Towards Automated Risk Analysis of "One-day" Vulnerabilities

Clément Elbaz
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
clement.elbaz@inria.fr

Louis Rilling
DGA
Rennes, France
louis.rilling@irisa.fr

Christine Morin
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
christine.morin@inria.fr

***Abstract*—Common Vulnerabilities and Exposures (CVE) databases such as Mitre's CVE List and NIST's NVD database identify every disclosed vulnerability affecting any public software. However, during the early hours of a vulnerability disclosure, the metadata associated with these vulnerabilities is either missing, wrong, or at best sparse. This creates a challenge for robust automated analysis of new vulnerabilities. We present a new technique based on TF-IDF to map newly disclosed vulnerabilities to the most probably affected software products, formulated as an ordered list of relevant entries in the Common Platform Enumeration (CPE) database. For doing so we rely only on the human readable description of the vulnerability without any need for metadata.**

***Index Terms*—Security, Cloud, CVE, CPE, TF-IDF**

I. Introduction

The disclosure of a vulnerability is the most critical part of its life cycle. As a confidential zero-day, a vulnerability is a high value asset used sparingly to attack high value targets. On the other hand, well known public vulnerabilities can be mitigated using standard security practices such as applying software updates diligently, or using a signature-based intrusion detection system (IDS). Bilge et al [1] showed that during the disclosure, the usage of exploits of the vulnerability in the wild increase as high as five orders of magnitude while transitioning from a zero-day to a public vulnerability. A software patch is sometimes already available, but its adoption may not be widespread. At this early stage the vulnerability is not understood well enough to author a proper signature rule for an IDS. All these factors contribute in making the disclosure a dangerous time, since a lot of systems are vulnerable in practice.

In a nod to the well known zero-day term, we call *one-day* these newly disclosed vulnerabilities that are still in the critical part of their life cycle. *One-day* should not be taken literally here: a vulnerability disclosure can be 72 hours old and still be at its most threatening.

The vulnerability disclosure process is coordinated by the Common Vulnerabilities and Exposures (CVE) system overseen by Mitre's Corporation [2]. Newly disclosed vulnerabilities are first published by the CVE List managed by Mitre. They are then forwarded to other security databases, such as NIST's NVD database [3] or SCAP data feeds [4], where they will eventually be annotated by multiple security experts. These annotations include metadata such as the affected software, as described by an entry from the Common Platform Enumeration (CPE) [5]. It also includes a Common Vulnerability Scoring System (CVSS) score and vector [6].

None of these steps are instantaneous: at any given time the NVD database is approximately 24 hours behind Mitre's CVE List, and NIST security experts take at least a few more days to analyze and annotate a vulnerability. As of writing this article, we can trivially find six-days-old vulnerabilities that are still not analyzed by NVD (for example CVE-2019-3908, disclosed by CVE List on 2019/18/01, has no NVD analysis as of 2019/24/01).

These delays mean that in order to reliably analyze one-day vulnerabilities, one should not rely on enriched databases such as NVD at all. Instead one should focus on Mitre's CVE List, which only provides three elements: a unique CVE identifier, a free-form human readable description, and at least one public reference [7].

Our end-goal is to propose an automated system that uses free-form descriptions of one-day vulnerabilities to infer a preliminary threat level for a given vulnerability, in the context of a specific information system (IS). Our approach is two-fold: first we intend to design a robust similarity metric between vulnerabilities, so that we can quickly associate new vulnerabilities to older, better understood ones. Second, we want to let the IS administrator annotate a set of past vulnerabilities to assess their relevance to the IS to be protected. By measuring the similarity between new and previously annotated vulnerabilities, the system can act as a first layer of analysis against new vulnerabilities, deciding in real time if an urgent action is required.

In this paper, we present a first contribution towards this end-goal: a new automated mapping technique that associates CVE vulnerabilities to CPE software entries, quickly pointing out the most probable affected software. To the best of our knowledge this is the first attempt at doing so while only relying on the free-form description of the vulnerability without any metadata help. In Section II we discuss related work and the real world challenges of such a mapping. In Section III we present the architecture of our solution. In Section IV we evaluate the accuracy of our proposed mapping technique. We conclude in Section V.

II. Related work and open problems

Extracting information and insights from the CVE corpus is not a new idea. Multiple works [8], [9], [10] brought meaningful insights using statistical analyses of historical vulnerabilities in the NVD database.

Frei et al [8] found a statistical correlation between the availability of exploits and patches and the number of days since disclosure.

Clark et al [9] brought to light a "honeymoon effect" where more recent software is less subject to new vulnerabilities than older software, all else being equal.

In our own previous work [10] we showed that cloud providers could exploit the predictability in the vulnerabilities life cycle to design new and profitable Service-Level Agreements about improving the security of their clients.

The closest work to ours is by Ganz et al [11]. They attempted to automatically enrich the quality of the metadata in NVD, by blending the existing metadata with textual analysis of the description. However their technique still requires the availability of existing metadata, while ours does not.

All of these studies point out inconsistent metadata as a major difficulty when working with the corpus. By being a theoretical database authored manually by security experts, CPE can not map perfectly to actual software binaries and packages in a production system [12], [13]. This situation creates a lot of discrepancies when doing analysis, such as associating a CVE vulnerability to a CPE entries with an incorrect version or name.

Moreover, even if this situation was solved and it was possible to fully map CVE to CPE, and CPE to actual software, this mapping would still be done manually by security experts. This is however impractical when dealing with one-day vulnerabilities because the analysis arrives too late in the vulnerability life cycle.

III. Our approach

We consider the explainability of automated decision-making as a paramount quality of security systems. Therefore we deliberately choose to avoid elaborated machine learning methods when their accuracy comes at the expense of explainability. We evaluated several approaches for this work which we detail below, from the most simple to the most accurate.

The simplest possible approach we could think of is exact pattern matching.

We view the CPE database as a highly dimensional space (one dimension per CPE entry, the CPE database having 175 200 entries as of 2019/23/01), and the CVE mapping as a vector in this space. With exact pattern matching, the components of the vector are binary: if a CPE entry title is exactly present in the description, we consider it a match and we put the scalar component to 1. Otherwise, we leave it to 0.

We found this approach to fail in practice, as most CVE do not spell exact CPE entries explicitly. For instance Table I details the matching results of a randomly chosen vulnerability, CVE-2016-5181 [14], with a sample of entries from the CPE database. CVE-2016-5181 is an actual vulnerability affecting Google Chrome. We notice that the description refer to "Google Chrome prior to 54.0.2840.59 for Windows, Mac, and Linux" instead of spelling every affected version and every affected platform. This kind of shortened enumerations are common enough to prevent naive exact pattern matching to work.

CVE-2016-5181		
Blink in Google Chrome prior to 54.0.2840.59 for Windows, Mac, and Linux; 54.0.2840.85 for Android permitted execution of v8 microtasks while the DOM was in an inconsistent state, which allowed a remote attacker to inject arbitrary scripts or HTML (UXSS) via crafted HTML pages.		
CPE entry	Exact matching	Partial matching
Linux Kernel 4.10.14	N	1
Linux Kernel 4.10.15	N	1
Google Chrome 54.0.2840.59	N	3
Google Chrome 54.0.2840.85	N	3
Google Chrome 53.0.2785.143	N	2
Microsoft Windows 10 64-bits	N	1
Juniper Remote Security Client	N	1
Oracle HTML DB	N	1
Apache Tomcat 8.0.21	N	0

TABLE I
Matching results using exact and partial pattern matching for CVE-2016-5181.

We then tried partial pattern matching, by tokenizing every CVE and CPE entry into individual words, and incrementing the associated scalar component of the CVE vector for each individual word match (case insensitive). We can then identify the most probable CPE as the one associated with the highest component of the vector. We can also compare two CVE vectors using a standard distance metric such as the euclidean distance. However this naive approach proved to be too noisy.

If we look again at table I, we can see that although correct entries are matched, many irrelevant entries are also matched as false positives.

This high number of false positive makes partial pattern matching unworkable in practice. Entries such as "IBM Tivoli Service Request Manager" have a partial match with any CVE description that uses the word "request" or "service"!

We improved the results by using weighted partial pattern matching. We still increment the scalar component of the CVE vector for each word match, but instead of incrementing it by a fixed amount we add the term frequency-inverse document frequency (TF-IDF) [15] value of the word, in the context of the CVE corpus.

TF-IDF is a numerical statistic reflecting the importance of a word to a document, in the context of a corpus. In our context, we consider a CVE description as an individual document, and the set of the descriptions of all past vulnerabilities as a corpus. TF-IDF is formally defined in Equation 1:

$$\mathrm{TFIDF}(t, d, D) = \mathrm{TF}(t, d) \times \mathrm{IDF}(t, D), \tag{1}$$

where t is a word and d is a document belonging to a corpus

of documents D. TF is defined as the number of occurrences of the word t in document d. IDF is defined in Equation 2:

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|d \in D : t \in d|}, \quad (2)$$

where $|D|$ is the number of documents in the corpus, and $|d \in D : t \in d|$ is the number of documents of the corpus containing the word t.

TF-IDF therefore allows more specific words to have a bigger impact on the mapping than common words. As an intuitive example, if we look back at CVE-2016-5181, the word "54.0.2840.59" is much more specific than "remote". Therefore the matching score should be higher for "Google Chrome 54.0.2840.59" than "Juniper Remote Security Client".

We evaluate the results of this approach in the next section.

IV. Evaluation

We used the CVE corpus for the year 2016 as an evaluation dataset. It includes 8068 actual (non-rejected) vulnerabilities. We mapped it to the CPE dictionary in version 2.3, extracted in December 2017. As mentioned in section II, the corpus metadata proved to be too inconsistent to be used as a reliable ground truth. We found that more than 20% of vulnerabilities had CPE references pointing to nonexistent CPE entries, and more than 10% did not have any CPE reference at all.

Therefore instead of relying on metadata, we evaluated our results manually. We randomly sampled 229 vulnerabilities from the CVE corpus, applied our mapping, and annotated manually the correctness of the proposed CPE entries.

We chose a broad definition of correctness: we consider the top three CPE entries proposed by the mapping. If at least one of them is a software affected by the vulnerability, we consider the vulnerability as correctly classified. Also we do not require the version to be correct.

With this definition of correctness, we found 151 correctly classified vulnerabilities out of 229, a 65.94% correctness rate.

As we outlined in section II we did not find any comparable result in the literature : past research focused on analyzing and enriching present metadata instead of limiting the reliance on it. To provide some context, the version of the CPE dictionary we used for this experiment describes 124681 CPE entries related to 17631 unique pieces of software (CPE entries relates to a specific version of a piece of software). Therefore our result is far better than a random choice (three random picks in a set of 17631 elements).

On the other hand, we acknowledge that our technique is giving incorrect results one third of the time, which makes many potential applications impractical at this stage.

Regarding performance, a full mapping of the 2016 corpus took under 2mn30 on a commodity laptop with 16 Gb of RAM and an Intel Core i7-7600U CPU @ 2.80GHz. We released all our experiment code under AGPL licence [16].

V. Conclusion

We introduced a method to automatically propose the most probable CPE software entry for a CVE vulnerability, relying only on its human readable description.

Our results are promising, as a quite simple technique already brings quite accurate results. However there is room for improvement, and we believe a better accuracy can be achieved in the future using more elaborated techniques, while still preserving the explainability of the decision.

So far we considered CPE entries to be completely independent from each other, as we treated them as different dimensions of an euclidean space. However, they are actually not independent. Two vulnerabilities matching respectively the CPE entry "Google Chrome 54.0.2840.59" and the CPE entry "Google Chrome 54.0.2840.85" are definitely closer to each other than two vulnerabilities matching respectively "Oracle HTML DB" and "Apache Tomcat 8.0.21". This is not currently reflected in the algorithm.

Also, we would like to introduce a "confidence" metric to the prediction, allowing the mapping to express a confidence score about the accuracy of the selected CPE entries for a CVE vulnerability.

In the future we hope to use our mapping technique to create a high quality similarity metric for vulnerabilities. This metric would allow us to quickly compare newly disclosed vulnerabilities to older ones, providing an immediate automated risk analysis mechanism for one-day vulnerabilities.

References

[1] L. Bilge and T. Dumitraş, "Before We Knew It: An Empirical Study of Zero-day Attacks in the Real World," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 833–844.

[2] Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/.

[3] National Vulnerability Database. https://nvd.nist.gov/.

[4] Security Content Automation Protocol. https://csrc.nist.gov/projects/security-content-automation-protocol.

[5] NVD - CPE. https://nvd.nist.gov/products/cpe.

[6] Common Vulnerability Scoring System. https://www.first.org/cvss/.

[7] CVE and NVD Relationship. https://cve.mitre.org/about/cve_and_nvd_relationship.html.

[8] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale Vulnerability Analysis," in *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense*, ser. LSAD '06. New York, NY, USA: ACM, 2006, pp. 131–138.

[9] S. Clark, S. Frei, M. Blaze, and J. Smith, "Familiarity Breeds Contempt: The Honeymoon Effect and the Role of Legacy Code in Zero-day Vulnerabilities," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 251–260.

[10] C. Elbaz, L. Rilling, and C. Morin, "Mesurer et prévenir l'évolution de la menace dans un cloud d'infrastructure," in *ComPas'18 - Conférence d'informatique en Parallélisme, Architecture et Système*, Toulouse, France, Jul. 2018, pp. 1–7.

[11] L. Glanz, S. Schmidt, S. Wollny, and B. Hermann, "A Vulnerability's Lifetime: Enhancing Version Information in CVE Databases," in *Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business*, ser. i-KNOW '15. New York, NY, USA: ACM, 2015, pp. 28:1–28:4.

[12] A. Dulaunoy. (2016) The Myth of Software and Hardware Vulnerability Management. https://www.foo.be/2016/05/The_Myth_of_Vulnerability_Management/.

[13] L. A. B. Sanguino and R. Uetz, "Software Vulnerability Analysis Using CPE and CVE," *CoRR*, vol. abs/1705.05347, 2017.

[14] NVD - CVE-2016-5181. https://nvd.nist.gov/vuln/detail/CVE-2016-5181.

[15] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, pp. 11–21, 1972.

[16] Firres. https://gitlab.inria.fr/celbaz/firres_ressi.