Empirical Validation of Automated Vulnerability Curation and Characterization

Ahmet Okutan, Peter Mell, Mehdi Mirakhorli, Igor Khokhlov, Joanna C. S. Santos, Danielle Gonzalez, and Steven Simmons

Abstract—Prior research has shown that public vulnerability systems such as US National Vulnerability Database (NVD) rely on a manual, time-consuming, and error-prone process which has led to inconsistencies and delays in releasing final vulnerability results. This work provides an approach to curate vulnerability reports in real-time and map textual vulnerability reports to machine readable structured vulnerability attribute data. Designed to support the time consuming human analysis done by vulnerability databases, the system leverages the Common Vulnerabilities and Exposures (CVE) list of vulnerabilities and the vulnerability attributes described by the National Institute of Standards and Technology (NIST) Vulnerability Description Ontology (VDO) framework. Our work uses Natural Language Processing (NLP), Machine Learning (ML) and novel Information Theoretical (IT) methods to provide automated techniques for near real-time publishing, and characterization of vulnerabilities using 28 attributes in 5 domains. Experiment results indicate that vulnerabilities can be evaluated up to 95 hours earlier than using manual methods, they can be characterized with F-Measure values over 0.9, and the proposed automated approach could save up to 47% of the time spent for CVE characterization.

Index Terms—Software Vulnerability, CVE, Vulnerability Characterization, NIST Vulnerability Description Ontology

1 Introduction

Cybersecurity professionals and product vendors rely on vulnerability reports to make informed decisions about the security of their products and reduce the number of related attack vectors [1]–[6]. Vulnerability databases analyze and release reports about known vulnerabilities. They publish information such as vulnerability description, fixes/patches, underlying software weaknesses, root causes, attack vectors, complexity of attack, privileges required, victim interactions required, scope of impact, and consequences [7]–[11].

Public vulnerability databases, such as the NIST National Vulnerability Database (NVD) [12] and those owned by the private sector, rely on manual processes to analyze vulnerabilities. They review free-form text based reports, advisories, and patch information. While this often can be done efficiently for each vulnerability (i.e., consuming several minutes of analyst time), given the tens of thousands of vulnerabilities published every year this is a time-consuming process taking considerable human effort. The analysis effort represents a bottleneck slowing down vulnerability publication. In addition, studies show that this is an error-prone process that has resulted in inconsistencies and delays in releasing final vulnerability results [13]–[18].

Many vulnerability databases leverage the Common Vulnerability and Exposure's (CVE) listing of vulnerability records. To speed up maintenance of this list, software vendors can directly add vulnerabilities by becoming a *CVE Numbering Authority* (CNA). Non-CNA provided vulnerabilities have to be gleaned from public advisories and security announcements. Ideally, all vulnerabilities would be reported by CNAs and those CNAs would report vulnerability details in a structured machine-readable format. This would remove the human analysis bottleneck. Unfortunately, this is not the situation today, and the publication of new CVEs may take weeks or even months [19]. Even in an ideal future where CNAs fill out structured forms to enter vulnerability attribute data, it is likely that only a minority portion of the tens of thousands of CVEs published each year would be covered, since many vendors (especially small ones) may never participate. For this reason, there is a great need for a system that can automate the analyst task by reading free-form vulnerability description text and outputting structured vulnerability attribute data.

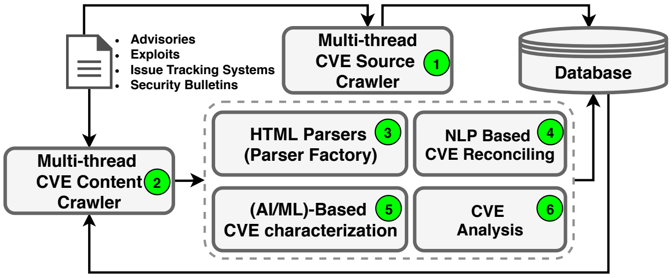

Advisories
Exploits
Issue Tracking Systems
Security Bulletins
Multi-thread CVE Source Crawler 1
Database
HTML Parsers (Parser Factory) 3
NLP Based CVE Reconciling 4
Multi-thread CVE Content Crawler 2
(AI/ML)-Based CVE characterization 5
CVE Analysis 6

Fig. 1: An overview of the proof of concept system that is designed to illustrate the applicability of automated CVE collection and characterization.

In this work, we provide a solution that can currently assist human analysts and someday perform the work automatically under human supervision. It is designed to automatically curate vulnerability data and characterize them by populating the attributes in the draft National

- *M. Mirakhorli, D. Gonzalez, S. Simmons were with the Department of Software Engineering, Rochester Institute of Technology, Rochester, NY. E-mail: mxmvse@rit.edu*
- *A. Okutan is with Leidos, Reston, VA, 20190.*
- *P. Mell is with the National Institute of Standards and Technology.*
- *I. Khokhlov is with the Sacred Heart University, Fairfield, CT, 06825.*
- *J. C. S. Santos is with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, 46556.*

Institute of Standards and Technology (NIST) Vulnerability Description Ontology (VDO) [20]. VDO is designed to enable the characterization of vulnerabilities to support vulnerability databases (especially aspects of vulnerability scoring). Figure 1 shows a general overview of our proof of concept experimental environment that we use to examine the applicability of the proposed methodology.

Our methodology relies on *real-time* data and web-mining techniques to collect vulnerability information as soon as they are disclosed. It uses multiple sources, such as *security bulletins*, *advisories*, *exploit databases*, *issue tracking systems*, *code repositories*, *blogs*, and *mailing lists*. Furthermore, it makes a novel use of Natural Language Processing (NLP), Machine Learning (ML), and Information Theoretical (IT) techniques to process extracted vulnerability reports and characterizes them using VDO. Each CVE is characterized in five domains to identify (1) where an attack may come from (*Attack Theater*), (2) the context of the vulnerability (*Context*), (3) potential impact methods used by exploits (*Impact Method*), (4) likely consequences (*Logical Impact*), and (5) applicable mitigation strategies (*Mitigation*). This automated characterization of vulnerabilities can assist security professionals in their manual review process and therefore reduce the time to disclose or report vulnerabilities. The **contributions** of this work are four-fold:

- First, it demonstrates the feasibility of developing an automated technique to collect and characterize vulnerabilities as soon as they are discovered. In addition to conventional ML-based text classification approaches, it introduces a novel *entropy-based* CVE characterization method. To extract vectors of features from CVE reports, it uses a new context-aware feature extraction and CVE vectorization technique that takes into account the words and their context. Using an entropy-based approach, extracted word vectors derived from CVE descriptions are used to create non-parameterized term histograms. Shannon's entropy is used to estimate the stochastic nature of each histogram, and the Kullback–Leibler divergence (KLD) [21] and Cross-Entropy (CE) are used with entropy redistribution to measure the dissimilarities of different term histograms. The empirical evaluations successfully demonstrate the feasibility of the entropy-based vulnerability characterization method.
- This paper's second contribution is in providing an empirical study based on *mixed-methods*. It uses both quantitative (cross-validations) and qualitative studies (security subject-matter experts (SME) case study) to evaluate whether the process of vulnerability curation and characterization can be improved using ML and information retrieval.
- This paper's third *practical contribution* is in enhancing vulnerability communication by using the NIST VDO framework for automated CVE characterization. For each CVE, our system outputs the *attack vector* (e.g., remote or local), the *vulnerable component* (e.g., hypervisor or application), the *impact method* (e.g., code execution or man in the middle), the *consequences* (e.g., service interruption or privilege escalation), and applicable *mitigation approaches* (e.g., multi-factor authentication or sandboxes).
- Finally, as a case study we investigate how our system could directly support the NVD. For this, we partially map VDO attributes to the Common Vulnerability Scoring System (CVSS) attributes used by NVD and then estimate CVSS scores. This is done only partially as a proof of concept because the current version of VDO does not yet fully support all needed CVSS attributes.

Reproducibility of the Results. The training data sets used to characterize CVEs will be publicly released on GitHub to contribute to the research and development studies for vulnerability intelligence and characterization. All the scripts, algorithms and their configurations will be released publicly. The entire software will be packaged and released as open source.

The rest of the paper is organized as follows: Section 2 describes the research methodology used to build ML and IT methods to characterize CVEs and explains designed experiments. Section 3 presents the results for different characterization methods. Section 4 discusses the validation of performance on real-life uses cases. Section 5 describes the use cases for our automated characterization approach. Section 6 reviews prior works about existing vulnerability platforms and characterization services. Section 7 discusses threats to the validity of the work, and Section 8 provides concluding remarks.

2 Methodology: Vulnerability Characterization

This work aims to minimize the manual effort needed to characterize and describe vulnerabilities based on the NIST VDO framework and therefore provide near real-time curation and disclosure of vulnerabilities. To do so, we develop a web-mining technique that crawls thousands of vulnerability sources and then parses and records the attributes of the crawled CVEs. Apache Open NLP toolkit [22] is used to reconcile CVEs based on part-of-speech (PoS) diversity and novel AI/ML approaches are leveraged for just-in-time CVE characterization. The characterization methodology relies on using a *context-aware* feature extraction and vectorization technique along with conventional supervised learning methods and novel *information theoretical* approaches. This section explains the research methodology, the training data collection process and the experiments used to evaluate the proposed CVE characterization approaches.

2.1 Vulnerability Description Ontology (VDO)

Figure 2 shows the essential pieces of the NIST framework used for CVE characterization, where each vulnerability attribute (*i.e.,* **noun group**) is represented by a box. The mandatory, recommended, and optional noun groups are shown with black, white, and gray headers, respectively. Within each noun group, the set of applicable noun group *values* are listed. For example, the context of a vulnerability is represented by the mandatory noun group *Context*, whose noun group values are: *Application, Hypervisor, Firmware, Host OS*, and *Guest OS*. Similarly, the technique used by an attacker to execute an exploit is represented by the noun group *Impact Method*, while *Trust Failure, Context Escape,*

TABLE 1: VDO's vulnerability attribute domains (noun groups) included in this study. From now on, we use the term *noun group* to refer to the core building blocks of the VDO framework.

Attack Theater	Remote	A vulnerability is characterized as Remote if the cyberattack originates from locations outside of the target network.
	Limited Remote	The exploit is executed from closer locations, using Cellular, Wireless, Bluetooth, Infrared, or Line-Of-Sight technologies.
	Local	The attacker has to have logical local access to a target computer or system to execute the exploit.
	Physical	The attacker is required to have physical access to the target system to carry out the exploit
Context	Application (App)	CVE is related to a program that is designed to accomplish a specific task within an operating system or firmware.
	Hypervisor (Hyp)	Allows an attacker to get access or manipulate resources that are shared among controlled guest operating systems.
	Firmware (Fw)	An attacker exploits a vulnerability in the software that is built-in to a device.
	Host OS (HOS)	This is a vulnerability in the operating system and the Hypervisor is not applicable.
	Guest OS (GOS)	This is a vulnerability in the operating system that is controlled by a Hypervisor.
	Channel (Ch)	A flaw in the logical communication medium, such as the incorrect implementation of a cipher algorithm.
	Physical Hardware (Hw)	This represents a flaw in the actual physical hardware, such as processors, storage, memory cells, etc.
Impact Method	Trust Failure	A vulnerability is exploited if an assumed trust relationship between two parties leads to unexpected impacts.
	Context Escape	Attackers exploit a trust mechanism by breaking out of a sandbox.
	Authentication Bypass	The exploit is related to a failure to identify the adversary properly.
	Man in the Middle (MitM)	The attackers access acommunication channel that might lead to sensitive data disclosures, impersonation, data modification or denial of communication.
	Code Execution	A vulnerability exploit allows an attacker to execute unauthorized code
Logical Impact	Write	A vulnerability is characterized with Write if an attacker can do unauthorized modifications on the data.
	Read	Indicate whether the attacker is able to gain unauthorized access to data.
	Resource Removal	Resource Removal is used to represent an unauthorized removal (deletion) of data.
	Service Interrupt	An attacker causes a loss in the availability of a target system.
	Indirect Disclosure	Attacker can learn information about the target, not through a direct read operation, but indirect methods like side-channel attacks or traffic analysis.
	Privilege Escalation	An adversary gains a level of privilege that is not intended for him/her.
Mitigation	Address Space Layout Randomization (ASLR)	A vulnerability is characterized by ASLR mitigation, if ASLR is an applicable protection mechanism to guard against buffer overflows.
	HPKP/HSTS	If HTTP Public Key Pinning (HPKP) or HTTP Strict Transport Security (HSTS) is applicable as a mitigation strategy.
	Multi-Factor Authentication (MFA)	It is used if MFA is a viable protection technique for a vulnerability.
	Physical Security	If ensuring physical security provides protection from the exploits that are caused by a vulnerability.
	Sandboxed	If deploying a software product in the sandbox provides protection.

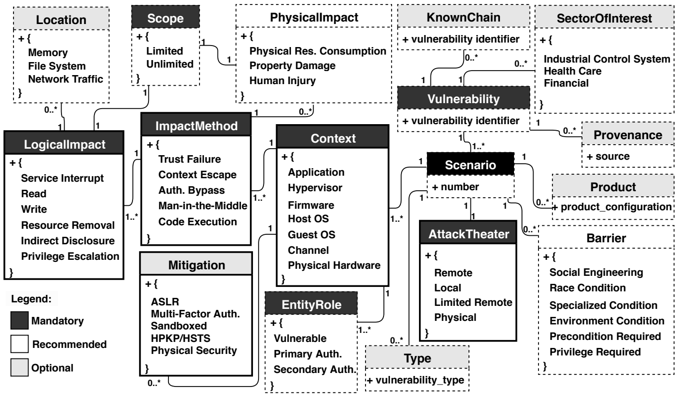


Fig. 2: Partial View of the Vulnerability Description Ontology (VDO) [20]

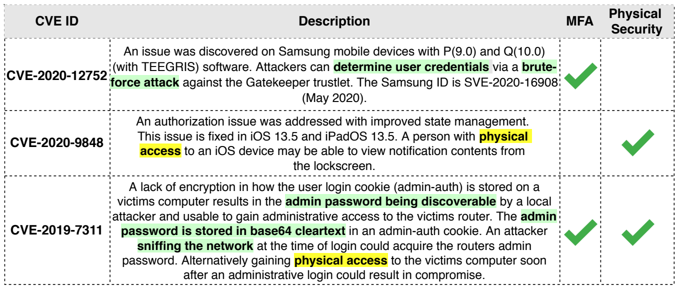
Authentication Bypass, *Man in the Middle*, and *Code Execution* are listed as viable methods. For the scope of this paper, to characterize CVEs, five noun groups were selected from the VDO framework based on the following criteria:

- Some noun groups are ignored because they are assumed to apply to a more limited scope of problems, *e.g.*, *Physical Impact* or are not easy to interpret accurately, compared to other groups, *e.g.*, *Scope*.
- Mandatory noun groups are applicable to all vulnerabilities. Therefore, mandatory groups that have more than one label and provide a higher value for the security community are considered first while determining the set of noun groups that will be studied. The security community is usually more interested to know (1) the attack avenues from which cyberattacks occur, (2) vulnerable pieces in a target system, (3) attack methods utilized, (4) potential impacts when exploitation is successful, and (5) applicable mitigation techniques that can be used when a vulnerability is exploited [14]. To address these concerns, five noun groups are studied: *Attack Theater*, *Context*, *Impact Method*, *Logical Impact*, and *Mitigation*. Figure 2 shows these noun groups as boxes with solid line borders.

Attack Theater defines the attack surface from which an attack may come. The *Context* noun group defines the entities where the impacts are observed when a vulnerability is exploited. The *Impact Method* noun group describes methods used to exploit a software vulnerability. The *Logical Impact* noun group describes the impacts that an exploit can create. The *Mitigation* noun group describes the techniques that can be used to limit the impact of a vulnerability, even if it is exploited. Table 1 provides a brief summary of the noun group values in the studied VDO domains.

2.2 Data Collection & Labelling

The preparation of the training data sets is a critical stage that may affect the performance of the designed system.



CVE ID	Description	MFA	Physical Security
CVE-2020-12752	An issue was discovered on Samsung mobile devices with P(9.0) and Q(10.0) (with TEEGRIS) software. Attackers can **determine user credentials** via a **brute-force attack** against the Gatekeeper trustlet. The Samsung ID is SVE-2020-16908 (May 2020).	✓	
CVE-2020-9848	An authorization issue was addressed with improved state management. This issue is fixed in iOS 13.5 and iPadOS 13.5. A person with **physical access** to an iOS device may be able to view notification contents from the lockscreen.		✓
CVE-2019-7311	A lack of encryption in how the user login cookie (admin-auth) is stored on a victims computer results in the **admin password being discoverable** by a local attacker and usable to gain administrative access to the victims router. The **admin password is stored in base64 cleartext** in an admin-auth cookie. An attacker **sniffing the network** at the time of login could acquire the routers admin password. Alternatively gaining **physical access** to the victims computer soon after an administrative login could result in compromise.	✓	✓

Fig. 3: Three example CVEs labeled for two values in the Mitigation noun group, *i.e.*, Multi-Factor Authentication (MFA) and Physical Security.

Therefore, we followed a systematic methodology to generate training data sets for noun groups in VDO. We collected a set of CVEs from NVD for each studied noun group, and the relationships between each CVE and the labels in each group were identified. Each CVE, in each noun group, is labeled with one or more noun group values, as shown in Figure 3.

In the next subsections, we explain how we collected the set of CVEs used for training and performed labelling.

2.2.1 Data

The most recent CVEs (before CVE-2020-14000) are queried from the `NVD Vulnerability Search Engine` (https://nvd.nist.gov/vuln/search) to derive near to equal number of CVEs that can be characterized with the noun group values within a studied group. CVEs after CVE-2020-14000 are reserved for the case study to test the performance of the trained models with hands-on, real-world examples.

2.2.2 Data Labeling Process

Five security researchers with 2-15 years of experience spent 3000+ person-hours to generate training data sets for the five VDO noun groups. All reports were peer-reviewed and examined to avoid biases, and uncertainties were discussed collectively to ensure consistency.

The main steps of the systematic approach we used to collect, label, and review CVE data for each noun group are:

1) **SMEs' Understanding of the Noun Groups:** Subject-matter experts were all already familiar with VDO vulnerability attribute domain (noun group). However, each SME were requested to carefully study each noun group value (label) within a group. Then each noun group value was discussed among all SMEs to make sure that its distinctive attributes are understood clearly.
2) **Labeling and Annotation Sessions:** Each SME reviewed and annotated the noun group dataset that they were assigned randomly (without considering any specific preference). They had to provide three pieces of information, the labels, annotation of the text that implies the label, and a confidence score. Most of the time, a CVE is labeled with one label within a noun group, however it sometimes needs to be labeled with more than one label. For example, CVE-2019-7311 in Figure 3 is marked with both *Multi Factor Authentication* and *Physical Security* for the Mitigation group, because both provide a mitigation strategy for the vulnerability.
3) **Self-Reported Confidence:** To minimize guessing and distinguish high-confidence responses from others, we collected the SME's confidence score with a numerical score ranging from one to three, where a score of one indicated no-confidence in making a judgement; a score of 2 indicated a low-confidence in the generated label, and a score of 3 indicated high-confidence responses. High-confidence responses were reviewed by a second labeler.
4) **Peer-Discussion Sessions:** CVEs tagged with no-confidence or low-confidence were discussed by two SMEs. A confidence column is included in each data set to represent the confidence of the researcher while assigning each noun group label to a CVE. During *Peer-Discussion Sessions*, reviewers went through additional sessions to double-check the labels that were assigned a low confidence (a confidence value of 1 or 2).
5) **Peer Review and Discussion Sessions:** After the first review sessions, a draft version of the data set for each studied noun group is recorded. The percentage of CVEs labeled with a high confidence for *Attack Theater, Context, Impact Method, Logical Impact*, and *Mitigation* noun groups were 94%, 58%, 87%, 80%, and 72%, respectively. A new annotator peer-reviewed all CVEs and confidence scores and highlighted the CVEs she/he wanted to discuss. Once all CVEs had a confidence value of 3, the final version of each data set was recorded for experiments.

For each noun group, each CVE is labeled with one or more labels. As shown in Figure 3, the SMEs highlighted parts of the CVE's description that were relevant in the labeling process, *i.e.*, in assigning the CVE to a specific noun-group. For example, CVE-2019-7311 has text highlighted in *green*, which are related to *MFA*, and text highlighted in *yellow*, which are related to *Physical Security*.

Generated training data sets include two fields, the CVE description and the corresponding label(s) for the noun group. The total number of CVEs for *Attack Theater*, *Context*, *Impact Method*, *Logical Impact*, and *Mitigation* noun groups are 293, 798, 465, 562, and 474, respectively. The distribution of the labels for each group is shown in Figure 4. The data sets generated for noun groups include between 70 and 120 CVEs for each label (noun group value).

2.3 CVE Pre-processing

We perform a number of standard NLP pre-processing steps to ensure all CVE descriptions are cleaned, tokenized, and normalized before ML/IT models are trained. First, CVE descriptions are cleaned to make sure that any duplicate white spaces, punctuation, and numbers are removed, and the text is reduced to characters readable by humans. Then, all characters are converted to lower case, and all stop words are removed, *i.e.*, commonly occurring words such as 'this' and 'shall' which are not helpful for classification purposes. Next, each word in the vulnerability descriptions is stemmed to its morphological root [23].

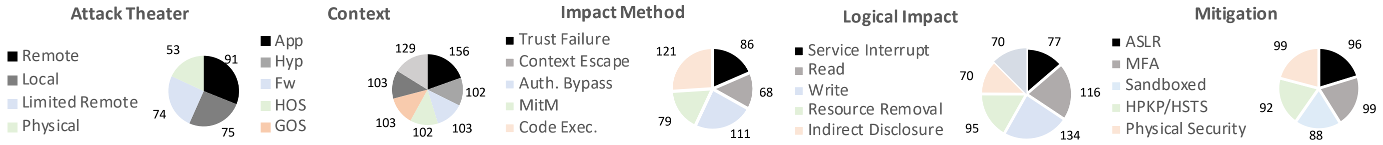


Fig. 4: The distribution of noun group values in each studied VDO domain.

2.4 Context-Aware Feature Extraction and Vectorization of CVEs

To extract features from CVE descriptions, we converted each description into a vector of terms using the *Term Frequency-Inverse Document Frequency* (TF-IDF) approach. We did not use word embedding (word2vec) as we did not have an extensive domain-specific corpus to train a word2vec model. Instead, we followed TF-IDF, a very well-known and rather simple approach, such that we can use the CVE descriptions only, without a need to create a domain-specific corpus.

Although TF-IDF can help identify the terms most relevant to a specific noun group, in the security domain, the context of the terms plays an important role for their meaning. For instance, *"user authorization"* and *"control"* may appear in any context in a vulnerability description, but knowing the context of the terms (*e.g.*, *"control"* appearing right after "user", *i.e.*, "User-Controlled") can provide richer information about the text which is otherwise lost in a bag of words approach. Therefore, to extract rich features describing the vulnerability reports, we augment the TF-IDF approach by using *n-grams*. This ensures that the order and the context of tokens in the CVE descriptions are taken into account by the trained models.

For a VDO noun group G, let the data set D_G be composed of N CVE instances where each CVE description is represented by

$$\mathbf{x^i} = (x_1^i, x_2^i, ..., x_M^i) \tag{1}$$

$\mathbf{x^i}$ is a term frequency vector generated from a pre-processed CVE description i where each x_j^i represents the number of a uni, bi or 3-gram j in the description. The feature vector $\mathbf{x^i}$ of each CVE description in D_G is associated with one or more VDO noun group values y^i where $y^i \in \{y_1, y_2, ..., y_K\}$ and K represents the number of labels in G.

The n-grams that have higher frequencies within a CVE description should have higher importance for that CVE. However, we should penalize the frequencies of such terms, if they frequently appear in all other descriptions within the CVE corpus of a noun group, to make sure that the n-grams that are distinctive are emphasized more. The tf-idf approach ensures that the scores of the terms that have a high frequency in all documents are penalized properly using an inverse document frequency (idf) term. Given a CVE i and a term j, if the term frequency of j (in i) and its inverse document frequency are represented by p_j^i and d_j^i, respectively, the tf-idf score w_j^i of the term j in CVE i is calculated by

$$w_j^i = p_j^i * d_j^i \tag{2}$$

where

$$p_j^i = \frac{x_j^i}{\sum_{j=1}^{M} x_j^i} \tag{3}$$

and

$$d_j^i = \log \frac{N}{D_j + 1} \tag{4}$$

where N is the total number of CVEs, and D_j is the number of CVE descriptions that include term j. A word vector $\mathbf{w^i}$ that is composed of the tf-idf scores of all n-grams is created for each CVE where $\mathbf{w^i} = (w_1^i, w_2^i, ..., w_M^i)$. For each noun group G, generated word vectors $\mathbf{w^i}$ and their associated VDO labels y^i are used to train binary and multi-class classification models for CVE characterization.

2.5 CVE Characterization Methods

After creating a context-aware vectorization of CVEs, this paper uses two main approaches for CVE characterization:

1) **Machine Learning (ML) Approach:** Labeled CVE data sets are used to train supervised ML models with commonly used classifiers listed in Section 2.5.1.
2) **Information-Theoretical (IT) Approach:** IT methods are widely used in Statistics and Computer Science for various tasks, including Inference and Natural Language Processing. In the Information Theory, "Entropy" is a key metric used to measure the amount of uncertainty for the value of a random variable. This work uses a novel *entropy-based* approach to characterize CVEs, by deriving n-gram frequencies from CVE descriptions and using them as non-parameterized histograms. Individual CVE histograms P^i with the same noun group value are unified to generate a non-parameterized histogram Q^i for the noun group value. While calculating the divergence between P^i and Q^i, the entropy of Q^i is evenly distributed to all n-grams in Q^i (including the unseen ones), not to completely rule out the possibility of the existence of unseen n-grams in Q^i. The novel IT approach that is based on entropy distribution is explained in Section 2.5.2.

2.5.1 *Classifiers*

Once all vulnerability descriptions in each noun group G are converted to feature vectors ($\mathbf{w^i}$) as described in Section 2.4, a set of classifiers are trained using CVE feature vectors $\mathbf{w^i}$ as features, and VDO noun group labels y^i as classes. There are between 80-100 CVE instances marked positive for each label in the training data sets used for each noun group G. Since the amount of labeled data is limited, a set of commonly used supervised algorithms from different domains are used to characterize CVEs, rather than following a Deep Learning approach:

1) **Support Vector Machines (SVM):** SVM is a supervised learning algorithm that maximizes the margin between the training patterns and the decision boundaries [24]. We use the SVM algorithm with a sequential minimal optimization approach and a polynomial kernel to characterize CVEs [25].
2) **Naïve Bayes (NB):** Given a set of features and a target label, the algorithm assumes feature independence and uses the Bayes rule while predicting the target label. NB classifiers have proven to be successful in solving various text classification problems, including authorship attribution, spam email detection, and sentiment analysis [26]. This paper uses the NB classifier [27] to predict the noun group value y^i of a CVE description represented by feature vector $\mathbf{w^i}$.
3) **C4.5 Decision Tree (DT) and Random Forest (RF):** We use the C4.5 DT algorithm [28] with pruning to predict VDO labels for CVEs. As an alternative method, the Random Forest algorithm [29] that uses an ensemble learning approach is used. To characterize a CVE, we construct a forest of random trees and use the VDO label that is agreed on by the majority of the constructed trees.
4) **Ensemble Learning (Voting):** Majority voting is one of the widely used ensemble methods to boost the performance of the ML classifiers [30]. This paper uses a voting based ensemble approach which combines the results of SVM, NB, DT, and RF classifiers.

The ML classifiers described above are used to predict the VDO noun group values for each CVE with binary and multi-class classification approaches. The experiment process is automated using the Weka Workbench API [31]. To keep the training and experimentation process simple and easy to reproduce, each algorithm is used with its default parameter set in Weka 3.8.0.

2.5.2 Information Theory based CVE Characterization

This work uses a novel *entropy-based* approach to characterize CVE descriptions. Each feature vector $\mathbf{x^i}$ that is created from a CVE description i is composed of the frequencies of n-grams where $n >= 1$ and $n <= 3$. The probability of each n-gram j within a CVE description i (p^i_j) is calculated by Equation 3 and a feature histogram P^i is generated where $P^i = (p^i_1, p^i_2, ..., p^i_M)$ and $\sum_j^M p^i_j = 1$.

Assuming that there are n feature histograms generated from n CVEs labeled with the same VDO label y^i, they are unified to generate a new feature histogram $\mathbf{z^i} = (z^i_1, z^i_2, ..., z^i_M)$, where the frequency of each n-gram z^i_j is calculated by

$$z^i_j = \sum_i^n x^i_j \tag{5}$$

The unified feature histogram $\mathbf{z^i}$ is considered as a new distribution Q^i, where the probability of each n-gram in Q^i is calculated by

$$q^i_j = \frac{z^i_j}{\sum_{j=1}^M z^i_j} \tag{6}$$

n-grams that have higher q^i_j values within a distribution Q^i better represent the semantic attributes of the associated CVE descriptions, compared to other n-grams. However, CVE feature distributions are usually sparse, and the probability values for many n-grams are zero. Although setting the probabilities of the n-grams that do not appear in Q^i to zero is mathematically correct, it may not accurately represent the uncertainties in real life. A key novelty of this work is the use of the Shannon's entropy to estimate the stochastic nature of a feature distribution and then distribute its entropy to all n-grams in the distribution (including the unseen ones) to reflect the possibility of having an unobserved n-gram. Given a feature distribution Q^i, where q^i_j represents the probability of an n-gram j, its entropy is calculated by

$$H(Q^i) = -\sum_j^M q^i_j \log(q^i_j) \tag{7}$$

where M is the total number of n-grams in the distribution. This entropy is normalized and then evenly distributed to all n-grams in the distribution, by defining r^i_j as

$$r^i_j = q^i_j(1 - H(Q^i)) + \frac{H(Q^i)}{M} \tag{8}$$

where $0 \leq H(Q^i) \leq 1$. The IT approach uses the smoothed n-gram probabilities r^i_j calculated by Equation 8 while measuring the dissimilarity between a CVE represented by P^i and a distribution Q^i. For example, for a distribution Q^i the probabilities of n-grams before and after the entropy is distributed are shown in Figure 5. The probabilities q^i_j that are shown with a dashed black line are smoothed to the probabilities shown with a solid red line (r^i_j) after the entropy is distributed.

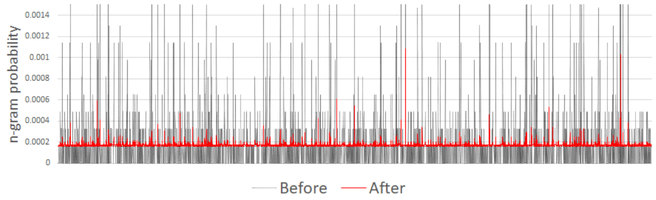


Fig. 5: The probabilities of n-grams in a feature distribution derived from CVEs that are marked positive for *Trust Failure (Impact Method)*, before and after the entropy is distributed to all n-grams.

Given a feature distribution P^i that represents a CVE description i, and a distribution Q^i that represents the CVEs with the same VDO label y^i, the KLD or relative entropy [21] from P^i to Q^i is calculated by

$$KL\left(P^i \quad Q^i\right) = \sum_j^M p^i_j \log \frac{p^i_j}{q^i_j} \tag{9}$$

The cross-entropy measures the relative entropy between two probability distributions. The cross-entropy of the distribution Q^i relative to the distribution P^i is defined as

$$H(P^i, Q^i) = H(P^i) + KL\left(P^i \quad Q^i\right) \tag{10}$$

Substituting Equations 7 and 9 into Equation 10, we get

$$H(P^i, Q^i) = -\sum_j^M p^i_j \log(q^i_j) \tag{11}$$

2.6 Experiment Design

For each ML and IT-based CVE characterization method, two sets of experiments are designed to evaluate the performance. This section briefly explains the details of the binary and multi-class characterization experiments designed for the ML and IT methods.

2.6.1 *Binary CVE Characterization*

In a use-case scenario where the security analyst wants to see if certain noun group values are applicable or not (, e.g., check if the multi-factor authentication (MFA) mitigation technique is applicable), the binary characterization could be helpful. A binary model is built for each noun group value y^i in each VDO domain G. For each y^i, CVEs that are labeled with y^i are marked positive, and the remaining ones in G are marked negative to generate a binary classification data set for y^i. Proposed ML and IT methods are used to build a binary classifier for each y^i. Given a CVE and a noun group value y^i, each ML and IT model uses a binary classification approach to identify whether y^i is applicable to the CVE or not.

In the ML approach, commonly used classifiers from different domains and an ensemble of them are used for CVE characterization. In the IT approach, KL divergence (KLD) [21] and Cross-Entropy (CE) are used as divergence metrics to calculate the dissimilarity between two CVE term distributions. Two n-gram distributions (Q^i and R^i) are created for each noun group value y^i, based on the CVEs that are marked positive and negative for y^i, respectively. The KLD (or CE) between a CVE represented by P^i and these two probability distributions is calculated, and the label of the distribution that has the lowest divergence from P^i is assigned to the CVE.

2.6.2 *Multi-class CVE Characterization*

For the ML approach, algorithms described in Section 2.5.1 are used to build multi-class classifiers for CVE characterization. In the IT approach, one term distribution (Q^i) is created for each noun group value y^i. Given a CVE i that is represented by distribution P^i and a set of distributions (unified histograms, one for each noun group value) $\{Q^1, Q^2, ..., Q^K\}$, where $K \geq 2$, the dissimilarity of P^i from each Q^i is calculated, and the label of the Q^i that has the lowest divergence from P^i is used to assign the label of CVE i.

A multi-class classifier is built for each VDO domain to predict the noun group values as labels. In multi-class classification, we keep track of the confidences predicted for each VDO label y_i within each noun group G. The goal is to see the confidence of the second or even third best predictions. If the confidence of the first prediction is below a defined threshold ρ ($p(y^i) < \rho$), we predict multiple labels $y^i_1, y^i_2, ..., y^i_k$ where $\sum_{j=1}^{k} p(y^i_j) >= \rho$. Providing the confidence values of the second or third best predictions could aid security practitioners to focus on the most likely labels first, which can help to decrease the amount of manual work during the CVE characterization. Therefore, this work defines an alternative multi-class classification approach, where each classier is given a second prediction chance. In summary, each ML/IT model reports two sets of results for multi-class characterization:

- **Single-chance:** This is the conventional multi-class classification approach where the performance of a model is measured based on its predicted labels.
- **Double-chance:** To decrease the manual effort needed for CVE characterization and limit the set of noun group values that are potentially applicable, two labels are predicted for each CVE. Each model is given a second chance to predict a second VDO label for each noun group, which corresponds to the label with the second-best confidence. The prediction of the multi-class classification is assumed to be correct if either the first or second prediction is correct. The goal of the double chance prediction is to support the decision-making process of the security analyst who is doing the manual characterization. Both predictions are provided to the analyst in an ordered list. If the first prediction seems to be unrelated, the analyst can look at the second prediction without needing to analyze the CVE description further.

We use single (top-1) and double chance (top-2) while evaluating our approach for two reasons. First, we want to evaluate the practicality of the approach in a real scenario for an end-user (analyst) that may use these results. Therefore, analyzing results for more than top-2, would require more effort from the analysts. Second, the double chance prediction emulates a real scenario to support the decision-making process of the security analyst who is doing the manual characterization. Our approach provides both predictions to the analyst in an ordered list. If the first prediction seems to be unrelated, the analyst can look at the second prediction without needing to analyze the CVE description and do research about the related CVE.

2.6.3 *Evaluation Setup*

For both ML and IT methods, each experiment is run with classical 10-folds cross-validation, and the average F-Measure values are reported to evaluate results. The main reason for using cross-validation instead of the conventional train/test split approach was the availability of limited data for training. Considering the fact that each noun group value (label) has limited number of instances in the underlying dataset (Figure 4), it would be hard to maintain a balanced label distribution between the training sets and their corresponding tiny test sets if the train/test split approach was used. To assign confidence scores to the predictions of IT methods, divergence measures are converted to normalized proximity scores favoring lower divergences. Assuming the divergences of K distributions from a CVE P^i are $\{\delta^1, \delta^2, ..., \delta^K\}$, and the minimum and maximum divergence values are δ_{mn} and δ_{mx}, we define $\epsilon = \delta_{mn}/100$ and derive a proximity score λ^i for each Q^i, where $\lambda^i = (\delta_{mx} + \epsilon - \delta^i)$. While classifying P^i, the normalized λ^i scores are used to assign a confidence score to each VDO label represented by each Q^i.

2.6.4 *Performance Metrics*

Depending on the outcome of a classification, there are four possible cases for a label y^i_k that represents the noun group value k for CVE i:

- **True Positive (TP)**: If a CVE had true VDO label y^i_k and it is correctly classified to y^i_k.

- **False Positive (FP)**: If a CVE had true VDO label y_l^i, but it is incorrectly classified to y_k^i where $y_k^i <> y_l^i$.
- **True Negative (TN)**: If a CVE had true VDO label y_l^i and it is correctly classified to y_l^i where $y_l^i <> y_k^i$.
- **False Negative (FN)**: If a CVE had true VDO label y_k^i, but it is incorrectly classified to another label y_l^i where $y_k^i <> y_l^i$.

For a given label, the Precision metric represents the fraction of the true positives within all positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

The Recall metric represents the fraction of the true positives among the instances that are actually positive.

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

We calculate the harmonic mean of the Precision and Recall to derive F-Measure and use it as a performance metric to compare different characterization methods. F-Measure provides a more realistic performance measure across different class distributions and characterization methods.

$$F - Measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{14}$$

3 Vulnerability Characteization Results

This section presents the empirical evaluation of our approach and provides a detailed comparison of different characterization techniques.

3.1 Binary CVE Characterization

For binary characterization, the average F-Measure values for the **positive** class are used to evaluate the performance of different methods. Sections 3.1.1, 3.1.2, 3.1.3, 3.1.4, and 3.1.5 provide the binary characterization results for *Attack Theater*, *Context*, *Impact Method*, *Logical Impact*, and *Mitigation*, respectively.

3.1.1 Attack Theater

The mandatory noun group *Attack Theater* defines the attack avenue from which an attack may come. Each vulnerability is characterized by four noun group labels: *Remote*, *Local*, *Pyhsical*, and *Limited Remote*. The F-Measure values for the *Attack Theater* group for different ML and IT methods are shown in Table 2. Depending on the underlying method used, the models achieve F-Measure values up to 0.98. The top three methods to predict *Attack Theater* are KLD, Vote, and CE with an average F-Measure value of 0.92, 0.91, and 0.91, respectively.

3.1.2 Context

The *Context* noun group defines the entities where the impacts are observed, when a vulnerability is exploited. It includes seven noun group labels: *Application*, *Hypervisor*, *Firmware*, *Host OS*, *Guest OS*, *Channel*, and *Hardware*. The F-Measure values of *Context* group for ML and IT methods are shown in Table 3. The models achieve F-Measure values up to 0.95 while predicting noun group labels. The top three methods to predict *Context* are Vote, SVM, and DT, with an average F-Measure value of 0.90, 0.87, and 0.85, respectively.

3.1.3 Impact Method

The *Impact Method* noun group describes methods used to exploit a software vulnerability and includes five labels: *Trust Failure*, *Context Escape*, *Authentication Bypass*, *Man in the Middle (MitM)*, and *Code Execution*. The F-Measure values for the *Impact Method* noun group for different ML and IT methods are shown in Table 4. The top three methods to predict *Impact Method* are Vote, DT, and KLD with an average F-Measure value of 0.95, 0.95, and 0.93, respectively.

3.1.4 Logical Impact

The *Logical Impact* domain describes the impacts that an exploit can create. One vulnerability may incorporate multiple logical impacts simultaneously, because the applicable noun group values are not mutually exclusive. The domain contains six labels: *Service Interrupt*, *Read*, *Write*, *Resource Removal*, *Indirect Disclosure*, and *Privilege Escalation*. The F-Measure values obtained for the *Logical Impact* noun group are shown in Table 5 for ML and IT approaches. The top three methods while predicting *Logical Impact* are Vote, DT, and SVM with an average F-Measure value of 0.91, 0.91, and 0.89, respectively.

3.1.5 Mitigation

The *Mitigation* noun group describes the techniques that can be used to limit the impact of a vulnerability, even if it is exploited. Five mitigation techniques from the VDO model are included: *ASLR*, *MFA*, *Sandboxed*, *HPKP/HSTS*, and *Physical Security*. The F-Measure values while predicting different mitigation strategies are shown in Table 6 for ML and IT approaches. The top three methods are Vote, DT, and SVM, with an average F-Measure value of 0.94, 0.94, and 0.89, respectively.

3.2 Multi-class CVE Characterization

The Attack Theater, Context, and Impact Method VDO domains have mutually exclusive noun group values in the training data sets. Therefore, the multi-class classification, described in Section 2.6.2, is used for them as an alternative CVE characterization approach. The results of the multi-class classification method for the *Attack Theater* noun group are shown in Figure 6. In the single-chance approach, average F-Measure values while predicting the *Attack Theater* noun group values range between 0.8 and 0.97. The double-chance evaluation approach increases the average F-measure between 2% and 5%, except NB. 5% increased F-Measure values for KLD (KLD-M) and CE (CE-M) are shown with a dashed line and scattered square points in Figure 6.

The results of the multi-class classification for the *Context* noun group are shown in Figure 7. In the single-chance evaluation approach, average F-Measure values are between 0.74 and 0.96 for all noun group values, with all characterization methods except NB, which achieved 0.73 on average. The double-chance evaluation approach increases the average F-measure between 6% and 8%, except NB and DT. 8% increased F-Measure values for RF (RF-M), and CE (CE-M) are shown with the dashed line and scattered square points in Figure 7.

TABLE 2: F-Measure values for *Attack Theater*.

	Remote	Local	Physical	Limited Remote
KLD	**0.94**	**0.92**	**0.9**	**0.91**
CE	**0.94**	0.89	**0.91**	**0.91**
SVM	**0.92**	0.77	**0.96**	0.79
NB	0.89	0.8	0.85	0.75
DT	**0.97**	0.8	**0.98**	0.87
RF	**0.9**	0.75	**0.9**	0.67
Vote	**0.94**	0.82	**0.98**	0.88

TABLE 3: F-Measure values for *Context*.

	App	Hyp	Fw	HOS	GOS	Ch	Hw
KLD	0.8	0.76	0.85	0.88	0.76	0.76	0.73
CE	0.79	0.77	0.85	0.88	0.76	0.78	0.73
SVM	0.84	**0.95**	0.89	0.84	**0.93**	0.82	0.82
NB	0.74	0.57	0.73	0.76	0.7	0.69	0.68
DT	0.7	**0.95**	**0.95**	0.76	**0.95**	0.82	0.8
RF	0.68	0.76	0.82	0.81	0.88	0.7	0.73
Vote	0.87	**0.95**	**0.94**	0.84	**0.94**	**0.9**	0.85

TABLE 4: F-Measure values for *Impact Method*.

	Trust Failure	Context Escape	Auth. Bypass	MitM	Code Exec.
KLD	**0.91**	**0.94**	0.88	**0.96**	**0.94**
CE	**0.9**	**0.93**	0.86	**0.96**	**0.94**
SVM	0.67	0.79	0.84	**0.96**	**0.9**
NB	0.84	0.84	0.85	0.82	0.88
DT	**0.92**	**0.98**	**0.97**	**0.99**	**0.93**
RF	0.39	0.76	0.62	0.78	0.81
Vote	**0.93**	**0.94**	**0.92**	**1**	**0.94**

TABLE 5: F-Measure values for *Logical Impact*.

	Service Int.	Read	Write	Resource Rem.	Indirect Disc.	Privilege Esc.
KLD	0.86	0.86	0.83	0.88	**0.94**	0.78
CE	0.86	0.85	0.82	0.87	**0.94**	0.78
SVM	0.89	**0.94**	**0.95**	0.78	0.88	0.88
NB	0.84	0.71	0.76	0.83	0.83	0.71
DT	**0.97**	**0.92**	**0.94**	0.84	**0.94**	0.88
RF	0.74	0.87	0.88	0.64	0.82	0.82
Vote	**0.95**	**0.95**	**0.96**	0.84	**0.91**	0.87

TABLE 6: F-Measure values for *Mitigation*.

	ASLR	MFA	Sandboxed	HPKP HSTS	Physical Security
KLD	**0.91**	0.78	0.73	0.77	0.89
CE	**0.92**	0.76	0.73	0.77	0.88
SVM	**0.94**	0.85	0.88	0.84	**0.92**
NB	0.74	0.76	0.78	0.71	0.78
DT	**0.96**	**0.92**	**0.93**	**0.97**	**0.96**
RF	0.83	0.71	0.72	0.79	0.85
Vote	**0.96**	**0.92**	**0.92**	**0.92**	**0.96**

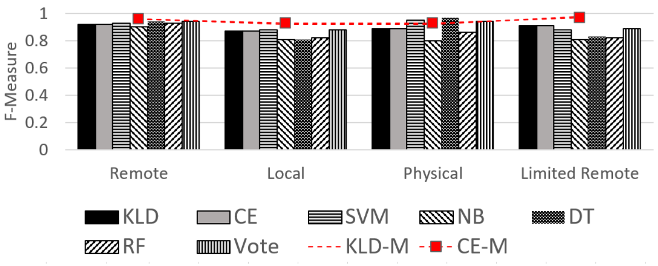


Fig. 6: F-Measure values of the IT and ML approaches while predicting *Attack Theater* with multi-class classification approach.

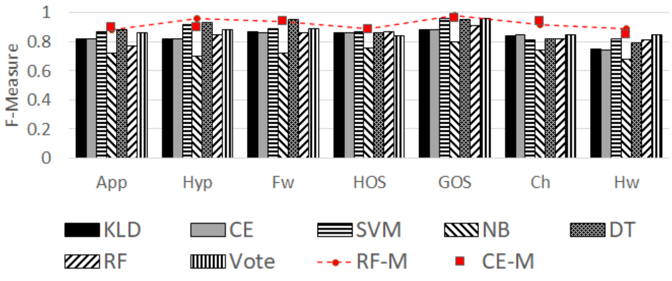


Fig. 7: F-Measure values of the IT and ML approaches while detecting *Context* with multi-class classification approach.

The results of the *Impact Method* noun group using a multi-class classification method are shown in Figure 8. With a single chance prediction, the average F-Measure values are higher than 0.90 for all noun group values, with all characterization methods except NB and RF. With the double-chance evaluation approach, increases ranging between 4% to 7% are observed in the average F-Measure values of different prediction models, except NB and DT, which stay at 0.84 and 0.96, respectively. 7% and 5% increased F-Measure values for RF (RF-M) and KLD (KLD-M) are shown with the dashed line and scattered square points in Figure 8.

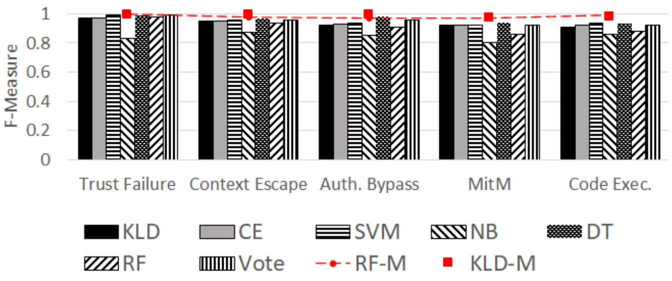


Fig. 8: F-Measure values of the IT and ML approaches while detecting the *Impact Method* with multi-class classification approach.

3.3 Overall Evaluation

The proposed methodology uses a context-based feature extraction and a set of ML and IT methods to characterize vulnerabilities based on the NIST VDO framework. Five VDO noun groups are used to characterize vulnerabilities for *Attack Theater*, *Context*, *Impact Method*, *Logical Impact*, and *Mitigation*. The overall average of the F-Measure values obtained in cross-validation in the binary characterization for the ML and IT methods are both 0.85. For multi-class characterization, the F-Measure values increase to 0.88 for both ML and IT methods. During binary characterization, the IT methods appear to be more consistent than the ML methods. The lowest F-Measure value observed with an IT method is 0.73. However, the performance of some ML techniques varies depending on the VDO noun group or label. The box plot of the F-Measure values observed for each ML and IT method is shown in Figure 9 for binary characterization. Figure 10 shows the box plot of the F-Measure values for multi-class characterization for each ML and IT method. The average F-Measure values obtained for noun group values are between 0.87 and 0.91 for all ML and IT methods, except NB, which has an average F-Measure value of 0.79. A paired t-test with $\alpha = 0.05$ is used to run statistical significance tests on the F-Measure values obtained with different ML and IT methods during binary and multi-class characterization. Figure 11 shows the results of the t-tests obtained for binary (A) and multi-class (B) characterization. In Figure 11, when a method in a row is compared with another method in a column, statistically significant positive and negative differences are represented with up and down arrows, respectively. The right arrow shows that the F-Measure values of the two methods (in the corresponding row and column) are comparable. During binary characterization, the F-Measure values obtained with Vote are statistically better than other methods except DT and all methods perform better than NB, except RF. The overall performances of the KLD, CE and SVM are comparable. In multi-class characterization, the F-Measure values of all methods are comparable, except NB and RF.

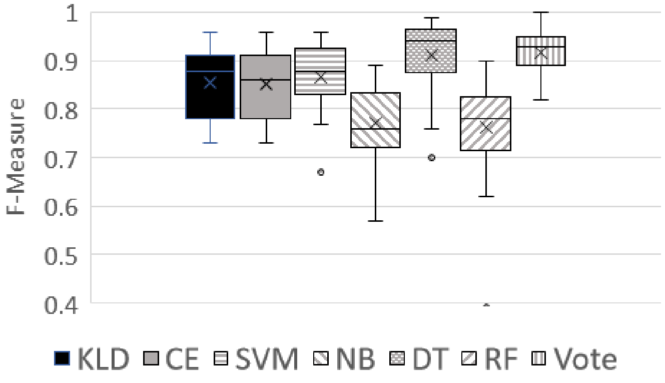


Fig. 9: The box plot of the F-Measure values of ML and IT methods for binary characterization.

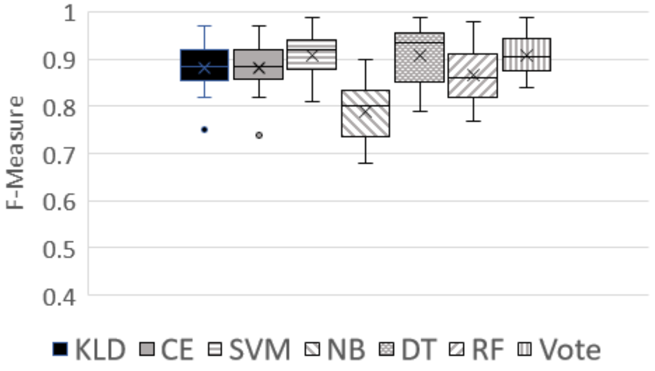
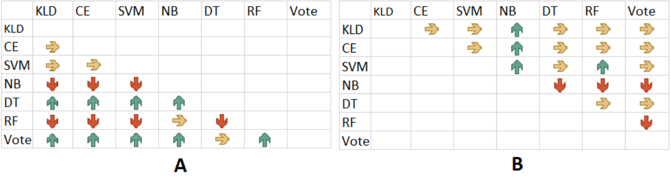


Fig. 10: The box plot of the F-Measure values of ML and IT methods for multi-class characterization.



	KLD	CE	SVM	NB	DT	RF	Vote
KLD							
CE	→						
SVM	→	→					
NB	↓	↓	↓				
DT	↑	↑	↑	↑			
RF	↓	↓	↓	→	↓		
Vote	↑	↑	↑	↑	→	↑	

A

	KLD	CE	SVM	NB	DT	RF	Vote
KLD		→	→	↑	→	→	→
CE			→	↑	→	→	→
SVM				↑	→	↑	→
NB					↓	↓	↓
DT						→	→
RF							↓
Vote							

B

Fig. 11: T-test results to compare F-Measure values in binary (A) and multi-class (B) characterization where each cell shows the difference when the method in the row is compared with the one in the column.

Model Performance in Cross Validation: During binary characterization, the performance of the IT methods is consistent. However, the performance of ML techniques varies depending on the VDO noun group or label (noun group value).

In multi-class characterization, the IT methods are more consistent than the ML methods. The average F-Measure values obtained for noun group values are between 0.87 and 0.91 for all ML and IT methods, except NB, with an average F-Measure value of 0.79.

4 Instrumentation: Real-time Mining and Characterization of Raw Vulnerability Data

To examine our automated vulnerability characterization in the context of vulnerability management workflow, we have instrumented a platform to curate vulnerability data in real-time, and used our characterization approach to describe them using NIST VDO and share vulnerabilities publicly. This platform aims to automate the manual vulnerability curation activities currently in place in NIST's NVD.

Through instrumentation we investigate two main research questions (1) whether this automated instrumentation would help curate and disseminate CVEs faster than current systems (e.g. NVD and MITRE CVE programs), and (2) can ML techniques reduce the manual effort for review and characterization of CVEs. In particular we will carry out a set of qualitative and quantitative studies to answer these two overall questions.

The reminder of this instrumentation section is organized as follows: sub-section 4.1 describes the instrumentation platform; sub-section 4.2 reports an empirical study of CVE disclosure timing analysis, therefore demonstrating that our proposed instrumentation outperforms NVD in terms of timely disclosure of vulnerabilities; sub-section 4.3 further evaluates the accuracy of our approach in a scenario simulating if it was deployed publicly for the use in the wild; sub-section 4.4 reports a qualitative study demonstrating the practical-significance of our approach in supporting the *analysts* in their daily job and reducing their effort in terms of time spent to review and characterize CVEs; lastly sub-section 4.5 demonstrate how the automated characterization can further assist the analysts by automatically generating CVE's severity scores–a labor-intensive activity that is performed manually today.

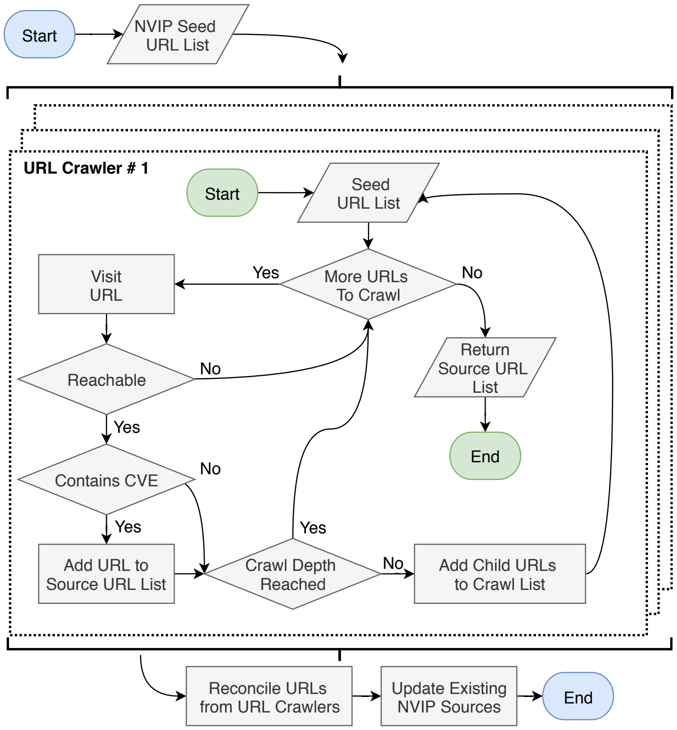


Fig. 12: The process flow of the NVIP Source URL Crawler.

4.1 Automated Web-Mining to Curate Raw Vulnerability Data (CVEs)

The instrumented platform includes several different components as illustrated in Figure 1.

CVE Source Crawler: The first component is the *CVE Source Crawler* responsible for continuously identifying and maintaining an updated list of potential CVE sources, *i.e.*, web sources that actively disclose software vulnerabilities. To curate vulnerability data as soon as they are disclosed, we compiled an extensive *seed* list of web-sources that disclose vulnerabilities. These seed lists contain *CNAs*[1], *security advisories*, and other public sources that disclose software vulnerabilities. To extend these web-sources, we developed *multi-thread CVE source crawler* module to help maintain an updated list of potential CVE sources. It takes a list of seed URLs as input and spawns multiple processes to scrape all these pages and all outgoing links from each page to reach additional potential sources. Source URL Crawler is a configurable software component that has a crawl depth parameter. It can continue to crawl all the links in each page until the configured depth is reached. Each page content is checked with regular expressions to test if a valid CVE ID is included or not. Only the pages that include a valid CVE ID are included in the final source list, and all other pages are ignored. Detected CVE sources are checked against the existing sources of the NVIP, and newly detected sources are added to the database. Meanwhile, CVE Source Crawler keeps track of the status of each source URL and removes it from the database if it cannot be reached anymore due to a variety of HTTP errors. Source Crawler can be scheduled to run as a service, to ensure all CVE sources are updated in a timely manner. The process flow of the Source URL Crawler is shown in Figure 12.

CVE Content Crawler The second component leverages various parsers to extract CVE descriptions from each web-source. These CVEs are then processed by NLP-Based Reconciling methods to analyze CVEs with the same identification number (CVE ID) and make sure the correct and most updated version of each CVE description is kept. Each source might be disclosing new CVEs, or updating existing ones multiple times in a week, day or even hour. CVE Content Crawler keeps track of the CVE contents at each source, to make sure that curated CVEs are up-to-date. It fetches CVE sources from the database and spawns multiple CVE crawlers to look at each source in parallel. The list of CVEs returned by different content crawler processes might have similar CVE instances (with the same CVE ID). Therefore, the outputs of these processes need to be reconciled to ensure that only the most recently updated instance of each CVE is included. Crawled CVEs are reconciled using the CVE reconciliation process. All crawled and reconciled CVEs are checked against the CVEs that exist in the database, to go through another reconciliation process and identify the ones that are new since the most recent run or have updated content.

CVE Reconciling: The most challenging problem for the CVE reconciliation algorithm is to decide whether an existing CVE description has to be updated or not. To address this challenge, a rule-based reconciliation method is developed which uses NLP to extract informative metrics from CVE descriptions. The method uses four characteristics, *i.e.*, the length of the CVE description, the number of sentences in the description, part-of-speech (PoS) diversity, and the number of unidentified PoS. Based on these characteristics, four metrics are developed to identify if the new CVE description is longer ("Longer"), it has less unidentified PoS ("Less Unknown"), it has more sentences ("More Sentences"), and it has more diverse PoS ("More Diverse"). Using these four metrics we decide if a CVE description needs to be updated as shown in Table 7.

TABLE 7: Truth table used for CVE description update rule set.

Less Unknown	Longer	More Sentences	More Diverse	**Update**
False	False	True	True	**True**
True	False	False	True	**True**
True	False	True	False	**True**
True	False	True	True	**True**
True	True	False	True	**True**
True	True	True	False	**True**
True	True	True	True	**True**

The last components of the system, leverage VDO attributes and historical vulnerability data to automatically characterize CVEs, calculate a severity score for each CVE. and carry out the CVE analysis described in Section 4.2.

4.2 Timing Analysis of CVE Curation

This section performs a CVE disclosure timing analysis to compare the CVE disclosure times of our instrumented approach with the NVD. We use automated and scheduled crawlers to find software vulnerabilities as soon as they are disclosed. Then, we use the characterization techniques to

[1] CVE Numbering Authorities

identify the attributes of the vulnerabilities. In an experiment, we measured the gaps between the time CVEs are found and characterized by our system and the time they first appear in the NVD. Between 5/23/2020 and 7/31/2020, our approach has detected 575 vulnerabilities earlier than NVD, with time gaps ranging from 7 to 95 hours. The distribution of the time gaps for these 525 vulnerabilities is shown in Figure 13.

For example, CVE-2020-13753 is a vulnerability that allows access outside the sandbox of `WebKitGTK and WPE WebKit` (https://webkitgtk.org/), prior to version 2.28.3 and was found at 7/11/2020 08:18:00. The same vulnerability was published on NVD at 07/14/2020 and included in the `NVD Data Feeds` (https://nvd.nist.gov/vuln/data-feeds) approximately 95 hours after the vulnerability was found by our system.

While NVD publishes a received CVE, still many attributes of the CVE may be incomplete. For instance, as of September 3rd 2020, the status of CVE-2020-24717 was 'Undergoing Analysis' and CVE details were missing. The vulnerability characterization is aimed to convert this pure manual process to a semi-automated process. Furthermore, characterization of CVEs can automate the generation of vulnerability severity score (CVSS) and other metrics [20]. Additional analysis showed that the developed proof-of-concept tool was able to identify 100% of CVEs in NVD in a significantly shorter amount of time. Furthermore, it was able to find additional CVEs that did not exist in NVD at all, because either they were reserved by a CVE Naming Authority or going through an analysis process before getting published.

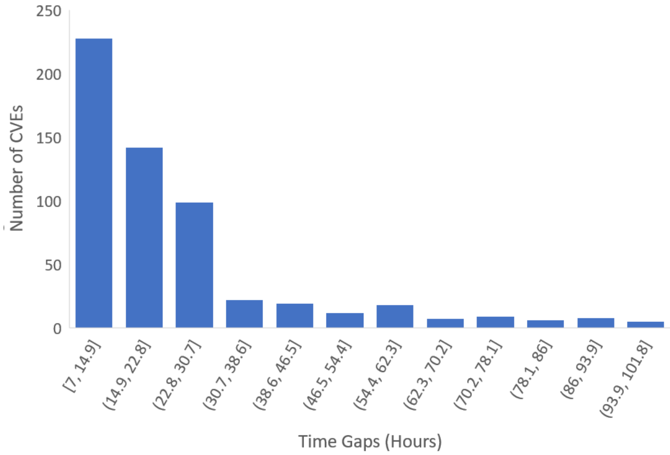


Fig. 13: The distribution of the time gaps (hours) for CVEs that were detected earlier than NVD between 5/23/2020 and 7/31/2020.

Rapid CVE Curation: Our initial results indicate that the proposed approach can detect and characterize vulnerabilities up to 95 hours earlier than NVD. It can significantly reduce the manual effort in the analysis process and reduce the time gap between the disclosure of CVEs and their publication.

4.3 Case Study: Model Performance on 2020 CVEs

The goal of this case study is to examine how our solution performs if it was deployed publicly and used on unseen data. We created a case study by pulling a set of recent CVEs from NVD (with CVE-ID $>$ 2020-14000) to measure the performance of the trained models using real-world CVE examples. The models were trained on the data collected earlier and tested on this new data-set. CVEs included in the case study were not used while training the models.

First, each CVE was labeled with the applicable noun group values in each of the five VDO domains. Then, 50 CVEs were selected from the labeled CVEs, with the common criteria of maintaining a balanced label distribution in each test set. Except some rarely observed noun group values like *Physical* and *Local* in *Attack Theater*, *Guest OS* in *Context* and *Resource Removal* in *Logical Impact*, all noun group values are represented with at least 15% of the test instances. Using each method, all CVEs in the case study were characterized by the binary or multi-class characterization methods described in Section 2.6. *Attack Theater*, *Context* and *Impact Method* noun groups were used to test multi-class characterization (Section 3.2). Therefore, the multi-class characterization with the single-chance and double-chance approaches were used to characterize CVEs for these groups. For the *Logical Impact* and *Mitigation* noun groups, the binary classification approach was used.

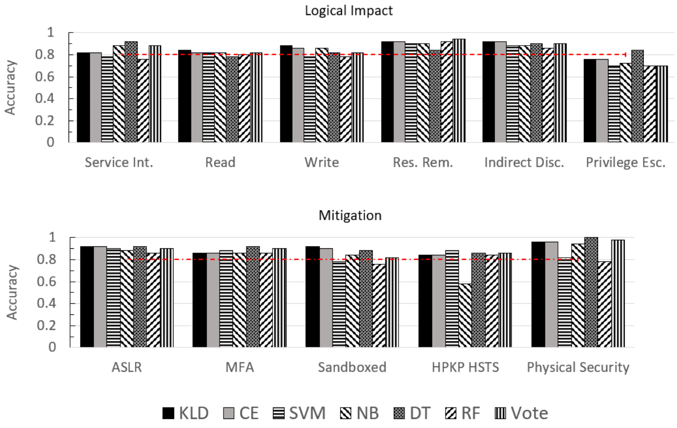


Fig. 14: The accuracy of different methods in the case study with binary characterization (80% levels are marked with a dashed red line).

Figure 14 shows the accuracy metrics of the ML and IT methods with binary characterization. Based on the overall average accuracy metrics across all noun group values, KLD and Vote achieve the best average accuracy of 0.88. Figure 15 shows the accuracy metrics of the ML and IT methods for multi-class characterization with single-chance and double-chance approaches. In the single-chance approach, KLD, CE and SVM methods achieve accuracy between 78% and 90%, and these accuracies increase up to 96% with the double-chance approach. With the single-chance approach, SVM, KLD, and CE achieve an average accuracy of 0.85, 0.81, and 0.81, respectively, across the three VDO noun groups. With the double-chance approach, these accuracies increase to 0.90, 0.92, and 0.92 for SVM, KLD, and CE, respectively.

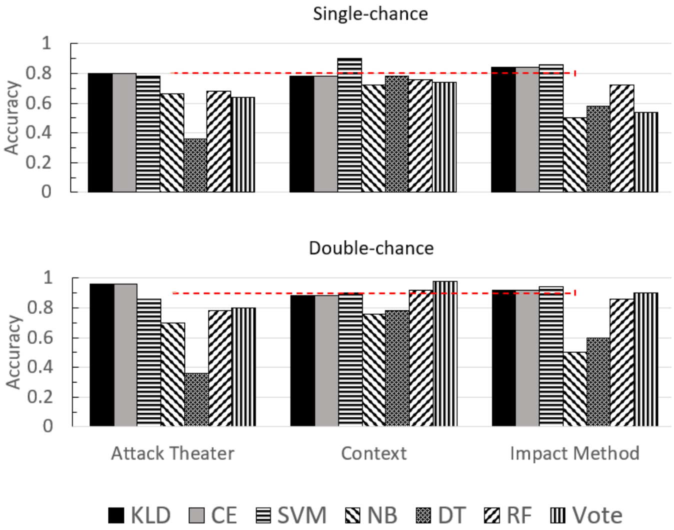


Fig. 15: The accuracy of different methods in the case study with multi-class characterization (80% and 90% levels are marked with a dashed red line).

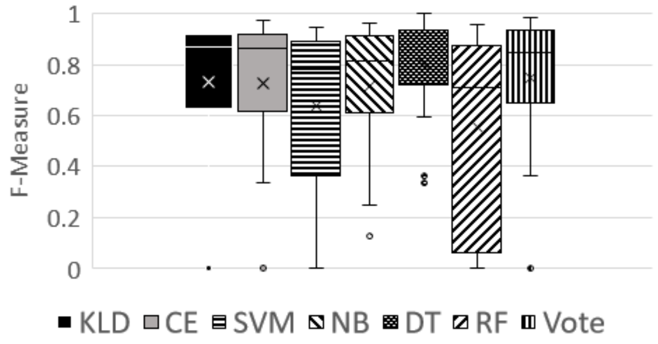


Fig. 16: The box plot of the F-Measure values of binary characterization in the case study.

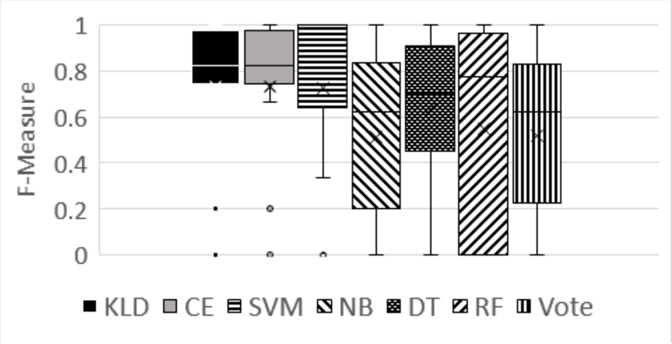


Fig. 17: The box plot of the F-Measure values of multi-class characterization in the case study.

To compare different characterization methods in the case study, we use the F-Measure values of the noun group values obtained during binary and multi-class characterization. Figure 16 and 17 show the box plots of the F-Measure values obtained for binary and multi-class characterizations, respectively. Figure 18 shows the statistically significant differences between the F-Measure values of each method during the case study. During binary characterization, for *Logical Impact* and *Mitigation*, DT models had a better performance with 10 folds cross-validation, however, they did not perform that well in the case study. The change in the performance of the DT model led to a loss in the performance of the ensemble voting classifier as well. The average F-Measure values of DT and Vote decreased from 0.93 and 0.92 to 0.80 and 0.75, respectively. The performance decrease of the DT models could be explained by the slight change in their underlying training data sets (due to using 100% of the data for training this time, instead of cross validation). In spite of pruning, changes in the training data of a decision tree model can cause a significant difference in the tree structure, which might lead to a performance variation [32], [33].

	KLD	CE	SVM	NB	DT	RF	Vote
KLD							
CE	➡						
SVM	⬇	⬇					
NB	➡	➡	⬆				
DT	➡	➡	⬆	➡			
RF	⬇	⬇	⬇	⬇	⬇		
Vote	➡	➡	⬆	➡	➡	⬆	

A

	KLD	CE	SVM	NB	DT	RF	Vote
KLD		➡	➡	⬆	➡	⬆	⬆
CE			➡	⬆	➡	⬆	⬆
SVM				⬆	➡	⬆	⬆
NB					➡	➡	➡
DT						➡	➡
RF							➡
Vote							

B

Fig. 18: T-test results to compare F-Measure values for binary (A) and multi-class (B) characterization in the case study, where each cell shows the difference when the method in the row is compared with the one in the column.

We observe a more consistent prediction performance with the IT methods during the case study. Figure 18 shows that both KLD and CE achieve comparable results with NB, DT, and Vote and perform better than SVM and RF during binary characterization. Similarly, they achieve comparable F-Measure values with SVM and DT and do significantly better than NB, RF, and Vote during multi-class (single chance) characterization. We confirm the same observation with the multi-class double-chance approach as well.

Model Performance on 2020 CVEs: The results of the *Model Performance* case study indicate that the trained models are able to characterize CVEs in the wild with high accuracy, and these models can be leveraged to decrease the manual CVE characterization efforts in the community significantly.

4.4 Qualitative Study: Effort Reduction in Manual CVE Review and Characterization

We conduct a human-subject study to investigate whether the automated CVE characterization approach can enhance the analysts productivity in CVE reviews and characterization. In particular, we will measure the amount of time spent to characterize CVEs for five studied VDO noun groups *Attack Theater*, *Context*, *Impact Method*, *Logical Impact*, and *Mitigation*. Six researchers (from the authors of the paper) who had up to 10 years of experience with CVE analysis and were familiar with the NIST Vulnerability Description Ontology used the descriptions of the 20 most recent CVEs pulled by automated crawlers of the developed tool to characterize them for five VDO noun groups. Included CVEs had varying sizes between 651 and 5752 characters (including spaces) and all characterizations were done using the CVE descriptions only, without searching for any additional information about them.

We conducted a one factor with two-treatment controlled experiment and followed a crossover design or within-group where each subject receives the two treatments, and we get repeated measures [34]. Another option was a parallel or between-group design where each subject receives only one treatment and we obtain independent measures. There are advantages and disadvantages to each design. For instance, the benefits of a crossover design are the elimination of the effects of confounding variables, such as experience, as each subject serves as his/her own matched control, and a higher statistical power with fewer subjects. On the other hand the parallel design minimizes the learning effects, a known challenge in controlled experiments on program comprehension [35]. In our study, there also be a third challenge related to treatment shift that may impact the mental model and productivity of participants. We chose crossover design for this qualitative case study.

We use various blocking techniques to control sources of variation that will reduce error variance. This include source such as participants experience, or learning biases because of the order of tasks or exposure to other vulnerability data.

To block the co-founding impact of software and cybersecurity experiences researchers were scored according to the amount of time they worked on (1) secure software development, (2) software vulnerabilities, (3) vulnerability characterization, and (4) the Vulnerability Description Ontology. Researchers were sorted according to their experience, where the average of the number of months spent on each of the aforementioned four areas is used to measure the experience of each subject. The case study was composed of two stages and two subject groups were created by including the first, third, and fifth subjects in the first user group U_1 and the second, fourth and sixth subjects in the second user group U_2, to make sure the average experience of each group is close.

To block the impact of learning biases due to the order of exposure to the vulnerability data and treatment, we reverse the order of treatment from one group to the other group. In such setting, the software security experience across two groups is balanced, while we do not believe the order of treatment will have an impact, we take a further action to minimize the impact of learning biased due to the order of treatment. Half of participants receive the treatment in one order and other half in another order.

Twenty CVEs derived from the developed tool were divided into two groups C_1 and C_2, and on each stage of the case study each user group worked on one CVE group using either a pure *manual characterization* or *semi-automated characterization* approach leveraging the trained AI/ML models. For the *semi-automated characterization*, the system was configured to use the multi-class characterization with Vote.

During Stage 1, researchers in U_1 were given the descriptions of the CVEs in C_1 and went through a pure manual characterization process where they recorded the time spent to read, understand and characterize each CVE for each of the five VDO noun groups. For each CVE and VDO noun group, researchers read the description of each CVE carefully, used their best judgment to select one or more relevant noun group values (labels) and recorded the amount of time elapsed while reading the CVE description and characterizing it for the noun group. Similarly, in addition to the descriptions of the CVEs in C_2, subjects in U_2 were provided with the set of labels that were predicted for each CVE in C_2. Each researcher reviewed the CVE descriptions in C_2 to verify the automatically assigned labels, and recorded the amount of time spent to finalize the characterization process for each CVE noun group pair. Researchers evaluated the automatically assigned labels based on their subjective judgment, therefore the amount of time spent varied depending on the subjects' judgment and the match between their labels and the automatically assigned ones. For example, if automatically assigned labels included a noun group value that the subject did not expect to see, then the verification process took a longer period of time to clarify if the assigned label was wrong or the subject was missing or ignoring a fact. During the second stage of the case study, this time subjects in U_1 used the *semi-automated characterization* approach to characterize CVEs in C_2 and subjects in U_2 used the *manual characterization* to characterize CVEs in C_1.

To have an overall insight about the value of the proposed automated approach in terms of characterization timing, the average of the time values recorded by each subject for each noun group are calculated for manual and semi-automated characterization. Figure 19 shows the comparison of the average times for the manual characterization process and the semi-automated one using the trained AI/ML models. Based on the average of the time values recorded by six subjects, we observe that a significant amount of time is saved when the proposed automated approach is used for CVE characterization. The percentage of time saved for the *Attack Theater*, *Context*, *Impact Method*, *Logical Impact*, and *Mitigation* noun groups are observed to be 48.4%, 44.7%, 51.5%, 37.0%, and 39.1%, respectively.

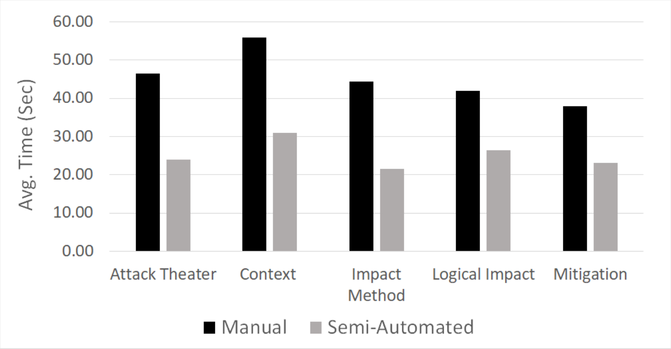


Fig. 19: Average characterization time of five noun groups for the manual CVE characterization process and the semi-automated one leveraging the trained AI/ML models.

To evaluate the performance of the manual and semi-automated characterization processes, we calculate the true positive and false negative counts for the noun group values assigned by each subject during the case study. Figure 20 shows the recall values of the two approaches for each noun group. We observe higher recall values for each noun group during the *semi-automated characterization*, where the subjects are provided with labels that are automatically assigned by the AI/ML models. The percentage of improvement in the recall values of the *Attack Theater*, *Context*, *Impact Method*, *Logical Impact*, and *Mitigation* noun groups are observed to be 5.1%, 12.6%, 8.8%, 15.8%, and 5.0%, respectively. A paired t-test with $\alpha = 0.05$ is used to run a statistical significance test on the recall values obtained for the noun group values during the manual and semi-automated characterization processes. The paired t-test result indicates that the difference between the recall values of the two processes is significant with a p value of 0.0009.

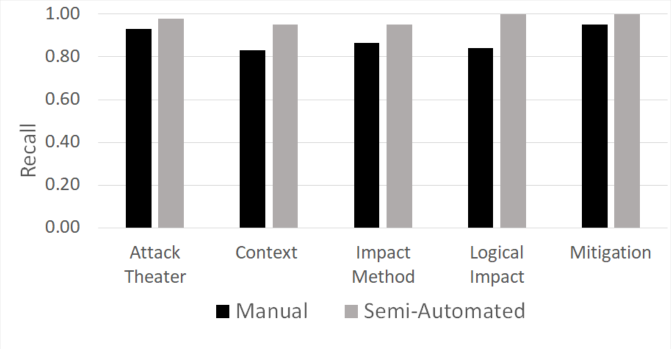


Fig. 20: Recall values for each noun group during the manual CVE characterization process and the semi-automated one leveraging the trained AI/ML models.

Manual Effort Reduction: The results of the qualitative study with security subject matter experts (SME) shows that when the proposed automated approach is used by security SMEs, up to 51.5% of the time spent for CVE characterization could be saved in comparison to a full manual process. Furthermore, the recall value of the characterization process increases up to 15.8%, making the effort more accurate.

4.5 Automated CVSS Scoring using the NIST Vulnerability Description Ontology

To further support the analysts in the manual review of CVEs to generate severity scores, we investigate whether our predicted VDO labels can be further used to score the severity of published CVEs automatically. This scoring capability is currently limited because the draft version of the VDO is not expressive enough to provide a thorough mapping from VDO to CVSS. However, a goal of VDO is to support such mappings and thus the finalized version to be developed by NIST may remedy the weaknesses revealed by this work.

Our high level approach is to identify available mappings from the VDO labels to CVSS attributes. Then, for a set of CVEs for which we've identified VDO labels, we create partial CVSS vectors (where wildcards are used for some attributes and some attribute value sets). We then use historical CVE data from the NVD to find all CVEs that match the partially specified vectors. Lastly, we estimate a CVSS score for each partial vector by taking the median CVSS score from all matching CVEs.

More specifically, we first define a set of rules to map the five VDO noun groups [20] studied in this paper to the CVSS version 3.1 [36] base metrics. For each of the entries in the CVSS base metric group, the following set of rules are used to establish the mapping:

- **Attack Vector (AV)**: The VDO labels under the *Attack Theater* noun group are *Remote*, *Limited Remote*, *Local*, and *Physical*). These four map respectively to the four CVSS *Attack Vector* metric values: Network (AV:N), Adjacent (AV:A), Local (AV:L), and Physical (AV:P).
- **Attack Complexity (AC)**: According to the CVSS specification, the complexity of an attack can either be Low (AC:L) or High (AC:H). The default value of the *Attack Complexity* is assumed to be AC:L, however when a *Man in the Middle* attack is identified the attack complexity is set to AC:H. This is the only VDO value that appears to affect AC.
- **Scope (S)**: The scope of an attack is assumed to be Unchanged (S:U) by default. A scope value of Changed (S:C) is assigned when a *Context Escape* impact method or *Sandboxed* mitigation tactic is detected.
- **Confidentiality (C), Integrity (I), Availability (A)**: The CVSS impact metrics of Confidentiality, Integrity and Availability (CIA) are each allowed to have one of three values: None (N), Low (L), and High (H). For our work, we add a value of Unknown (U) which indicates that the metric is not N but that the VDO does not enable us to distinguish between L and H. Each CIA metric is assumed to be N unless there is evidence otherwise.
  We map each CIA metric to a distinct metric specific set of logical impacts: *Read* and *Indirect Disclosure* for C, *Write* and *Resource Removal* for I, and *Service Interrupt* for A. U is tentatively assigned for a CIA metric if at least one of the metric specific set of logical impacts applies. A U value is upgraded to an H if the VDO logical impact *Privilege Escalation* also applies. A logical impact of *Privilege Escalation* (with no other logical impacts specified) upgrades a N or U value to H as the VDO specification states that *Privilege Escalation* by itself implies all other logical impacts.
- **Privileges Required (PR)** and **User Interaction (UI)**: For the PR and UI base metrics of the CVSS specification, no mapping is defined. This is because the studied VDO noun groups do not convey the level of required privileges and the requirement of user interaction. The values of the CVSS metrics that had no mapping were set to 'X' to indicate no value was assigned.

During the case study, the system was configured to use the multi-class characterization with ensemble voting, and all CVEs published during 2020 and 2021 were characterized automatically to predict VDO noun group values for each CVE. CVEs published before CVE-2020-14000 were excluded from the case study, because they were included in the training data sets. Given a CVE description, the labels for the five studied noun groups were predicted using the trained AI/ML models and the aforementioned mapping rules were used to generate a partial CVSS feature vector for each CVE. For example, if *Remote*, *Man in the Middle*, *Write* and *Mitigation* labels are predicted for the *Attack Theater*, *Impact Method*, *Logical Impact*, and *Mitigation* noun groups, then the CVSS vector $[N, H, X, X, C, N, U, N]$ is created based on the defined rules, where each entry in the vector represents the *Attack Vector*, *Attack Complexity*, *Privileges Required*, *User Interaction*, *Scope*, *Confidentiality*, *Integrity*, and *Availability* component, respectively. Please note that the PR and UI metrics are set to X, because no mapping is defined, and the value of the Integrity is set to U to indicate the ambiguity of the level of integrity violation. Once a partial CVSS vector is generated from the VDO labels derived from a CVE description, it was matched against the CVSS vectors of the previously published CVEs between 2015 and 2019 that match the partial vector and that had a score assigned by NVD. The median CVSS score of the matching CVEs is used to assign the severity score of the CVE. Median was used instead of mean to minimize the sum of the distances between the chosen value and all applicable CVSS values.

Our analysis showed that out of the 6777 CVEs published in 2020 (after CVE-2020-14000) and 2021 (up until 2/25/2021), 317 did not have a base CVSS score assigned by NVD. For each of the remaining 6460 CVEs, the partial CVSS vector generated from VDO is matched with the CVSS vectors of CVEs published between 2015 and 2019. Figure 21 shows the distribution of the 6460 CVSS scores assigned by the proposed automated approach and the NVD. The mean of the CVSS scores assigned by our method and NVD are 6.8 and 7.2, respectively. Furthermore, the mean absolute error of the scores assigned by our approach is found to be 1.6 when compared to the scores assigned by NVD. This proof of concept case study provides promising results and shows that automated CVSS score calculation could be possible. Please note that the performance of this case study could be improved further if additional VDO noun group values enabled more thorough coverage of the CVSS metrics (especially for the impact of an attack) in the next releases of VDO.

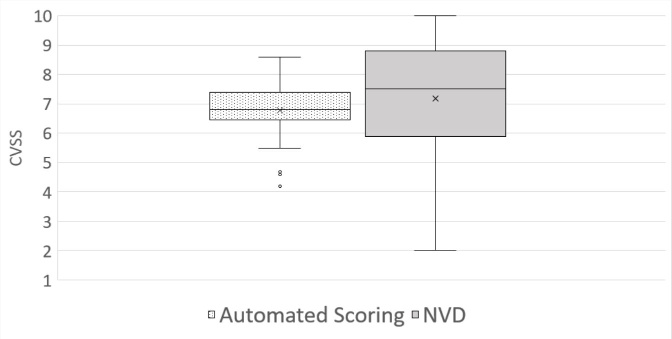


Fig. 21: The box plot showing the distribution of the median CVSS scores calculated by the described automated approach and the scores assigned by NVD.

VDO-based Automated CVSS Calculation: The results of the CVE scoring case study shows that predicted VDO labels can be used to score the severity of CVEs automatically. For 6460 CVEs analyzed during the case study, the mean values of the CVSS scores assigned by the proposed automated scoring approach and the NVD are found to be 6.8 and 7.2, respectively.

In summary, Section 4 first briefly described an automated system developed to mine the web and curate raw vulnerability data and then provided a set of quantitative and qualitative studies to showcase its contributions by:

1) Testing the system for 68 continuous days to perform a timing analysis for CVE curation and show that CVEs could be found and curated earlier than NVD that heavily rely on manual processes.
2) Monitoring the system's performance on real-life CVEs to simulate its use by security analysts and show that CVEs with free-form vulnerability descriptions can be characterized automatically with high accuracy.
3) Demonstrating the effort reduction brought by the developed system. A human-subject study was performed to show that a significant portion of the time that is currently spent on manual CVE curation could in fact be saved.
4) Conducting a quantitative study to show the developed system can be used for automated CVSS severity scoring.

5 Use Cases

Semi-Automated Analysis of Vulnerability Reports: Vulnerability reports, bug reports, and issue tracking tickets exist in a free text format. Automated characterization methods can extract valuable information from textual reports to easily interpret them and draw conclusions from them. While in this work we only applied the characterization techniques to CVE data, our approach can also be applied to other sources of information such as bug reports, vulnerability reports, developer discussions forums, and any other textual artifact.

Enable automated analysis using metrics such as the Common Vulnerability Scoring System (CVSS): Conversion of CVE description into VDO ontology will enable developers to obtain an estimated value for vulnerability CVSS scores on demand. NIST provides a method that describes how to map VDO noun-groups to CVSS scores and provides the reasoning for the mapping [20]. Using the automated vulnerability characterization with this mapping, it is possible to build a just-in-time CVSS score generation method.

Support CNAs: NVD relies on voluntary submissions by various CNAs, however some of the CNAs do not have adequate resources, guidelines, or tools to establish a sustainable submission method. A semi-automated approach will reduce the cost of describing and reporting software vulnerabilities for these CNAs.

Help Developers Characterize and Reason about Security Bugs: The proof of concept system developed during this work will provide the minimum information needed to properly inform all the stake-holders and facilitate the sharing of vulnerability information across language barriers.

Rapid Access to Vulnerability Data: The system developed for this work discovers and characterizes vulnerabilities as soon as they are disclosed. The real-time nature of this process can help many organizations access the newest vulnerabilities and make informed decisions in-time.

6 Related Work

Accurate vulnerability information is crucial for program repair [5], [37], vulnerability modeling/prediction/detection [38]–[40], exploit generation [41], and conducting empirical studies [1], [2], [14], [42], [43]. Security practitioners need vulnerability information in a timely manner to patch their systems properly [1]–[5], [44], [45]. Prior works have studied and enumerated the challenges of aggregating vulnerability information in an automated fashion, to allow reasoning [44]–[47] as well as create a common vocabulary for sharing vulnerability data [48], [49]. Currently, there are many security-related databases that contain information about vulnerabilities, *i.e.*, descriptions, related vendor advisories, patches, exploits, root causes, and consequences. Examples of these databases are Exploit DB [11], Security Focus [10], VulnDB [9], IBM X-Force [50], VulnCode-DB [8] and NVD [7]. However, vulnerability data in these systems may not always provide all information needed to conduct empirical studies. For instance, ExploitDB is focused on providing code snippets for breach exploits, whereas NVD lacks such information built-in. Therefore, security researchers often create their own datasets, but these datasets are not always reusable by other researchers [51], [52].

NVD is widely used as the main source of vulnerability information in previous research, motivating empirical studies on the accuracy and consistency of information available in NVD [13], [16], [17], [53], [54]. Dong *et al.* [17] described VIEM, an approach to detect vulnerable software names and their versions from free text and used it to examine NVD's vulnerability information consistency. They found that NVD contains incorrect information about vulnerable software versions. Similarly, Nguyen and Massaci [53] studied CVEs associated with Google Chrome, and found non-negligible errors in NVD's vulnerable software versions data that could affect conclusions drawn in empirical studies that rely on it. Zhang *et al.* [16] explored the use of NVD's data to perform vulnerability prediction. They found difficulty in building prediction models that perform well based on NVD data. Among the reasons identified in this work are data quality problems regarding affected vulnerable software versions in NVD and missing information for CVE instances. Dongliang *et al.* [13] scrutinized 368 vulnerability reports to quantify their reproducibility. They found that one single source of vulnerability information is not enough for reproducibility, because it is often incomplete and forces security professionals to manually debug and guess, to infer missing information.

Because of the data quality problems in the existing vulnerability databases, some prior works focused on creating vulnerability datasets that researchers could reuse in their work. Ponta *et al.* [55] described a dataset of vulnerability fixes, curated by extracting CVEs from NVD and performing a manual analysis of these reports in order to identify the commit(s) that fix the CVEs. Similarly, Fan *et al.* [56] released a dataset of commit fixes for C/C++ programs. Gkortzis *et al.* [57] described VulinOSS, a dataset of vulnerabilities that correlates software metrics from 8,694 open-source software with their particular vulnerabilities. Namrud *et al.* [58] presented AndroVul, a dataset of vulnerabilities in Android apps to be used as a benchmark for security research. Although these datasets are useful for researchers and practitioners, they are limited in terms of the underlying programming language/domain of the sample programs (e.g., C/C++, Android apps) as well as the vulnerability information types they cover (e.g., commit fixes, software metrics).

Other prior works focused on investigating some of the problems observed in the current vulnerability management systems, attempting to automatically characterize vulnerabilities [59]–[65]. Some works investigated approaches to characterize specific attributes related to the life-cycle of a vulnerability (i.e., time of introduction, disclosure, and patch) [66]–[68] as well as their severity [18], [69], [70]. Joshi *et al.* [60] attempt to partially solve the problem of vulnerability characterization by proposing the use of techniques that automatically extract a limited set of security entities from free-text. However, an automated end-to-end approach is needed to characterize vulnerabilities and reduce the manual effort needed for CVE management. Some prior research studies attempt to automate the process of accurately identifying all software product versions that are affected by a vulnerability [62], [63]. However, these automated processes of tracing vulnerable software releases suffered from weaknesses that result in a high number of false positives. In a relatively recent study [61], CVEs derived from a vulnerability report platform are used to categorize vulnerabilities, predict their risk level, and identify solutions required to address them. However, vulnerabilities were derived from a single source, and the generated data set was very limited, *i.e.*, only 39,417 vulnerability entries in 16 categories were considered. A more comprehensive approach is needed to design and develop a continuously running automated platform that considers multiple vulnerability sources. Furthermore, it should provide intelligent analysis and characterization services to the software community, identifying the vulnerability context, impacted target assets, attack methods, and potential mitigation techniques based on a well-known vulnerability ontology.

7 Threats to Validity

In this section, we briefly summarize the internal, external, and construct validity threats and the measures taken to mitigate each threat [71].

An *internal* validity threat occurs when the cause-effect relationship between the dependent and independent variables are not trustworthy. To help mitigate internal validity threats, first we selected a set of real-life CVEs and used CVE descriptions as the sole source of context-aware feature extraction and vectorization. We used a variety of machine learning and information theoretical methods to cross-check our findings, and applied cross validation to report empirical findings.

External validity determines how well the results of a study can be generalized, and it is threatened when the results observed on one data set are not applicable to others. To mitigate external validity threats, first we collected data from a diverse set of real-life CVEs which report real vulnerabilities from a variety of domains. While evaluating whether the process of vulnerability curation and characterization can be improved using ML and information retrieval, we used cross-validation during quantitative analysis and a cross-over design approach during qualitative studies to minimize potential biases. However, one of the limitations of our study is the size of the datasets used during the quantitative and qualitative analyses. Since manual CVE characterization is a time-consuming and expensive process, we used a small set of CVEs during these analyses to demonstrate the practical contribution of the proposed approach. More experiments with larger sets of more diversified CVEs might be needed to cross-check our findings.

Construct validity defines how well an experiment can measure its claims and can be threatened by a variety of factors, including errors in the experiment pipeline and biases in the experiment design. To mitigate the construct validity threats, we automated all experiment steps including CVE pre-processing, feature extraction, vectorization, model training and testing. To mitigate any potential biases, we followed a cross-over design approach during the qualitative analyses. However, it should be noted that predicted CVE noun group values are dependent on the natural language description of the underlying CVE, and the wording of the vulnerabilities matters. To ensure repeatability, we used the CVE descriptions from the NVD, but an AI/ML model may predict different noun group values for the same CVE if its description is pulled from a different vulnerability database. Therefore, it could be good to perform the same set of experiments on the same set of CVEs pulled from different CNAs. Furthermore, we used the default parameter set in Weka to carry out the ML experiments. Conducting parameter tuning may improve the performance of some ML algorithms (*e.g.*, SVM) and that may boost the performance of the voting algorithm as well. Therefore, the comparison between the ML and IT methods may produce slightly different results when model parameters are tuned. And finally, we used a threshold value of $\rho = 0.7$ (and not 0.5, like classifiers do by default) during our experiments for multi-class CVE characterization. This was chosen heuristically to increase the probability of seeing the second or even the third predictions. Using a different threshold might affect the performance of multi-class characterization. The script used to run the experiments will be shared in the GitHub repository and different ρ values could be tested by future works.

8 Conclusion

This paper uses novel Natural Language Processing (NLP), Machine Learning (ML), and Information Theoretical (IT) methods to show the feasibility of automated CVE collection and characterization. It develops a configurable, scalable, and portable proof of concept experimental system which crawls CVEs from security bulletins, advisories, exploit databases, issue tracking systems, and provides automated vulnerability characterization services.

It uses the public NIST's VDO framework to characterize CVEs and help unify vulnerability communication in the security community. For each CVE, it identifies where an attack comes from, the vulnerability context, impact methods used in the exploits, potential consequences (logical impacts) and mitigation strategies. Five researchers followed a systematic methodology and spent 3000+ person-hours to generate labeled CVE data sets for five domains (noun groups) in the VDO framework. Using conventional ML classifiers and novel IT methods, binary and multi-class prediction models are built to characterize CVEs in five domains and provide automated vulnerability intelligence capabilities.

The performance of a set of conventional ML methods and novel IT approaches are compared for binary and multi-class characterization. First, a ten folds cross-validation approach is used to test each method, and then a case study is created to evaluate the performance of each method on real-life CVE examples. Experiment results indicate that CVEs can be characterized instantly, with relatively high F-Measure values. Our analysis reveals that the proposed *entropy-based* IT technique achieves similar or even better performance compared to the conventional ML classifiers. A second case study is created to measure the value of the proposed approach in terms of the amount of time spent in the characterization process. The case study results indicate that the proposed methodology could save up to 47% of the time spent for vulnerability characterization and could significantly speed up the CVE publication process.

Our vectorization method combines TF-IDF with n-gram to be able to capture contextual information about each term. In particular, as we described earlier, this is important for the security domain that often context plays an important role in reasoning. This paper demonstrates the possibility of automating a task that today is done manually. Given the current analysis pipeline and the promising results, we demonstrate the practical significance of the approach. Future research could improve the performance of the algorithms further by leveraging word2vec models trained on a domain-specific corpus and using tuned parameters for ML methods.

9 Acknowledgments

This work was partially funded by the U.S. Department of Homeland Security, contract 70RSAT19CB0000020.

References

[1] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson, "You've got vulnerability: Exploring effective vulnerability notifications," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1033–1050.

[2] C. Sabottke, O. Suciu, and T. Dumitras, "Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 1041–1056. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sabottke

[3] F. Li, G. Ho, E. Kuan, Y. Niu, L. Ballard, K. Thomas, E. Bursztein, and V. Paxson, "Remedying web hijacking: Notification effectiveness and webmaster comprehension," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 1009–1019.

[4] M. Vasek and T. Moore, "Do malware reports expedite cleanup? an experimental study." in *5th Workshop on Cyber Security Experimentation and Test (CSET 12)*. Bellevue, WA: USENIX Association, Aug. 2012. [Online]. Available: https://www.usenix.org/conference/cset12/workshop-program/presentation/Vasek

[5] A. Machiry, N. Redini, E. Camellini, C. Kruegel, and G. Vigna, "Spider: Enabling fast patch propagation in related software repositories," in *2020 IEEE Symposium on Security and Privacy (SP). IEEE*, 2020.

[6] Z. Huang, M. DAngelo, D. Miyani, and D. Lie, "Talos: Neutralizing vulnerabilities with security workarounds for rapid response," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 618–635.

[7] National Vulnerability Database, "Nvd dashboard," https://nvd.nist.gov/general/nvd-dashboard#, 2018, (Accessed on 05/04/2018).

[8] R. Habalov and T. Schmid, "Vulncode-db," https://www.vulncode-db.com/, (Accessed on 09/02/2020).

[9] "Vulndb," https://vulndb.cyberriskanalytics.com/, (Accessed on 09/01/2020).

[10] "Securityfocus," https://www.securityfocus.com/, (Accessed on 09/01/2020).

[11] "Exploit database - exploits for penetration testers, researchers, and ethical hackers," https://www.exploit-db.com/, (Accessed on 09/01/2020).

[12] National Institute of Standards and Technology (NIST), "National vulnerability database (nvd)," https://nvd.nist.gov, 2020, (Accessed on 10/29/2020).

[13] D. Mu, A. Cuevas, L. Yang, H. Hu, X. Xing, B. Mao, and G. Wang, "Understanding the reproducibility of crowd-reported security vulnerabilities," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 919–936. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/mu

[14] F. Massacci, "Which is the right source for vulnerabilities studies? an empirical analysis on mozilla firefox," in *In Proceedings of MetriSec'10*. Citeseer, 2010, pp. 1–8.

[15] F. Farahmand, S. Navathe, P. Jr, and G. Sharp, "Managing vulnerabilities of information systems to security incidents," in *Proceedings of the 5th international conference on Electronic commerce*, 09 2003, pp. 348–354.

[16] S. Zhang, D. Caragea, and X. Ou, "An empirical study on using the national vulnerability database to predict software vulnerabilities," in *International Conference on Database and Expert Systems Applications*. Springer, 2011, pp. 217–231.

[17] Y. Dong, W. Guo, Y. Chen, X. Xing, Y. Zhang, and G. Wang, "Towards the detection of inconsistencies in public security vulnerability reports," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 869–885. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/dong

[18] J. Ruohonen, "A look at the time delays in cvss vulnerability scoring," *Applied Computing and Informatics*, vol. 15, no. 2, pp. 129–135, 2019.

[19] B. Ladd, "The race between security professionals and adversaries," https://www.recordedfuture.com/vulnerability-disclosure-delay/, June 2017, (Accessed on 09/02/2020).

[20] H. Booth, "Draft NISTIR 8138, Vulnerability Description Ontology (VDO)," National Institute of Standards and Technology (NIST), Tech. Rep., 2016.

[21] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 03 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[22] "Apache OpenNLP," http://opennlp.apache.org, 2011, (Accessed on 09/02/2020).

[23] M. Mirakhorli and Jane Cleland-Huang, "Detecting, tracing, and monitoring architectural tactics in code," *IEEE Trans. Software Eng.*, 2015. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TSE.2015.2479217

[24] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[25] J. Platt, "Fast training of support vector machines using sequential minimal optimization," *Advances in Kernel Methods: Support Vector Learning*, pp. 185–208, 02 1999.

[26] D. Jurafsky and J. H. Martin, *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River, N.J.: Pearson Prentice Hall, 2009. [Online]. Available: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y

[27] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, ser. UAI'95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, p. 338–345.

[28] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[29] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, p. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[30] L. I. Kuncheva, *Bagging and Boosting*. John Wiley & Sons, Ltd, 2004, ch. 7, pp. 203–235. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471660264.ch7

[31] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques*, 4th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.

[32] R. Li and G. G. Belford, "Instability of decision tree classification algorithms," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 570–575. [Online]. Available: https://doi.org/10.1145/775047.775131

[33] K. Dwyer and R. Holte, "Decision tree instability and active learning," in *Machine Learning: ECML 2007*, J. N. Kok, J. Koronacki, R. L. d. Mantaras, S. Matwin, D. Mladenič, and A. Skowron, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 128–139.

[34] R. Conradi and A. I. Wang, *Empirical Methods and Studies in Software Engineering: Experiences from Esernet*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003.

[35] J. Quante, "Do dynamic object process graphs support program understanding? - a controlled experiment." in *2008 16th IEEE International Conference on Program Comprehension*, June 2008, pp. 73–82.

[36] "Common Vulnerability Scoring System version 3.1: Specification Document," https://www.first.org/cvss/specification-document, (Accessed on 02/25/2021).

[37] Z. Huang, D. Lie, G. Tan, and T. Jaeger, "Using safety properties to generate vulnerability patches," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 539–554.

[38] F. Xiao, J. Zhang, J. Huang, G. Gu, D. Wu, and P. Liu, "Unexpected data dependency creation and chaining: A new attack to sdn," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 264–278.

[39] 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*. IEEE, 2012, pp. 75–81.

[40] D. Brumley, J. Newsome, D. Song, Hao Wang, and Somesh Jha, "Towards automatic generation of vulnerability-based signatures," in *2006 IEEE Symposium on Security and Privacy (S P'06)*, 2006, pp. 15–16.

[41] D. Brumley, P. Poosankam, D. Song, and J. Zheng, "Automatic patch-based exploit generation is possible: Techniques and implications," in *2008 IEEE Symposium on Security and Privacy (SP 2008)*, 2008, pp. 143–157.

[42] T. Scholte, D. Balzarotti, and E. Kirda, "Have things changed now? an empirical study on input validation vulnerabilities in web applications," *Computers & Security*, vol. 31, no. 3, pp. 344–356, 2012.

[43] ——, "Quo vadis? a study of the evolution of input validation vulnerabilities in web applications," in *International Conference on Financial Cryptography and Data Security*. Springer, 2011, pp. 284–298.

[44] Madhusudhanan Chandrasekaran, Mukkarram Baig, and Shambhu Upadhyaya, "Avare: aggregated vulnerability assessment and response against zero-day exploits," in *2006 IEEE International Performance Computing and Communications Conference*, 2006, pp. 8 pp.–610.

[45] Sufatrio, R. H. Yap, L. Zhong *et al.*, "A machine-oriented integrated vulnerability database for automated vulnerability detection and processing," *Large Installation System Administration (LISA)*, 2004.

[46] A. Arnold, B. M. Hyla, and N. C. Rowe, "Automatically building an information-security vulnerability database," in *Proc of 2006 IEEE Workshop on Information Assurance US Military Academy*, 2006.

[47] V. Mulwad, W. Li, A. Joshi, T. Finin, and K. Viswanathan, "Extracting information about security vulnerabilities from web text,"

in *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, vol. 3, 2011, pp. 257–260.

[48] Z. Syed, A. Padia, T. Finin, L. Mathews, and A. Joshi, "Uco: A unified cybersecurity ontology," *UMBC Student Collection*, 2016.

[49] M. Iannacone, S. Bohn, G. Nakamura, J. Gerth, K. Huffer, R. Bridges, E. Ferragut, and J. Goodall, "Developing an ontology for cyber security knowledge graphs," in *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, 2015, pp. 1–4.

[50] IBM, "IBM X-Force Exchange," https://exchange.xforce.ibmcloud.com/, (Accessed on 09/01/2020).

[51] M. Zheng, H. Robbins, Z. Chai, P. Thapa, and T. Moore, "Cybersecurity research datasets: taxonomy and empirical analysis," in *11th USENIX Workshop on Cyber Security Experimentation and Test (CSET 18)*, 2018.

[52] C. Grajeda, F. Breitinger, and I. Baggili, "Availability of datasets for digital forensics–and what is missing," *Digital Investigation*, vol. 22, pp. S94–S105, 2017.

[53] V. H. Nguyen and F. Massacci, "The (un) reliability of NVD vulnerable versions data: An empirical experiment on google chrome vulnerabilities," in *Proceedings of the 8th ACM SIGSAC symposium on Information, Computer and Communications Security*. ACM, 2013, pp. 493–498.

[54] A. Ozment and S. E. Schechter, "Milk or wine: does software security improve with age?" in *USENIX Security Symposium*, vol. 6, 2006.

[55] S. E. Ponta, H. Plate, A. Sabetta, M. Bezzi, and C. Dangremont, "A manually-curated dataset of fixes to vulnerabilities of open-source software," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 383–387.

[56] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A c/c++ code vulnerability dataset with code changes and cve summaries," in *2020 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2020, pp. 383–387.

[57] A. Gkortzis, D. Mitropoulos, and D. Spinellis, "Vulinoss: A dataset of security vulnerabilities in open-source systems," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 18–21. [Online]. Available: https://doi.org/10.1145/3196398.3196454

[58] Z. Namrud, S. Kpodjedo, and C. Talhi, "Androvul: a repository for android security vulnerabilities," in *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering*, 2019, pp. 64–71.

[59] G. Huang, Y. Li, Q. Wang, J. Ren, Y. Cheng, and X. Zhao, "Automatic classification method for software vulnerability based on deep neural network," *IEEE Access*, vol. 7, pp. 28 291–28 298, 2019.

[60] 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 (ICSC)*. IEEE, 2013, pp. 252–259.

[61] X. Zhang, H. Xie, H. Yang, H. Shao, and M. Zhu, "A general framework to understand vulnerabilities in information systems," *IEEE Access*, vol. 8, pp. 121 858–121 873, 2020.

[62] C. Cabrey, "Identifying the presence of known vulnerabilities in the versions of a software project," Master's thesis, Rochester Institute of Technology, 2016.

[63] V. H. Nguyen, S. Dashevskyi, and F. Massacci, "An automatic method for assessing the versions affected by a vulnerability," *Empirical Software Engineering*, vol. 21, no. 6, pp. 2268–2297, 2016.

[64] X. Gong, Z. Xing, X. Li, Z. Feng, and Z. Han, "Joint prediction of multiple vulnerability characteristics through multi-task learning," in *2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2019, pp. 31–40.

[65] D. Gonzalez, H. Hastings, and M. Mirakhorli, "Automated characterization of software vulnerabilities," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 135–139.

[66] W. A. Arbaugh, W. L. Fithen, and J. McHugh, "Windows of vulnerability: a case study analysis," *Computer*, vol. 33, no. 12, pp. 52–59, Dec 2000.

[67] A. Jumratjaroenvanit and Y. Teng-amnuay, "Probability of attack based on system vulnerability life cycle," in *2008 International Symposium on Electronic Commerce and Security*, Aug 2008, pp. 531–535.

[68] R. Wita, N. Jiamnapanon, and Y. Teng-Amnuay, "An ontology for vulnerability lifecycle," in *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*. IEEE, 2010, pp. 553–557.

[69] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale vulnerability analysis," in *Proceedings of the 2006 SIGCOMM Workshop on Large-scale Attack Defense*, ser. LSAD '06. New York, NY, USA: ACM, 2006, pp. 131–138. [Online]. Available: http://doi.acm.org.ezproxy.rit.edu/10.1145/1162666.1162671

[70] H. Holm and K. K. Afridi, "An expert-based investigation of the common vulnerability scoring system," *Computers & Security*, vol. 53, pp. 18 – 30, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404815000620

[71] P. Runeson and M. Hoest, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, pp. 131–164, 2009.