Toward Semantic Assessment of Vulnerability Severity: A Text Mining Approach

Yongjae Lee and Seungwon Shin
Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
{ylee.cs, claude}@kaist.ac.kr

Abstract

A security vulnerability is a flaw in software or hardware systems that an adversary could exploit to compromise resources. Despite the never ending effort to reduce and prevent the vulnerabilities, its number has been constantly increasing until today. To deal with the vulnerabilities that are increasingly found in diverse systems, various methods to prioritize and manage the vulnerabilities have been proposed. The de facto standard method used to assess and prioritize the vulnerability based on severity is using CVSS (Common Vulnerability Scoring System), and many organizations have been using this system for vulnerability management. However, CVSS is limited in that it only takes some properties (e.g., ease of exploit, impact, etc.) of a vulnerability into account when measuring severity, and hence, CVSS scores are often considered inaccurate or impractical. In this paper, we present a semantic approach to assess the severity of vulnerabilities by ranking them. Our ranking method uses the relational information of how strongly two vulnerabilities are related or similar to each other. With this ranking method, we try to find which vulnerability has more common characteristics than others, since we believe that if a vulnerability has more common and popularly used characteristics, then the vulnerability is likely to attract more attack trials. Based on this insight, we evaluate our ranking method with real vulnerability data and show that our method can sift out more critical vulnerabilities effectively.

1 Introduction

A vulnerability is a flaw which exists in either hardware or software systems and can be used to threaten the systems [ALRL04]. A vulnerability itself is not a problem unless an adversary exploits it for the purpose of making the systems fail in terms of security. In other words, the vulnerability can be used by malicious people to violate the systems' important security properties such as Confidentiality, Integrity, and Availability (CIA) [ALRL04]. Therefore, swiftly finding and patching the vulnerabilities is one of the most significant concerns for hardware or software manufacturers, security software vendors, and researchers.

Unfortunately, it is so labor intensive and time consuming to fix the ever increasing vulnerabilities, and thus people want to prioritize vulnerabilities to know much more critical ones. For example, we can decide to fix remotely exploitable vulnerabilities prior to locally exploitable ones because the former can be easily exploited by most attackers. To this end, the vulnerabilities are managed in a systematic manner where they are given a unique ID and stored in a central database, NVD (National Vulnerability Database) [NIS17], and their severity is assessed by a severity assessment system.

Common Vulnerabilities and Exposures (CVE) [MSR06, MIT18] is the most popular vulnerability management scheme, which is operated by NVD. If someone finds and asks to register a newly discovered vulnerability, NVD issues a unique ID (CVE identifier) to the vulnerability. Once the vulnerability is registered to NVD, one can check its related information by the issued CVE identifier, which includes a short description, references, a list of affected products, and a severity score. In particular, the severity score is evaluated by Common Vulnerability Scoring System (CVSS) [FIR17], the de facto standard to quantify a vulnerability's severity. With CVSS score, one can sort vulnerabilities from highest to lowest, which helps prioritize the vulnerabilities to fix.

However, many researchers have argued that the CVSS does not account for what security experts perceive in the wild [AM12, AM14]. For example, a vulnerability with a low CVSS score is ranked in a higher position in bug bounty programs [MM16], and the CVSS scores of randomly selected vulnerabilities are not correlated well with severity scores manually evaluated by experts in the security field [HA15]. More specifically, let's take Heartbleed as an example. Heartbleed (CVE-2014-0160) is one of the most well-known vulnerabilities that received worldwide attention lately. It is an implementation flaw of OpenSSL, the most used open source encryption library and TLS implementation [Ope18]. This vulnerability can make servers leak confidential data including the encryption key of the servers, which makes the problem much worse, but its CVSS score is just 5.0 out of 10.0 with Medium severity level.

In this paper, we present a semantic approach to assess the severity of vulnerabilities (specifically CVEs) by analyzing descriptions for a CVE. There are many text descriptions for a CVE, such as NVD entries, security blog posts, and manufacturers' web bulletins, and such text descriptions explain how to exploit the CVE and what kind of damage can be caused if the CVE is exploited by attackers. Since those descriptions commonly present various characteristics about the CVE in human readable natural language, we can glean insightful information from them with the help of Natural Language Processing (NLP) techniques.

To this end, we first collect text descriptions illustrating the characteristics of CVEs from various sources: NVD, blogs, and web bulletins. Next, we extract information from the text descriptions, which includes the type of product where a CVE is found, which version of the product that has the CVE, whether there exists an easy-to-use exploit for the CVE, and so forth. Once such information is extracted, then we apply a ranking method to understand the severity of CVEs clearly. Based on the extracted information, our ranking method first tracks how strongly CVEs are related or similar to one another. This relation can reveal whether characteristics of a CVE are also shared by other CVEs or not. Finally, our ranking method sorts CVEs in order, i.e., a CVE with more common characteristics will be ranked higher. The intuition behind our ranking method is that if characteristics of a CVE are more general, which means that they could be commonly/widely adopted by attackers, then the CVE is more serious. To get the rank of each CVE, we employ the TextRank algorithm [MT04], and it is an unsupervised ranking algorithm that can summarize and extract important sentences or words within a text.

To initially evaluate our proposed ranking approach, we have collected real CVE data and apply our method to the data. In addition, we compare our ranking results with CVSS score to understand whether our approach can clearly reflect real world opinions. Our initial results show that our approach provides much more realistic (and reasonable) ranking results than CVSS.

2 Method

Before we give the detailed explanation on our vulnerability ranking method, we illustrate our system overview in Figure 1. Our method operates in three phases: (1) corpus building, (2) graph building, and (3) vulnerability ranking. Our method is a text-oriented vulnerability ranking, and thus we need a lot of text descriptions about CVEs. Fortunately, NVD compiles related information in a database and allows to access the information freely, and we glean CVE descriptions from NVD to build CVE description corpus. After building the corpus, our method generates a vulnerability ranking graph where a vertex represents a CVE. In the graph, vertices are linked to one another when there is a certain relation between two CVEs. We will discuss the relation that the CVEs can have in the following sections. Once we completed the graph building, we run TextRank algorithm on the graph to obtain importance scores of the CVEs, by which the CVEs are sorted.

2.1 Vulnerability ranking

Ranking model Our vulnerability ranking method is based on TextRank [MT04], which is a graph-based and unsupervised ranking model. TextRank summarizes a text by ranking sentences in the text according to their importance and singling out a set of higher ranked sentences. In the model, a sentence is represented as a vertex, and two sentences are linked to each other if they share similar contents, or words. Although TextRank is an application of Google's PageRank [BP98] to text summarization, the two ranking methods are different in that TextRank operates on an undirected graph. This is because, unlike web pages, sentences do not have explicit reference relations. Therefore, TextRank cannot use the graph structure information which denotes that a node votes another one. Instead, TextRank assigns a similarity weight on each link between two nodes, and they exchange the weight when calculating the importance score.

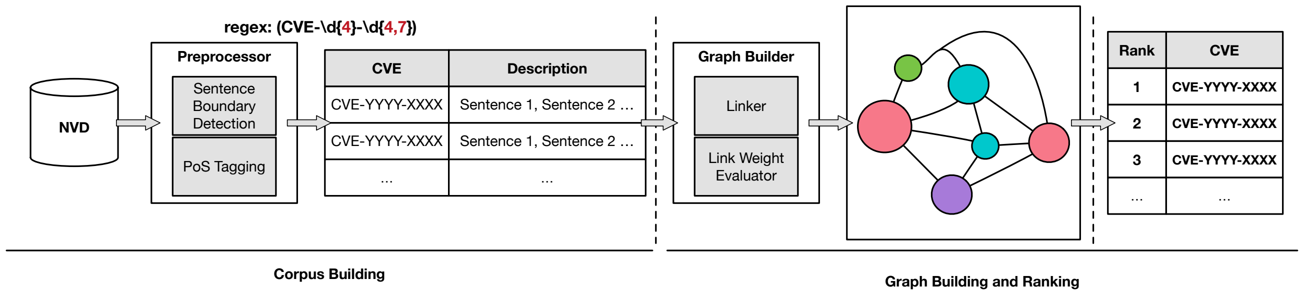

Figure 1: Vulnerability ranking system overview

In the ranking graph G=(V, E), where V is the set of vertices and E is the set of edges, let's assume that there is a vertex V_i. For V_i, let $In(V_i)$ and $Out(V_i)$ be the set of predecessors of V_i and the set of successors of V_i, respectively. In addition, if there is a vertex V_j that belongs to $In(V_i)$, the similarity weight between V_i and V_j is defined as w_{ji}. Then, the importance score of the vertex V_i can be computed as below:

$$IS(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} IS(V_j) \tag{1}$$

where d is a damping factor, which denotes the probability (1 - d) for a random surfer on the graph to jump from a vertex to another one randomly [BP98]. In this model, the importance score of a vertex is distributed to its successors proportionally to the similarity weight. Therefore, a vertex that is similar to majority of other vertices within the graph tends to have a higher importance score.

In our vulnerability ranking problem, we believe that a vulnerability that has similar properties with all kinds of vulnerabilities is important and thus needs to be fixed first. This is because, if such a vulnerability is found in a hardware or software product, it means that the vulnerability makes the product have broad attack surfaces. In other words, the product can be attacked in various ways. Here, for two vulnerabilities to have similar properties means that they can be used by similar types of attacks or violate one of the CIA triads alike.

How to represent a vulnerability in a graph? In our ranking graph, a vertex represents the short description of a CVE, which is less than 10 sentences and recorded for every CVE in NVD. For example, a vertex labeled with CVE-2014-0160 represents the text description presented in Figure 2 which is excerpted from NVD [1]. From content words in the description, we can grasp how the vulnerability can be exploited by attackers and what kind of damages can be caused after the vulnerability is successfully exploited. In other words, in the CVE description, the characteristics of the CVE are expressed in natural language, and the presence of common characteristics between two vertices determines whether they are linked together or not. Notice that the text description cannot be used directly to be drawn as a vertex and needs to pass the predefined preprocessing steps to be converted to a bag-of-words such as part-of-speech tagging, lemmatization, and so forth.

[1]http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160

The (1) TLS and (2) DTLS implementations in OpenSSL 1.0.1 before 1.0.1g do not properly handle Heartbeat Extension packets, which allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read, as demonstrated by reading private keys, related to d1_both.c and t1_lib.c, aka the Heartbleed bug.

Figure 2: Description of CVE-2014-0160 (Heartbleed)

How to define similarity between two vulnerabilities? For two CVEs to share similar characteristics can be defined as having similar words in both of their bags-of-words simultaneously. If the two bags-of-words describing the two CVEs have similar words, we compute the similarity between them by employing text similarity measures such as Jaccard index or TF-IDF cosine similarity [BYRN99]. In this work, we employ Jaccard index [Pau12] as presented in Equation 2, where X and Y are the sets of unique words that constitute each bag-of-words of the two vulnerability descriptions.

$$JaccIndex(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|} \tag{2}$$

Using this metric, we can measure how similar two CVEs are. For instance, we present three CVEs and their descriptions in Figure 3 and summarize their similarities in Table 1. Since both CVE-2015-0311 and CVE-2015-7645 are vulnerabilities of Adobe Flash Player and affect the same operating systems (i.e., MS

Unspecified vulnerability in Adobe Flash Player through 13.0.0.262 and 14.x, 15.x, and 16.x through 16.0.0.287 on Windows and OS X and through 11.2.202.438 on Linux allows remote attackers to execute arbitrary code via unknown vectors, as exploited in the wild in January 2015.

(a) CVE-2015-0311

Adobe Flash Player 18.x through 18.0.0.252 and 19.x through 19.0.0.207 on Windows and OS X and 11.x through 11.2.202.535 on Linux allows remote attackers to execute arbitrary code via a crafted SWF file, as exploited in the wild in October 2015.

(b) CVE-2015-7645

Unspecified vulnerability in Oracle MySQL Server 5.6.23 and earlier allows remote authenticated users to affect availability via unknown vectors related to Server : Security : Privileges.

(c) CVE-2015-2567

Figure 3: Three random CVEs registered in 2015 and their NVD descriptions.

Windows and Apple OS X), they are the most similar vulnerability pair among the three vulnerability pairs. Next, CVE-2015-0311 and CVE-2015-2567 are similar to each other because they have the same property that remote attackers can exploit the vulnerabilities via unknown vectors. In consequence, CVE-2015-0311 has various factors to be exploited by attackers such as Adobe Flash Player, OS, and unknown attack vectors, and we can conclude that it should be handled earlier than others.

3 Evaluation

To evaluate our ranking method, we randomly selected 100 CVEs from NVD, which were registered in 2014, and then constructed a small corpus consisting of the 100 CVE descriptions. The descriptions are converted to bags-of-words, and the conversion requires preprocessing that normally consists of sentence boundary detection, stop word removal, and lemmatization. After that, we run TextRank algorithm on the CVE description corpus and obtain the rank of the 100 CVEs. In Table 2, we present the CVEs ranked in the top 10 out of 100. Due to the page limitation, we could not specify the whole description about each of the CVEs but present some keywords in the table.

CVE pair	Jaccard index
CVE-2015-0311 & CVE-2015-7645	0.6182
CVE-2015-0311 & CVE-2015-2567	0.2223
CVE-2015-7645 & CVE-2015-2567	0.1132

Table 1: Pairs of the three CVEs registered in 2015 and their Jaccard index

CVE ID	CVSS	Keyword
CVE-2014-5694	5.4	Android, X.509 certificate, SSL, MITM
CVE-2014-3169	7.5	Use-after-free, DOM, Google Chrome, DoS, remote attack
CVE-2014-6707	5.4	7Sage LSAT Prep - Proctor, Android, X.509 certificates, SSL, MITM, spoof
CVE-2014-1741	7.5	Integer overflows, Blink, Google Chrome, remote attackers, DoS
CVE-2014-1316	5.0	Heimdal, Apple OS X, remote attackers, DoS, Kerberos 5
CVE-2014-2536	4.3	Multiple directory traversal, McAfee, remote authenticated users
CVE-2014-2279	6.4	Multiple directory traversal, SeedDMS, remote authenticated users, read arbitrary files, .. (dot dot) in the logname parameter
CVE-2014-5836	5.4	GittiGidiyor, Android, X.509 certificates, SSL, MITM, spoof
CVE-2014-0885	6.8	CSRF, Admin Web UI, IBM Lotus Protector for Mail Security, remote authenticated users, unknown vectors
CVE-2014-5780	5.4	Bouncy Bill, Android, X.509 certificates, SSL, MITM, spoof

Table 2: Top 10 CVEs generated by our ranking method and their CVSS scores and keywords. CVSS score ranges from 0.0 (not severe) to 10.0 (the most severe) and it is increased by 0.1.

Taking a look at Table 2, we know that most of the CVEs are related to an X.509 public key certificate problem and can cause some remote attacks such as Man-In-The-Middle (MITM) attacks and Denial of Service (DoS) attacks. In addition, the CVEs are reported to be found in widely used software products including Android, Google Chrome, and Apple OS X. In summary, our ranking method ranks CVEs higher, which (1) are related to a security hole (e.g., certificate verification bypass), (2) are found in popularly used products (e.g., Android) , and (3) can cause well-known types of attacks (e.g., MITM and DoS).

Note that, while the CVSS score for CVE-2014-5694 is lower than that for CVE-2014-3169, CVE-2014-5694 is located at a higher position than CVE-2014-3169 by our ranking method. This is because the keywords contained in the description of CVE-2014-5694 (i.e., Android, X.509, SSL, etc.) are commonly found in other CVEs' descriptions more frequently than those of CVE-2014-3169 (i.e., DOM, Google Chrome, DoS), which has an impact on the weights of the links to which the CVE is connected.

4 Discussion

Although our ranking method gives a new rank of the vulnerabilities, reflecting other facets that were never used for assessing the vulnerabilities' severity, the ranking method has issues to be addressed further. First, using NVD descriptions, we make nodes of the vulnerability ranking graph. However, the descriptions are a short text and provide limited information. On the contrary, there are useful sources from which we can glean detailed information about vulnerabilities. For example, one can search Microsoft Security Bulletins [Mic18] for web documents explaining about newly discovered or patched vulnerabilities in their own words, and many researchers and practitioners post security-related information on their own blogs [Fee18]. In addition, exploits, which are a set of commands to infringe a system using a vulnerability, are archived with related information in a database [Sec18], and thus we can collect another type of information that explains how to use the vulnerability from the viewpoint of practitioners.

Furthermore, to measure the similarity of given two vulnerabilities, we use Jaccard index based on their CVE descriptions. However, we can extend our similarity measure, considering not only such topological semantics but also distributional semantics such as word embeddings. In addition, it is not sufficient to measure two vulnerabilities' similarity only considering the textual descriptions because there are other factors to determine whether given two vulnerabilities are similar to each other, which may not be expressed in the descriptions. For instance, there are bug bounty programs that are operated by many organizations such as Google, Mozilla, and Facebook, and they give a bounty for a bug or a vulnerability to the bug discoverer. Then, we can assume that two vulnerabilities are similar if their bounties are set in a similar level.

5 Conclusion

In this work, we present a semantic way to assess vulnerabilities by examining their textual descriptions from which we can grasp characteristics of the vulnerabilities. We then build the vulnerability ranking graph by representing each vulnerability's characteristics as a node, which are expressed in natural language, and run the TextRank algorithm on the graph to obtain the rank of the vulnerabilities. As our future work, we will address the issues discussed in Section 4 to improve the performance of our ranking method to the degree to which security experts and practitioners can agree with our ranking result. To this end, we are going to carry out expert-based performance evaluation for our ranking method, inspired by the existing research work [HA15].

Acknowledgment

This research is (in part) based on the work supported by Samsung Research, Samsung Electronics.

References

[ALRL04] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan 2004.

[AM12] Luca Allodi and Fabio Massacci. A preliminary analysis of vulnerability scores for attacks in wild: The ekits and sym datasets. In *Proceedings of the 2012 ACM Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, BADGERS '12, pages 17–24, New York, NY, USA, 2012. ACM.

[AM14] Luca Allodi and Fabio Massacci. Comparing vulnerability severity and exploits using case-control studies. *ACM Trans. Inf. Syst. Secur.*, 17(1):1:1–1:20, August 2014.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International Conference on World Wide Web 7*, WWW7, pages 107–117, Amsterdam, The Netherlands, The Netherlands, 1998. Elsevier Science Publishers B. V.

[BYRN99] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Fee18] Feedspot. Top 100 information security blogs for data security professionals. `http://blog.feedspot.com/information_security_blogs`, 2018.

[FIR17] FIRST. Common vulnerability scoring system. `https://www.first.org/cvss`, 2017.

[HA15] Hannes Holm and Khalid Khan Afridi. An expert-based investigation of the common vulnerability scoring system. *Computers & Security*, 53:18 – 30, 2015.

[Mic18] Microsoft. Microsoft security bulletins. `https://technet.microsoft.com/en-us/security/bulletins.aspx`, 2018.

[MIT18] MITRE. Common vulnerabilities and exposures. `https://cve.mitre.org/`, 2018.

[MM16] Nuthan Munaiah and Andrew Meneely. Vulnerability severity scoring and bounties: Why the disconnect? In *Proceedings of the 2nd International Workshop on Software Analytics*, SWAN 2016, pages 8–14, New York, NY, USA, 2016. ACM.

[MSR06] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security and Privacy*, 4(6):85–89, November 2006.

[MT04] Rada Mihalcea and Paul Tarau. TextRank: Bringing order into texts. In *Proceedings of the 2004 Conference on Empirical Methods Natural Language Processing*, EMNLP '04, 2004.

[NIS17] NIST. National vulnerability database. `https://nvd.nist.gov/`, 2017.

[Ope18] OpenSSL Software Foundation. Openssl. `https://www.openssl.org/`, 2018.

[Pau12] Jaccard Paul. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, 1912.

[Sec18] Offensive Security. The exploit database. `https://www.exploit-db.org/`, 2018.