A Vulnerability's Lifetime: Enhancing Version Information in CVE Databases

Leonid Glanz
Software Technology Group
TU Darmstadt, Germany
glanz@cs.tu-darmstadt.de

Sebastian Schmidt
Multimedia Communications Lab (KOM)
TU Darmstadt, Germany
schmidt@kom.tu-darmstadt.de

Sebastian Wollny
KOM
TU Darmstadt, Germany
sebastian.wollny@kom.tu-darmstadt.de

Ben Hermann
Software Technology Group
TU Darmstadt, Germany
hermann@cs.tu-darmstadt.de

ABSTRACT

The National Vulnerability Database (NVD) is a rich source of information for system administrators, software engineers, IT security consultants, and researchers in software security. Relevant information is provided in machine readable form and hence can be used for automated software security management. However, we discovered that information on affected software versions and fix information is not always available in structured form. We therefore propose to enrich the NVD database with this information and use a rule-based approach to extract this information from the informal vulnerability description. Such information is useful in software development to exchange or avoid vulnerable components as well as in security research for directed cause analysis.

1. INTRODUCTION

The secure operation of software systems requires current and accurate updates on the vulnerability status of the software itself and all of the used components. In order to aid system developers and operators to keep their software safe from known vulnerabilites, machine readable descriptions of vulnerabilites are provided to automated vulnerability management systems. These systems in turn notify developers and operators to trigger necessary actions.

We found that for this process to work efficiently, the version information in the machine readable vulnerability description has to be accurate. The most used source of vulnerability descriptions is the *Common Vulnerability Enumeration (CVE)*[1] which was originally published to provide a common name-space for vulnerabilities, facilitating inter-organisational software security management. The CVE[1] is used by the *National Vulnerability Database (NVD)*[2] to provide detailed machine readable information on software vulnerabilities in XML files. It, however, lacks information on the affected software version ranges and fixed versions. The schemata define a listing for affected software versions, but these lists are often not exhaustive. However, in order to determine if the software product currently in use is vulnerable this information needs to be complete. While the information on ranges is interesting for research in software vulnerabilites in order to analyze the root cause of the vulnerability, the fix information is useful for system developers and operators to use the next fixed version that most likely will not break compatibility.

In order to make this version information available, we contribute the following:

- a dataset containing detailed version information for each CVE entry
- a mechanism for the automated gathering of this dataset, mining the required information from multiple sources: the *CVE* and the *Common Platform Enumeration (CPE)* [3]

2. DATA USAGE

In order to find possibly affected components in a system it is necessary to clearly identify the components' versions that are vulnerable. Moreover, in order to prevent future vulnerabilities, information on vulnerability fixes is highly relevant. In this section we give two usage scenarios in which this information is helpful.

[1]https://cve.mitre.org/
[2]http://nvd.nist.gov/
[3]https://cpe.mitre.org/

2.1 Software Development

In cases where vulnerable third-party software is the cause for security flaws of a software system, (1) identification of vulnerable software components and (2) replacement of these software components by non-affected code are common steps to fill the security gap.

For the first task, the machine-readable part of the NVD CVE data can be used in order to check if any of the used software components is known to be vulnerable.

Having information on fixed versions of the previously vulnerable software components, automated vulnerability management systems could advise the usage of these versions as part of the second task outlined before. The CVE data released in the NVD does not contain information on fixed versions in machine-readable form, but we observed that this information can often be found in the informal summary.

Since version numbering schemata are often neither continuous (v2.0 might be the follow-up of v1.7) nor linear (e.g. v1.7 and v2.0 can be developed as parallel branches) information about only the first fixed version is not sufficient. A fix in v1.7 does not always mean that v2.0 is fixed with respect to this vulnerability. Hence, information on all fixed versions w.r.t. their base version is needed to fulfill the requirements. Information on fixes in combination with vulnerability lifetime ranges can be used by version recommender systems to select the correct software version to use.

2.2 Vulnerability Cause Analysis

In order to understand the cause of vulnerabilities researchers and practitioners inspect source code of vulnerable software products in order to derive new detection and protection mechanisms to prevent vulnerabilities of the same type in the future. In this process, information on the precise introduction and fix of the vulnerability is valuable. Correlated with the source code repository of the software product the version information of our dataset can help inspectors find the source code related to the vulnerability and its fix as well as failed attempts for fixes fast and easily. With our tool, inspectors are empowered to review the complete evolution of a vulnerability which will likely help them understanding its cause.

The source code files of the version before introduction, after introduction and after fix can be compared automatically. Depending on the size of the change set, inspectors can then easily point out the problem cause and begin to derive detection and protection mechanisms.

3. DATA SCHEMA

To allow usage and extension of the provided dataset we describe the existing data schema used[4] in the NVD and our extension of it.

Input Data Schema.

The source dataset we use was gathered from the NVD and consists of multiple files in XML-format containing CVE entries grouped by year. A CVE entry contains a match expression that matches the vulnerability to software products (`<vulnerable-software-list>`[5]), the CVE id (`<cve-id>`), a categorization in terms of the Common Weakness Enumeration (CWE)[6], references to related articles, the vulnerability summary (`<summary>`) and other vulnerability specific information. The CWE reference (`<cwe>`) categorizes the vulnerability. For example, CWE id 119 is labeled *Improper Restriction of Operations within the Bounds of a Memory Buffer*, which is commonly known as a buffer overflow. Listing 1 shows an XML-snippet before our transformation. The `<vulnerable-software-list>` contains the CPE identifiers for Apache Tomcat Version 6.0.24 and Version 7.0.52, additionally there is a description of the vulnerability in the `<summary>` element.

```
<entry id="CVE-2014-0099">
  <vuln:vulnerable-software-list>
    <vuln:product>
     cpe:/a:apache:tomcat:6.0.24
    </vuln:product>
    <vuln:product>
     cpe:/a:apache:tomcat:7.0.52
    </vuln:product>
    <!--...-->
  </vuln:vulnerable-software-list>
  <!--...-->
  <vuln:summary>
    Integer overflow in ... Apache Tomcat before
    6.0.40, 7.x before 7.0.53, and 8.x before
    8.0.4, when operated behind ...
  </vuln:summary>
</entry>
```

Listing 1: NVD CVE entry for CVE-2014-0099

The list of vulnerable software products (`<vulnerable-software-list>`) may contain several entries for software affected by the described vulnerability. The syntax of these entries follows the specification of the *CPE*.

Output Data Schema.

In the output data, we add information extracted from the summary of the CVE entry. We enrich the dataset with two kinds of information for each software product in the list of vulnerable software products:

1. The first version where the vulnerability is fixed in the element `<fix>`.
2. The version range which is affected by the vulnerability starting with the version in element `<start>` and ending with the version in element `<end>`. The `<end>`-tag is only provided if no `<fix>` is found, as the change of an end version is more likely than the fix version.

To aid automated tools, we provide the version information in the same CPE syntax as in other parts of the vulnerability description. In cases, where the starting, ending or fixing version cannot be extracted from the summary, only the one that can be safely inferred is provided. As software versions are sometimes maintained in parallel, vulnerabilities might affect multiple ranges at the same time. In Figure 1, we illustrate an example from CVE-2014-0099 where versions 7.0 and 8.0 have clearly defined start and fix points. In contrast, only the fixed version and no other information can be extracted for version 6.0.

Listing 2 shows the extended version of the previously introduced XML-snippet. We insert our extensions at the end of the entry tag to avoid issues with algorithms that sequentially traverse the XML-file.

```
<entry id="CVE-2014-0099">
```

[4]https://nvd.nist.gov/schema/nvd-cve-feed_2.0.xsd

[5]For better readability we omit the namespace prefixes of element names given in this section.

[6]http://cwe.mitre.org/

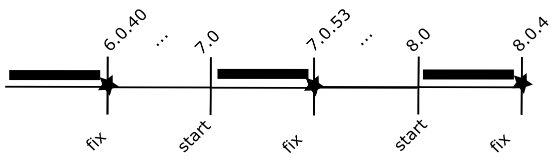

Figure 1: Extracted version information from CVE-2014-0099 (affecting Apache Tomcat). Thick lines denote ranges, stars are fixed versions.

```
<!--as before-->
<ext:ranges>
  <ext:range>
    <ext:fix>cpe:/a:apache:tomcat:6.0.40</ext:fix>
  </ext:range>
  <ext:range>
    <ext:start>cpe:/a:apache:tomcat:7.0</ext:start>
    <ext:fix>cpe:/a:apache:tomcat:7.0.53</ext:fix>
  </ext:range>
  <ext:range>
    <ext:start>cpe:/a:apache:tomcat:8.0</ext:start>
    <ext:fix>cpe:/a:apache:tomcat:8.0.4</ext:fix>
  </ext:range>
  <!--...-->
</ext:ranges>
</entry>
```

Listing 2: NVD CVE entry for CVE-2014-0099 after processing

4. DATA GATHERING

This section presents our approach for mining the two target pieces of information from the informal summary: the ranges of affected software versions and the hint information on fixed versions, which are not always the ascendant of the last affected version.

The extraction method consists of a set of extraction rules and is applied to each CVE entry separately. A schematic view on the process can be found in Figure 2.

1) Tokenization. A white space tokenizer is used to initially split the summary text into tokens.

2) Token enrichment. Each token is enriched by a feature vector where the single features are later used as hints to whether a token is part of a software name, version etc. The feature vector consists of punctuation, capitalization, regular expressions[7] and word list comparison[8] features.

3) Snippet generation. Since software names and version numbers normally do not consist of a single token, the tokens are grouped to snippets. This grouping is performed by different rules based on the feature vectors. The single feature vectors are combined to a common feature vector for the snippet. To ensure that software names are grouped in one single snippet, all sequences of tokens starting with a capital letter are combined. For example

`<t1>Molisoft</t1><t2>PDF</t2><t3>Reader</t3>`

[7]E.g. [\d]+[\p{Punct}\w]* is used as a feature to detect a major software version, or numeric regular expressions are used to detect revision numbers

[8]The word list contains key words such as *revision*, *update*, *rc* and *build* which are hints for software revisions as well as *before*, *earlier* and *after* for interval boundary detection

transforms to `<s1>Molisoft PDF Reader</s1>`.
This combination will be applied, if the first token starting with a capital letter is succeeded by a comma or a period. This ensures that entities will still be separated in enumerations or at the end of sentences. Furthermore, rules which define software versions are used. Tokens with feature hints for a major software version, a revision or a revision number are grouped as they define a software version. In this context a token sequence like

`<t1>7.2</t1><t2>update</t2><t3>3</t3>`
will be transformed to `<s1>7.2 update 3</s1>`.

4) Entity detection. To identify entities such as complete version information or software names, the whole snippet sequence of the summary is analyzed. Major software versions can be identified precisely. After the combination step, the whole software version entity holds the characteristic feature vector. Thus, an identification of software versions can be easily determined. The related software name of a software version is always mentioned beforehand. This is why another extraction definition can be used for software names: Because the vulnerability description is written in English language, only proper nouns and the first word of the sentence are written with capital initial letters. We then assume that the related software name is a previous snippet in the same sentences as the version, if the snippet is starting with a capital letter and is not located at the start of the sentence. For an example see Listing 3.

```
Cross-site scripting (XSS) vulnerability in <name>
    Managemoney Suite</name> before <version>7.4</
    version>.
```

Listing 3: Extractor annotations of a software version and name

Through previous manual inspection, we determined that related software names and versions are in close distance from one another. For that reason, we apply a maximum token limit of six tokens between a software name and version.

5) Entity refinement. We next need to refine the entity value for software versions. This is done by analyzing the snippet's context. We are interested in the first affected, last affected and fixed versions. For each of these specifications, keyword-based features like the following can be used:

- Keywords for first affected versions: e.g. **after**
  (e.g. in Software X after 11.5 update 2)
- Keywords for last affected versions: e.g. **and earlier**
  (e.g. in Software X 11.5 update 2 and earlier)
- Keyword for fixed versions : **before**
  (e.g. in Software X before 11.5 update 2)

6) Range generation. In the previous step we identified single version numbers, but not ranges. Hence, in this step we generate ranges from the collected data. To identify an affected software range, related software versions must be grouped together and sorted. We use the following procedure to identify affected software ranges (i.e. the first and last affected version):

a) All mentioned software versions of a CVE entry are inserted in sorted lists, which contain software versions of the same software name.

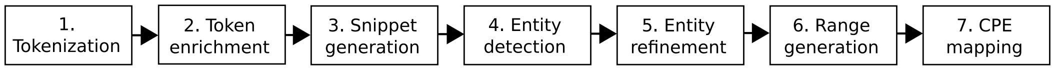

Figure 2: Schematic overview of the proposed data gathering process

b) The shortest[9] software version in this list is now defined as software branch root.
c) Potential sub versions of this software branch root will be searched and associated.
d) All related versions of a software branch root are defined as a software range.

Steps (b) to (d) are repeated until all software versions are associated to a software range.

7) CPE mapping. The results from step 6 are then mapped to the resolution of CPE identifiers in the `<vulnerable-software-list>` of the entry by calculating the Jaro-Winkler distance [6] to the identified software names.

5. EVALUATION

In order to verify the quality of the proposed approach, we manually annotated a set of 300 randomly chosen CVE entries based on annotation guidelines. Afterwards, the annotated entries were compared with the output of our language processing algorithm. The annotated dataset contains 686 (start: 274, end: 270, fixed: 142) tags. Using this dataset as gold standard, the algorithm shows to have an average F1-score of 0.70 (precision: 0.71, recall.: 0.69). The best results (F1: 0.79, prec.: 0.84, rec.:0.75) were obtained for the fix-tag. Examining the automatically extracted results, it can be observed that the most common error cause is a non-standardized usage of CPE-identifiers. In particular, for sub-version information (e.g. as beta versions or release numberings) a large variety is used, which does not adhere to the CPE-ID naming conventions. Often, the approach identifies the version information correctly, builds a CPE-ID which adheres to the naming conventions but its structure is not similar to the structure used in the existing CVE entries.

6. PUBLISHED DATASET

The complete dataset, its schema and the code for our approach can be downloaded from our GitHub repository at: https://github.com/stg-tud/cveenhance.

Applying the approach on the 70,716 CVE entries currently available in the NVD[10] results in additional information for 80.78% of the entries, fix information for 28.01%, vulnerability interval start information for 56.13% and end information for 57.80%. Examining the gold standard revealed that for 12% of the entries, the available description does not contain the targeted information making an extraction of that information impossible (e.g. when a vulnerability has not been fixed yet).

7. RELATED WORK

Proposals for extensions of the NVD and the CVE for automated security management of the software engineering process has been the focus of various work. A clustering-based approach for categorization of vulnerabilities where only the category titles have to be added manually is presented by Venter et al. [3]. Incorporating information from the *Common Weakness Enumeration* and other standards, the *Ontology for Vulnerability Management (OVM)* was proposed and populated with CVE entries [4].

The usage of CVE data for security management has been leveraged by Wang et al. [5] who propose a software security metric based on the CVE data. In particular, the vulnerability scores introduced in the *Common Vulnerability Scoring System (CVSS)* [2] are used.

[9]A short software version is one, which is near to the software version tree root.
[10]Data from 17-05-2015

8. CONCLUSION

In this paper, we present an approach to enhance the National Vulnerability Database with the version ranges of each vulnerable software component together with the version numbers in which the vulnerabilities are fixed. We extract this information through analyzing the textual descriptions in the NVD CVE entries. Such information is useful for root cause analysis of vulnerabilities as well as for automated software version recommendation.

Our approach has its limitations in cases where the summary of the entry does not contain the required information or when the versioning scheme does not follow common rules.

In future work, we aim to integrate additional external information in order to improve the overall coverage. Furthermore, the usage of the obtained data for vulnerability evolution studies will be evaluated.

Acknowledgment

This work was supported by the BMBF within EC SPRIDE, by the BMBF within the Software Campus initiative (01IS12054) and by the Hessian LOEWE excellence initiative within CASED.

9. REFERENCES

[1] D. W. Baker, S. M. Christey, W. H. Hill, and D. E. Mann. The Development of a Common Enumeration of Vulnerabilities and Exposures. In *Recent Advances in Intrusion Detection*, volume 7, page 9, 1999.
[2] P. Mell, K. Scarfone, and S. Romanosky. Common Vulnerability Scoring System. *IEEE Security Privacy*, 4(6):85–89, Nov. 2006.
[3] H. S. Venter, J. H. P. Eloff, and Y. L. Li. Standardising Vulnerability Categories. *Computers & Security*, 2008.
[4] J. A. Wang and M. Guo. OVM: An Ontology for Vulnerability Management. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, CSIIRW '09.
[5] J. A. Wang, H. Wang, M. Guo, and M. Xia. Security Metrics for Software Systems. In *Proceedings of the 47th Annual Southeast Regional Conference*, 2009.
[6] W. E. Winkler. The state of record linkage and current research problems. In *Statistical Research Division, US Census Bureau*. Citeseer, 1999.