Software Vulnerability Analysis Using CPE and CVE

Luis Alberto Benthin Sanguino, Rafael Uetz
Fraunhofer FKIE, Bonn, Germany

Abstract
In this paper, we analyze the Common Platform Enumeration (CPE) dictionary and the Common Vulnerabilities and Exposures (CVE) feeds. These repositories are widely used in Vulnerability Management Systems (VMSs) to check for known vulnerabilities in software products. The analysis shows, among other issues, a lack of synchronization between both datasets that can lead to incorrect results output by VMSs relying on those datasets. To deal with these problems, we developed a method that recommends to a user a prioritized list of CPE identifiers for a given software product. The user can then assign (and, if necessary, adapt) the most suitable CPE identifier to the software so that regular (e.g., daily) checks can find known vulnerabilities for this software in the CVE feeds. Our evaluation of this method shows that this interaction is indeed necessary because a fully automated CPE assignment is prone to errors due to the CPE and CVE shortcomings. We implemented an open-source VMS that employs the proposed method and published it on GitHub.

Keywords: elsarticle.cls, LaTeX, Elsevier, template

1. Introduction

Cybercriminals often infect their victims by exploiting vulnerabilities in software products. For example, a cybercriminal can persuade a user to open a malicious PDF document using social engineering techniques like spear phish-
ing. If the user employs a vulnerable version of a PDF reader, malicious code embedded in the document exploits the vulnerability, resulting in the user being infected with malware (e.g., Ransomware).

Vulnerabilities on software products seem to be a never-ending problem. In 2015, 5,585 new vulnerabilities were found [1]. Similarly, in 2016, more than 6,000 software flaws were published in the National Vulnerability Database (NVD) [1]. The number of undisclosed or zero-day vulnerabilities has increased over the last years [1]. Nonetheless, publicly known vulnerabilities still are the highest risk for organizations. As reported by the SANS institute, in more than 80% of the security incidents known flaws were exploited [2]. The most exploited vulnerabilities in 2015 were vulnerabilities published in 2011 and 2007 [3].

To avoid data breaches caused by software vulnerabilities, organizations deploy Vulnerability Management Systems (VMSs) inside their IT infrastructures. Among the main functionalities, a VMS scans for software products installed inside a company that has potential vulnerabilities. To perform this task, the system manages an inventory containing information of the installed products, and moreover, it correlates this information with vulnerability information obtained from private (e.g., software vendors) and public sources, such as Computer Emergency Response Teams (CERTs), the NVD or Bugtraq [2].

Within the most used sources of vulnerability information are the Common Vulnerabilities and Exposures (CVE) feeds [3], which are a subset of the NVD and offered by the National Institute of Standards and Technology (NIST). The CVE feeds contain identifiers for known vulnerabilities (e.g., CVE-2015-1538) and information related to them. To identify software products affected by a vulnerability, a CVE includes a list of vulnerable software, in which the description of a product follows the specifications of the Common Platform Enumeration (CPE) standard [4].

[1] https://nvd.nist.gov
[2] http://bugtraq-team.com
[3] https://nvd.nist.gov/vuln/data-feeds
[4] https://cpe.mitre.org
The use of standards like CVE and CPE helps IT security vendors, security experts, research institutes, etc. to publish and exchange information efficiently and effectively. Additionally, the standards are employed by VMSs to assign structured identifiers to software products (inventory) and search for vulnerabilities related to them (vulnerability scanning). In this paper, we present an analysis of the official CPE dictionary and CVE feeds that show inconsistencies in these datasets that could prevent VMSs from delivering reliable results when scanning for vulnerabilities.

The paper is structured as follows. We first introduce the CPE and CVE standards and their role in VMSs (Section 2). Then, we provide an analysis of the CPE dictionary and CVE feeds (Section 3). In Section 4, we describe a method that aims at dealing with the issues found in the CPE dictionary and CVE feeds. Section 5 contains an evaluation of our method. Section 6 describes related work. Finally, Section 7 gives a conclusion.

2. Background

This section briefly describes the standards that are used in our method to scan for vulnerabilities: Common Platform Enumeration (CPE) and Common Vulnerabilities and Exposures (CVE). In addition, we explain the relation between both standards and how they are employed by VMSs.

2.1. Common Platform Enumeration (CPE)

CPE is a method that specifies a naming scheme for applications, hardware devices, and operating systems. CPE is part of the Security Content Automation Protocol (SCAP) standard, which was proposed by the National Institute of Standards and Technology (NIST). Currently, there exist two versions of the CPE specification: CPE 2.2 and CPE 2.3. Version 2.3 defines a stack formed by five specifications, including the CPE naming specification [4] and the CPE dictionary specification [5].
The CPE naming scheme is defined by a set of attributes called *Well-Formed CPE Name* (WFN). The following attributes are part of this format: *part*, *vendor*, *product*, *version*, *update*, *edition*, *language*, *sw_edition*, *target_sw*, *target_hw*, and other. Listing 1 shows the WFN format for the software product *Microsoft Internet Explorer 8*. The value of the attribute *part* indicates that the IT asset is an application. The logical value *NA* means not applicable or not used. *NA* is assigned to attributes that have no meaning for a software product. The logical value *ANY* indicates that there are no restrictions for an attribute.

**Listing 1: Possible WFN format for Microsoft Internet Explorer 8**

\[
\text{part}: \text{a}, \quad \text{vendor}: \text{microsoft}, \quad \text{product}: \text{internet\_explorer}, \quad \text{version}: 8, \\
\text{update}: \text{NA}, \quad \text{edition}: \text{ANY}, \quad \text{language}: \text{ANY}, \quad \text{sw\_edition}: \text{ANY}, \\
\text{target\_sw}: \text{ANY}, \quad \text{target\_hw}: \text{ANY}, \quad \text{other}: \text{ANY}
\]

The CPE specification is used to assign identifiers to assets inside an IT infrastructure. Currently, CPE supports two formats: URI (defined in CPE version 2.2) and formatted string (defined in CPE version 2.3). A URI or formatted string identifier is generated from the WFN of an IT asset. This process is called URI or formatted string binding. Listing 2 and Listing 3 show examples for the URI binding and formatted string binding respectively.

**Listing 2: WFN of Listing 1 bound to a URI**

\[
cpe:/a:microsoft:internet\_explorer:8:–
\]

**Listing 3: WFN of Listing 1 bound to a formatted string**

\[
cpe:2.3:a:microsoft:internet\_explorer:8:–:*:*:*:*:*
\]

Besides the naming specification, the CPE 2.3 stack encompasses the CPE dictionary specification, which defines the structure (e.g., names and metadata) of a repository containing CPE identifiers for classes of IT products. Each entry in the dictionary contains the bound form (URI and/or formatted string) of a product’s WFN. The National Vulnerability Database (NVD) hosts and maintains the official CPE dictionary which is provided in XML format. Figure 1

---

6 [https://nvd.nist.gov/products/cpe](https://nvd.nist.gov/products/cpe)
Figure 1: CPE entry for Mozilla Firefox 38.0 in the official CPE dictionary

shows a CPE entry of the official dictionary version 2.3. The entry contains the URI (for backward compatibility with CPE version 2.2) and formatted string that identify the application Mozilla Firefox 38.0.

2.2. Common Vulnerabilities and Exposures (CVE)

CVE, which is also part of the SCAP specifications, is a method used to assign identifiers to publicly known vulnerabilities found in IT products and to provide information (e.g., affected products) about the vulnerabilities. Across organizations, anti-virus vendors, and security experts, CVE has become the de facto standard to share information on known vulnerabilities and exposures. Assignment of unique identifiers to vulnerabilities is managed by the MITRE Corporation\(^7\) and the NVD provides documents in XML format (CVE feeds\(^8\)) that contain the CVE identifiers defined by MITRE along with additional information (e.g., severity, vulnerable configuration, and a list of vulnerable software). Figure 2 shows a CVE entry obtained from the CVE feed for 2016. Note that some information has been omitted to save space.

2.3. CVE and CPE in a Vulnerability Management System (VMS)

One of the goals of a VMS is the detection of vulnerable IT assets (e.g., vulnerable software products) in an IT infrastructure. A VMS also includes other aspects like risk management and remediation. In this paper, we focus on the vulnerability detection capabilities of a VMS. In this context, the CPE dictionary and CVE feeds play an important role. VMSs employing CPE and

\(^7\) https://cve.mitre.org
\(^8\) https://nvd.nist.gov/vuln/data-feeds
Figure 2: Entry of the CVE feed for 2016. The identifier of this vulnerability is CVE-2016-0006.

CVE depend on the compatibility between both datasets. The CPE dictionary is queried by a VMS to find CPE identifiers that match IT products in the infrastructure where the VMS operates. Since the CVE entries of the CVE feeds contain a list of vulnerable software in CPE format (see Figure 2), it allows the VMS to find related vulnerabilities for the IT products using their assigned CPE identifier.

3. CPE Dictionary and CVE Feeds Analysis

This section discusses issues that we found in the official CPE dictionary and CVE feeds that can lead to incorrect results of VMSs relying on those

[9] https://nvd.nist.gov/cpe.cfm
[10] https://nvd.nist.gov/download.cfm
repositories to find vulnerabilities in software products.

### 3.1. CVE entries without CPE entries

The CVE feeds (as of February 14th, 2017) contain 895 CVE entries that do not have any CPE identifier. For example, CVE-2016-9748 is a vulnerability for the products *IBM Rational DOORS Next Generation 5.0* and *6.0*. This CVE has no CPE identifier, even though the official CPE dictionary has CPE identifiers for these products, as shown in Listing 4.

*Why this is a problem.* VMSs that only use CPE identifiers to find related CVE entries for software products deliver incomplete results. For example, if a vulnerability is found and a CVE entry is created for it without CPE identifiers, a VMS would not generate any alert for the IT assets affected by that vulnerability.

Listing 4: CPE identifiers found in the official CPE dictionary (accessed on February 14th, 2017) for IBM Rational DOORS Next Generation versions 5.0 and 6.0.0

```plaintext
cpe:/a:ibm:rational_doors_next_generation:5.0.
cpe:/a:ibm:rational_doors_next_generation:6.0.0
```

### 3.2. Software Products without assigned CPE

There are software products for which no CPE entry has been created in the official CPE dictionary. For example, the product *Microsoft Search Server* does not have an entry in the dictionary. Nonetheless, according to CVE-2008-4032, version 2008 of this product is vulnerable, and the CVE entry contains two CPE identifiers for it.

*Why this is a problem.* Full automation (i.e., without human interaction) of the process of assigning a CPE identifier to a software product can lead to false positives or false negatives. Since there are software products for which no CPE identifier exists in the CPE dictionary, a fully automated system trying to

---

1. [https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-4032](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-4032)
assign an existing CPE to a software product would either assign an incorrect
best match or not assign a CPE at all. Consequently, either incorrect CVE
entries are found for these software products (false positives), or the actual
CVE entries related to those products are not found (false negative).

3.3. CPE Dictionary Deprecation Process

According to the CPE dictionary specification version 2.3 [5], CPE identifiers
are deprecated for three reasons: identifier name correction, identifier name
removal, or additional information discovery. Up to February 14th, 2017, the
CPE dictionary contained 2,614 deprecated entries; all of them are due to name
correction. Listing 5 shows one of the deprecated identifiers found in the official
CPE dictionary. In this case, the deprecation was caused by a typo in the word
player.

Why this is a problem. If a VMS assigns a CPE identifier to a software product
that is later corrected, then CVEs containing the corrected CPE could not be
found. Following the example shown in Listing 5, if a system tries to obtain
CVE entries that fulfill the condition: vendor equals adobe and name equals
flash_player_for_linux, it is likely that the query does not return any results,
since the typo is probably present only in the CPE dictionary and not in the
CVE feeds.

Listing 5: Deprecated identifier in the official CPE dictionary

cpe:/a:adobe:flash_player_for_linux:9.0.115.0 (deprecated)
cpe:/a:adobe:flash_player_for_linux:9.0.115.0 (corrected)

3.4. NVD Synchronization

The CVE feeds (as of February 14th, 2017) contain 105,591 CPE entries that
do not exist in the CPE dictionary. The analysis showed that this problem is
caused by several reasons. One of them was already pointed out in Section 3.2
in the official CPE dictionary, there are vulnerable IT products without CPEs.
In addition, there are CPEs which are semantically equal, but nonetheless, their
WFNs’ attributes contain different values, resulting in different CPEs. Listing 6 shows an example of this case. The CPE found in the CPE dictionary uses the update attribute. On the other hand, the CPE of the CVE feeds concatenates the update (beta1) with the version (1.4.0_beta1). This clearly demonstrates that synchronization between the official CPE dictionary and the CVE feeds do not take place, though both datasets are hosted in the NVD which is managed by the NIST.

Listing 6: CPEs semantically equal but with different WFN attributes. The first CPE is found in the official CPE dictionary and the second one in the CVE feeds.

cpe:/a:digium:asterisk:1.4.0:beta1

cpe:/a:digium:asterisk:1.4.0_beta1

Why this is a problem. If a VMS assigns a CPE to a software product using the entries of the CPE dictionary only, there is a risk that no CVE is found for that product.
4. Method

The method described in this paper comprises three steps: CPE matching, CPE assignment, and CVE matching. An overview is shown in Figure 3.

In the first step, CPE identifiers that could be assigned to a software product (CPE candidates) are searched for. The search is based on the product’s vendor, name, and version strings (e.g., Microsoft Internet Explorer 8) that is stored in an inventory database. As shown in Section 3, automatically assigning CPEs to software products is problematic. The evaluation in Section 5 supports this finding. Therefore, in the second step, human interaction is required to assign the most suitable CPE to a product. In the third step, CVEs are looked up for a software product based on its assigned CPE. The CPE and CVE matching is carried out using the CPE WFN format. Hence, the URIs of the CPEs are first unbound to WFNs. The described method is the core of an open-source VMS that we will present in Section 5. In the following sections, the conversion of a URI to a WFN is shortly described, and thereafter, the CPE and CVE matching algorithms are explained.

4.1. Well-Formed Name (WFN) Conversion

Before carrying out the steps shown in Figure 3, the URIs (CPE identifiers) of both CPE dictionary and CVE feeds are converted into their WFN format. The conversion is based on the CPE Naming Specification Version 2.3 [4]. Listing 7 shows the conversion of a URI binding into its WFN format.

Listing 7: URI to WFN conversion example

CPE URI
cpe:/a:microsoft:internet_explorer:8.*:en~"windows"x86
CPE WFN

part: a, vendor: microsoft, product: internet_explorer,
version: 8.*, update: ANY, language: en, edition: NA,
sw_edition: ANY, target_sw: windows, target_hw: x86, other: ANY

The idea of this conversion is to compare two CPEs based on their WFN attributes (e.g., vendor) and not using their URIs. This allows to determine
which attributes are unequal so that weights can be assigned to the attributes according to their importance. For instance, the attributes vendor, product, and version have higher weights than the attributes target_sw, target_hw, or other. In addition, the WFN format allows to perform logical comparisons between two CPEs’ versions. For example, the version 8.* is equal to 8.2.7, due to the special character *.

As explained in Section 3, the CPE dictionary can contain incorrect CPE identifiers (e.g., Listing 5 shows a URI binding with a typo in the word player). This can prevent from finding potential CVE matches for a software product. This can however be overcome with string similarity algorithms such as Levenshtein edit distance [6]. Nonetheless, if we apply this algorithm for the whole URI string, we cannot establish which attributes (e.g., vendor, product, version) of the CPEs being compared are not equal.

4.2. CPE Matching

The CPE matching algorithm aids the correct assignment of a CPE to a software product by suggesting CPE candidates found in the CPE dictionary. As depicted in Figure 4, the algorithm receives as input the product’s information obtained from an IT assets database (e.g., GLPI\(^\text{12}\)). Then, three steps are performed: Generate Search Terms, Search by Product and Vendor, and Sort by Version.

**Step 1: Generate Search Terms.** In this step, using the information provided by the inventory database, search terms to query the CPE dictionary are generated. Listing 8 shows the information obtained from a GLPI database for a Microsoft product. Two groups of search terms are generated: vendor and product search terms. The former are generated with the value of the field “product”, and the latter with the value of the field “vendor”. For both groups the following operations are performed: First, the value is split by spaces and converted

\(^{12}\)http://glpi-project.org
to lower case. For example, from the string “Microsoft .NET Framework 4.5.2”, the following terms are generated: microsoft, .net, framework, and 4.5.2. If more than one term is generated, joined terms are formed by concatenating terms with an underscore between them (e.g., microsoft.net.framework). Underscores are used in the CPE dictionary to replace spaces. If a vendor term (e.g., microsoft) is present in the product search terms, the joined terms of the product are duplicated but removing the vendor term (e.g., .net_framework). During the analysis of the CPE dictionary, we observed that some CPE identifiers include the vendor in the product attribute (e.g., cpe:/a:adobe:adobe_air:1.0) while other CPE identifiers exclude it (e.g., cpe:/a:adobe:air:15.0.0.293). Therefore, product search terms with and without the vendor are generated. Listing 9 shows the search terms generated with the information of Listing 8.

The additional search terms increase the search cost but improve the accuracy of the CPE matching algorithm. For instance, to find CPEs similar to .net_framework 4.5.2, the algorithm should also consider CPEs with a difference of at least six characters (4.5.2) in the attribute product of their WFNs. Doing this would generate inaccurate results, e.g., the string player would match the string joomla, if the threshold of the edit distance is set to six.

Listing 8: Software product obtained from a GLPI database

vendor: Microsoft Corporation
product: Microsoft .NET Framework 4.5.2
version: 4.5.51209

Listing 9: Search terms generated with the information shown in Listing 8

vendor terms: [microsoft_corporation, microsoft, corporation]
product terms: [microsoft.net_framework.4.5.2, microsoft.net_framework, microsoft.net, microsoft, .net_framework.4.5.2, .net_framework, framework, .net, 4.5.2]

Step 2: Search by Product and Vendor. Using the generated search terms, CPEs for a software product are searched for in the CPE dictionary by comparing the search terms with the CPE WFN attributes vendor and product.
consider a CPE as match, the following condition must be met:

\[(cpe.wfn.vendor \simeq \text{vendor\_search\_term}) \land (cpe.wfn.product \simeq \text{product\_search\_term})\]

Note that the symbol similar or equal \(\simeq\) and not the symbol equal \(=\) is employed. As discussed in Section 3, the CPEs can contain typographical errors; therefore, to determine the similarity between a search term and the WFN attribute, the Levenshtein distance algorithm \cite{6} is used. When the Levenshtein distance is less or equal to two, the CPE is considered a match (it can be discarded in the next steps). To establish this threshold, we analyzed the deprecated CPEs in the CPE dictionary. We observed that the CPEs deprecated due to a typo in the attribute product have a Levenshtein distance of one or two.

**Step 3: Order by Version.** Finally, the CPEs found in the previous step are ordered by version. CPEs with similar or equal versions as the software product version are prioritized. Two versioning schemes are considered: versions defined by years (e.g., 2008) and dotted versions like 1.2.49. The latter are split by the dots.

4.3. CVE Matching

In this section we describe how the CVE matching algorithm searches for CVEs related to a software product. The algorithm performs three steps: Search by CVE-CPE Entries, Search by CVE Summary, and Merge CVE Candidates.

**Step 1: Search by CVE-CPE Entries.** As explained in Section 2, each entry in the CVE feeds contains a list of software products that are affected by the CVE. The list is written in the form of CPE URIs, which facilitates finding CVEs for a software product using its assigned CPE. To determine whether a CVE is a match for a software product, the product, vendor, and version attributes of the software product’s CPE are compared with the product, vendor, and version of each entry of the CVE vulnerable software list. Note that
to compare those attributes the URIs of the CVE must be first unbound to the WFN format (see Subsection 4.1).

Algorithm 1 summarizes the process of finding CVEs for a software product in the CVE feeds. The algorithm receives as input the software’s CPE and the CVE feeds. It compares the CPEs of each CVE with the software’s CPE. To perform the comparison two functions are defined: `similar()` and `same_version()`. The function `similar()` compares two strings (e.g., `similar(microsoft, microsof`) using the Levenshtein distance algorithm \[6\] and returns true if and only if the distance is less than three. This allows dealing with possible typographical errors in either the entries of the CPE dictionary or in the entries of the CVE feeds. The function `same_version()` compares the version of a software product with the version of a CVE-CPE. The function uses the asterisk symbol to map a version to a group of versions. For example, the version 1.2.* is equal to the versions 1.2.3, 1.2.3.5256 or any version which starts with 1.2.
Input: cve_feeds, software
Output: cve_candidates

\[\text{sw.product} = \text{software.cpe.wfn.product}\]
\[\text{sw.vendor} = \text{software.cpe.wfn.vendor}\]
\[\text{sw.version} = \text{software.cpe.wfn.version}\]

for \( \text{cve} \in \text{cve.feeds} \) do
  for \( \text{cve.cpe} \in \text{cve.cpe.entries} \) do
    if similar(\text{sw.product}, \text{cve.cpe.wfn.product}) AND
      similar(\text{sw.vendor}, \text{cve.cpe.wfn.vendor}) AND
      same_version(\text{sw.version}, \text{cve.cpe.wfn.version}) then
      \text{cve.candidates.add(cve)}
    end
  end
end

Algorithm 1: Search CVE candidates for a software product by CVE-CPE entries

**Step 2: Search by CVE Summary.** As pointed out in Section 3, there are CVE entries which do not have CPE entries. To tackle this problem, the summaries of the CVEs (see Figure 2) are used to find CVE candidates for a software product. The algorithm compares, using the Levenshtein distance algorithm [6], all the words of a CVE summary with the product and vendor attributes of the software’s CPE WFN. Algorithm 2 implements this step.

**Step 3: Merge CVE Candidates.** In this step, the CVE candidates delivered by the steps “Search by CVE-CPE entries” and “Search by CVE Summary” are merged. The merged list is the final result of this phase.
Input: cves\_without\_cpe\_entries, software
Output: cve\_candidates

\[\text{sw\_product} = \text{software.cpe.wfn.product}\]
\[\text{sw\_vendor} = \text{software.cpe.wfn.vendor}\]

\[
\text{for} \ cve \in \text{cves\_without\_cpe\_entries} \ \text{do} \\
\quad \text{for} \ word \in \text{cve.summary} \ \text{do} \\
\quad \\n\quad \quad \text{if} \ \text{similar}(word, \text{sw\_product}) \ \text{AND} \ \text{similar}(word, \text{sw\_vendor}) \\
\quad \quad \quad \text{then} \\
\quad \quad \quad \quad \text{cve\_candidates.add(cve)} \\
\quad \quad \text{end} \\
\quad \text{end} \\
\text{end}
\]

Algorithm 2: Search CVE candidates for a software product by CVE summary

5. Evaluation

As part of this work, we implemented an open-source VMS software called Inventory Vulnerability Analysis (IVA)\footnote{https://github.com/fkie-cad/iva} which core is the method described in Section 4. In this section, we evaluate IVA and show its results when performing fully automated analysis. First, the environment on which IVA was evaluated is described.

5.1. Environment

To evaluate the proposed method and IVA, we set up an environment (see Figure 5) that is composed of two Windows 7 virtual machines. On one machine, we installed known vulnerable versions of twelve software products, and on the other machine, the latest versions (at the moment of carrying out the evaluation) of the same products were installed. The list of software products (see Table 1) was generated based on the top 50 products by total number of

\footnote{https://github.com/fkie-cad/iva}
distinct vulnerabilities\textsuperscript{14} (the list was accessed on March 24th 2017). From the 50 products, only those compatible with Windows 7 was considered. To collect the information of the software installed on the Windows 7 machines, Fusion-Inventory Agent\textsuperscript{15} was employed. This tool gathers information from different sources (e.g., the registry) and sends it to an inventory database (in this case, GLPI) in XML format. The software information is then read by IVA from the GLPI database.

5.2. Fully Automated Vulnerability Analysis

In this context, fully automated vulnerability analysis refers to the capability of a system to assign a CPE identifier to a software product and then scan for related vulnerabilities without human interaction. In this section, we

\textsuperscript{14}http://www.cvedetails.com/top-50-products.php  
\textsuperscript{15}http://fusioninventory.org
| Vendor  | Product        | Vulnerable | Latest       |
|---------|----------------|------------|--------------|
| P1      | Mozilla Firefox| 48.0.2     | 52.0         |
| P2      | Adobe Flash Player | 23.0.0.207 | 25.0.0.127   |
| P3      | Microsoft Internet Explorer | 8.0.7601.17514 | 11.0.9600.17843 |
| P4      | Mozilla Thunderbird | 38.6.0     | 45.8.0       |
| P5      | Mozilla SeaMonkey | 2.35       | 2.46         |
| P6      | Adobe Adobe Reader | 11.0.17    | 15.023.20070 |
| P7      | Oracle Java | 8.0.1120.15 | 8.0.1210.13  |
| P8      | Oracle Java SE Development Kit | 8.0.1120.15 | 8.0.1210.13  |
| P9      | Wireshark Wireshark | 2.0.0      | 2.2.5        |
| P10     | Apple Inc. iTunes | 12.1.3.6    | 12.5.5.5     |
| P11     | Oracle MySQL Server | 5.7.15     | 5.7.17       |
| P12     | Adobe Adobe AIR | 20.0.0.260  | 25.0.0.134   |

Table 1: List of software products used for the evaluation of IVA. The fourth column shows the vulnerable versions of the products, and the fifth column shows the latest versions (as of March 24th 2017).

| Vendor          | Product                      | Version             |
|-----------------|------------------------------|---------------------|
| Adobe Systems   | Adobe Flash Player           | 23.0.0.207          |
| Incorporated    | 23 NPAPI                     |                     |
| Microsoft Corporation | Internet Explorer | 8.0.7601.17514       |
| Oracle Corporation | Java SE Development Kit 8 Update 112 | 8.0.1120.15       |
| The Wireshark developer community, https://www.wireshark.org | Wireshark 2.0.0 (32-bit) | 2.0.0             |

Table 2: Examples of the software information collected by FusionInventory Agent from the Windows 7 machines
analyze the results of IVA when assuming that a user assigns the first (best) CPE candidate to a software product. Since IVA was not designed to do this assignment automatically, we simulate this behavior. Moreover, we do not only analyze the accuracy of IVA to assign a CPE for a software product, but also its capability to find CVEs related to the assigned CPE.

Table 3 and Table 4 show the results for the vulnerable and latest versions of the software products installed on the Windows 7 machines. The first column shows the product label according to Table 1 (e.g., P1 is the label for Mozilla Firefox). The second column represents the best CPE candidate found by IVA for a given product. In this case, this is the CPE that would be automatically assigned to the product. The third column indicates whether the CPE completely matches the product information collected by FusionInventory Agent. Note that the collected information differs from the one shown in Table 1. Table 2 shows examples of the actual information collected by FusionInventory Agent. The fourth column shows the number of CVEs that IVA found related to an assigned CPE. The fifth column shows the number of positives, i.e., CVEs that have CPE entries exactly matching the assigned CPE. The last column represents the number of false positives, CVEs with CPE entries not exactly matching the assigned CPE. For example, if a CVE has CPE entries where only the main version matches the version of the assigned CPE (e.g., 2.35 ≠ 2.32), then the CVE is considered as a false positive.

5.2.1. CPE Matching

As can be seen in Table 3, IVA would be able to automatically assign the correct CPEs to ten products; however, to the products P6 and P11 incorrect CPEs would be assigned. In the former case, the target software (android) is incorrect, and in the latter case, the product (jserver) is incorrect. As depicted in Figure 6, the correct CPE (cpe:/a:oracle:mysql:5.7.15) for P11 is suggested by IVA in the third position of the candidates list. Likewise, for the latest versions of the software products, ten out of twelve products were assigned correctly. For P6, the software edition (classic) is not the correct one. In this case, the edition
Table 3: Evaluation results for vulnerable versions of twelve software products installed on Windows 7

| CPE Matching       | CVE Matching          |
|--------------------|-----------------------|
| **Assigned CPE**   | **Correct?** | **CVEs** | **Correct** | **Incorrect** |
| P1 cpe:/a:mozilla:firefox:48.0.2 | Yes | 19 | 18 | 1 |
| P2 cpe:/a:adobe:flash_player:23.0.0.207 | Yes | 39 | 17 | 22 |
| P3 cpe:/a:microsoft:internet_explorer:8.0.7601.17514 | Yes | 349 | 340 | 9 |
| P4 cpe:/a:mozilla:thunderbird:38.6.0 | Yes | 27 | 9 | 18 |
| P5 cpe:/a:mozilla:seamonkey:2.35 | Yes | 475 | 1 | 474 |
| P6 cpe:/a:adobe:adobe_reader:11.0.17:~~~android~~~ | No | 237 | 75 | 162 |
| P7 cpe:/a:oracle:jre:8.0.1120.15 | Yes | 0 | 0 | 0 |
| P8 cpe:/a:oracle:jdk:8.0.1120.15 | Yes | 0 | 0 | 0 |
| P9 cpe:/a:wireshark:wireshark:2.0.0 | Yes | 85 | 84 | 1 |
| P10 cpe:/a:apple:itunes:12.1.3.6 1 | Yes | 146 | 3 | 143 |
| P11 cpe:/a:oracle:jserver:5.7.15 | No | - | - | - |
| P12 cpe:/a:adobe:adobe_air:20.0.0.260 | Yes | 64 | 23 | 41 |
| Assigned CPE                                                                 | Correct? | CVEs | Correct | False |
|-----------------------------------------------------------------------------|----------|------|---------|-------|
| cpe:/a:mozilla:firefox:52.0                                                  | Yes      | 0    | 0       | 0     |
| cpe:/a:adobe:flash_player:25.0.0.127                                       | Yes      | 0    | 0       | 0     |
| cpe:/a:microsoft:internet_explorer:11.0.9600.17843:-                       | Yes      | 525  | 523     | 2     |
| cpe:/a:mozilla:thunderbird:45.8.0                                           | Yes      | 0    | 0       | 0     |
| cpe:/a:mozilla:seamonkey:2.46                                               | Yes      | 475  | 0       | 475   |
| cpe:/a:adobe:acrobat_reader:dc:15.023.20070::~classic~~~                   | No       | 259  | 0       | 259   |
| cpe:/a:oracle:jre:8.0.1210.13                                               | Yes      | 0    | 0       | 0     |
| cpe:/a:oracle:jdk:8.0.1210.13                                               | Yes      | 0    | 0       | 0     |
| cpe:/a:wireshark:wireshark:2.2.5                                            | Yes      | 85   | 0       | 85    |
| cpe:/a:apple:itunes:12.5.5.5                                                | Yes      | 146  | 0       | 146   |
| cpe:/a:oracle:jserver:5.7.17                                                | No       | -    | -       | -     |
| cpe:/a:adobe:adobe_air:25.0.0.134                                            | Yes      | 0    | 0       | 0     |

Table 4: Evaluation results for the latest versions of twelve products installed on Windows 7
The results show that the method described in Section 4 fails to assign correct CPEs to some software products if the assignment is performed automatically. One problem is that the CPE dictionary does not contain CPEs for all products, as analyzed in Section 3. Furthermore, it is a challenging task to find the correct CPEs using the software information collected from a computer. In some cases the information of a software product significantly differs from its CPE. For example, in the case of Wireshark, the system must match the string “Wireshark” with the string “The Wireshark developer community, https://www.wireshark.org”. For Java, the system must match the string “jdk” with the string “Java SE Development Kit 8 Update 112”.

5.2.2. CVE Matching

In this section, we evaluate the ability of IVA to find CVEs related to a software product using its automatically assigned CPE. We call a CVE correct if the software is actually affected by it, and incorrect otherwise.
For a vulnerable version of a software product, IVA should find at least one correct CVE. Nevertheless, as shown in Table 3 this is not the case for all products. Due to inconsistencies between the version collected by FusionInventory Agent, the version schemes defined by the product’s vendor, and the version used in the CPE dictionary and CVE feeds, no CVE was found for the Java products (P7 and P8). As of version 5, a new versioning system was introduced for Java products\(^\text{16}\) For example, 5 refers to the product version and 1.5 is the developer version. However in the information gathered by the inventory agent from the Windows machine neither the product version nor the developer version is used. For instance, for Java 8 update 112, the version 8.0.1120.15 was provided. The first candidate suggested by IVA keeps the version collected by the inventory agent, but in the CVE feeds all the CPEs for Java products use the developer version (e.g., 1.8.0). Therefore, no CVEs are found for JDK and JRE when employing the version collected by the agent.

We can also observe in Table 3 that some products (e.g., P5) have a high number of incorrect CVEs. This is the case especially when the main version of different product releases are the same (e.g. 2.35 and 2.56), since IVA only considers the main version to suggest CVEs. Nevertheless, IVA prioritizes CVE CPEs with equal version as the version of the software product. Figure 7 shows the CVE matches found by IVA for Mozilla SeaMonkey 2.35. As one can see, the first CVE match has a CPE with the same version as the software product. Also note that only the CPEs that are related to the software product are shown.

When CVE matches for a software product are found, IVA does not automatically generate alerts. The user must confirm that a CVE is indeed a vulnerability for the software product. To facilitate this task, IVA can group CVEs with equal CPEs. Thus, the user can confirm or discard several CVEs at the same time. Figure 8 shows an example of grouped CVEs for Mozilla SeaMonkey 2.35. In this case, the user can discard all CVEs since they do not have any CPE that matches the version (2.35) of the product.

\(^{16}\)https://en.wikipedia.org/wiki/Java_version_history#Versioning_change
To the products P6 and P11 incorrect CPEs were assigned. Nonetheless, since IVA compares only the vendor, product, and version, for P6 75 correct CVEs were found. This is not the case for P11, since “jserver” does not correspond to the correct product name “mysql”.

For the latest versions of the software products (see Table 4) we analyze the following cases. For the products P1, P2, P4, and P5, no CVE was found, which is the result that is expected for the latest versions of software products. For P5, P6, P9, P10 CVEs were found, but none of them are correct. As explained before, when determining whether two CPEs are equal or not, apart from the vendor and product name, IVA only considers the main version (e.g., 2.35 = 2.45). The results for the Java products (P7 and P8) seem to be correct; however, as explained before, IVA employed the version collected by the inventory agent (e.g., 8.0.1120.15), which format does not correspond to the one used in the CVE feeds for Java products (1.8.0).

Even though the latest version of Internet Explorer (version 11) was installed, 523 CVEs were found for P3. In these CVEs, only the main version of
Internet Explorer 11 (e.g., cpe:/a:microsoft:internet_explorer:11) is mentioned. The CVEs description indicates however which security updates must be installed to patch the vulnerabilities. Therefore, in this case, IVA just provides a hint to the user that the software product can be vulnerable. The user needs to verify whether the required security updates have been installed.

6. Related Work

Takahashi et al. describe a system and architecture to monitor software vulnerabilities in a company’s network [7]. Similar to the method explained in this paper, the CPE dictionary is employed to look up CPE identifiers to be assigned to software products based on the information collected by agents and network scanners. In addition, the CVE feeds are used to find vulnerabilities in software products using the assigned CPEs. The proposed technique aims at carrying out this task without human interaction. The aspects discussed in Section 3 and Section 5 are not considered. Among others, the inconsistencies in the official CPE dictionary and CVE feeds could lead to a poor efficacy, risking the security of an IT infrastructure. When searching for CVEs, it is assumed
that the correct CPE was assigned. Furthermore, the employed CPE and CVE matching algorithms are not explained.

Torkura and Meinel propose an approach to improve vulnerability assessment for IT assets deployed on cloud environments (e.g., the Elastic Computing Cloud) [8]. They implemented a system using OpenVAS\(^\text{17}\) as a vulnerability scanner, and NVD and Amazon Web Service (AWS) security advisories as vulnerability information sources. Other sources of information are employed, since, based on previous studies, the authors are aware of the inaccuracy of the NVD datasets. However, solutions to deal with the NVD drawbacks are not proposed. Based on this approach, a system called CAVAS is implemented, and its ability to correctly detect cloud components (e.g., Amazon Linux) is evaluated. Nonetheless, an evaluation on the capability of CAVAS to correctly find vulnerabilities in the detected components is not provided.

Other works have already pointed out the disadvantages when using the NVD datasets for vulnerability management. Zhang et al. presented a study in which it is observed that the NVD is not adequate to find software vulnerabilities due to missing software information, different version schemas, vendors’ bad practices in vulnerability release time, and data errors [9]. By means of an experiment, Nguyen and Massacci demonstrated that a significant number of CVEs for Google Chrome erroneously refer to some other versions [10]. Fitzgerald and Foley show that inconsistencies in SCAP can cause name ambiguities in the CPE dictionary. They explain that the CPE naming specification [4] does not offer a mechanism to define relationships between CPEs, and thus, ambiguities are likely to exist [11].

7. Conclusion

In this paper, we analyzed the official CPE dictionary and the CVE feeds. Four major issues were observed: CVE entries without CPE references, software

\(^{17}\)

\[\text{http://www.openvas.org/index.de.html}\]
products without assigned CPEs, typographical errors, and a lack of synchronization between both datasets. To cope with these problems, we proposed a novel method comprised of three steps: CPE matching, CPE assignment, and CVE matching. The method is based on string similarity algorithms (e.g., edit distance), and moreover, instead of using the URI of a CPE, its WFN format is employed, which allows to determine the similarity of each attribute (e.g., vendor, product, version) separately between two compared CPEs. We also presented IVA (Inventory Vulnerability Analysis), a Vulnerability Management System that implements the proposed method. IVA is available as open-source software on GitHub\footnote{https://github.com/fkie-cad/iva}.

Our evaluation (employing IVA) showed that a fully automated assignment of a CPE identifier to a software product is impractical and error prone. IVA therefore proposes a prioritized list of CPE candidates and lets the user select (and, if necessary, change) the most suitable CPE.

During the evaluation other obstacles that can hinder the process of vulnerability analysis were encountered: name ambiguities in the CPE dictionary, versioning format inconsistencies, and significant differences between the information collected by the inventory agent and the information in the CPE and CVE repositories for a software product. The erroneous assignment of a CPE identifier to a software product can prevent from finding CVEs related to that product, thus increasing the risk of a successful cyberattack. Therefore, when using the CPE dictionary and CVE feeds for vulnerability scanning, the issues discussed in this paper should be considered. Moreover, the NIST, which is responsible for maintaining these repositories, should define a mechanism (e.g., synchronization between the datasets when adding, modifying or deleting a CPE) to overcome these issues.
References

[1] Symantec, Internet Security Threat Report, Tech. rep., Symantec (April 2016).

[2] P. John, Cyber Security Trends: Aiming Ahead of the Target to Increase Security in 2017, Tech. rep., SANS Institute (March 2017).

[3] Verizon, Data Breach Investigations Report, Tech. rep., Verizon (2016).

[4] B. A. Cheikes, D. Waltermire, K. Scarfone, Common Platform Specification: Naming Specification Version 2.3, http://csrc.nist.gov/publications/nistir/ir7695/NISTIR-7695-CPE-Naming.pdf (2011).

[5] P. Cichonski, D. Waltermire, K. Scarfone, Common Platform Specification: Dictionary Specification Version 2.3, http://csrc.nist.gov/publications/nistir/ir7697/NISTIR-7697-CPE-Dictionary.pdf (2011).

[6] P. Christen, Data Matching Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection, Springer, 2012, Ch. Field and Record Comparison. doi:10.1007/978-3-642-31164-2

[7] T. Takahashi, D. Miyamoto, K. Nakao, Toward Automated Vulnerability Monitoring Using Open Information and Standardized Tools, in: 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), 2016, pp. 1–4. doi:10.1109/PERCOMW.2016.7457049

[8] K. A. Torkura, C. Meinel, Towards Cloud-Aware Vulnerability Assessments, in: 2015 11th International Conference on Signal-Image Technology Internet-Based Systems (SITIS), 2015, pp. 746–751. doi:10.1109/SITIS.2015.63

[9] S. Zhang, D. Caragea, X. Ou, An Empirical Study on Using the National Vulnerability Database to Predict Software Vulnerabilities, Springer
[10] V. H. Nguyen, F. Massacci, The (Un)Reliability of NVD Vulnerable Versions Data: An Empirical Experiment on Google Chrome Vulnerabilities, in: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS ’13, ACM, New York, NY, USA, 2013, pp. 493–498. doi:10.1145/2484313.2484377.

[11] W. M. Fitzgerald, S. N. Foley, Avoiding Inconsistencies in the Security Content Automation Protocol, in: 2013 IEEE Conference on Communications and Network Security (CNS), 2013, pp. 454–461. doi:10.1109/CNS.2013.6682760.