A Recommender System for Tracking Vulnerabilities

PHILIP HUFF, University of Arkansas, USA
KYLIE MCCLANAHAN, University of Arkansas, USA
THAO LE, Bastazo, Inc., USA
QINGHUA LI, University of Arkansas, USA

Mitigating vulnerabilities in software requires first identifying the vulnerabilities with an organization's software assets. This seemingly trivial task involves maintaining vendor product vulnerability notification for a kludge of hardware and software packages from innumerable software publishers, coding projects, and third-party package managers. On the other hand, software vulnerability databases are often consistently reported and categorized in clean, standard formats and neatly tied to a common software product enumerator (i.e., CPE). Currently it is a heavy workload for cybersecurity analysts at organizations to match their hardware and software package inventory to target CPEs. This hinders organizations from getting notifications for new vulnerabilities, and identifying applicable vulnerabilities. In this paper, we present a recommender system to automatically identify a minimal candidate set of CPEs for software names to improve vulnerability identification and alerting accuracy. The recommender system uses a pipeline of natural language processing, fuzzy matching, and machine learning to significantly reduce the human effort needed for software product vulnerability matching.

CCS Concepts: • **Security and privacy** → **Vulnerability management**; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: software vulnerability, machine learning, natural-language processing

ACM Reference Format:
Philip Huff, Kylie McClanahan, Thao Le, and Qinghua Li. 2021. A Recommender System for Tracking Vulnerabilities. In *NG-SOC '21: Next Generation Security Operations Centers, August 17–20, 2021.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

An organization wishing to mitigate its software vulnerabilities has clear disadvantages against an adversary. No obvious tie exists between the installed software of the entity and the software publishers that maintain the vulnerability reporting and mitigation for a given piece of software. The installed software and hardware inventory observed by a consumer is referred to as *software inventory package name*. Organizations have tens of thousands of software packages represented by hundreds to thousands of software publishers. These software publishers vary significantly from some of the largest companies on earth to mostly abandoned software projects.

Most cybersecurity operation centers have a process to identify vulnerabilities through a combination of scanning, asset management, and assessments. However, the fastest and surest strategy in preventing vulnerabilities from being exploited is an immediate notification from the vendor. For some vendors, this may occur through a paid support contract. Prominent operating system vendors, such as Microsoft, Apple, and Android, regularly push new vulnerability patches,

which consumers then automatically deploy. However, many more software publishers rely on public notification of vulnerabilities through services such as the National Vulnerability Database (NVD).

In these cases, consumers can only act on public notifications by matching the vulnerable product to their hardware on software inventory. Timely analysis and remediation are critical in computing environments such as data centers, industrial control systems, and Internet of Things (IoT) systems.

The matching process is more complicated than it sounds. Hardware and software inventories often represent an amalgamation of different software publishers packaged up and deployed together. Then, the public vulnerability notifications use a standard naming convention which is well-formed but distinct from software package listings in the system inventory.

To further complicate the problem, software and hardware device inventory is a private data set. Organizations cannot simply post their software inventory for assistance in matching vulnerability reports. Revealing the deployed hardware and software names provides reconnaissance information adversaries might use to breach an organization, and revealing the data often violates regulatory requirements. Instead, cybersecurity analysts must painstakingly match their deployed software to public vulnerability reports.

Automatically matching enterprise software inventory with vulnerability reports frees up time for cybersecurity operations, but it also allows the convergence of the vulnerability data features with the asset and operational data features. Bringing these together paves the way for even more automation in vulnerability management and mitigation [12, 20].

Our approach solves the matching problem through a pipeline of natural language processing (NLP), fuzzy matching, and machine learning. A cybersecurity analyst using our technique can immediately obtain a shortlist of candidate matches to their software inventory or conclude the absence of any matching vulnerability identification source. While not perfect, the automation mimics the work of a human analyst to obtain a situational understanding of their environment and dramatically shortens the time to make a match decision. *To the best of our knowledge, this is the first work to solve the aforementioned matching problem.*

This paper is organized as follows. We present prior work in section 2 and then describe the vulnerability reporting process and software name matching problem in section 3 from the perspective of cybersecurity operations. We present our recommender solution in section 4. Finally, we present our implementation and test results in section 5 and then conclude this paper.

2 PRIOR WORK

[5] defines software vulnerabilities as specific flaws in a piece of software that allows attackers to do something malicious. The problem of determining whether the software is vulnerable is undecidable [7], and as [9, 21] observe, vulnerabilities have become a normal process of cybersecurity operations. Entities with the responsibility of protecting against software vulnerabilities face the challenge of timely identifying vulnerabilities through public vulnerability alert repositories.

Researchers have studied the problem of public vulnerability reporting in several contexts. [16] examines the time delay between the reporting of vulnerabilities and the reporting of Common Vulnerability Scoring Scheme (CVSS) attributes. Several works enhance publicly reported CVSS data using machine learning and NLP to obtain new features and improve vulnerability and patch management decisions. [20] uses the CVSS metrics to automatically recommend vulnerability remediation actions. [22] uses vulnerability features to predict the probability of exploit for vulnerabilities and optimize patch scheduling. [12] uses natural language processing to automatically localize vulnerability information

from online resources. In [10], a model is provided for NLP Named Entity Recognition (NER) over vulnerability descriptions to extract cause, consequence analysis, and impact ratings automatically. [19] demonstrates temporal difference over time in the vulnerability CVSS when used for machine learning. In [18], machine learning improves the accuracy of impact scoring, and [11] shows the drift over time of NLP models when applied to vulnerability descriptions. Similarly, [17] parses the CVE description to obtain vulnerability characteristics automatically, and [8] extracts network services from vulnerability descriptions using NLP to automatically perform threat analysis.

However, entities cannot realize the benefits of enhanced vulnerability intelligence without mapping the Common Vulnerabilities and Exposures (CVE) to the software inventory of an entity through a Common Product Enumerator (CPE). For this task, we look to recommender systems. The term *recommender system* originated with [15] as an approach used to filter through large datasets to present the most relevant and desired selection back to the user. [3] defines recommender systems as those guiding the user to interesting or useful objects in a large space of possible options. Although most recommender systems [2] focus on user experience, there has been recent work to apply the approach to cybersecurity [1, 6, 14]. To the best of our knowledge, this is the first approach to address the problem of matching raw software inventory to standardized software product names.

3 BACKGROUND

Currently, the largest open vulnerability database is maintained as the U.S. National Vulnerability Database (NVD) with over 1,500 vulnerability being reported monthly. Each of these vulnerabilities contains a section of products to which the vulnerabilty applies. The NVD relies on CVE Numbering Authorities (CNAs) to report vulnerabilities for products for which the CNAs have responsibility. As of this writing there are 167 CNAs representing 28 countries. A summary of the top ten CNAs reporting the most vulnerabilities in 2020 is provided in Table 1.

Table 1. Estimated Number of Vulnerabilities Reported in 2020 by CNA

CVE Numbering Authority	**Reported**
MITRE Corporation	3,669
Microsoft Corporation	1,082
Oracle	807
Cisco Systems, Inc.	436
Android (Google Inc. or Open Handset Alliance)	413
GitHub, Inc.	407
IBM Corporation	353
Adobe Systems Incorporated	308
Apple Inc.	272
ICS-CERT	225
Jenkins Project	209

When reporting vulnerabilities, CNAs identify the new vulnerability by a Common Vulnerability Enumerator (CVE). The CVE list is an open database sponsored by the United States Department of Homeland Security [1]. The CVE is commonly used by vulnerability databases, bug bounty programs, and exploit databases. The NVD uses the Common Product Enumeration (CPE) software product naming standard when reporting vulnerability product applicability.

[1] https://cve.mitre.org/cve/

The CPE is based off the Uniform Resource Identifier (URI) syntax and provides a set theoretic naming convention in which a source and target software product name may be structured and compared as sets [4]. The naming standard is most commonly used to define the set of software to which a vulnerability applies using the following keys:

- Part - Type of software. A value *a* indicates software, *h* indicates hardware, and *o* indicates operating system.
- Vendor - Name of the manufacturer or publisher of the software.
- Product - Name of the product.
- Version - Version of the product.
- Update - A major update of the product such as a service pack for Windows. Currently, this key is only specified in 10% of the products reported to the NVD.
- Edition - Used to specify different types of the same product. However, in practice, the edition is commonly specified in the product name.

The CPE URI reads from general to specific in the format cpe:2.3: <part> : <vendor> : <product> : <version> : <update> : <edition> using the key value indicated in the URI. A '*' character is used as a wildcard for the source CPE. When matching sources to target URIs, the wildcard defines a superset based on other key value pairs in the URI. For example *CPE:2.3:a:microsoft:*:*:** defines the superset of all Microsoft software.

Vulnerabilities in the NVD define software applicability using CPEs in a boolean expression. In most cases, this means a simple *OR* operation to define applicability to multiple products. However, certain CNAs use the *AND* operation when the vulnerability only applies to a certain platform of the product. For example, a vulnerability for Adobe Acrobat on Android as opposed to the Windows operating system, would use the boolean *AND* operation to distinguish the applicability. An example with AND and OR operations is the Wordpress vulnerability CVE-2020-10257 [2] that has 62 vulnerable configurations.

The intention of the NVD is for organizations to match their hardware and software inventory to CVEs and receive the applicable vulnerability notifications. However, in most cases, there is no clean way to do so. Table 2 provides a summary of CNAs who provide an online mapping between CVEs and software products and packages.

As shown, most Linux operating systems can automatically match to CVEs using package listings available through the operating system. Cisco, Intel, and Microsoft provide up to date mappings from update packages to the CVE. Microsoft is unique in providing a tool for automatically extracting applicable packages (i.e., KBs) for an operating system.

As far as we can tell, no other CNAs provide an automated software inventory to CVE listing. Therefore, organizations wanting to receive notifications of new vulnerabilities are left to manually map their remaining inventory to CPE source URIs. Alternatively, they may review all new vulnerabilities for applicability, but the mapping still must regularly occur to perform vulnerability patching and mitigation.

4 FUZZY MATCHING TECHNIQUE

This section presents a solution for automatically matching target CPEs against an enterprise hardware and software inventory. Doing so not only allows for automated vulnerability notifications but also enables organizations to combine data features between vulnerabilities and vulnerable assets to improve cybersecurity risk understanding and mitigation decisions.

[2] https://nvd.nist.gov/vuln/detail/CVE-2020-10257

Table 2. CVE Notifications in Practice

Hyperlinked CNA	Matching Approach
Cisco, Inc.	Each Cisco advisory is mapped to multiple CVEs, but product matching is not known to be automated
Debian	CVEs are listed by package name, which can be matched to Debian APT listing
Intel Corp.	Each Intel advisory is mapped to multiple CVEs, but product matching is not known to be automated
Microsoft Corp.	A spreadsheet can be downloaded from the source URI with Microsoft Knowledge Base (KB) mapping to CVEs. Product matching comes through their Windows Security Update Service (WSUS) tool which can be configured to output a KB listing.
RedHat, Inc.	CVEs are mapped to packages, and product listing can be matched from the output of RedHat Package Manager.
Suse	Package to CVE mappings can be scraped from the given URI, and packages are listed on the operating system through RedHat Package Manager.
Ubuntu	Package to CVE mappings can be scraped from the given URI, and packages can be matched through an APT listing on the operating system.

The approach to fuzzy matching approximates the process a security analyst would take in finding matching CPEs for their hardware and software inventory. A security analyst seeking to map their unknown software package inventory to a vulnerability source begins first by finding a suitable vendor name and filtering down further to find the product, but the search becomes messier when they cannot easily find a suitable source CPE for a specific product. If a vendor match is found, then the target CPE set might contain only the vendor, which is too coarse-grained. For large vendors, this might produce several hundred false positive CVE notifications each year.

We account for the complexity in decision making by determining candidate target CPEs through a three phase automation process. First, we extract the word vectors using the core english vocabulary from spaCy[3]. Then, based on the word vectors, we fuzzy match the software inventory package names to a set of CPEs using multiple metrics. Since many results might be returned in this step, finally, we use machine learning to order these CPEs and produce a small set (e.g., 5 to 10) of candidate target CPEs for a human to select.

4.1 Natural Language Processing

The inherent variation in natural language means that computers struggle to extract meaning in language. Irregular verb conjugations, for example, are difficult to detect using word stems. In our work, we find that software names, in particular, are often not dictionary words, and so many pre-trained or ready-to-use models have no context with which to recognize them.

Because of this, there is great value in representing words in some standardized, machine-readable format; Word2Vec is a recent but well-established method for this task [13]. Word2Vec is an algorithm which processes a text corpus and projects the corpus vocabulary as vectors in a multi-dimensional space. Word2Vec can be trained on any corpus and will create word vectors for all tokens within it. Using word vectors, many NLP tasks become programmatic in nature. Measures of vector similarity can predict synonyms or antonyms. Similarly, word vectors can be used to infer analogies. For example, if a direction is calculated between the vector for "man" and the vector for "king", that direction can be applied to the vector for "woman" to obtain "queen".

[3]https://spacy.io/

Word vectors are simply an array of the same size as the number of dimensions, often normalized in the range $[-1, 1]$ before storage or comparison. Once a Word2Vec model has been trained, only the word vectors must be stored, minimizing the storage space needed.

4.2 Fuzzy Matching

String similarity tests occur through multiple calculations. First, the similarity between the software package inventory and the CPE is measured using the cosine between two-word vectors A and B as shown in the following equation where equal vectors have a cosine angle of one, and maximally dissimilar vectors have a cosine of 0.

$$\cos\Theta = \frac{A \cdot B}{\|A\|\|B\|} \tag{1}$$

Software names commonly have multiple words and symbols. The comparison is normalized by first splitting the string by common stop words and then removing unnecessary symbols. Subsequently, calculating the cosine similarity occurs through the average of all distinct word vectors in the string. This approach allows for similar words to appear in a different order between two strings without impacting the similarity score.

However, the concatenation of vendor and product strings can have misleading cosine similarity scores. For instance, software products commonly get bought out by other companies, but the CPE name may remain the same, or product names can span several words and adversely impact the matching vendor. Consequently, the cosine similarity between the software package inventory name A, and the vendor B_{vendor}, and the cosine similarity between the software package inventory name A and the product $B_{product}$ are calculated separately.

Finally, software package inventory names may not have a word vector representation due to misspellings or other variance in the software naming. Likewise, the cosine similarity calculation returns 0 even when the words closely align. In these cases, the similarity gets measured as the cosine similarity of the strings using the letter count where c represents the letter, and $C(x)$ represents the count of the letter x.

$$\cos\Theta = \frac{\sum_{\forall c \in A \cap B} C(A.c) \times C(B.c)}{\sqrt{C(A.c)^2} \times \sqrt{C(B.c)^2}} \tag{2}$$

4.3 Machine Learning

Our matching solution uses machine learning as the final step in the pipeline to better distinguish the results of fuzzy matching. The machine learning model assists in ordering the results from fuzzy matching and presenting a smaller focused set to human operators. This approach improves the accuracy of the identified vulnerabilities by considering both fuzzy matching and historical vulnerability reporting for the software product.

The number of vulnerabilities between different vendors and different products varies greatly. For example, accidentally matching the vendor Goomeo instead of Google for a software package drastically affects the number of accurately identified vulnerabilities because Google has several vulnerabilities identified each month. Thus, using the string similarity in the previous fuzzy matching step does not produce effective enough results. That is why machine learning is further used. The data features for the machine learning model include:

(1) **Actual Type** (*categorical feature*) - The type can be either software or hardware.
(2) **Part** (*categorical feature*) - CPE part as either "a" for *application*, "o" for *operating system*, or "h" for *hardware*.
(3) **CNA Source** (*categorical feature*) - The CVE Naming Authority is most closely associated with the vendor and product. Although the CNA does not currently factor much in the prediction, there is a clear distinction in

product naming among CNAs. We leave this to future research to determine if this should have more weight in the model's output.

(4) **Word Vector Cosine Similarity** (*between [0, 1]*) - The cosine similarity between word vectors in the software inventory package name and the concatenated vendor and product from the CPE.
(5) **Vendor Word Vector Cosine Similarity** (*between [0, 1]*) - The cosine similarity between word vectors in the software inventory package name and only the CPE vendor.
(6) **Product Word Vector Cosine Similarity** (*between [0, 1]*) - The cosine similarity between word vectors in the software inventory package name and only the CPE product.
(7) **Vendor Word Character Cosine Similarity** (*between [0, 1]*) - The cosine similarity between the word characters in the software inventory package name and the vendor.
(8) **Product Word Character Cosine Similarity** (*between [0, 1]*) - The cosine similarity between the word characters in the software inventory package name and the product.
(9) **Vendor Product Count** - The number of software products per vendor reported in the CPE. A vendor with a large product count has the risk of producing a false positive in the product matching.
(10) **Vendor Vulnerability Count** - The historical number of vulnerabilities reported through the NVD for the given vendor. A high number of vulnerabilities increases the likelihood of a CPE match producing false positives. A low number of vulnerabilities indicates that products for this vulnerability can match at the less granular VENDOR level without much risk of false positives.
(11) **Product Vulnerability Count** - Similar to the vendor vulnerability count, this is the count of vulnerabilities specific to the software product.

For each candidate CPE generated in the fuzzy matching step, the machine learning prediction outputs three classifications:

(1) **Order** - An ordered set (HIGHEST, HIGH, MEDIUM, LOW, LOWEST, REJECT) on the confidence this target CPE is a good recommendation. The ordering classifier allows for a sorted presentation of recommendations back to the security operator. A *REJECT* output indicates the recommender should not present the option to the security operator.
(2) **Level** - Either *VENDOR* or *PRODUCT*. If the target CPE went only to VENDOR for a software publisher with numerous products and vulnerabilities (e.g., Google, Microsoft, etc.), it would have many false positive vulnerability notifications. In contrast, a target CPE for a vendor with relatively few vulnerability notifications could safely match only to the vendor level.

Then the candidate CPEs are ordered based on Order classification first and then the Level classification. CPEs with a higher Order classification go before CPEs with a lower Order classification. When the Order classification is the same, CPEs with the Level classification of PRODUCT go before CPEs with the Level classification of VENDOR. Using the outputs of Order and Level, the recommender can present to human operators the most likely target CPEs first.

We generated training data samples using over 7,000 software product strings (i.e., software inventory package names) found on Wikipedia using the MediaWiki API [4] and Wikipedia categories related to software products. These names match against the CPE word vectors to produce 23,048 samples of training and test data for machine learning.

For each CPE in the NVD, the vendor and product strings receive pre-processing to remove non-alphanumeric characters. Words split based on the convention of using underscores and dashes to separate words in the CPE strings.

[4] https://www.mediawiki.org/wiki/API

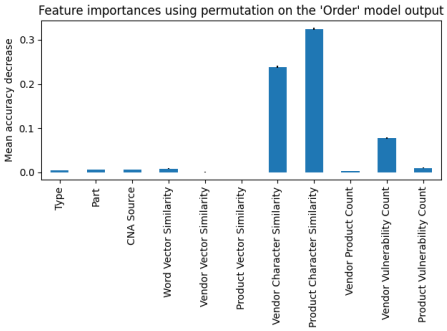

Fig. 1. Feature Importance for 'Order' Output

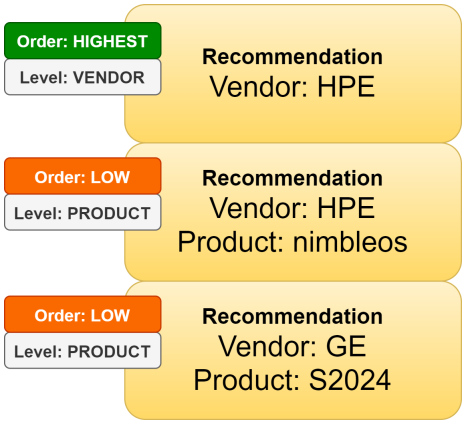

Fig. 2. Example Recommendation Output

Then, the word vectors are calculated for each vendor and product and stored in a hashed reference to the CPE. Training samples are pre-processed and formed into word vectors in the same way. The top 100 of the most similar CPE word vectors are extracted as initial candidates using word vector cosine similarity tests on the product and vendor. Training data labels are generated based on manual inspection and general guidelines to avoid false positives. For instance, decisions to classify a match as a VENDOR typically only apply to vendors having less than 100 vulnerabilities per year. Otherwise, the match will likely produce far too many false positives.

The machine learning model uses a random forest classifier using a Gini impurity measure for the decision split, 100 decision trees in the forest, and an 80/20 training/testing split. Using a random forest classifier reduces the variance between the models and allows for more feature analysis. Table 3 shows the result for the *Order* classification. The highest and lowest order recommendations have very high F-scores because HIGHEST, LOWEST, and REJECT have more determinism and samples. The ordering in between (i.e., HIGH, MEDIUM, and LOW) requires more subjective judgment and more variance. Model classification performance for *Level* is almost 100% accurate. These classifications indicate high determinism in the output. The dataset used for training and testing is available on GitHub [5].

The primary output of the machine learning model is the *order* of presenting the results back to the security operator. Likewise, of the three outputs, *order* has the most number of options and highest variance. Figure 1 shows the feature importance for the *order* prediction using permutations as a measure. Each feature is arbitrarily modified ten times over the training and test data. The importance calculation, i, for feature j, is calculated as the difference in the overall model accuracy score, s and the mean permuted accuracy score:

$$i_j = s - \frac{1}{K}\sum_{j=1}^{K} s_{j_k} \tag{3}$$

The permuted feature measurement shows that the *Vendor Character Similarity* and *Product Character Similarity* have the highest importance in the model performance. Vulnerability count also has a significant impact on the model performance because our *order* labeling factored in the impact of choosing the wrong CPE. If a vendor or product has a higher vulnerability count, then the label is often ordered higher to minimize the potential for an operator missing

[5]https://github.com/pdhuff/cpe_recommender

applicable vulnerabilities. The lower feature importance for word vector similarity tests does not discount their use in the model. Indeed, word vector similarity is used to identify the input candidate CPEs for machine learning, but the variance in word vector similarity does not appear helpful in ordering.

Table 3. Machine Learning 'Order' Classification Results for CPE Matching

Recommended Order	Precision	Recall	F-Score	Support
HIGHEST	100%	98%	99%	937
HIGH	80%	66%	72%	232
MEDIUM	72%	46%	56%	211
LOW	89%	87%	88%	1,076
LOWEST	98%	99%	98%	5,854
REJECT	99%	100%	100%	14,738
Weighted Average	**98%**	**98%**	**98%**	**23,048**

5 IMPLEMENTATION AND EVALUATION

We tested the recommender system and compared it to a human analyst using i) 50 software inventory package names of commonly used software on the Microsoft Window (MS) platform, and ii) 50 hardware inventory names of commonly used network and computing hardware in an enterprise. The search involved inspecting these products on the NVD to find vulnerabilities associated with them, and validating the search result. Overall, the manual process took approximately 6.5 hours to identify target CPEs for the hardware and 70 minutes to identify target CPEs for the software.

Figure 2 provides examples for each prediction. Users have presented a reduced set of options priortized first by *Order* and then by *Level*. For example, if the machine learning finds a product match, the user first sees the product match before a less granular vendor match.

For software, the average number of manual search results was 119, and only 8% of those searches returned a conclusive result. In contrast, our recommender returned an average of 2 results with 40% conclusively identified, since it can accurately map software names to CPEs which in turn can accurately map to vulnerabilities. A sample of the results generated by the recommender is shown in figure 2.

For hardware, the average number of manual search results by the analyst was approximately 34 with only 28% returning a conclusive result, and the average number returned by the recommender was 2 with 48% returning a conclusive result. We consider no results returned as inconclusive. Since hardware is not as commonly represented in the NVD, no search results is not surprising with hardware assets.

For this small dataset of 100 hardware and software inventory package names, the recommender already saved over 7 hours and provided more accurate results. For larger systems with more hardware and software assets, the time saving will be even more. Furthermore, with the recommender automatically finding results, the cybersecurity operation has less risk of missing vulnerability notifications.

6 CONCLUSION

We have presented a new approach for organizations to automatically match their hardware and software inventory to CPEs used by standard vulnerability sources. The recommendation system outputs a small set of target CPE names for identifying applicable vulnerabilities based on the input of software package inventory. Using a pipeline of NLP, fuzzy

matching and machine learning, the system produces a much more accurate and useful result based on the ecosystem of software vendors and CVE Naming Authorities.

Our initial testing showed significant time savings, but more importantly, the results showed an improvement in accuracy. In future work, we plan to validate these results with a larger, more realistic dataset.

ACKNOWLEDGMENT

This work is supported in part by NSF under award number 1751255.

REFERENCES

[1] Abdullah Abuhussein, Sajjan Shiva, and Frederick T Sheldon. 2016. CSSR: cloud services security recommender. In *2016 IEEE world congress on services (SERVICES)*. IEEE, 48–55.

[2] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-based systems* 46 (2013), 109–132.

[3] Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction* 12, 4 (2002), 331–370.

[4] Brant A Cheikes, Brant A Cheikes, Karen Ann Kent, and David Waltermire. 2011. *Common platform enumeration: Naming specification version 2.3*. US Department of Commerce, National Institute of Standards and Technology.

[5] Mark Dowd, John McDonald, and Justin Schuh. 2006. *The art of software security assessment: Identifying and preventing software vulnerabilities*. Pearson Education.

[6] Muriel Figueredo Franco, Bruno Rodrigues, and Burkhard Stiller. 2019. MENTOR: the design and evaluation of a protection services recommender system. In *2019 15th international conference on network and service management (CNSM)*. IEEE, 1–7.

[7] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Computing Surveys (CSUR)* 50, 4 (2017), 1–36.

[8] Philip Huff and Qinghua Li. 2021. Towards Automated Assessment of Vulnerability Exposures in Security Operations. In *EAI International Conference on Security and Privacy in Communication Networks*.

[9] Andreas Kuehn and Milton Mueller. 2014. Shifts in the cybersecurity paradigm: zero-day exploits, discourse, and emerging institutions. In *Proceedings of the 2014 New Security Paradigms Workshop*. 63–68.

[10] H. T. Le and P. K. K. Loh. 2011. Using Natural Language Tool to Assist VPRG Automated Extraction from Textual Vulnerability Description. In *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*.

[11] Triet Huynh Minh Le, Bushra Sabir, and Muhammad Ali Babar. 2019. Automated software vulnerability assessment with concept drift. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 371–382.

[12] Kylie McClanahan and Qinghua Li. 2020. Automatically Locating Mitigation Information for Security Vulnerabilities. In *IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids*.

[13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[14] Fitzroy D Nembhard, Marco M Carvalho, and Thomas C Eskridge. 2019. Towards the application of recommender systems to secure coding. *EURASIP Journal on Information Security* 2019, 1 (2019), 1–24.

[15] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.

[16] Jukka Ruohonen. 2019. A look at the time delays in CVSS vulnerability scoring. *Applied Computing and Informatics* 15, 2 (2019), 129–135.

[17] Ernesto Rosario Russo, Andrea Di Sorbo, Corrado A Visaggio, and Gerardo Canfora. 2019. Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities. *Journal of Systems and Software* 156 (2019), 84–99.

[18] Peichao Wang, Yun Zhou, Baodan Sun, and Weiming Zhang. 2019. Intelligent Prediction of Vulnerability Severity Level Based on Text Mining and XGBboost. In *2019 Eleventh International Conference on Advanced Computational Intelligence (ICACI)*. 72–77.

[19] Y. Yamamoto, D. Miyamoto, and M. Nakayama. 2015. Text-Mining Approach for Estimating Vulnerability Score. In *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. 67–73.

[20] Fengli Zhang, Philip Huff, Kylie McClanahan, and Qinghua Li. 2020. A Machine Learning-based Approach for Automated Vulnerability Remediation Analysis. In *IEEE Conference on Communications and Network Security (CNS)*.

[21] Fengli Zhang and Qinghua Li. 2018. Security Vulnerability and Patch Management in Electric Utilities: A Data-Driven Analysis. In *The 1st Radical and Experiential Security Workshop (RESEC)*.

[22] Fengli Zhang and Qinghua Li. 2020. Dynamic Risk-Aware Patch Scheduling. In *IEEE Conference on Communications and Network Security (CNS)*.