with the Polytechnic University of Cartagena, where he is a member of the INDIe project development team.

Juan Manuel Carrillo de Gea received the BS, MS, and PhD degrees from University of Murcia, Murcia, Spain, in 2000, 2009, and 2016, respectively. He is an assistant professor with the University of Murcia. He has published more than 30 articles on software engineering, requirements engineering, and applications in the e-health and e-learning domains in relevant journals and conferences. His current research interests include software engineering, sustainability, medical informatics, and education.

José Luis Fernández-Alemán received the BS and PhD degrees from University of Murcia, Murcia, Spain, in 1994 and 2002, respectively. He is currently an associate professor with University of Murcia, where he is a member of the Software Engineering Research Group. He has published more than 50 JCR papers in the areas of software engineering and requirements engineering and their application to the fields of e-health and e-learning. Currently, his main research interest is m-health and m-learning and their application to computer science, medicine, and nursing.

Jesús Garcerán received the BS degree from University of Murcia, Murcia, Spain, in 2018. He has worked for 8 months in HOP Ubiquitous S.L. as a software developer, where he was managing some apps, and he has experience with IoT and FIWARE. He has specialized in information systems, but he is interested in other areas inside computing, like front-end development with AngularJS, cybersecurity, and how to protect an enterprise and, finally, in the possibilities of IoT agents, especially from the side of software. He is pending to publish a research work about Semantically-Enhanced System for Pest Recognition in AISC Springer as an author.

Ambrosio Toval received the BS degree from University Complutense of Madrid, Madrid, Spain, in 1983, and the PhD degree from Technical University of Valencia, Valencia, Spain, in 1994. He is currently a full professor with University of Murcia, Spain, where he is the head of the Software Engineering Research Group. He has conducted a variety of research and technology transfer projects in the areas of requirements engineering processes and tools, privacy and security requirements, sustainable requirements, and applications in the e-health, e-learning, and mobile development domains. He has published in the same topics in international journals, such as *IEEE Software, Information and Software Technology, Requirements Engineering, Computer Standards & Interfaces, IET Software, International Journal of Information Security*, etc.