Automated Keyword Extraction from "One-day" Vulnerabilities at Disclosure

Clément Elbaz, Louis Rilling, Christine Morin

Automated Keyword Extraction from "One-day" Vulnerabilities at Disclosure

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 assess the software products most probably affected by newly disclosed vulnerabilities, formulated as an ordered list of relevant keywords. For doing so we rely only on the human readable description of a new vulnerability without any need for its metadata. Our evaluation results suggest real world applicability of our technique.**

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 at disclosure, the usage of exploits of a vulnerability in the wild increases 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 to making the disclosure a dangerous time, since a lot of systems are vulnerable in practice. 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 period.

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 on the *CVE List* data feed 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]. NIST security experts take at least a few days to analyze and annotate a vulnerability, and often weeks (see Section III-A). It is common to find vulnerabilities that have been disclosed for several days that are still not analyzed by NVD. For example CVE-2019-9084, disclosed on the CVE List on 06/07/2019, has no NVD analysis as of 06/11/2019. This delay means that in order to reliably analyze one-day vulnerabilities, one should not rely at all on enriched metadata provided by databases such as NVD. Instead one should focus on the data available when the vulnerability is first disclosed on Mitre's CVE List, which consists of three elements only: a unique CVE identifier, a free-form human readable description, and at least one public reference [7].

The vulnerability analysis ecosystem presented above makes it expensive for organizations to analyze one-day vulnerabilities at disclosure. On the one hand achieving real-time threat evaluation of new vulnerabilities through manual analysis requires extensive man power as hundreds of vulnerabilities are disclosed daily. On the other hand there is not enough machine-readable metadata available at disclosure for automated analysis. Real-time threat analysis is therefore prohibitively expensive for most organizations, although it would benefit them as severe vulnerabilities such as Shellshock have been massively exploited within hours of their disclosure [8].

Automating real-time threat evaluation for newly disclosed vulnerabilities would make it affordable for more organizations. This would allow cloud service providers (CSP) and information systems to react in real-time to vulnerability disclosures. Examples of automated reactions include reconfiguring security policies by elevating logging levels for critical systems, switching these systems into degraded mode or even shutting them down while waiting for a remediation to be applied. Such a reaction service could help the CSP to protect both its internal systems and tenants (the latter constituting a potential source of revenues for the CSP).

We propose an automated system that uses free-form descriptions of newly-disclosed one-day vulnerabilities to extract the most probable affected software from the description, and can do so in near real-time (at most seconds after the disclosure). Identifying which systems are vulnerable can be achieved by extracting relevant keywords from the free-form vulnerability description and forwarding them to an alert service monitoring specific keywords related to these systems

(such as names of public software used in the system).

Our system associates CVE vulnerabilities to keywords extracted from past CPE URIs to quickly point 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 vulnerabilities, without using their metadata.

In Section II we discuss related work and the real world challenges of working with vulnerability data. In Section III we present our approach. In Section IV we evaluate the accuracy of our proposed technique. We conclude in Section V.

II. Related Work and Open Problems

Most cloud providers provide Intrusion Prevention System (IPS) or Web Application Firewall (WAF) capabilities among their commercial offering [9] [10] [11]. However, to the best of our knowledge, the process of monitoring new vulnerabilities and adding related rules is always done manually [12]. Extracting information and insights from the CVE corpus is not a new idea. Multiple works brought meaningful insights using statistical analyses of historical vulnerabilities in the NVD database. Frei et al. [13] found a statistical correlation between the availability of exploits and patches and the number of days since disclosure. Clark et al. [14] brought to light a "honeymoon effect" where more recent software is less subject to new vulnerabilities than older software, everything else being equal. Ganz et al. [15] 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 for the enriched vulnerability, while ours does not. The closest work to ours is by Jacobs et al. [16] who proposed the Exploit Prediction Scoring System (EPSS). Like our work, EPSS is to be used at vulnerability disclosure: they try to determine a new vulnerability's probability of exploitation in the next twelve months. They use a logistic regression model trained using both public and non-public data-sources that can infer probabilities for new vulnerabilities using only public data-sources. Our work does not require any non-public data-source. 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 cannot map perfectly to actual software binaries and packages in a production system [17], [18]. This situation creates a lot of discrepancies when doing analysis, such as associating a CVE vulnerability to incorrect or inexisting CPE entries. 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. Last, all these studies except [16] considered the NVD database as static historical data to be studied retroactively. To the best of our knowledge, our technique and evaluation protocol are the first to focus on disclosure time analysis of new vulnerabilities, only considering vulnerabilities and metadata publicly available at disclosure.

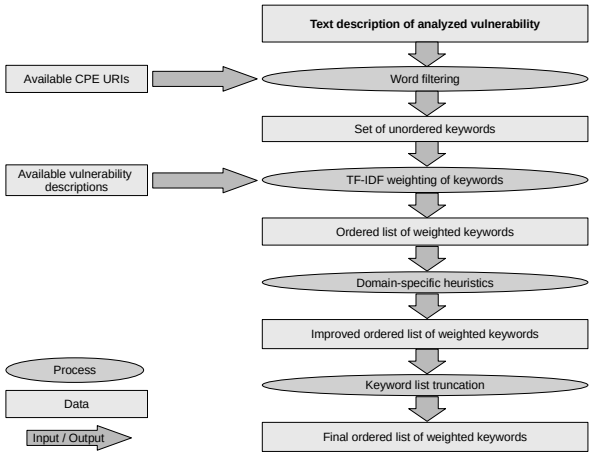

Fig. 1: Overview of the vulnerability description processing pipeline.

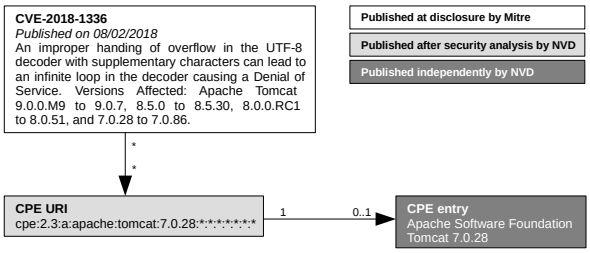

Fig. 2: The relationships between the vulnerability CVE-2018-1336 and its associated CPE URI and entries. The vulnerability, its metadata, and the CPE entries have three different publication processes.

III. Our Approach

In this section we present our keyword extraction pipeline. Its input is the free-form description of a new vulnerability. It is analyzed using all vulnerability descriptions and metadata available at the time of disclosure. It outputs an ordered list of keywords, where each keyword is given a weight representing its estimated relevance. As an intuitive example, Table I presents a sample of vulnerabilities, including their free-form description and the corresponding keywords extracted by our analysis technique. We consider the explainability of automated analysis as a paramount quality of security systems. Therefore we deliberately chose to avoid elaborated machine learning methods (such as deep learning) when their accuracy comes at the expense of explainability. A high-level overview of the proposed vulnerability analysis pipeline is shown in Figure 1. We now describe each stage of the pipeline in more details, starting with our choice of data sources.

A. Data Sources Considerations

The CVE and the CPE corpus are linked together using a metadata called the *CPE URI* defined in NIST-IR 7695 [19].

CVE ID	Disclosure date	Description	
CVE-2016-6808	04/12/2017	Buffer overflow in Apache Tomcat Connectors (mod_jk) before 1.2.42.	
		Weight	**Keywords**
		27.54	tomcat connectors
		12.41	mod jk
		11.01	connectors
		8.37	tomcat
CVE-2017-0155	04/12/2017	The Graphics component in the kernel in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; and Windows 7 SP1 allows local users to gain privileges via a crafted application, aka "Windows Graphics Elevation of Privilege Vulnerability."	
		Weight	**Keywords**
		18.46	windows server 2008
		12.30	windows vista
		12.28	windows 7
		12.18	graphics
		11.84	server 2008
		11.74	windows server
		8.70	windows
		5.95	r2
CVE-2015-3421	07/21/2017	The eshop_checkout function in checkout.php in the Wordpress Eshop plugin 6.3.11 and earlier does not validate variables in the "eshopcart" HTTP cookie, which allows remote attackers to perform cross-site scripting (XSS) attacks, or a path disclosure attack via crafted variables named after target PHP variables.	
		Weight	**Keywords**
		26.29	eshop plugin
		19.10	eshop
		12.33	checkout php
		5.71	wordpress
CVE-2015-5194	07/21/2017	The log_config_command function in ntp_parser.y in ntpd in NTP before 4.2.7p42 allows remote attackers to cause a denial of service (ntpd crash) via crafted logconfig commands.	
		Weight	**Keywords**
		14.69	ntp
		5.07	y
		3.50	parser
		3.44	config

TABLE I: A sample of keyword extraction results, using our analysis pipeline (with all heuristics enabled and a keyword list truncation target of 95% of the norm).

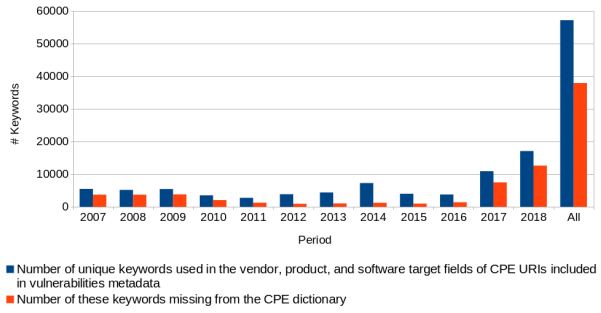

Fig. 3: Historical rate of software names used in vulnerabilities CPE URIs that are missing from the CPE dictionary.

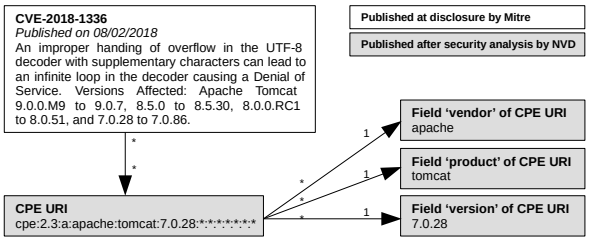

Fig. 4: When discarding the CPE dictionary we get a more robust data life cycle while retaining most of the inherent data.

A CPE URI is a unique reference to a specific entry in the CPE database, a specific version of a piece of software. An example of the relationships between CVE, CPE URI and CPE entries can be found in Figure 2.

It is tempting to consider these corpus as two relational tables linked together using a foreign key. However we discovered two drawbacks of this approach. The first one is that the life cycles of the two databases are very different, resulting in an ever-increasing number of "dead" CPE URIs entries referenced in vulnerabilities metadata that do not actually exist in the CPE database. Figure 3 illustrates the problem. While both databases were mostly kept consistent from 2011 to 2016, the inconsistencies grew substantially in 2017 and 2018, to the point that in 2018 73.8% of the software names mentioned in CPE URIs included in vulnerabilities metadata are not present in the CPE dictionary. From 2007 to 2018, on average 66.3% of the software names are missing. We want to emphasize

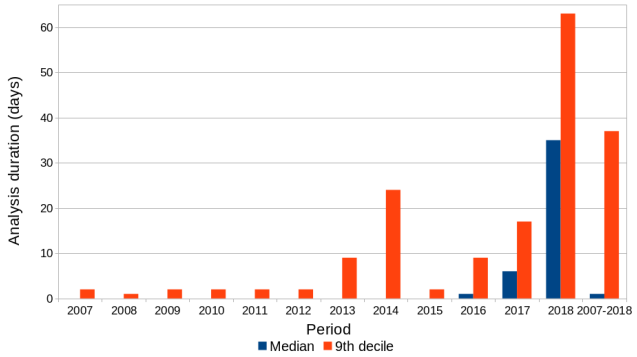

Fig. 5: Number of days between vulnerability disclosure and analysis in NVD from 2007 to 2018.

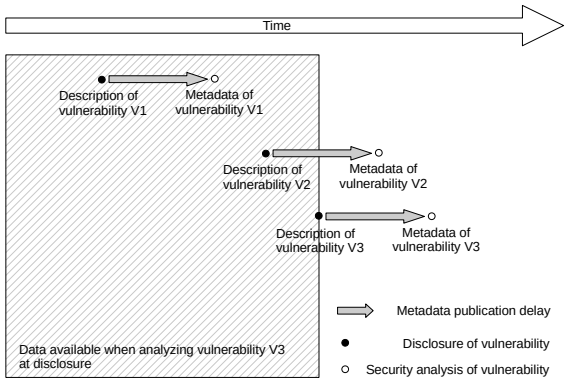

Fig. 6: In this example, when analyzing the text description of vulnerability V3 at disclosure time we have access to the text descriptions for V1 and V2 and to the metadata for V1, but not the metadata for V2 or V3.

that this is an important problem that, if left unchecked, will greatly decrease the relevance and real world usefulness of the CPE dictionary. A second drawback is the lack of a date-of-inclusion field for CPE entries in the CPE database. While this would have no impact in a production system, it prevents us from properly evaluating our results using a journaled view of the corpus. In order to properly simulate an analysis at disclosure time, we want to only consider CVE and CPE published "in the past" compared to the disclosure date of the analyzed vulnerability.

We solved both problems by discarding the CPE database completely and instead use the data embedded in the fields of the CPE URIs, as described in Figure 4. As CPE URIs are part of a CVE vulnerability's metadata, we can reuse the date-of-publication field of the vulnerability for the included CPE URIs. However as we saw in Section I, metadata is not published at disclosure time, but authored by security experts several days after. Figure 5 shows the number of days between vulnerability disclosure and analysis publication in NVD from 2007 to 2018. Historically the median analysis duration has been zero day while the 9th decile has been two days. While this remained true until 2016 (for the median) and 2012 (for the 9th decile), there have been sharp drops in NVD analyses timeliness since then. In 2018 the median and 9th decile analysis duration reached 35 days and 63 days respectively. Therefore we decided to set the notion of a fixed *metadata publication delay* for all vulnerabilities which we fixed at sixty days. This means that when a vulnerability is disclosed on day N, we consider that its metadata will be published on day $N + 60$. Conversely, when analyzing a vulnerability disclosed on day N, we have access to all vulnerability descriptions up to day N and to all vulnerability metadata up to day $N - 60$. The choice of 60 days ensures analysis conditions that are overall realistic (albeit simplified) but strictly worse than any recorded median case, and close to the worst recorded 9th decile. Therefore if our analysis technique performs well during evaluation, we can be highly confident that it will perform as well or better in the real world. Figure 6 shows a simplified example of how time impacts the data available when analyzing vulnerabilities at disclosure.

Description	SQL injection vulnerability in register.php in GeniXCMS before 1.0.0 allows remote attackers to execute arbitrary SQL commands via the activation parameter.
Keyword set (in alphabetical order)	activation, before, commands, genixcms, in, parameter, php, register, remote, sql, the, to, via, vulnerability

TABLE II: Description and extracted keyword set for CVE-2016-10096, a vulnerability disclosed on 01/01/2017. The filtering list included all CPE URIs published between 01/01/2007 and 12/24/2016.

B. *Word filtering*

All available CPE URIs (considering the metadata publication delay) are parsed as a keyword filter list. Fields extracted from the CPE URI are the software vendor, software product, and target software. After extraction all these fields are tokenized into individual words. When analyzing vulnerabilities descriptions, only words belonging to this filter list are considered, the others are discarded. This filtering is a trade-off: it vastly reduces analysis noise but may filter out some relevant information. Specifically a new vulnerability affecting a never-seen-before software product will not have any relevant CPE URI in available historical data therefore the name of the product will be filtered out. However, we argue this is the right trade-off in our context, as our goal is to feed keyword alerts to a security monitoring system: it is probably more relevant to output an alert because we did not find any highly relevant keyword than outputing an alert on a keyword that has never been seen before and is probably not monitored. This filtering gives us a set of keywords for every vulnerability. However this set is unordered and contains a lot of irrelevant keywords, as illustrated in Table II. As we can see the filtering list includes very common words such as "before" that are not related to the present vulnerability ("before" was added to the filtering list through vulnerability CVE-2011-5107 affecting the Wordpress plugin *Alert Before You Post*). It is clear that the mere presence of a keyword in a

Keyword list (after TF-IDF weighting)		Keyword list (after heuristics)	
Keyword	**Weight**	**Keyword**	**Weight**
genixcms	6.36	genixcms	12.71
activation	5.24	sql	5.46
register	3.40	activation	5.24
sql	2.73	register	3.40
commands	1.62	commands	1.62
parameter	1.22	parameter	1.22
php	1.19	php	1.19
vulnerability	0.57	vulnerability	0.57
before	0.56	before	0.56
the	0.21	the	0.21
in	0.18	in	0.18
remote	0.18	remote	0.18
via	0.12	via	0.12
to	0.01	to	0.01

TABLE III: Keywords order and weight for CVE-2016-10096, after TF-IDF weighting (left) and heuristics (right). TF-IDF corpus included all vulnerabilities disclosed between 01/01/2007 and 01/01/2017. Capitalization heuristic doubled the scores of "genixcms" and "sql" (spelled "GeniXCMS" and "SQL" in the description).

vulnerability description is not enough to assess its relevance for the given vulnerability.

C. TF-IDF Weighting

Instead of treating the presence of a keyword in a vulnerability as a binary event, we weight each keyword by the term frequency-inverse document frequency (TF-IDF) [20] 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 the set of keywords extracted from a CVE description as an individual document, and the set of these sets as a corpus. TF-IDF is formally defined in Equation 1:

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

where t is a word and d is a document belonging to a corpus of documents D. TF is based on the number of occurrence of the word in the document. Several formulas have been proposed, such as using the raw count directly or treating the presence or absence of a word as a boolean event. We chose to use the logarithmic version [21] of TF, defined in Equation 2, as it better reflects the diminishing returns of repeating the same term several times.

$$\text{TF}(t, d) = \log(1 + |t \in d|), \tag{2}$$

IDF is defined in Equation 3:

$$\text{IDF}(t, D) = \log \frac{|D|}{|d \in D : t \in d|}, \tag{3}$$

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, let us consider an actual software named *IBM Tivoli Service Request Manager*. The word "Tivoli" is much more specific than "Request", therefore its weight should be higher. Every keyword in the set is now weighted, which allows us to order them by relevance. The left side of Table III gives an example of such weighting.

D. Domain-specific Heuristics

A number of additional heuristics can be applied to the existing ordering to improve it even further. In this section we propose three of them which we detail below. We want to emphasize that an important part of our contribution is to give security experts the ability to formulate such kind of heuristics and reliably evaluate their accuracy. While these heuristics are very simple and domain-specific, we show in Section IV that each of them increases the accuracy of the analysis.

Multiple-words keywords. When a CPE URI field contains entries that are multiple words long, such as "linux kernel", this heuristic treats "linux", "kernel", and "linux kernel" as three different individual terms with individual TF-IDF values. Therefore this heuristic allows the existence of keywords that are actually multiple words long. A match on "linux kernel" is considered more relevant than two separate matches on "linux" and "kernel", so this heuristic also multiplies the score of a keyword linearly by the number of words it is made of.

Capitalized words. We observed that software names in vulnerability descriptions are often capitalized. This heuristic doubles the score of every keyword that is capitalized in the vulnerability description. The software industry has a somewhat peculiar grasp of English capitalization rules, so this heuristic is triggered for any capitalized letter in a word and not just the first one: "iPhone" or "openSUSE" are considered capitalized.

Words starting by "lib-". We empirically observed that words starting by "lib-" that are not "library" are rare in English but are commonly used as software names (*libxml2*, *libssh*, *libpng*, etc.). This heuristic doubles the score of every keyword starting by "lib-" that is not "library".

The right side of Table III gives an example of how applying all three heuristics alter the weights and order of keywords. We evaluate these heuristics in Section IV.

E. Keyword List Truncation

The list of keywords extracted from a vulnerability can be long. Assuming the analysis did a good job at sorting the relevant keywords first, the end of the list is empty of meaningful information. It is therefore desirable to truncate the list and only keep the beginning, as this operation retains most of the information while removing most of the noise. However it is not straightforward to decide where to cut. Keyword lists lengths vary greatly (from 0 to 196 keywords in our evaluation dataset) and the amount of relevant keywords inside them too. This makes static truncation threshold not appropriate, as it could remove too much information or retain too much noise. Instead we propose a dynamic truncation scheme based on the euclidean norm. At the truncation step of the pipeline, we view the untruncated weighted keyword list as an euclidean vector and we compute its norm. We then compute a truncation

Untruncated keyword list		Truncated keyword list (target norm = 95%)	
Keyword	**Weight**	**Keyword**	**Weight**
genixcms	12.71	genixcms	12.71
sql	5.46	sql	5.46
activation	5.24	activation	5.24
register	3.40		
commands	1.62		
parameter	1.22		
php	1.19		
vulnerability	0.57		
before	0.56		
the	0.21		
in	0.18		
remote	0.18		
via	0.12		
to	0.01		
Norm		Norm	
15.38		14.79	

TABLE IV: Untruncated and truncated weighted keyword lists for CVE-2016-10096, with a target norm of 95% for the truncated list.

budget by defining a *target for the truncated norm*, such as staying above 95% of the untruncated norm. Because most of the norm of the vector comes from the most relevant keywords, it is possible to cut out most irrelevant keywords while staying under budget. Table IV gives a practical example of such a truncation. We evaluate experimentally the impact of keyword list truncation in Section IV-D.

IV. Evaluation

A. Experimental Setup

We analyzed all 31156 CVE vulnerabilities disclosed between January 1st, 2017 and January 1st, 2019, using all 57640 CVE vulnerabilities disclosed between January 1st, 2007 and December 31st, 2016 as past historical data. This experimental setup simulates the behavior of a production system put online on January 1st, 2017, initially fed with historical information from ten years before, which then monitored all newly disclosed vulnerabilities continuously for the next two years. Each vulnerability was analyzed using the data available on its disclosure day only, as described in Figure 6. As discussed in Section III-A we chose a metadata publication delay of sixty days. Any non-zero metadata publication delay implies the actual metadata of a vulnerability including its CPE URIs is not available during analysis. We can therefore use the CPE URIs from the vulnerability metadata as a ground truth for evaluation. We propose two metrics to evaluate the quality of the ordered list of keywords: the position of the first relevant keyword, and the number of keywords necessary to reconstruct the software name. We evaluate our solution using the first metric in Section IV-B and using the second in Section IV-C.

B. Position of the First Relevant Keyword

As our goal is to identify the software affected by a vulnerability, we formally define a *relevant* keyword as a substring of any software name or software vendor fields present in the CPE URIs included in this vulnerability metadata. As

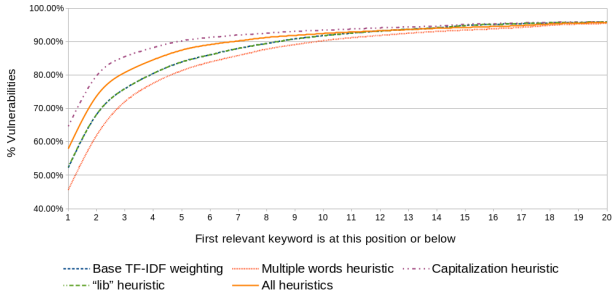

Fig. 7: First relevant keyword position.

our analysis gives us a sorted list of keywords, we can expect relevant keywords to be placed at the beginning of the list before irrelevant ones. Therefore the position of the first relevant keyword is directly tied to the usability of the results. It should be emphasized that the number of CPE URIs included in a vulnerability metadata can vary greatly, as well as the reason these CPE URIs were included in the first place. Usually at least one CPE URI is included as a machine-readable summary of the affected software described in the text description of the vulnerability. However security analysts sometimes include additional CPE URIs for software absent from the text description (but still relevant for the vulnerability) to provide context after the fact. This metric sidesteps the problem of choosing the most appropriate CPE URI as an evaluation ground truth and instead focuses on extracting the relevant information actually present in the vulnerability text description. The experimental results for this metric are described in Figure 7, for the base TF-IDF weighting, each heuristic described in Section III-D, and all heuristics combined. We can see that in all cases at least 70% of vulnerabilities have a relevant keyword in the top three keywords of their ordered list, at least 80% of vulnerabilities have a relevant keyword in their top five keywords, and 90% of vulnerabilities have a relevant keyword in the top ten. How to interpret these scores? In a control trial we randomly sampled 200 vulnerabilites and asked a security expert to guess the software product(s) affected by a vulnerability from the top three keywords without reading the vulnerability description. He gave 161 (80%) correct answers, which is very close to our metric's (81%) in the same configuration (all heuristics combined). 3% of the vulnerabilities have no relevant keyword at all. For these vulnerabilities there is not a single common word between the description and the CPE URIs of the vulnerability, creating a plateau for the metric. We randomly sampled 20 of these vulnerabilities to find out the reason for the absence of keyword matching. In 18 cases the vulnerability disclosure is about a software that has never been seen before at the time. In the two other cases, the software was seen only one day and four days before, a time period within the metadata delay we analyzed in Section III-A. In all cases this leads to the CPE index not being populated with the proper software name, which is filtered out at the keyword extraction stage. Examples of such vulnerabilities are CVE-2016-1132

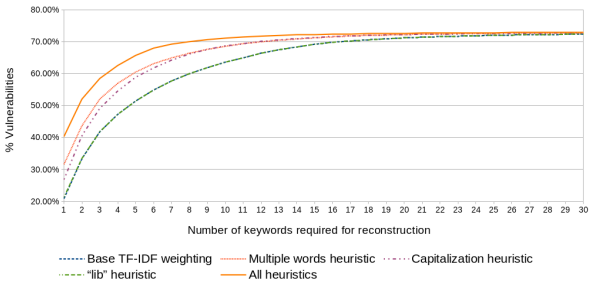

Fig. 8: Number of keywords necessary for name reconstruction.

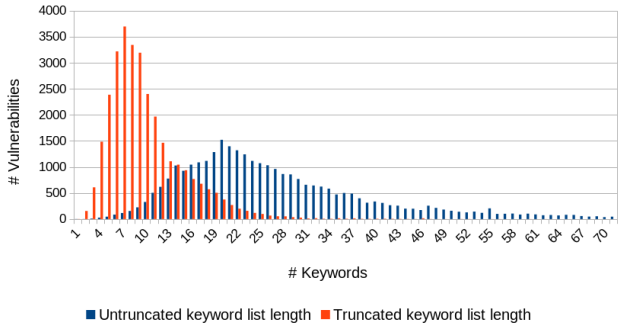

Fig. 9: Distribution of keyword list lengths before and after truncation with a target of preserving at least 95% of the norm.

(first vulnerability disclosed for the *Shoplat* iOS application) and CVE-2016-1198 (*Photopt* Android application). Regarding individual heuristics, we can see that our *capitalization* heuristic brings a substantial accuracy increase compared to the base TF-IDF weighting. The number of vulnerabilities with a relevant keyword at position 3 or below is increased from 76% to 86%. The *multiple-words* heuristic decreases the accuracy under this metric. The reason is that multiple-words keywords are aggressively pushed to the beginning of the list most of the time in front of single word keywords. When the software and vendor names are only one word long, they might lose one rank because of an irrelevant multiple-words keyword. However the reconstruction metric sheds a different light on this heuristic's accuracy, as discussed in the next section. The *lib* heuristic, while being strictly superior to the base TF-IDF weighting, provides such insignificant gains that it probably doesn't justify its maintenance cost. All heuristics combined provide a measurable improvement over the base TF-IDF weighting without heuristics.

C. Number of Keywords Necessary for Software Name Reconstruction

Our second metric is about measuring our ability to fully reconstruct a software name using the smallest amount of keywords. Formally, this means finding a permutation of a subset of keywords equal to a full software name string from a CPE URI of the vulnerability, then measuring the highest keyword position in this group. As an intuitive example, if we want to reconstruct the software name "linux kernel" and our keyword list is, in order, "kernel", "overflow", "linux", and "buffer", we can reconstruct the software name using the first 3 keywords (disregarding "overflow"). This metric is strictly more difficult than the previous one, as we now want to reconstruct full strings instead of substrings, and we are focusing on the software name only and not the software vendor. However it is also more indicative of real world usefulness, as reconstructing a complete software name provides more useful information than finding a substring of it. The experimental results for this metric are described in Figure 8. As expected reconstructing a full software name is more difficult than finding a relevant keyword. Using the base TF-IDF weighting, a software name can be reconstructed using the first three keywords only 42% of the time. 27% of the vulnerabilities do not have enough keywords in their description to reconstruct a software name at all. This leads to a lower plateau for the metric compared to *first relevant keyword position*. We sampled 20 of these vulnerabilities at random to investigate the cause of reconstruction failure. In 14 cases the affected software has never been seen before, leading to the same problem as described in Section IV-B. In the six other cases the software name is worded differently in the vulnerability description and the associated CPE URIs. As an example, CVE-2017-3814's description describes a vulnerability affecting the software *Cisco Firepower System Software* while the associated CPE URIs are referencing *Cisco Firepower Management Center*. One of these 6 cases, while technically a wording problem, can be attributed to excessive strictness in our parsing logic. Regarding individual heuristics, the multiple-words heuristic now provides the biggest improvement. This makes sense, as having multiple-words keywords provides opportunities to drastically shorten reconstruction of multiple-words software name. For instance, in a single word setup, the software "linux kernel" takes at least two keywords to be reconstructed ("linux" and "kernel"), while it can be fully reconstructed in a single multiple-words keyword ("linux kernel"). The capitalization heuristic again brings a substantial improvement under this metric. This time again the *lib* heuristic brings a very small improvement such that its maintenance cost is probably not justified. All heuristics combined together yield the best accuracy of all configurations. Using this setup we can reconstruct the full name of an affected software in 9 keywords or less for 71% of the vulnerabilities in the evaluation dataset.

D. Keyword List Truncation Evaluation

In this section we evaluate two effects of the truncation step: the keyword list length reduction ratio and a possible accuracy loss due to excessive truncation. All our evaluations were done with a truncation target of preserving at least 95% of the original norm. Figure 9 shows the distribution of keyword list lengths before and after truncation. The median untruncated keyword list length is between 23 and 24 words long, while the median truncated keyword list length is between 8 and 9 words long. The average reduction ratio is 3.22. We can conclude that keyword truncation has a substantial effect on keyword list length and is particularly effective at bringing keyword lists to sizes more easily readable by humans. Does

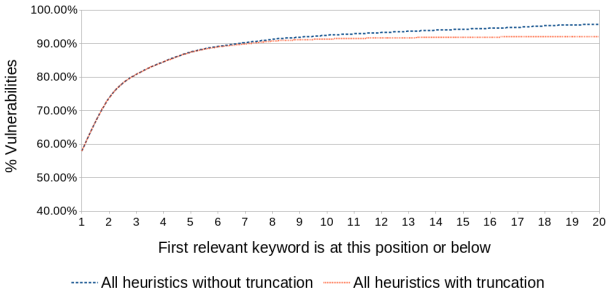

Fig. 10: Impact of keyword truncation on first relevant keyword position. (Truncated norm target = 95%)

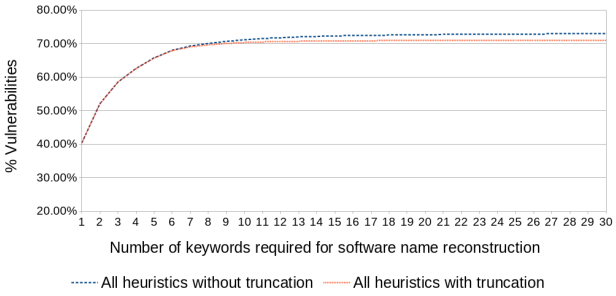

Fig. 11: Impact of keyword truncation on number of keywords necessary for name reconstruction. (Truncated norm target = 95%)

this reduction have an impact on the keyword list accuracy? Figure 10 and 11 show the impact of truncation on the two accuracy metrics studied before. We can see that while the effects of truncation are negligible on most vulnerabilities, the relevant keywords of a few hard-to-analyze vulnerabilities are lost during keyword truncation. 1446 vulnerabilities (4.64%) went from having a low ranking first relevant keyword to having no relevant keyword at all. 649 vulnerabilities (2.08%) had a software name that could be reconstructed before the truncation (albeit with difficulty) but not after. The proper trade-off between truncation and accuracy probably depends on the nature of the keyword consumers downstream. Humans might prefer shortened keyword lists, as reading a 23 word long keyword list is probably less convenient than reading the actual vulnerability description. Meanwhile machine monitoring systems might or might not prefer untruncated lists, depending on their ability to properly handle keyword noise and detect weak signals in low-ranked keywords. In either case it is not straightforward to make good use of a relevant keyword at rank #20 or #25 when all preceding keywords have been irrelevant, which makes a good case for truncation.

E. Performance of the Analysis Pipeline

While performance was not a major concern for us at this stage, analyzing a day worth of vulnerability historical data takes under a second on a commodity laptop with 16 Gb of RAM and an Intel Core i7-7600U CPU @ 2.80GHz, making the pipeline suitable for near real-time analysis at disclosure. Indexing ten years of historical data, which would be a one-time operation on a production system, takes between 5 and 30 seconds on the same hardware, depending on the heuristics used. The multiple-words heuristic creates more keywords to index, increasing the time of indexing when this heuristic is activated. This fast turnaround time enables a security expert to easily formulate a new heuristic hypothesis, quickly reindex the full historical dataset using the new heuristic and get a prompt evaluation of how this heuristic increases or decreases the accuracy of the analysis. All the code and data used for our experiment are available at [22].

V. Conclusion

We introduced a method to automatically extract from a CVE vulnerability the most relevant keywords with regard to the affected software, relying only on its human readable description. Our results are promising, as a simple technique brings results that are accurate enough to be useful in the real world. As discussed in Section I, our keyword extraction technique is the first step toward automated reaction to new vulnerabilities disclosures.

In future work we intend to use this keyword extraction technique to build a complete threat analysis system at disclosure. The goal is to assess automatically at disclosure time how much a vulnerability is a threat for a given information system. We can consider the final weighted keyword list (truncated or not) as an euclidean vector, which makes euclidean distance between two analyses an interesting similarity metric between two vulnerabilities. This would help assessing the threat resulting from a new vulnerability by comparing it to older, annotated vulnerabilities, providing an immediate automated risk analysis mechanism for one-day vulnerabilities.

The automated risk analysis and reaction mechanisms made possible by our technique could become invaluable tools for security engineers defending cloud infrastructure and information systems against day-to-day threats.

Acknowledgment

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

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 (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] Cloudflare - Inside Shellshock: How hackers are using it to exploit systems. https://blog.cloudflare.com/inside-shellshock/.
[9] AWS WAF - Web Application Firewall. https://aws.amazon.com/waf/.
[10] Google Cloud Armor. https://cloud.google.com/armor/.
[11] Cloudflare Web Application Firewall. https://www.cloudflare.com/waf/.

[12] Cloudflare - Stopping SharePoint's CVE-2019-0604. https://blog.cloudflare.com/stopping-cve-2019-0604/.
[13] 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 (LSAD '06)*. New York, NY, USA: ACM, 2006, pp. 131–138.
[14] 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 (ACSAC '10)*. New York, NY, USA: ACM, 2010, pp. 251–260.
[15] 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 (i-KNOW '15)*. New York, NY, USA: ACM, 2015.
[16] J. Jacobs, S. Romanosky, B. Edwards, M. Roytman, and I. Adjerid, "Exploit Prediction Scoring System (EPSS)," in *Black Hat 2019*, 2019. [Online]. Available: http://i.blackhat.com/USA-19/Thursday/us-19-Roytman-Predictive-Vulnerability-Scoring-System-wp.pdf
[17] A. Dulaunoy. (2016) The Myth of Software and Hardware Vulnerability Management. https://www.foo.be/2016/05/The_Myth_of_Vulnerability_Management/.
[18] L. A. B. Sanguino and R. Uetz, "Software Vulnerability Analysis Using CPE and CVE," *CoRR*, vol. abs/1705.05347, 2017.
[19] NIST IR 7695 — Common Platform Enumeration: Naming Specification Version 2.3. http://csrc.nist.gov/publications/nistir/ir7695/NISTIR-7695-CPE-Naming.pdf.
[20] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, pp. 11–21, 1972.
[21] Term Frequency - Inverse Document Frequency statistics. https://jmotif.github.io/sax-vsm_site/morea/algorithm/TFIDF.html.
[22] Firres. https://gitlab.inria.fr/celbaz/firres_noms.