Automated Characterization of Software Vulnerabilities

Preprint · September 2019

CITATIONS
0

READS
8

3 authors, including:

Danielle Gonzalez
Rochester Institute of Technology
8 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Mehdi Mirakhorli
Association for Computing Machinery
69 PUBLICATIONS **636** CITATIONS
SEE PROFILE

Automated Characterization of Software Vulnerabilities

Danielle Gonzalez, Holly Hastings, Mehdi Mirakhorli
Rochester Institute of Technology, Rochester, NY
{dng2551,hmh6182,mxmvse}@rit.edu

***Abstract*—Preventing vulnerability exploits is a critical software maintenance task, and software engineers often rely on Common Vulnerability and Exposure (CVEs) reports for information about vulnerable systems and libraries. These reports include descriptions, disclosure sources, and manually-populated vulnerability characteristics such as root cause from the NIST Vulnerability Description Ontology (VDO). This information needs to be complete and accurate so stakeholders of affected products can prevent and react to exploits of the reported vulnerabilities. However, characterizing each report requires significant time and expertise which can lead to inaccurate or incomplete reports. This directly impacts stakeholders ability to quickly and correctly maintain their affected systems.**

In this study, we demonstrate that VDO characteristics can be automatically detected from the textual descriptions included in CVE reports. We evaluated the performance of 6 classification algorithms with a dataset of 365 vulnerability descriptions, each mapped to 1 of 19 characteristics from the VDO. This work demonstrates that it is feasible to train classification techniques to accurately characterize vulnerabilities from their descriptions. All 6 classifiers evaluated produced accurate results, and the Support Vector Machine classifier was the best-performing individual classifier. Automating the vulnerability characterization process is a step towards ensuring stakeholders have the necessary data to effectively maintain their systems.

***Index Terms*—software maintenance, vulnerability characterization, text classification, CVE, VDO**

I. Introduction

Preventing exploits of existing and emergent vulnerabilities is a critical software maintenance task. Whenever a new vulnerability or exploit affecting a software product is discovered, the software must be updated to remove the vulnerability and/or add appropriate mitigation technique. Stakeholders of a software product must have access to comprehensive and accurate data about vulnerabilities affecting their system to perform this maintenance. The standardized source for vulnerability data are Common Vulnerability and Exposure (CVE) reports, stored in the online and publicly available National Vulnerability Database [2] maintained by the National Institute of Standards and Technology (NIST). Listing 1 is an example of a vulnerability description from a NIST CSV report.

CVE ID: CVE-2017-6725
Overview: A vulnerability in the web framework code of Cisco Prime Infrastructure could allow an unauthenticated, remote attacker to conduct a cross-site scripting (XSS) attack against a user of the web interface of an affected system. More Information: CSCuw65833 CSCuw65837. Known Affected Releases: 2.2(2).
References: http://www.securityfocus.com/bid/99202, **[...]**

Listing 1. Vulnerability Description from the NVD

Characterizing software vulnerabilities is a critical step in identifying the root cause of the vulnerability, understanding its consequences, attack mechanisms and appropriate mitigation techniques. A vulnerability characterization is an important property for making informed decisions to fix or mitigate the vulnerability. NIST has developed a standardized *Vulnerability Description Ontology* (VDO) [1] to characterize software vulnerabilities. The VDO defines the description attributes required to effectively provided actionable intelligence to the software developers aiming to fix/mitigate the vulnerability.

Figure 1 demonstrates various attributes of VDO ontology used to characterize software vulnerabilities. For example, the *Impact Method* characterization category represents how a vulnerability can be exploited. The characterizations in the category reflect specific techniques an attacker can use to take advantage of a vulnerability: *Authentication Bypass*, *Trust Failure*, *Context Escape*, *Man-in-the-Middle Attack*, and *Code Execution*. Other characterizations specify consequences, domain, location, and mitigations for vulnerability exploits.

Stakeholders and affected users of the vulnerable product rely on characteristic data to determine how an exploit affects their systems and how to prevent exploitation. Unfortunately, identifying a vulnerability's characteristics as described in the VDO is a manual, labor-intensive, and asynchronous process. Proper characterization of a vulnerability requires reviewing its descriptions with sufficient security background and familiarity with the ontology. This intensive process has lead to problems with the quality of the vulnerability reports. Studies have found that NVD's vulnerability reports are often left incomplete or lack specific characteristics [10], [13].

In this study, we conducted an experiment in which we used information retrieval, natural language processing, and supervised machine learning techniques to characterize vulnerabilities based on the descriptions present in every CVE report. We manually curated a dataset of 365 vulnerability descriptions, each mapped to 1 of 19 characteristics from the NIST Vulnerability Description Ontology (VDO). This data was run through several stratified 10-fold cross validation experiments to train and evaluate the performance of 6 classification algorithms to determine which was most suited for the task. We initially found that the classifier with the best performance was a Majority Vote ensemble learner, which assigned the majority-chosen label from 1 instance each of the Naïve Bayes, Decision Tree, Support Vector Machine (SVM), AdaBoost-SVM, and Random Forest classifiers. However,

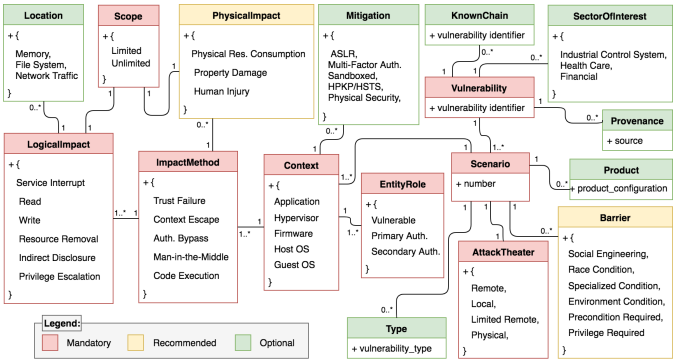

Fig. 1. The NIST Vulnerability Description Ontology (VDO)

further statistical tests showed that it was not significantly better than an individual Support Vector Machine classifier, which requires less resources to train.

We conclude from our findings that it is feasible to rely only on vulnerability description to characterize the CVE attributes. This automated approach can provide valuable information to the software developers so they can better understand a reported vulnerability and its characteristics.

The remainder of this paper is organized as follows. Our data collection and experiment methodology are explained in Section II. The results of our experiment are reported and discussed in Section III. We acknowledge related work in Section IV, and we conclude in Section V.

II. Automated Characterization of Vulnerability Reports

Towards the goal of automating the vulnerability characterization process, we developed an approach that applies natural language processing (NLP), information retrieval (IR), and supervised machine learning techniques to train a classifier to analyze CVE reports and automatically infer their VDO characteristics. This is based on an assumption that the terms used in CVE descriptions offer insights into these characteristics of the vulnerability, and classifiers can learn such insight based on historical data. To test the feasibility of this approach, we trained multiple classification algorithms and compared their performance to determine the best performing classifier for this specific task.

In this section we describe our evaluation approach. First, we created a labeled dataset by manually curating relevant vulnerability descriptions for each of the 19 characteristics Next, 6 classifiers were trained and tested using stratified 10-fold cross validation. The results were used to calculate a set of evaluation metrics, which were used to compare the classifiers' performance as reported in Section III.

A. Labeled Dataset Creation

Using textual descriptions to characterize vulnerability reports is considered a multi-class text classification task [15]. In binary and multi-class classification, each item in the training dataset is manually assigned 1 class (label). Therefore, our first task was to curate a set of vulnerability descriptions for each of the 19 characteristics.

We peer-reviewed numerous CVEs and their vulnerability descriptions in the National Vulnerability Database (NVD) [2] and selected CVE reports that had previously been characterized by the original developers. These CVE reports and their instantiated characteristics were peer reviewed by two security experts from our research group. Descriptions were only kept if the two members of our team agreed with the labeling performed by the original developer. The end result was a manually curated training dataset of 365 labeled descriptions. The number of CVE descriptions mapped to each VDO attribute ranged from 12 to 26, and the median was 19. To mitigate the risks associated with imbalanced data, Proper precautions were taken to ensure that the evaluations handled this situation appropriately (see Section II-D below). Figure 2 shows the distribution of descriptions across the characteristics.

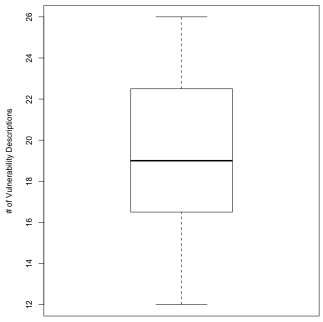

Fig. 2. Distribution of Vuln. Descriptions per Characteristic

B. Training Data Preparation

Before the training data can be provided as input to the classification algorithms, it must be *pre-processed*. This is a necessary step in the natural language processing (NLP) data pipeline which cleans the text data and converts it to a numerical representation for use in classic machine learning algorithms. The vulnerability descriptions mined from the NVD's online vulnerability reports are considered "free text", meaning there were limited restrictions on what characters were permitted. This means it can be "dirty" and contain non-textual characters which will affect the quality of the classifications if left in the input data.

To prepare our text data for classification, we cleaned each vulnerability description. First, all characters were converted to lowercase and urls were removed. Next, the text was *tokenized* by splitting sentences in the text by spaces into individual words. The set of tokens were then filtered to remove any non-word tokens (eg.!,#,?). We also removed stop words (eg. "the","a"). Next, each word in a vulnerability description was *stemmed* to its root. Then the text data was converted to a numerical representation known as a *Term Frequency-Inverse Document Frequency (TF-IDF) Matrix*. Each row in the matrix is a vulnerability description, and each pre-processed word in the entire dataset is a column. Therefore, the TF-IDF weights in the matrix are calculated per-description for every word. These values are proportions of how often the word occurs

within the description and how many descriptions it occurs in. This matrix is then provided as input to each classification algorithm. For our work, we used the StringToWordVec filter in the Weka tool [7] to perform all these operations.

C. Classification

We compared the performance of 6 classification algorithms, using implementations from the Weka data mining tool. We ran preliminary experiments of all algorithms available in the tool and empirically selected 3 "individual" classifiers (Naïve Bayes, Decision Tree, Support Vector Machine) and 3 "ensemble" classifiers (Random Forest, AdaBoost-SVM, Majority Vote).

D. Stratified 10-Fold Cross-Validation

Cross validation is a standard technique for training a classifier and evaluate its performance for unseen data. To mitigate risks related to imbalanced data [5], we applied *stratified* 10-fold cross validation which splits the labeled data into 10 sets, representing each class proportionally in each set. To tune the hyper-parameters for each of the 6 classifiers, we ran the 10-fold experiments using multiple configurations. The best performing configurations were used in our evaluation. These configurations are provided in Table I in the Weka [7] "scheme" format used to run the experiments.

III. Results

After the stratified 10-fold cross validation was conducted for all classifiers, we calculated 6 standard evaluation metrics from Weka's output to compare their performance.

The baseline metrics for each classifier are shown in Table II. The Majority Vote ensemble classifier had the highest accuracy, or total percentage of correctly classified data, at 74.52%. Majority Vote also had the highest Kappa statistic of 0.73. SVM and AdaBoost-SVM tied for 2nd best accuracy (72.88%) and Kappa (0.71).

Per-characteristic information retrieval metrics (precision, recall, and F-measure) were computed for each of the 6 classifiers as shown in Table III. The best F-measure for each characteristic is highlighted (ties are all counted as best) and the number of highlights per classifier was used to calculate what we call the *Ratio of Best Performance* (RBF) for each classifier. This a ratio of the number of characteristics that the classifier "won" (highest F1) to the total number of characteristics evaluated (19). Using this metric, the Majority Voting ensemble was the best performing classifier, for 8 of the characteristics ($RBF = 0.42$).

A. Analysis of Results

The baseline metrics in Table II indicate that all classifiers had an accuracy higher than 50%, and outperformed random assignment. While these metrics can be used to measure the performance of each classifier individually, they are not suitable for *comparing* classifiers. However, these findings indicate that text classification algorithms are well-suited for the task of automatically characterize vulnerability reports using their description.

The best performance metrics in Table III were used instead to compare classifiers, and these results show that each classification algorithm performed best for at least 3 of the characteristic classes. The worst performing classifiers in this respect were Naïve Bayes and Random Forest, each with $RBF = 0.16$. The individual Support Vector Machine (SVM) and the ensemble AdaBoost-SVM had identical F-measures, tying for second-best classifier by performing best for 7 of 19 classes. These 2 classifiers also had the same accuracy and kappa statistic, which indicates that applying Adaptive Boosting did not improve the SVM classifier at all.

To confirm our RBP-based findings, we also conducted statistical significance tests. First, we conducted a Friedman test [15] which confirmed a statistically significant ($p < 0.05$) performance difference between *any* of the classifiers, as shown in Table IV. We also performed a post-hoc Conover test [3] to investigate if differences in performance between individual classifiers was statistically significant. The p-values from this analysis are listed in Table V. This data indicates that the SVM classifier's performance was significantly ($P < 0.05$) different from the Random Forest and Decision tree classifiers but not the Majority Vote or AdaBoost-SVM.

The statistical tests confirm that while the Majority Vote classifier had the highest accuracy and Ratio of Best Performance, it did not significantly outperform the individual Support Vector Machine (SVM). Therefore, there is no motivation to spend more resources training the ensemble Majority Vote and instead a Support Vector Machine is suited for this classification task.

Classifier Evaluation Key Findings:

- Classification learners can be trained to accurately characterize vulnerability reports using their free-text descriptions.
- The classifier with the highest accuracy and Ratio of Best Performance in our evaluation was the Majority Vote ensemble learner, with a Ratio of Best Performance of 0.42 (performing best for 8 of 19 characteristics) and overall accuracy of 74.52%.
- The Support Vector Machine (SVM) and AdaBoost-SVM classifiers tied for 2nd-best with a Ratio of Best Performance of 0.37.
- Majority Vote did not *significantly* outperform these classifiers, and the AdaBoost-SVM ensemble did not *significantly* outperform the individual Support Vector Machine.
- Considering these evaluations and time-to-train,an individual **Support Vector Machine is concluded to be the best-performing classifier for this task**.

IV. Related Work

A. Analysis of Vulnerability Report Data

Several studies have evaluated the *quality* of vulnerability reports which motivated our work to improve them. An early and extensive study by Ozment [13] noted the inconsistent

TABLE I
CLASSIFICATION ALGORITHM HYPER-PARAMETER CONFIGURATION SCHEMES USED IN WEKA TOOL

Classifier	**Weka Schema (Hyper-parameters)**
Random Forest	weka.classifiers.trees.RandomForest,-P 100 -O -I 320 -num-slots 1 -K 1 -M 1.0 -V 0.001 -S 123 -B
Decision Tree	weka.classifiers.trees.J48 -C 0.4 -M 0
SVM	weka.classifiers.functions.SMO -C 0.5 -L 0.001 -P 1.0E-12 -N 0 -V 10 -W 123 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"
AdaBoost-SVM	weka.classifiers.meta.AdaBoostM1,-P 100 -S 123 -I 100 -W weka.classifiers.functions.SMO – -C 0.5 -L 0.001 -P,1.0E-12 -N 0 -V 10 -W 123 -K,"weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C,250007" -calibrator "weka.classifiers.functions.Logistic -R 1.0E-8,-M -1 -num-decimal-places 4"
MajorityVote	weka.classifiers.meta.Vote,-S 123 -B "weka.classifiers.bayes.NaiveBayes " -B,"weka.classifiers.functions.SMO -C 0.5 -L 0.001 -P 1.0E-12 -N 0 -V 10 -W,123 -K \"weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C,250007\" -calibrator \"weka.classifiers.functions.Logistic -R,1.0E-8 -M -1 -num-decimal-places 4\"" -B,"weka.classifiers.trees.J48 -C 0.4 -M 0" -B,"weka.classifiers.trees.RandomForest -P 100 -O -I 320 -num-slots 1 -K 1,-M 1.0 -V 0.001 -S 123" -B "weka.classifiers.meta.AdaBoostM1 -P 100,-S 123 -I 100 -W weka.classifiers.functions.SMO – -C 0.5 -L 0.001 -P 1.0E-12,-N 0 -V -1 -W 123 -K,\"weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C,250007\" -calibrator \"weka.classifiers.functions.Logistic -R,1.0E-8 -M -1 -num-decimal-places 4\"" -R MAJ

TABLE II
BASELINE METRICS FOR CLASSIFIERS

Classifier	**Accuracy**	**Kappa statistic**
Nave Bayes	66.85%	0.65
Decision Tree	64.93%	0.63
SMO-SVM	72.88%	0.71
Random Forest	71.51%	0.70
AdaBoost-SVM	72.88%	0.71
Majority Vote	74.52%	0.73

terminology used at the time for characterizing vulnerabilities and also highlighted the inconsistent report quality. Massacci and Nguyen [10] compared characterizations included in reports from 14 vulnerability databases, maintained by specific and multi-software vendors (eg. Bugzilla and NVD). They corroborated the earlier findings, noting NVD lacked or had incomplete data for many temporal and code-related characterizations. Ladd [9] compared the report generation processes of the NVD and its Chinese equivalent the CNNVD, concluding that the asynchronous reporting processes of the NVD cause the US falls behind on vulnerability management.

B. Characterizing Vulnerabilities

Other relevant works aimed to develop or identify of vulnerability characterizations, but do not use the NIST Vulnerability Description Ontology, which was published in late 2016 [1]. Tierney [16] used similar data mining and machine learning techniques and CVE report data, including a precursor of NIST's CVSS severity scores [11], to learn vulnerability patterns and co-ocurring vulnerabilities, and to classify vulnerabilities by severity. Zhang et al. [17] extracted temporal, affected version, and severity score data from CVE reports and evaluated 6 regression and classification algorithms on the task of predicting a system's time to next vulnerability with limited success, which they attributed to the poor quality of the NVD data. Edkrantz and Said [4] extracted *n-grams* from CVE report summaries along with other CVE data such as the severity score to evaluate the performance of 5 classification algorithms on the task of predicting the likelihood that a vulnerability would be exploited. SVM classifiers also performed best for their task. Santos et. al [14] curated a catalog of *architectural* weaknesses leveraging root cause data from vulnerability reports. Gonzalez et. al [6] also analyzed root cause data from vulnerability reports to identify weaknesses common in the domain of Industrial Control Systems.

Most similar to our work is that of Joshi et al. [8], who also characterized vulnerabilities using the textual descriptions from the CVE reports. While this work shares our goal, the approaches are very different. Instead of classifiers, they trained a custom version of Stanford CoreNLP's Named Entity Recognition implementation to identify and distinguish specific people, places, and things. They defined 10 characterization classes based on existing security literature, and also used terminology from a different ontology [12].

V. CONCLUSIONS & NEXT STEPS

This study was motivated by the existing challenges is maintaining vulnerability reports. We evaluated an approach towards automating the process of identifying characteristics for vulnerabilities using their descriptions. We curated 365 vulnerability descriptions from the National Vulnerability Database (NVD), each mapped to 1 of 19 characteristics from the NIST Vulnerability Description Ontology. We used this data train and evaluate the performance of 6 *classification* algorithms using stratified 10-fold cross validation. Standard information retrieval and machine learning metrics were used to identify the classification algorithm which performed best.

We found that all of the classifiers were able to accurately identify relevant vulnerability characteristics from the NIST Vulnerability Description Ontology (VDO). The best performing classifier for this task based on accuracy and the Ratio of Best Performance metric was a Majority Vote ensemble learner, which assigns the majority-chosen label from 1 instance each of the Naïve Bayes, Decision Tree, Support Vector Machine (SVM), AdaBoost-SVM, and Random Forest classifiers. However, further statistical tests showed that it

TABLE III
PER-CHARACTERISTIC RESULTS OF STRATIFIED 10-FOLD CROSS VALIDATION

	Individual Classifiers									Ensemble Classifiers								
	Naïve Bayes			Decision Trees			SVM			Random Forest			AdaBoost-SVM			Majority Vote		
characteristic	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
ASLR	1.00	0.80	0.89	0.75	0.75	0.75	1.00	0.75	0.86	1.00	0.80	0.89	1.00	0.75	0.86	1.00	0.85	0.92
Context Escape	0.76	0.96	0.85	0.77	0.65	0.71	0.85	0.85	0.85	0.77	1.00	0.87	0.85	0.85	0.85	0.83	0.92	0.87
File System	0.83	0.42	0.56	0.30	0.25	0.27	0.80	0.33	0.47	0.43	0.25	0.32	0.80	0.33	0.47	0.80	0.33	0.47
HPHK	0.64	0.75	0.69	0.88	0.58	0.70	0.89	0.67	0.76	0.89	0.67	0.76	0.89	0.67	0.76	0.90	0.75	0.82
HSTS	0.80	0.73	0.76	0.76	0.86	0.81	0.59	0.77	0.67	0.56	0.86	0.68	0.59	0.77	0.67	0.66	0.86	0.75
Indirect Disclosure	0.69	0.92	0.79	1.00	0.63	0.77	0.96	0.88	0.91	0.88	0.92	0.90	0.96	0.88	0.91	0.91	0.88	0.89
Limited	0.33	0.29	0.31	0.36	0.29	0.32	0.55	0.43	0.48	0.50	0.29	0.36	0.55	0.43	0.48	0.55	0.43	0.48
Man-in-the-Middle	0.75	0.71	0.73	0.93	0.82	0.88	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.77	0.81	0.77	0.79
Memory	0.56	0.41	0.47	0.58	0.68	0.63	0.64	0.64	0.64	0.48	0.55	0.51	0.64	0.64	0.64	0.64	0.64	0.64
Multi-Factor Auth	0.69	0.64	0.67	1.00	0.79	0.88	0.91	0.71	0.80	0.82	0.64	0.72	0.91	0.71	0.80	0.91	0.71	0.80
Network Traffic	0.53	0.63	0.57	0.53	0.56	0.55	0.89	0.50	0.64	0.88	0.44	0.58	0.89	0.50	0.64	0.89	0.50	0.64
Physical Security	0.71	0.56	0.63	0.82	0.78	0.80	0.94	0.83	0.88	1.00	0.67	0.80	0.94	0.83	0.88	0.94	0.83	0.88
Privilege Escalation	0.91	0.87	0.89	0.86	0.83	0.84	0.95	0.83	0.88	0.86	0.83	0.84	0.95	0.83	0.88	0.95	0.83	0.88
Read	0.77	0.71	0.74	0.47	0.63	0.54	0.64	0.75	0.69	0.64	0.75	0.69	0.62	0.75	0.68	0.69	0.75	0.72
Sandboxed	0.61	0.58	0.60	0.88	0.79	0.83	0.79	0.79	0.79	0.82	0.74	0.78	0.83	0.79	0.81	0.79	0.79	0.79
Service Interrupt	0.68	0.77	0.72	0.93	0.82	0.88	0.94	0.88	0.91	0.76	0.94	0.84	0.94	0.88	0.91	0.89	0.94	0.91
Trust Failure	0.57	0.77	0.66	0.33	0.62	0.43	0.74	0.89	0.81	0.76	0.85	0.80	0.74	0.89	0.81	0.70	0.89	0.78
Unlimited	0.71	0.55	0.62	0.61	0.64	0.62	0.52	0.77	0.62	0.64	0.73	0.68	0.52	0.77	0.62	0.57	0.73	0.64
Write	0.19	0.24	0.21	0.09	0.06	0.07	0.23	0.35	0.28	0.29	0.29	0.29	0.23	0.35	0.28	0.22	0.29	0.25
	RBF: 0.16			**RBF: 0.21**			**RBF: 0.37**			**RBF: 0.16**			**RBF: 0.37**			**RBF: 0.42**		

TABLE IV
TESTING FOR SIGNIFICANCE OF CLASSIFIER PERFORMANCE

# of Classes	19
Friedman's Chi-Squared	19.38312
df	5
p-value	0.002

TABLE V
P-VALUES FROM CONOVER POST-HOC TEST

	Decision Tree	**SVM**	**Random Forest**	**Adaboost-SVM**
SVM	0.000033072	NA	NA	NA
Random Forest	0.552778600	0.02808133	NA	NA
Adaboost-SVM	0.000033072	1.00	0.02808133	NA
Majority Vote	0.000000002	0.18663475	0.00000546	0.1866348

was not significantly better than an individual Support Vector Machine classifier, which requires less resources to train. This is an encouraging finding towards measuring the suitability of automated text classification algorithms for this task. With a learner that does not require extensive resources to train and test, vulnerability data can be easily updated and expanded.

With this work, we have demonstrated that the vulnerability management process can be improved with this approach thereby providing actionable information to programmers to effectively perform software engineering tasks. Future research will focus on expanding our dataset to include more descriptions and support the remaining characteristics, and conducting further classifier evaluations to increase the accuracy and scope of the automated approach. We also aim to utilize the CVE characteristics to generate intelligence about each vulnerability and consequently recommend appropriate mitigation strategies to software maintainers.

ACKNOWLEDGMENTS

This work was partially funded by the US National Science Foundation under grant number CNS-1816845 and IIP-0968959 under funding from the S2ERC I/UCRC program and US Department of Homeland Security.

REFERENCES

[1] H. Booth. Draft NISTIR 8138, Vulnerability Description Ontology (VDO). Technical report, National Institute of Standards and Technology (NIST), 2016.

[2] H. Booth, D. Rike, and G. Witte. The National Vulnerability Database (NVD): Overview. Technical report, National Institute of Standards and Technology (NIST), 2013.

[3] W. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, Hoboken, NJ, USA, 3 edition, 12 1999.

[4] M. Edkrantz and A. Said. Predicting exploit likelihood for cyber vulnerabilities with machine learning. *Chalmers Unıversıty of Technology Department of Computer Science and Engineering, Gothenburg, Sweden*, pages 1–6, 2015.

[5] G. Forman and M. Scholz. Apples-to-apples in cross-validation studies: Pitfalls in classifier performance measurement. *SIGKDD Explor. Newsl.*, 12(1):49–57, Nov. 2010.

[6] D. Gonzalez, F. Alhenaki, and M. Mirakhorli. Architectural security weaknesses in industrial control systems (ICS) an empirical study based on disclosed software vulnerabilities. In *2019 IEEE International Conference on Software Architecture (ICSA)*, pages 31–40, March 2019.

[7] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[8] A. Joshi, R. Lal, T. Finin, and A. Joshi. Extracting cybersecurity related linked data from text. In *2013 IEEE Seventh International Conference on Semantic Computing*, pages 252–259, Sep. 2013.

[9] B. Ladd. The Dragon Is Winning - U.S. Lags Behind Chinese Vulnerability Reporting. https://go.recordedfuture.com/hubfs/reports/cta-2017-1019.pdf, Oct. 2017. (Accessed on 05/03/2018).

[10] F. Massacci and V. H. Nguyen. Which is the right source for vulnerability studies?: An empirical analysis on mozilla firefox. In *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, MetriSec '10, pages 4:1–4:8, New York, NY, USA, 2010. ACM.

[11] P. Mell, K. Scarfone, and S. Romanosky. Common Vulnerability Scoring System. *IEEE Security Privacy*, 4(6):85–89, Nov 2006.

[12] S. More, M. Matthews, A. Joshi, and T. Finin. A knowledge-based approach to intrusion detection modeling. In *2012 IEEE Symposium on Security and Privacy Workshops*, pages 75–81. IEEE, 2012.

[13] J. A. Ozment. *Vulnerability discovery & software security*. PhD thesis, University of Cambridge, 2007.

[14] J. C. Santos, K. Tarrit, and M. Mirakhorli. A catalog of security architecture weaknesses. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pages 220–223. IEEE, 2017.

[15] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[16] S. Tierney. Knowledge discovery in cyber vulnerability databases. *Master of science, Computing and Software Systems, University of Washington*, 2005.

[17] S. Zhang, X. Ou, and D. Caragea. Predicting cyber risks through national vulnerability database. *Information Security Journal: A Global Perspective*, 24(4-6):194–206, 2015.