A Survey on Data-driven Software Vulnerability Assessment and Prioritization

TRIET H. M. LE and HUAMING CHEN, CREST - The Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia
M. ALI BABAR, CREST - The Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia and Cyber Security Cooperative Research Centre, Australia

Software Vulnerabilities (SVs) are increasing in complexity and scale, posing great security risks to many software systems. Given the limited resources in practice, SV assessment and prioritization help practitioners devise optimal SV mitigation plans based on various SV characteristics. The surges in SV data sources and data-driven techniques such as Machine Learning and Deep Learning have taken SV assessment and prioritization to the next level. Our survey provides a taxonomy of the past research efforts and highlights the best practices for data-driven SV assessment and prioritization. We also discuss the current limitations and propose potential solutions to address such issues.

CCS Concepts: • **General and reference** → *Surveys and overviews*; • **Security and privacy** → *Software security engineering*; • **Computing methodologies** → *Machine learning*; *Neural networks;*

Additional Key Words and Phrases: Software vulnerability, Vulnerability assessment and prioritization

ACM Reference format:
Triet H. M. Le, Huaming Chen, and M. Ali Babar. 2022. A Survey on Data-driven Software Vulnerability Assessment and Prioritization. *ACM Comput. Surv.* 55, 5, Article 100 (December 2022), 39 pages.
https://doi.org/10.1145/3529757

1 INTRODUCTION

Software Vulnerabilities (SVs) can negatively affect the confidentiality, integrity, and availability of software systems [66]. The exploitation of these SVs such as the Heartbleed attack[1] can damage the operations and reputation of millions of systems and organizations globally, resulting in huge financial losses as well. Therefore, it is important to remediate critical SVs as promptly as possible.

[1] https://heartbleed.com.

This work has been supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centres Programme.
Authors' addresses: T. H. M. Le and H. Chen, CREST - The Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia; emails: {triet.h.le, huaming.chen}@adelaide.edu.au; M. A. Babar, CREST - The Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia and Cyber Security Cooperative Research Centre, Australia; email: ali.babar@adelaide.edu.au.

0360-0300/2022/12-ART100 $15.00
https://doi.org/10.1145/3529757

Table 1. Comparison of Contributions between our Survey and the Existing Related Surveys/Reviews

Study \ Contribution	Focus on SV assessment & prioritization	Analysis of SV data sources	Analysis of data-driven approaches (NLP/ML/DL)
Ghaffarian et al. [66]	–	–	✓(Mostly ML)
Lin et al. [115] Semasaba et al. [170] Singh et al. [175] Zeng et al. [211]	–	–	✓(Mostly DL)
Pastor et al. [152]	–	✓(OSINT)	–
Sun et al. [184] Evangelista et al. [54]	–	✓(OSINT)	✓
Khan et al. [94]	✓(Rule-based methods)	–	–
Kritikos et al. [100]	✓(Static analysis)	✓	–
Dissanayake et al. [44]	✓(Socio-technical aspects)	–	–
Our survey	✓	✓	✓

Vulnerability assessment is required to prioritize the remediation of critical SVs among a large and increasing number of SVs each year [176] (e.g., more than 20,000 SVs were reported on **National Vulnerability Database (NVD)** [140] in 2021). SV assessment includes tasks that determine various characteristics such as the types, exploitability, impact, and severity levels of SVs [47, 113, 177]. Such characteristics help understand and select high-priority SVs to resolve early given the limited effort and resources. For example, an identified **cross-site scripting (XSS)** or SQL injection vulnerability in a web application will likely require an urgent remediation plan. These two types of SVs are well-known and can be easily exploited by attackers to gain unauthorized access and compromise sensitive data/information. However, an SV that requires admin access or happens only in a local network will probably have a lower priority, since only a few people can initiate an attack.

There has been an active research area to assess and prioritize SVs using increasingly large data from multiple sources. Many studies in this area have proposed different **Natural Language Processing (NLP), Machine Learning (ML)**, and **Deep Learning (DL)** techniques to leverage such data to automate various tasks such as predicting the **Common Vulnerability Scoring System (CVSS)** [57] metrics (e.g., References [75, 113, 177]) or public exploits (e.g., References [20, 23, 165]). These prediction models can learn the patterns automatically from vast SV data, which would be otherwise impossible to do manually. Such patterns are utilized to speed up the assessment and prioritization processes of ever-increasing and more complex SVs, significantly reducing practitioners' effort. Despite the rising research interest in data-driven SV assessment and prioritization, to the best of our knowledge, there has been no comprehensive survey on the state-of-the-art methods and existing challenges in this area.

Our Contributions. ❶ We are the first to review in-depth the research studies that automate *data-driven SV assessment and prioritization* tasks leveraging SV data and NLP/ML/DL techniques. ❷ We categorize and describe the key tasks performed in relevant primary studies. ❸ We synthesize and discuss the pros and cons of data, features, models, evaluation methods, and metrics commonly used in the reviewed studies. ❹ We highlight the challenges with the current practices and propose potential solutions moving forward. Our findings can provide useful guidelines for researchers and practitioners to effectively utilize data to perform SV assessment and prioritization. An online and up-to-date (by accepting external contributions) version of the survey can be found at https://github.com/lhmtriet/awesome-vulnerability-assessment.

Related Work. There have been several existing surveys/reviews on SV analysis and prediction, but they are fundamentally different from ours (see Table 1). Ghaffarian et al. [66]

conducted a seminal survey on ML-based SV analysis and discovery. Subsequently, several studies [115, 170, 175, 211] reviewed DL techniques for detecting vulnerable code. However, these prior reviews did not describe how ML/DL techniques can be used to assess and prioritize the detected SVs. There have been other relevant reviews on using **Open Source Intelligence (OSINT)** (e.g., phishing or malicious emails/URLs/IPs) to make informed security decisions [54, 152, 184]. However, these OSINT reviews did not explicitly discuss the use of SV data and how such data can be leveraged to automate the assessment and prioritization processes. Moreover, most of the reviews on SV assessment and prioritization have focused on either static analysis tools [100] or rule-based approaches (e.g., expert systems or ontologies) [94]. These methods rely on pre-defined patterns and struggle to work with new types and different data sources of SVs compared to contemporary ML or DL approaches presented in our survey [69, 72]. Recently, Dissanayake et al. [44] reviewed the socio-technical challenges and solutions for security patch management that involves SV assessment and prioritization after SV patches are identified. Unlike Reference [44], we focus on the challenges, solutions and practices of automating various SV assessment and prioritization tasks with data-driven techniques. We also consider all types of SV assessment/prioritization regardless of the patch availability.

2 SURVEY OVERVIEW

2.1 Background and Scope of the Survey

Our survey's focus is on *data-driven SV assessment and prioritization*. The *assessment* and *prioritization* phases are between the SV *discovery*/*detection* and SV *remediation*/*mitigation*/*fixing*/*patching* phases in an SV management lifecycle [61]. The *assessment* phase unveils the characteristics of the SVs found in the *discovery* phase to locate "hot spots" that contain many highly critical/severe SVs and require higher attention in a system. In the *prioritization* phase, practitioners use the assessment outputs to devise an optimal remediation plan, i.e., the order/priority of fixing each SV, based on available human and technological resources. SVs would then be mitigated/fixed accordingly to the prioritized plan in the *remediation* phase. Unlike the existing surveys on rule-based or experience-based SV assessment and prioritization [44, 94, 100] that hardly utilize the potential of SV data in the wild, this survey aims to review research papers that have leveraged such data to automate tasks in this area using data-driven models. To keep our focus, we do not consider papers that only perform manual analyses or descriptive statistics (e.g., taking mean/median/variation of data) without using any data-driven models, as these techniques cannot automatically assess or prioritize new SVs. We also do not directly compare the absolute performance of all the related studies, as they did not use exactly the same experimental setup (e.g., data sources and model configurations). While it is theoretically possible to perform a comparative evaluation of the identified techniques by establishing and using a common setup, this type of evaluation is out of the scope of this survey. However, we still cover the key directions/techniques of the studies in Sections 3, 4, 5, 6, and 7. We also provide in-depth discussion on the common practices and challenges of these studies and suggest some potential directions to advance the field in Sections 8 and 9.

2.2 Methodology

Study selection. Our study selection was inspired by the Systematic Literature Review guidelines [92]. Due to the space limit, we only included the key steps of our selection here. The full details can be found in Reference [108]. We first designed the search string: "*'software' AND vulner* AND (learn* OR data* OR predict*) AND (priority* OR assess* OR impact* OR exploit* OR severity*) AND NOT (fuzz* OR dynamic* OR intrusion OR adversari* OR malware* OR*

'vulnerability detection' OR 'vulnerability discovery' OR 'vulnerability identification' OR 'vulnerability prediction')." This search string covered the key papers (i.e., with more than 50 citations) in the area and excluded many papers on general security and SV detection. We then adapted this string[2] to retrieve an initial list of 1,765 papers up to April 2021 from various commonly used databases such as IEEE Xplore, ACM Digital Library, Scopus, SpringerLink, and Wiley. We also defined the inclusion/exclusion criteria (see Reference [108] for more details) to filter out irrelevant/low-quality studies with respect to our scope in Section 2.1. Based on these criteria and the titles and abstracts and keywords of 1,765 initial papers, we removed 1,550 papers. After reading the full-text and applying the criteria on the remaining 215 papers, we obtained 70 papers directly related to data-driven SV assessment and prioritization. To further increase the coverage of studies, we performed backward and forward snowballing on these 70 papers (using the above sources and Google Scholar) and identified 14 more papers that satisfied the inclusion/exclusion criteria. In total, we included 84 studies for our survey. We do not claim that we have collected all the papers in this area, but we believe that our selection covered most of the key studies to unveil the practices of data-driven SV assessment and prioritization.

Data extraction and synthesis of the selected studies. We followed the steps of thematic analysis [39] to identify the taxonomy of data-driven SV assessment and prioritization tasks in Sections 3, 4, 5, 6, and 7 as well as the key practices of data-driven model building for automating these tasks in Section 8. We first conducted a pilot study of 20 papers to familiarize ourselves with data to be extracted from the primary studies. After that, we generated initial codes and then merged them iteratively in several rounds to create themes. Two of the authors performed the analysis independently, in which each author analyzed half of the selected papers and then reviewed the analysis output of the other author. Any disagreements were resolved through discussions.

2.3 Taxonomy of Data-driven Software Vulnerability Assessment and Prioritization

Based on the scope in Section 2.1 and the methodology in Section 2.2, we identified five main themes of the relevant studies in the area of data-driven SV assessment and prioritization (see Figure 1). Specifically, we extracted the themes by grouping related SV assessment or prioritization tasks that the surveyed studies aim to automate/predict using data-driven models. Note that a paper is categorized into more than one theme if that paper develops models for multiple cross-theme tasks.

We acknowledge that there can be other ways to categorize the studies. However, we assert the reliability of our taxonomy as all of our themes (except theme 5) align with the security standards used in practice. For example, **Common Vulnerability Scoring System (CVSS)** [57] provides a framework to characterize exploitability, impact, and severity of SVs (themes 1–3), while **Common Weakness Enumeration (CWE)** [133] includes many vulnerability types (theme 4). Hence, we believe our taxonomy can help identify and bridge the knowledge gap between the academic literature and industrial practices, making it relevant and potentially beneficial for both researchers and practitioners. Details of each theme in our taxonomy are covered in subsequent sections.

3 EXPLOITATION PREDICTION

This section covers the *Exploitation* theme that automates the detection and understanding of both **Proof-of-Concept (PoC)** and real-world exploits[3] targeting identified SVs. This theme outputs

[2]This search string was customized for each database, and the database-wise search strings can be found at Reference [108].
[3]An *exploit* is a piece of code used to compromise-vulnerable software [165]. Real-world exploits are harmful and used in real host/network-based attacks. PoC exploits are unharmful and used to show the potential threats of SVs in penetration tests.

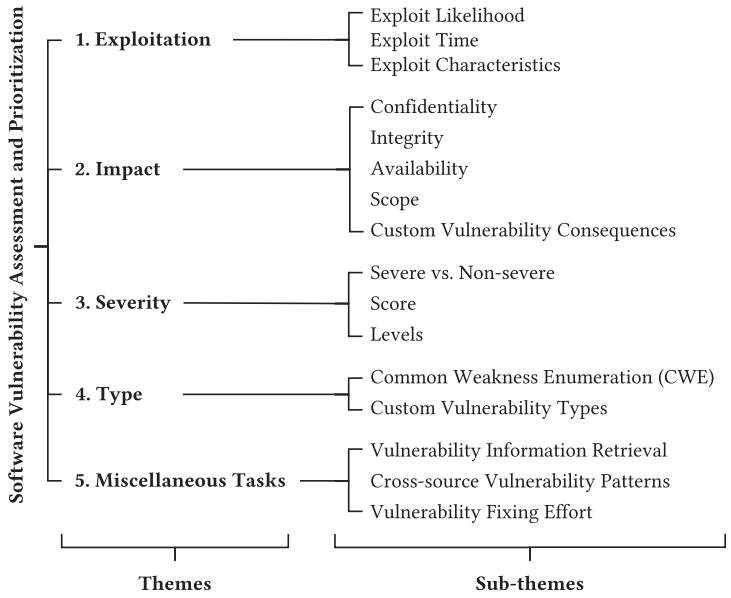

Fig. 1. Taxonomy of studies on data-driven SV assessment and prioritization.

the origin of SVs and how/when attackers would take advantage of such SVs to compromise a system of interest, assisting practitioners to quickly react to the more easily exploitable or already exploited SVs. The papers in this theme can be categorized into three groups/sub-themes: (*i*) Exploit likelihood, (*ii*) Exploit time, (*iii*) Exploit characteristics, as given in Tables 2, 3, and 4, respectively.

3.1 Summary of Primary Studies

3.1.1 Exploit Likelihood. The first sub-theme is *exploit likelihood* that predicts whether SVs would be exploited in the wild or PoC exploits would be released publicly (see Table 2). In 2010, Bozorgi et al. [20] were the first to use SV descriptions on **Common Vulnerabilities and Exposures (CVE)** [132] and **Open Source Vulnerability Database (OSVDB)**[4] to predict exploit existence based on the labels on OSVDB. In 2015, Sabottke et al. [165] conducted a seminal study that used Linear SVM and SV information on Twitter to predict PoC exploits on ExploitDB [168] as well as real-world exploits on OSVDB, Symantec's attack signatures [21] and private Microsoft security advisories [128]. These authors urged to explicitly consider real-world exploits, as *not* all PoC exploits would result in exploitation in practice. They also showed SV-related information on Twitter[5] can enable earlier detection of exploits than using expert-verified SV sources (e.g., NVD).

Built upon these two foundational studies [20, 165], the literature has mainly aimed to improve the performance and applicability of exploit prediction models by leveraging more exploit sources and/or better data-driven techniques/practices. Many researchers [5, 6, 50, 51, 86, 185] increased the amount of ground-truth exploits using extensive sources other than ExploitDB and Symantec in References [20, 165]. The sources were security advisories such as Zero Day Initiative [127], Metasploit [156], SecurityFocus [84], Recorded Future [63], Kenna Security [93], Avast,[6] ESET [53], Trend Micro [126], malicious activities in hosts based on traffic of spam/malicious IP

[4] http://osvdb.org Note that this database has been discontinued since [2016].
[5] https://twitter.com.
[6] https://avast.com/exploit-protection.php. This link was provided by de Sousa et al. [42], but it is no longer available.

Table 2. List of the Surveyed Papers in the *Exploit Likelihood* Sub-theme of the *Exploitation* Theme

Study	Data source	Data-driven technique
Bozorgi et al. [20]	CVE, Open Source Vulnerability Database (OSVDB)	Linear Support Vector Machine (SVM)
Sabottke et al. [165]	NVD, Twitter, OSVDB, ExploitDB, Symantec security advisories, private Microsoft security advisories	Linear SVM
Edkrantz et al. [50, 51]	NVD, Recorded Future security advisories, ExploitDB	Naïve Bayes, Linear SVM, Random forest
Bullough et al. [23]	NVD, Twitter, ExploitDB	Linear SVM
Almukaynizi et al. [5, 6]	NVD, ExploitDB, Zero Day Initiative security advisories & Darkweb forums/markets	SVM, Random forest, Naïve Bayes, Bayesian network, Decision tree, Logistic regression
Xiao et al. [204]	NVD, SecurityFocus security advisories, Symantec Spam/malicious activities based on daily blacklists from abuseat.org, spamhaus.org, spamcop.net, uceprotect.net, wpbl.info & list of unpatched SVs in hosts	Identification of malicious activity groups with community detection algorithms + Random forest for exploit prediction
Tavabi et al. [185]	NVD, 200 sites on Darkweb, ExploitDB, Symantec, Metasploit	Paragraph embedding + Radial basis function kernel SVM
de Sousa et al. [42]	NVD, Twitter, ExploitDB, Symantec Avast, ESET, Trend Micro security advisories	Linear SVM, Logistic regression, XGBoost, Light Gradient Boosting Machine (LGBM)
Fang et al. [55]	NVD, ExploitDB, SecurityFocus, Symantec	fastText + LGBM
Huang et al. [82]	NVD, CVE Details, Twitter, ExploitDB, Symantec security advisories	Random forest
Jacobs et al. [86]	NVD, Kenna Security Exploit sources: Exploit DB, Metasploit, FortiGuard Labs, SANS Internet Storm Center, Securewords CTU, Alienvault OSSIM, Canvas/D2 Security's Elliot Exploitation Frameworks, Contagio, Reversing Labs	XGBoost
Yin et al. [208]	NVD, ExploitDB, General text: Book Corpus & Wikipedia for pretraining BERT models	Fine-tuning BERT models pretrained on general text
Bhatt et al. [13]	NVD, ExploitDB	Features augmented by SV types + Decision tree, Random forest, Naïve Bayes, Logistic regression, SVM
Suciu et al. [182]	NVD, Vulners database, Twitter, Symantec, SecurityFocus, IBM X-Force Threat Intelligence Exploit sources: ExploitDB, Metasploit, Canvas, D2 Security's Elliot, Tenable, Skybox, AlienVault, Contagio	Multi-layer perceptron
Younis et al. [210]	Vulnerable functions from NVD (Apache HTTP Server project), ExploitDB, OSVDB	SVM
Yan et al. [207]	Executables (binary code) of 100 Linux applications	Combining ML (Decision tree) output & fuzzing with a Bayesian network
Tripathi et al. [187]	Program crashes from VDiscovery [27, 70] & LAVA [45] datasets	Static/Dynamic analysis features + Linear/Radial basis function kernel SVM
Zhang et al. [213]	Program crashes from VDiscovery [27, 70] dataset	*n*-grams of system calls from execution traces + Online passive-aggressive classifier

Note: The nature of task of this sub-theme is binary classification of existence/possibility of proof-of-concept and/or real-world exploits.

addresses [204], and Darkweb sites/forums/markets [143]. In addition to enriching exploit sources, better data-driven models and practices for exploit prediction were also studied. Ensemble models (e.g., Random forest, **eXtreme Gradient Boosting (XGBoost)** [34], **Light Gradient Boosting Machine (LGBM)** [91]) were shown to outperform single-model baselines (e.g., Naïve Bayes, SVM, Logistic regression, and Decision tree) for exploit prediction [42, 55, 82, 86]. Additionally, Bullough et al. [23] identified and addressed several issues with exploit prediction models (e.g., time sensitivity of SV data, already-exploited SVs before disclosure and training data imbalance), helping to improve the practical application of such models. Recently, Yin et al. [208] demonstrated that transfer learning is an alternative solution for improving the performance of exploit prediction

Table 3. List of the Surveyed Papers in the *Exploit Time* Sub-theme of the *Exploitation* Theme

Study	Nature of task	Data source	Data-driven technique
Bozorgi et al. [20]	*Binary classification*: Likelihood that SVs would be exploited within 2 to 30 days after disclosure	CVE, OSVDB	Linear SVM
Edkrantz [50]	*Binary classification*: Likelihood of SV exploits within 12 months after disclosure	NVD, ExploitDB, Recorded Future security advisories	SVM, K-Nearest Neighbors (KNN), Naïve Bayes, Random forest
Jacobs et al. [87]		NVD, Kenna Security Exploit sources: Exploit DB, Metasploit, D2 Security's Elliot & Canvas Exploitation Frameworks, Fortinet, Proofpoint, AlienVault & GreyNoise	Logistic regression
Chen et al. [30, 31]	*Binary classification*: Likelihood that SVs would be exploited within 1/3/6/9/12 months after disclosure *Regression*: number of days until SV exploits after disclosure	CVE, Twitter, ExploitDB, Symantec security advisories	Graph neural network embedding + Linear regression, Bayes, Random forest, XGBoost, Lasso/Ridge regression

with scarcely labeled exploits. Specifically, these authors pre-trained a DL model, BERT [43], on massive non-SV sources (e.g., text on Book Corpus [218] and Wikipedia [62]) and then fine-tuned this pre-trained model on SV data using additional pooling and dense layers. Bhatt et al. [13] also suggested that incorporating the types of SVs (e.g., SQL injection) into ML models can further enhance the predictive effectiveness. Suciu et al. [182] empirically showed that unifying SV-related sources used in prior work (e.g., SV databases [20], social media [165], SV-related discussions [185], and PoC code in ExploitDB [86]) supports more effective and timely prediction of *functional* exploits [60].

Besides using SV descriptions as input for exploit prediction, several studies in this sub-theme have also predicted exploits on the code level. Younis et al. [210] predicted the exploitability of vulnerable functions in the Apache HTTP Server project. Specifically, these authors used an SVM model with features extracted from the dangerous system calls [12] in entry points/functions [121] and the reachability from any of these entry points to vulnerable functions [80]. Moving from high-level to binary code, Yan et al. [207] first used a Decision tree to obtain prior beliefs about SV types in 100 Linux applications using static features (e.g., *hexdump*) extracted from executables. Subsequently, they applied various fuzzing tools (i.e., Basic Fuzzing Framework [25] and OFuzz [26]) to detect SVs with the ML-predicted types. They finally updated the posterior beliefs about the exploitability based on the outputs of the ML model and fuzzers using a Bayesian network. The proposed method outperformed *!exploitable*,[7] a static crash analyzer provided by Microsoft. Tripathi et al. [187] also predicted SV exploitability from crashes (i.e., VDiscovery [27, 70] and LAVA [45] datasets) using an SVM model and static features from core dumps and dynamic features generated by the Last Branch Record hardware debugging utility. Zhang et al. [213] proposed two improvements to Tripathi et al. [187]'s approach. These authors first replaced the hardware utility in Reference [187] that may not be available for resource-constrained devices (e.g., IoT) with sequence/*n*-grams of system calls extracted from execution traces. They also used an online passive-aggressive classifier [38] to enable online/incremental learning of exploitability for new crash batches on-the-fly.

3.1.2 Exploit Time. After predicting the likelihood of SV exploits in the previous sub-theme, this sub-theme provides more fine-grained information about *exploit time* (see Table 3). Besides

[7] https://microsoft.com/security/blog/2013/06/13/exploitable-crash-analyzer-version-1-6.

performing binary classification of exploits, Bozorgi et al. [20] and Edkrantz [50] also predicted the time frame (2–30 days in Reference [20] and 12 months in Reference [50]) within which exploits would happen after the disclosure of SVs. Jacobs et al. [87] then leveraged multiple sources containing both PoC and real-world exploits, as given in Table 3, to improve the number of labeled exploits, enhancing the prediction of exploit appearance within 12 months. Chen et al. [31] predicted whether SVs would be exploited within 1–12 months and the exploit time (number of days) after SV disclosure using Twitter data. The authors proposed a novel regression model whose feature embedding was a multi-layer graph neural network [98] capturing the content and relationships among tweets, respective tweets' authors and SVs. The proposed model outperformed many baselines and was integrated into the VEST system [30] to provide timely SV assessment information for practitioners. To the best of our knowledge, at the time of writing, Chen et al. [30, 31] have been the only ones pinpointing the exact exploit time of SVs rather than large/uncertain time-frames (e.g., months) in other studies, helping practitioners to devise much more fine-grained remediation plans.

3.1.3 Exploit Characteristics. *Exploit characteristics* is the final sub-theme that reveals various requirements/means of exploits (see Table 4), informing the potential scale of SVs; e.g., remote exploits likely affect more systems than local ones. The commonly used outputs are the Exploitability metrics provided by versions 2 [58] and 3 [59, 60] of **Common Vulnerability Scoring System (CVSS)**.

Many studies have focused on predicting and analyzing version 2 of CVSS exploitability metrics (i.e., Access Vector, Access Complexity, and Authentication). Yamamoto et al. [206] were the first to leverage descriptions of SVs on NVD together with a supervised Latent Dirichlet Allocation topic model [16] to predict these CVSS metrics. Subsequently, Wen et al. [199] used **Radial Basis Function (RBF)**-kernel SVM and various SV databases/advisories other than NVD (e.g., SecurityFocus, OSVDB, and IBM X-Force [171]) to predict the metrics. Le et al. [113] later showed that the prediction of CVSS metrics suffered from the *concept drift* issue; i.e., descriptions of new SVs may contain Out-of-Vocabulary terms for prediction models. They proposed to combine sub-word features with traditional **Bag-of-Word (BoW)** features to infer the semantics of novel terms/words from existing ones, helping assessment models be more robust against concept drift. Besides prediction, Toloudis et al. [186] used principal component analysis [202] and Spearman's ρ correlation coefficient to reveal the predictive contribution of each word in SV descriptions to each CVSS metric. However, this technique does not directly produce the value of each metric.

Recently, several studies have started to predict CVSS version 3 exploitability metrics including the new Privileges and User Interactions. Ognawala et al. [144] fed the features generated by a static analysis tool, Macke [145], to a Random forest model to predict these CVSS version 3 metrics for vulnerable software/components. Later, Chen et al. [30] found that many SVs were disclosed on Twitter before on NVD. Therefore, these authors developed a system built on top of a Graph Convolutional Network [97] capturing the content and relationships of related Twitter posts about SVs to enable more timely prediction of the CVSS version 3 metrics. Elbaz et al. [52] developed a linear regression model to predict the numerical output of each metric and then obtained the respective categorical value with the numerical value closest to the predicted value. For example, a predicted value of 0.8 for Attack Vector CVSS v3 is mapped to *Network* (0.85) [59]. To prepare a clean dataset to predict these CVSS metrics, Jiang et al. [88] replaced inconsistent CVSS values in various SV sources (i.e., NVD, ICS CERT, and vendor websites) with the most frequent value.

Instead of building a separate model for each CVSS metric, there has been another family of approaches predicting these metrics using a single model to increase efficiency. Gawron et al. [64]

and Spanos et al. [177] predicted multiple CVSS metrics as a unique string instead of individual values. The output of each metric is then extracted from the concatenated string. Later, Gong et al. [67] adopted the idea of a unified model from the DL perspective by using the multi-task learning paradigm [217] to predict CVSS metrics simultaneously. The model has a feature extraction module (based on a Bi-LSTM model with attention mechanism [9]) shared among all the CVSS metrics/tasks, yet specific prediction head/layer for each metric/task. This model outperformed single-task counterparts while requiring much less time to (re-)train.

Although CVSS exploitability metrics were most commonly used, several studies used other schemes for characterizing exploitation. Chen et al. [36] used Linear SVM and SV descriptions to predict multiple SV characteristics, including three *SV locations* (i.e., Local, LAN, and Remote) on SecurityFocus [84] and Secunia [85] databases as well as 11 *SV causes*[8] on SecurityFocus. Regarding the exploit types, Rouhonen et al. [160] used LDA [17] and Random forest to classify whether an exploit would affect a web application. This study can help find relevant exploits in components/sub-systems of a large system. For privileges, Aksu et al. [3] extended the Privileges Required metric of CVSS by incorporating the context (i.e., Operating system or Application) to which privileges are applied (see Table 4). They found MLP [76] to be the best-performing model for obtaining these privileges from SV descriptions. They also utilized the predicted privileges to generate attack graphs (sequence of attacks from source to sink nodes). Liu et al. [117] advanced this task by combining information gain for feature selection and **Convolutional Neural Network (CNN)** [96] for feature extraction. Regarding attack patterns, Kanakogi et al. [90] found Doc2vec [106] to be more effective than term-frequency **inverse document frequency (tf-idf)** when combined with cosine similarity to find the most relevant **Common Attack Pattern Enumeration and Classification (CAPEC)** [130] for a given SV on NVD. Such attack patterns can manifest how identified SVs can be exploited by adversaries, assisting the selection of suitable countermeasures.

3.2 Theme Discussion

In the *Exploitation* theme, the primary tasks are binary classification of whether **Proof-of-Concept (PoC)**/real-world exploits of SVs would appear and multi-classification of exploit characteristics based on CVSS. PoC exploits mostly come from ExploitDB [168]; whereas, real-world exploits, despite coming from multiple sources, are still much scarcer than PoC counterparts. Consequently, the models predicting real-world exploits have generally performed worse than those for PoC exploits. Similarly, the performance of the models determining CVSS v3 exploitability metrics has been mostly lower than that of the CVSS v2 based models. However, real exploits and CVSS v3 are usually of more interest to the community. The former can lead to real cyber-attacks, and the latter is the current standard in practice. To improve the performance of these practical tasks, future work can collect more exploit-related data from the relevant yet under-explored sources (see Section 9.1), as well as adapt the patterns learned from PoC exploits and old CVSS versions to real exploits and newer CVSS versions, respectively, e.g., using transfer learning [151].

There are other under-explored tasks targeting fine-grained prediction of exploits. Mitigation of exploits in practice usually requires more information besides simply determining whether an SV would be exploited. Information gathered from predicting *when* and *how* the exploits would happen is also needed to devise better SV fixing prioritization and mitigation plans. VEST [30] is one of the first and few systems aiming to provide such all-in-one information about SV

[8] Access/Input/Origin validation error, Atomicity/Configuration/Design/Environment/Serialization error, Boundary condition error, Failure on exceptions, Race condition error.

Table 4. List of the Surveyed Papers in the *Exploit Characteristics* Sub-theme of the *Exploitation* Theme

Study	Nature of task	Data source	Data-driven technique
Yamamoto et al. [206]	*Multi-class classification*: CVSS v2 (Access Vector & Access Complexity metrics)	NVD	Supervised Latent Dirichlet Allocation (LDA)
Wen et al. [199]		NVD, OSVDB, SecurityFocus, IBM X-Force	Radial basis function kernel SVM
Le et al. [113]	*Binary classification*: CVSS v2 (Authentication metric)	NVD	Concept-drift-aware models with Naïve Bayes, KNN, Linear SVM, Random forest, XGBoost, LGBM
Toloudis et al. [186]	*Correlation analysis*: CVSS v2	NVD	Principal component analysis & Spearman correlation coefficient
Ognawala et al. [144]	*Multi-class classification*: CVSS v3 (Attack Vector, Attack Complexity & Privileges Required metrics)	NVD (buffer overflow SVs) & Source code of vulnerable software/components	Combining static analysis tool (Macke [145]) & ML classifiers (Naïve Bayes & Random forest)
Chen et al. [30]	*Binary classification*: CVSS v3 (User Interaction metric)	CVE, NVD, Twitter	Graph convolutional network
Elbaz et al. [52]	*Multi-class/Binary classification*: CVSS v2/v3	NVD	Mapping outputs of Linear regression to CVSS metrics with closest values
Jiang et al. [88]		NVD, ICS Cert, Vendor websites (Resolve inconsistencies with a majority vote)	Logistic regression
Gawron et al. [64]	*Multi-target classification*: CVSS v2	NVD	Naïve Bayes, Multi-layer Perceptron (MLP)
Spanos et al. [177]		NVD	Random forest, boosting model, Decision tree
Gong et al. [67]	*Multi-task classification*: CVSS v2	NVD	Bi-LSTM with attention mechanism
Chen et al. [36]	*Multi-class classification*: Platform-specific vulnerability locations (Local, Remote, Local area network) & vulnerability causes (e.g., Access/Input/Origin validation error)	NVD, Secunia vulnerability database, SecurityFocus, IBM X-Force	Linear SVM
Ruohonen et al. [160]	*Binary classification*: Web-related exploits or not	ExploitDB	LDA + Random forest
Aksu et al. [3]	*Multi-class classification*: author-defined pre-/post-condition privileges (None, OS (Admin/User), App (Admin/User))	NVD	RBF network, Linear SVM, NEAT [181], MLP
Liu et al. [117]		NVD	Information gain + Convolutional neural network
Kanakogi et al. [90]	*Multi-class classification*: Common Attack Pattern Enumeration and Classification (CAPEC)	NVD, CAPEC	Doc2vec/tf-idf with cosine similarity

exploitation. However, this system currently only uses data from NVD/CVE and Twitter, which can be extended to incorporate more (exploit) sources and more sophisticated data-driven techniques in the future.

Most of the current studies have used SV descriptions on NVD and other security advisories to predict the exploitation-related metrics. This is surprising, as SV descriptions do not contain root causes of SVs. Instead, SVs are rooted in source code, yet there is little work on code-based exploit prediction. So far, Younis et al. [210] have been among the few using source code for exploit prediction, but their approach still requires manual identification of dangerous function calls in

Table 5. List of the Surveyed Papers in the *Impact* Theme

Study	Nature of task	Data source	Data-driven technique
Sub-themes: 1. Confidentiality, 2. Integrity, 3. Availability & 4. Scope (only in CVSS v3)			
Yamamoto et al. [206]	*Multi-class classification*: CVSS v2	NVD	Supervised Latent Dirichlet Allocation
Wen et al. [199]		NVD, OSVDB, Security Focus, IBM X-Force	Radial basis function kernel SVM
Le et al. [113]		NVD	Concept-drift-aware models with Naïve Bayes, KNN, Linear SVM, Random forest, XGBoost, LGBM
Toloudis et al. [186]	*Correlation analysis*: CVSS v2	NVD	Principal component analysis & Spearman correlation coefficient
Ognawala et al. [144]	*Multi-class classification*: CVSS v3 *Binary classification*: Scope in CVSS v3	NVD (buffer overflow SVs) & Source code of vulnerable software/components	Combining static analysis tool (Macke [145]) & ML classifiers (Naïve Bayes & Random forest)
Chen et al. [30]		CVE, NVD, Twitter	Graph convolutional network
Elbaz et al. [52]	*Multi-class classification*: CVSS v2/v3 *Binary classification*: Scope in CVSS v3	NVD	Mapping outputs of Linear regression outputs to CVSS metrics with closest values
Jiang et al. [88]		NVD, ICS Cert, Vendor websites (Resolve inconsistencies with a majority vote)	Logistic regression
Gawron et al. [64]	*Multi-target classification*: CVSS v2	NVD	Naïve Bayes, MLP
Spanos et al. [177]		NVD	Random forest, boosting model, Decision tree
Gong et al. [67]	*Multi-task classification*: CVSS v2	NVD	Bi-LSTM with attention mechanism
Sub-theme: 5. Custom Vulnerability Consequences			
Chen et al. [36]	*Multi-label classification*: Platform-specific impacts (e.g., Gain system access)	NVD, Secunia vulnerability database, SecurityFocus, IBM X-Force	Linear SVM

Note: We Grouped the First Four Sub-themes, as They Were Mostly Predicted Together.

C/C++. More work is required to employ data-driven approaches to alleviate the need for manually defined rules to improve the effectiveness and generalizability of code-based exploit prediction.

4 IMPACT PREDICTION

This section describes the *Impact* theme that determines the (negative) effects that SVs have on a system of interest if such SVs are exploited. There are five key tasks that the papers in this theme have automated/predicted: (*i*) Confidentiality impact, (*ii*) Integrity impact, (*iii*) Availability impact, (*iv*) Scope, and (*v*) Custom vulnerability consequences (see Table 5).

4.1 Summary of Primary Studies

4.1.1 Confidentiality, Integrity, Availability, and Scope. A majority of the papers have focused on the impact metrics provided by CVSS, including versions 2 [58] and 3 [59, 60]. Versions 2 and 3 share three impact metrics *Confidentiality*, *Integrity,* and *Availability*. Version 3 also has a new metric, *Scope*, that specifies whether an exploited SV would affect only the system that contains the SV. For example, *Scope* changes when an SV occurring in a virtual machine affects the whole host machine, in turn increasing individual impacts.

The studies that predicted the CVSS impact metrics are mostly the same as the ones predicting the CVSS exploitability metrics in Section 3. Given the overlap, we hereby only describe the

main directions and techniques of the *Impact*-related tasks rather than iterating the details of each study. Overall, a majority of the work has focused on classifying CVSS impact metrics (versions 2 and 3) using three main learning paradigms: single-task [30, 52, 88, 113, 144, 199, 206], multi-target [64, 177], and multi-task [67] learning. Instead of developing a separate prediction model for each metric such as the single-task approach, multi-target, and multi-task approaches only need a single model for all tasks. Multi-target learning predicts concatenated output; whereas, multi-task learning uses shared feature extraction for all tasks and task-specific softmax layers to determine the output of each task. These three learning paradigms were powered by applying and/or customizing a wide range of data-driven methods. The first method was to use single ML classifiers such as supervised Latent Dirichlet Allocation [206], Principal component analysis [186], Naïve Bayes [64, 113, 144], Logistic regression [88], Kernel-based SVM [199], Linear SVM [113], KNN [113], and Decision tree [177]. Other studies employed ensemble models combining the strength of multiple single models such as Random forest [113, 144], boosting model [177], and XGBoost/LGBM [113]. Recently, more studies moved towards more sophisticated DL architectures such as MLP [64], attention-based (Bi-)LSTM [67], and graph neural network [30]. Ensemble and DL models usually beat the single ones, but there is a lack of direct comparisons between these two emerging model types.

4.1.2 Custom Vulnerability Consequences. To devise effective remediation strategies for a system of interest in practice, practitioners may want to know *custom vulnerability consequences,* which are more interpretable than the levels of impact provided by CVSS. Chen et al. [36] curated a list of 11 vulnerability consequences[9] from X-Force [171] and Secunia [85] vulnerability databases. They then used a Linear SVM model to perform multi-label classification of these consequences for SVs, meaning that an SV can lead to more than one consequence. To the best of our knowledge, this is the only study that has pursued this research direction so far.

4.2 Theme Discussion

In the *Impact* theme, the common task is to predict the impact base metrics provided by CVSS versions 2 and 3. Similar to the Exploitation theme, the models for CVSS v3 still require more attention and effort from the community to reach the same performance level as the models for CVSS v2. These impact metrics are also usually predicted together with the exploitability metrics given their similar nature (multi-class classification) using either task-wise models or a unified (multi-target or multi-task) model. Multi-target and multi-task learning are promising, as they can reduce the time for continuous (re)training and maintenance when deployed in production.

Besides CVSS impact metrics, other fine-grained SV consequences have also been explored [36], but there is still no widely accepted taxonomy for such consequences. Thus, these consequences have seen less adoption in practice than CVSS metrics, despite being potentially useful by providing more concrete information about what assets/components in a system that an SV can compromise. In Section 9.2.1, we suggest potential ways to create a systematic taxonomy of custom SV consequences to pave the way for more data-driven research in this direction.

5 SEVERITY PREDICTION

This section discusses the work in the *Severity* theme. Severity is often a function/combination of Exploitation (Section 3) and Impact (Section 4). SVs with higher severity usually require more urgent remediation. There are three main prediction tasks in this theme: (*i*) Severe vs. Non-severe, (*ii*) Severity levels, and (*iii*) Severity score, shown in Tables 6, 7, and 8, respectively.

[9]Gain system access, Bypass security, Configuration manipulation, Data/file manipulation, Denial of Service, Privilege escalation, Information leakage, Session hijacking, Cross-site scripting (XSS), Source spoofing, Brute-force proneness.

Table 6. List of the Surveyed Papers in the *Severe vs. Non-severe* Sub-theme of the *Severity* Theme

Study	Data source (software project)	Data-driven technique
Kudjo et al. [102]	NVD (Mozilla Firefox, Google Chrome, Internet Explorer, Microsoft Edge, Sea Monkey, Linux Kernel, Windows 7, Windows 10, Mac OS, Chrome OS)	Term frequency & inverse gravity moment weighting + KNN, Decision tree, Random forest
Chen et al. [32]	NVD (Adobe Flash Player, Enterprise Linux, Linux Kernel, Foxit Reader, Safari, Windows 10, Microsoft Office, Oracle Business Suites, Chrome, QuickTime)	Term frequency & inverse gravity moment weighting + KNN, Decision tree, Naïve Bayes, SVM, Random forest
Kudjo et al. [101]	NVD (Google Chrome, Mozilla Firefox, Internet Explorer, Linux Kernel)	Find the best smallest training dataset using KNN, Logistic regression, MLP, Random forest
Malhotra et al. [120]	NVD (Apache Tomcat)	Chi-square/Information gain + bagging technique, Random forest, Naïve Bayes, SVM

Note: The nature of task here is binary classification of severe SVs with High/Critical CVSS v2/v3 severity levels.

Similar to the *Exploitation* and *Impact* themes, many studies in the *Severity* theme have used CVSS versions 2 and 3. According to both CVSS versions, the severity score share the same range from 0 to 10, with an increment of 0.1. Based on the score, the existing studies have either defined a threshold to decide whether an SV is severe (requiring high attention) or predicted levels/groups of severity score that require a similar amount of attention or determined the raw score value.

5.1 Summary of Primary Studies

5.1.1 Severe vs. Non-severe. The first group of studies has classified whether an SV is *severe or non-severe*, making it a binary classification problem (see Table 6). These studies have typically selected severe SVs as the ones with at least High severity level (i.e., CVSS severity score $\geq$ 7.0). Kudjo et al. [102] showed that using term frequency (BoW) with inverse gravity moment weighting [33] to extract features from SV descriptions can enhance the performance of ML models (i.e., KNN, Decision tree, and Random forest) in predicting the severity of SVs. Later, Chen et al. [32] confirmed that this feature extraction method was also effective for more projects and classifiers (e.g., Naïve Bayes and SVM). Besides investigating feature extraction, Kudjo et al. [101] also highlighted the possibility of finding Bellwether, i.e., the smallest set of data that can be used to train an optimal prediction model, for classifying severity. Recently, Malhotra et al. [120] revisited this task by showing that Chi-square and information gain can be effective dimensionality reduction techniques for multiple classifiers, i.e., bagging technique, Random forest, Naïve Bayes, and SVM.

5.1.2 Severity Levels. Rather than just performing binary classification of whether an SV is severe, several studies have identified one among multiple *severity levels* that an SV belongs to (see Table 7). This setting can be considered as multi-class classification. Spanos et al. [178] were to first to show the applicability of ML to classify SVs into one of the three severity levels using SV descriptions. These three levels are provided by NVD and based on the severity score of CVSS version 2 [58] and WIVSS [179], i.e., Low (0.0–3.9), Medium (4.0–6.9), High (7.0–10.0). Note that WIVSS assigns different weights for the Confidentiality, Integrity, and Availability impact metrics of CVSS, enhancing the ability to capture varied contributions of these impacts to the final severity score. Later, Wang et al. [194] showed that XGBoost [34] performed the best among the investigated ML classifiers for predicting these three NVD-based severity levels. Le et al. [113] also confirmed that ensemble methods (e.g., XGBoost [34], LGBM [91], and Random forest) outperformed single models (e.g., Naïve Bayes, KNN, and SVM) for this task. Predicting severity levels has also been tackled with DL techniques [119, 172] such as **Recurrent Convolutional Neural Network (RCNN)** [105], **Convolutional Neural Network (CNN)** [96], **Long-Short Term Memory (LSTM)** [77]. These studies showed potential performance gain of DL models compared to traditional ML counterparts. Han et al. [75] showed that DL techniques (i.e., 1-layer CNN) also achieved promising results for predicting a different severity categorization, namely,

Table 7. List of the Surveyed Papers in the *Severity Levels* Sub-theme of the *Severity* Theme

Study	Nature of task	Data source	Data-driven technique
Spanos et al. [178]	*Multi-class classification*: NVD severity levels based on CVSS v2 & WIVSS (High, Medium, Low)	NVD	Decision tree, SVM, MLP
Wang et al. [194]	*Multi-class classification*: NVD severity levels based on CVSS v2 (High, Medium, Low)	NVD (XSS attacks)	XGBoost, Logistic regression, SVM, Random forest
Le et al. [113]		NVD	Concept-drift-aware models with Naïve Bayes, KNN, Linear SVM, Random forest, XGBoost, LGBM
Liu et al. [119]		NVD, China National Vulnerability Database (XSS attacks)	Recurrent Convolutional Neural Network (RCNN), Convolutional Neural Network (CNN), Long-Short Term Memory (LSTM)
Sharma et al. [172]		CVE Details	CNN
Han et al. [75]	*Multi-class classification*: Atlassian categories of CVSS severity score (Critical, High, Medium, Low)	CVE Details	1-layer CNN, 2-layer CNN, CNN-LSTM, Linear SVM
Sahin et al. [166]		NVD	1-layer CNN, LSTM, XGBoost, Linear SVM
Nakagawa et al. [138]		CVE Details	Character-level CNN vs. Word-based CNN + Linear SVM
Gong et al. [67]	*Multi-task classification*: Atlassian categories of CVSS severity score (Critical, High, Medium, Low)	CVE Details	Bi-LSTM with attention mechanism
Chen et al. [36]	*Multi-class classification*: severity levels of Secunia (Extremely/ highly/ moderately/less/non-critical)	CVE, Secunia vulnerability database, Security Focus, IBM X-Force	Linear SVM
Zhang et al. [214]	*Multi-class classification*: Platform-specific levels (High/Medium/Low)	China National Vulnerability Database	Logistic regression, Linear discriminant analysis, KNN, CART, SVM, bagging/boosting models
Khazaei et al. [95]	*Multi-class classification*: 10 severity score bins (one unit/bin)	CVE & OSVDB	Linear SVM, Random forest, Fuzzy system

Atlassian's levels.[10] Such findings were successfully replicated by Sahin et al. [166]. Nakagawa et al. [138] further enhanced the DL model performance for the same task by incorporating the character-level features into a CNN model [215]. Complementary to performance enhancement, Gong et al. [67] proposed to predict these severity levels concurrently with other CVSS metrics in a single model using multi-task learning [217] powered by an attention-based Bi-LSTM shared feature extraction model. The unified model was demonstrated to increase both the prediction effectiveness and efficiency. Besides Atlassian's categories, several studies applied ML models to predict severity levels on other platforms such as Secunia [36] and China National Vulnerability Database[11] [214]. Instead of using textual categories, Khazaei et al. [95] divided the CVSS severity score into 10 bins with 10 increments each (e.g., values of 0–0.9 are in one bin) and obtained decent results (86%–88% Accuracy) using Linear SVM, Random forest, and Fuzzy system.

5.1.3 Severity Score. To provide even more fine-grained severity value than the categories, the last sub-theme has predicted the *severity score* (see Table 8). Using SV descriptions on NVD, Sahin et al. [166] compared the performance of ML-based regressors (e.g., XGBoost [34] and Linear regression) and DL-based ones (e.g., CNN [96] and LSTM [77]) for predicting the severity score of CVSS version 2 [58]. These authors showed that DL-based approaches generally outperformed the ML-based counterparts. For CVSS version 3 [59, 60], Chen et al. [29, 30] and Anwar et al. [7] also reported the strong performance of DL-based models (e.g., CNN and graph convolutional neural

[10] https://www.atlassian.com/trust/security/security-severity-levels.
[11] https://www.cnvd.org.cn.

Table 8. List of the Surveyed Papers in the *Severity Score* Sub-theme of the *Severity* Theme

Study	Nature of task	Data source	Data-driven technique
Sahin et al. [166]	*Regression*: CVSS v2 (0-10)	NVD	1-layer CNN, LSTM, XGBoost regressor, Linear regression
Wen et al. [199]		OSVDB, SecurityFocus, IBM X-Force	Radial basis function kernel SVM†
Ognawala et al. [144]	*Regression*: CVSS v3 (0-10)	NVD (buffer overflow SVs)	Combining a static analysis tool (Macke [145]) & ML classifiers (Naïve Bayes & Random forest)†
Chen et al. [29, 30]		CVE, NVD, Twitter	Graph convolutional network
Anwar et al. [7]		NVD	Linear regression, Support vector regression, CNN, MLP
Elbaz et al. [52]	*Regression*: CVSS v2/ (0-10)	NVD	Mapping outputs of Linear regression to CVSS metrics with closest values†
Jiang et al. [88]		NVD, ICS Cert, Vendor websites (Resolve inconsistencies with a majority vote)	Logistic regression†
Spanos et al. [177]	*Regression*: CVSS v2 & WIVSS (0-10)	NVD	Random forest, boosting model, Decision tree†
Toloudis et al. [186]	*Correlation analysis*: CVSS v2 & WIVSS (0-10)	NVD	Principal component analysis & Spearman correlation coefficient

Notes: †denotes that the severity score is computed from ML-predicted base metrics using the formula provided by an assessment framework (CVSS and/or WIVSS).

network [97]). Some other studies did not directly predict severity score from SV descriptions, instead they aggregated the predicted values of the CVSS Exploitability (see Section 3) and Impact metrics (see Section 4) using the formulas of CVSS version 2 [52, 88, 177, 199], version 3 [52, 88, 144], and WIVSS [177]. We noticed the papers predicting both versions (e.g., CVSS versions 2 vs. 3 or CVSS version 2 vs. WIVSS) usually obtained better performance for version 3 and WIVSS than version 2 [52, 88]. These findings may suggest that the improvements made by experts in version 3 and WIVSS compared to version 2 help make the patterns in severity score clearer and easier for ML models to capture. In addition to predicting severity score, Toloudis et al. [186] examined the correlation between words in descriptions of SVs and the severity values of such SVs, aiming to shed light on words that increase or decrease the severity score of SVs.

5.2 Theme Discussion

In the *Severity* theme, predicting the severity levels is the most prevalent task, followed by severity score prediction and then binary classification of the severity. In practice, severity score gives more fine-grained information (fewer SVs per value) for practitioners to rank/prioritize SVs than categorical/binary levels. However, predicting continuous score values is usually challenging and requires more robust models, as this task involves higher uncertainty to learn inherent patterns from data than classifying fixed/discrete levels. We observed that DL models such as graph neural networks [29, 30], LSTM [166], and CNN [7] have been shown to be better than traditional ML models for predicting severity score. However, most of these studies did not evaluate their models in a continuous deployment setting to investigate how the models will cope with changing patterns of new SVs over time. We distill recommendations for future work on real-world application and evaluation of these models in Section 9.2.

6 TYPE PREDICTION

This section reports the work done in the *Type* theme. Type groups SVs with similar characteristics, e.g., causes, attack patterns, and impacts, and thus facilitating the reuse of known prioritization and

Table 9. List of the Surveyed Papers in the *Type* Theme

Study	Nature of task	Data source	Data-driven technique
Sub-theme: 1. Common Weakness Enumeration (CWE)			
Wang et al. [193]	*Multi-class classification*: CWE classes	NVD, CVSS	Naïve Bayes
Shuai et al. [174]		NVD	SVM
Na et al. [136]		NVD	Naïve Bayes
Ruohonen et al. [161]		NVD, CWE, Snyk	tf-idf with 1/2/3-grams and cosine similarity
Huang et al. [81]		NVD, CWE	MLP, Linear SVM, Naïve Bayes, KNN
Aota et al. [8]		NVD	Random forest, Linear SVM, Logistic regression, Decision tree, Extremely randomized trees, LGBM
Aghaei et al. [1]		NVD, CVE	Adaptive fully connected neural network with one hidden layer
Das et al. [40]		NVD, CWE	BERT, Deep Siamese network
Zou et al. [220]		NVD & Software Assurance Reference Dataset (SARD)	Three Bi-LSTM models for extracting and combining global and local features from code functions
Murtaza et al. [135]	*Unsupervised learning*: sequence mining of SV types (over time)	NVD (CWE & CPE)	2/3/4/5-grams of CWEs
Lin et al. [116]	*Unsupervised learning*: association rule mining of CWE-related aspects (prog. language, time of introduction & consequence scope)	CWE	FP-growth association rule mining algorithm
Han et al. [74]	*Binary/Multi-class classification*: CWE relationships (CWE links, link types & CWE consequences)	CWE	Deep knowledge graph embedding of CWE entities
Sub-theme: 2. Custom Vulnerability Types			
Venter et al. [190]	*Unsupervised learning*: clustering	CVE	Self-organizing map
Neuhaus et al. [139]	*Unsupervised learning*: topic modeling	CVE	Latent Dirichlet Allocation (LDA)
Mounika et al. [134, 189]		CVE, Open Web Application Security Project (OWASP)	LDA
Aljedaani et al. [4]		SV reports (Chromium project)	LDA
Williams et al. [200, 201]	*Multi-class classification*: manually coded SV types	NVD	Supervised Topical Evolution Model & Diffusion-based storytelling technique
Russo et al. [163]		NVD	Bayesian network, J48 tree, Logistic regression, Naïve Bayes, Random forest
Yan et al. [207]		Executables of 100 Linux applications	Decision tree
Zhang et al. [214]	*Multi-class classification*: platform-specific vulnerability types	China National Vulnerability Database	Logistic regression, Linear discriminant analysis, KNN, CART, SVM, bagging/-boosting models

remediation strategies employed for prior SVs of the same types. Two key prediction outputs are: (*i*) **Common Weakness Enumeration (CWE)** and (*ii*) Custom vulnerability types (see Table 9).

6.1 Summary of Primary Studies

6.1.1 Common Weakness Enumeration (CWE). The first sub-theme determines and analyzes the patterns of the SV types provided by *CWE* [133]. CWE is currently the standard for SV types with more than 900 entries. The first group of studies has focused on multi-class classification of these CWEs. Wang et al. [193] were the first to tackle this problem with a Naïve Bayes model using the CVSS metrics (version 2) [58] and product names. Later, Shuai et al. [174] used LDA [17] with a location-aware weighting to extract important features from SV descriptions for building an effective SVM-based CWE classifier. Na et al. [136] also showed that features extracted from SV descriptions can improve the Naïve Bayes model in Reference [193]. Ruohonen et al. [161] studied an information retrieval method, i.e., **term-frequency inverse document frequency (tf-idf)** and cosine similarity, to detect the CWE-ID with a description most similar to that of a given SV collected from NVD and Snyk.[12] This method performed well for CWEs without clear patterns/keywords in SV descriptions. Aota et al. [8] utilized the Boruta feature selection algorithm [104] and Random forest to improve the performance of *base* CWE classification. Base CWEs give more fine-grained information for SV remediation than categorical CWEs used in Reference [136].

There has been a recent rise in using neural network/DL-based models for CWE classification. Huang et al. [81] implemented a deep neural network with tf-idf and information gain for the task and obtained better performance than SVM, Naïve Bayes, and KNN. Aghaei et al. [1] improved upon Reference [8] for both categorical (coarse-grained) and base (fine-grained) CWE classification with an adaptive hierarchical neural network to determine sequences of less to more fine-grained CWEs. To capture the hierarchical structure and rare classes of CWEs, Das et al. [40] matched SV and CWE descriptions instead of predicting CWEs directly. They presented a deep Siamese network with a BERT-based [43] shared feature extractor that outperformed many baselines even for rare/unseen CWE classes. Recently, Zou et al. [220] pioneered the multi-class classification of CWE in vulnerable functions curated from **Software Assurance Reference Dataset (SARD)** [141] and NVD. They achieved high performance (~95% F1-score) with DL (Bi-LSTM) models. The strength of their model came from combining global (semantically related statements) and local (variables/statements affecting function calls) features. Note that this model currently only works for functions in C/C++ and 40 selected classes of CWE.

Another group of studies has considered unsupervised learning methods to extract CWE sequences, patterns, and relationships. Sequences of SV types over time were identified by Murtaza et al. [135] using an *n*-gram model. This model sheds light on both co-occurring and upcoming CWEs (grams), raising awareness of potential cascading attacks. Lin et al. [116] applied an association rule mining algorithm, FP-growth [73], to extract the rules/patterns of various CWEs aspects including types, programming language, time of introduction, and consequence scope. For example, buffer overflow (CWE type) usually appears during the implementation phase (time of introduction) in C/C++ (programming language) and affects the availability (consequence scope). Lately, Han et al. [74] developed a deep knowledge graph embedding technique to mine the relationships among CWE types, assisting in finding relevant SV types with similar properties.

6.1.2 Custom Vulnerability Types. The second sub-theme is about *custom vulnerability types* other than CWE. Venter et al. [190] used Self-organizing map [99], an unsupervised clustering algorithm, to group SVs with similar descriptions on CVE. This was one of the earliest studies that automated SV type classification. Topic modeling is another popular unsupervised learning

[12] https://snyk.io/vuln.

model [4, 134, 139, 189] to categorize SVs without an existing taxonomy. Neuhaus et al. [139] applied LDA [17] on SV descriptions to identify 28 prevalent SV types and then analyzed the trends of such types over time. The identified SV topics/types had considerable overlaps (up to 98% precision and 95% recall) with CWEs. Mounika et al. [134, 189] extended Reference [139] to map the LDA topics with the top-10 OWASP [149]. However, the LDA topics/keywords did not agree well (<40%) with the OWASP descriptions, probably because 10 topics did not cover all the underlying patterns of SV descriptions. Aljedaani et al. [4] again used LDA to identify 10 types of SVs reported in the bug tracking system of Chromium[13] and found memory-related issues were the most prevalent topics.

Another group of studies has classified manually defined/selected SV types rather than CWE, as some SV types are encountered more often in practice and require more attention. Williams et al. [200, 201] applied a supervised topical evolution model [137] to identify the features that best described the 10 pre-defined SV types[14] prevalent in the wild. These authors then used a diffusion-based storytelling technique [10] to show the evolution of a particular topic of SVs over time; e.g., increasing API-related SVs requires hardening the APIs used in a product. To support user-friendly SV assessment using ever-increasing unstructured SV data, Russo et al. [163] used Bayesian network to predict 10 pre-defined SV types.[15] Besides predicting manually defined SV types using SV natural language descriptions, Yan et al. [207] used a decision tree to predict 22 SV types prevalent in the executables of Linux applications. The predicted type was then combined with fuzzers' outputs to predict SV exploitability (see Section 3.1.1). Besides author-defined types, custom SV types also come from specific SV platforms. Zhang et al. [214] designed an ML-based framework to predict the SV types collected from China National Vulnerability Database. Ensemble models (bagging and boosting models) achieved, on average, the highest performance for this task.

6.2 Theme Discussion

In the *Type* theme, detecting and characterizing coarse-grained and fine-grained CWE-based SV types are the frequent tasks. The large number and hierarchical structure of classes are the main challenges with CWE classification/analysis. In terms of solutions, deep Siamese networks [40] are more robust to the class imbalance issue (due to many CWE classes), while graph-based neural networks [74] can effectively capture the hierarchical structure of CWEs. Future work can investigate the combination of these two types of DL architectures to solve both issues simultaneously. We also recommend more solutions for addressing the data imbalance issue in Section 9.1.3. Besides model-level solutions, author-selected or platform-specific SV types have been considered to reduce the complexity of CWE. However, similar to custom SV consequences in Section 4.1.2, there is not yet a universally accepted taxonomy for these custom SV types. To reduce the subjectivity in selecting SV types for prediction, we suggest that future work should focus on the types that are commonly encountered and discussed by developers in the wild (see Section 9.1.1 for more details).

7 MISCELLANEOUS TASKS

The last theme is *Miscellaneous Tasks* covering the studies that are representative yet do not fit into the four previous themes. This theme has three main sub-themes/tasks: (*i*) Vulnerability

[13] https://bugs.chromium.org/p/chromium/issues/list.

[14] 1. Buffer errors, 2. Cross-site scripting, 3. Path traversal, 4. Permissions and Privileges, 5. Input validation, 6. SQL injection, 7. Information disclosure, 8. Resources Error, 9. Cryptographic issues, 10. Code injection.

[15] 1. Authentication bypass or Improper Authorization, 2. Cross-site scripting or HTML injection, 3. Denial of service, 4. Directory Traversal, 5. Local/Remote file include and Arbitrary file upload, 6. Information disclosure and/or Arbitrary file read, 7. Buffer/stack/heap/integer overflow, 8. Remote code execution, 9. SQL injection, 10. Unspecified vulnerability.

Table 10. List of the Surveyed Papers in the *Miscellaneous Tasks* Theme

Study	Nature of task	Data source	Data-driven technique
Sub-theme: 1. Vulnerability Information Retrieval			
Weeraward-hana et al. [198]	*Multi-class classification*: Extraction of entities (software name/version, impact, attacker/user actions) from SV descriptions	NVD (210 randomly selected and manually labeled SVs)	Stanford Named Entity Recognizer implementing a CRF classifier
Dong et al. [46]	*Multi-class classification*: Vulnerable software names/versions	CVE Details, NVD, ExploitDB, SecurityFocus, SecurityFocus Forum, SecurityTracker, Openwall	Word-level and character-level Bi-LSTM with attention mechanism
Gonzalez et al. [68]	*Multi-class classification*: Extraction of 19 Vulnerability Description Ontology [142] classes from SV descriptions	NVD	Naïve Bayes, Decision tree, SVM, Random forest, Majority voting model
Binyamini et al. [15]	Multi-class classification: Extraction of entities (attack vector/means/technique, privilege, impact, vulnerable platform/version/OS, network protocol/port) from SV descriptions to generate MulVal [148] interaction rules	NVD	Bi-LSTM with various feature extractors: word2vec, ELMo, BERT (pre-trained or trained from scratch)
Guo et al. [71]	*Multi-class classification*: Extraction of entities (SV type, root cause, attack type, attack vector) from SV descriptions	NVD, SecurityFocus	CNN, Bi-LSTM (with or without attention mechanism)
Waareus et al. [196]	*Multi-class classification*: Common Product Enumeration (CPE)	NVD	Word-level and character-level Bi-LSTM
Yitagesu et al. [209]	*Multi-class classification*: Part-of-speech tagging of SV descriptions	NVD, CVE, CWE, CAPEC, CPE, Twitter, PTB corpus [122]	Bi-LSTM
Sun et al. [183]	*Multi-class classification*: Extraction of entities (vulnerable product/version/component, type, attack type, root cause, attack vector, impact) from ExploitDB to generate SV descriptions	NVD, ExploitDB	BERT models
Sub-theme: 2. Cross-source Vulnerability Patterns			
Horawalavith-ana et al. [79]	*Regression*: Number of software development activities on GitHub after disclosure of SVs	Twitter, Reddit, GitHub	MLP, LSTM
Xiao et al. [205]	*Knowledge-graph reasoning*: modeling the relationships among SVs, its types and attack patterns	CVE, CWE, CAPEC (Linux project)	Translation-based knowledge-graph embedding
Sub-theme: 3. Vulnerability Fixing Effort			
Othmane et al. [147]	*Regression*: time (days) to fix SVs	Proprietary SV data collected at the SAP company	Linear/Tree-based/Neural network regression

information retrieval, (*ii*) Cross-source vulnerability patterns, and (*iii*) Vulnerability fixing effort (see Table 10).

7.1 Summary of Primary Studies

7.1.1 Vulnerability Information Retrieval. The first and major sub-theme is *vulnerability information retrieval* that studies data-driven methods to extract different SV-related entities (e.g., affected products/versions) and their relationships from SV data. The current sub-theme extracts assessment information appearing explicitly in SV data (e.g., SV descriptions on NVD) rather than predicting implicit properties as done in prior sub-themes. For instance, CWE-119, i.e., "Improper Restriction of Write Operations within the Bounds of a Memory Buffer," can be retrieved directly from CVE-2020-28022, but not from CVE-2021-2122.[16] The latter case requires techniques from Section 6.1.1.

[16] https://nvd.nist.gov/vuln/detail/CVE-2020-28022 and https://nvd.nist.gov/vuln/detail/CVE-2021-21220.

Most of the retrieval methods in this sub-theme have been formulated under the multi-class classification setting. One of the earliest works was conducted by Weerawardhana et al. [198]. This study extracted software names/versions, impacts, and attacker's/user's action from SV descriptions on NVD using Stanford **Named Entity Recognition (NER)** technique, a.k.a. CRF classifier [56]. Later, Dong et al. [46] proposed to use a word/character-level Bi-LSTM to improve the performance of extracting vulnerable software names and versions from SV descriptions available on NVD and other SV databases/advisories (e.g., CVE Details [222], ExploitDB [168], SecurityFocus [84], SecurityTracker [169], and Openwall [155]). Based on the extracted entities, these authors also highlighted the inconsistencies in vulnerable software names and versions across different SV sources. Besides version products/names of SVs, Gonzalez et al. [68] used a majority vote of different ML models (e.g., SVM and Random forest) to extract the 19 entities of **Vulnerability Description Ontology (VDO)** [142] from SV descriptions to check the consistency of these descriptions based on the guidelines of VDO. Since 2020, there has been a trend in using DL models (e.g., Bi-LSTM, CNNs, or BERT [43]/ELMo [154]) to extract different information from SV descriptions including required elements for generating MulVal [148] attack rules [15] or SV types/root cause, attack type/vector [71], **Common Product Enumeration (CPE)** [131] for standardizing names of vulnerable vendors/products/versions [196], part-of-speech [209], and relevant entities (e.g., vulnerable products, attack type, root cause) from ExploitDB to generate SV descriptions [183]. BERT models [43], pre-trained on general text (e.g., Wikipedia pages [62] or PTB corpus [122]) and fine-tuned on SV text, have also been increasingly used to address the data scarcity/imbalance for the retrieval tasks.

7.1.2 Cross-source Vulnerability Patterns. The second sub-theme, *cross-source vulnerability patterns*, finds commonality and/or discovers latent relationships among SV sources to enrich information for SV assessment and prioritization. Horawalavithana et al. [79] found a positive correlation between development activities (e.g., push/pull requests and issues) on GitHub and SV mentions on Reddit[17] and Twitter. These authors then used DL models (MLP [76] and LSTM [77]) to predict the appearance and sequence of development activities when SVs were mentioned on the two social media platforms. Xiao et al. [205] applied a translation-based graph embedding method to encode and predict the relationships among different SVs and the respective attack patterns and types. This work [205] was based on DeepWeak of Han et al. [74], but it still belongs to this sub-theme as they provided a multi-dimensional view of SVs using three different sources (NVD [140], CWE [133], and CAPEC [130]). Xiao et al. [205] envisioned that their knowledge graph can be extended to incorporate the source code introducing/fixing SVs.

7.1.3 Vulnerability Fixing Effort. The last sub-theme is *vulnerability fixing effort* that focuses on estimating SV fixing effort through proxies such as the SV fixing time, usually in days. Othmane and the co-authors were among the first to approach this problem. These authors first conducted a large-scale qualitative study at the SAP company and identified 65 important code-based, process-based, and developer-based factors contributing to the SV fixing effort [11]. Later, the same group of authors [147] leveraged the identified factors in their prior qualitative study to develop various regression models such as linear regression, tree-based regression, and neural network regression models, to predict time-to-fix SVs using the data collected at SAP. These authors found that code components containing detected SVs are more important for the prediction than SV types.

[17] https://reddit.com.

7.2 Theme Discussion

In the *Miscellaneous Tasks* theme, the key focus is on retrieving SV-related entities and characteristics from SV descriptions. The retrieval tasks are usually formulated as Named Entity Recognition from SV descriptions. However, we observed that NVD descriptions do not follow a consistent template [7], posing significant challenges in labeling the entities for retrieval. The affected versions and vendor/product names of SVs also contain inconsistencies [7, 46], making the retrieval tasks difficult. We recommend that data normalization and cleaning should be performed before labeling entities and building respective retrieval models to ensure the reliability of results.

Besides information retrieval, other tasks such as linking multi-sources, extracting cross-source patterns, or estimating fixing effort are also useful to obtain richer SV information for assessment and prioritization, yet these tasks are still in early stages. Linking multiple sources and their patterns is the first step towards building an SV knowledge graph to answer different queries regarding a particular SV (e.g., what systems are affected, exploitation status, how to fix, or what SVs are similar). In the future, such a knowledge graph can be extended to capture artifacts of SVs in emerging software types like AI-based systems (see Section 9.3). Moreover, to advance SV fixing effort prediction, future work can consider adapting/customizing the existing practices/techniques used to predict fixing effort for general bugs [2, 212].

8 ANALYSIS OF DATA-DRIVEN APPROACHES FOR SOFTWARE VULNERABILITY ASSESSMENT AND PRIORITIZATION

We extract and analyze the five key elements for data-driven SV assessment and prioritization: (*i*) Data sources, (*ii*) Model features, (*iii*) Prediction models, (*iv*) Evaluation techniques, and (*v*) Evaluation metrics, as given in Table 11. These elements correspond to the four main steps in building data-driven models: data collection (data sources), feature engineering (model features), model training (prediction models), and model evaluation (evaluation techniques/metrics) [72, 164].

8.1 Data Sources

Identifying and collecting rich and reliable SV-related data are the first tasks to build data-driven models for automating SV assessment and prioritization tasks. As shown in Table 11, a wide variety of data sources has been considered to accomplish the five identified themes.

Across the five themes, NVD [140] and CVE [132] have been the most prevalently used data sources. The popularity of NVD/CVE is mainly because they publish expert-verified SV information that can be used to develop prediction models. First, many studies have considered SV descriptions on NVD/CVE as model inputs. Second, the SV characteristics on NVD have been heavily used as assessment outputs in all the themes, e.g., CVSS Exploitability metrics for *Exploitation*, CVSS Impact/Scope metrics for *Impact*, CVSS severity score/levels for *Severity*, CWE for *Type*, CWE/CPE for *Miscellaneous tasks*. Third, external sources on NVD/CVE have enabled many studies to obtain richer SV information (e.g., exploitation availability/time [30] or vulnerable code/crashes [187, 207]) and extract relationships among multiple SV sources to develop a knowledge graph of SVs (e.g., References [74, 205]). However, NVD/CVE still suffer from information inconsistencies [7, 46] and missing relevant external sources (e.g., SV fixing code) [78]. Such issues motivate future work to validate/clean NVD data and utilize more sources for code-based SV assessment and prioritization (see Section 9.1.1).

To enrich the SV information on NVD/CVE, many other security advisories and SV databases have been commonly leveraged by the reviewed studies, notably ExploitDB [168], Symantec [21, 22], SecurityFocus [84], CVE Details [222], and OSVDB. Most of these sources disclose PoC

Table 11. The Frequent Data Sources, Features, Models, Evaluation Techniques, and Evaluation Metrics Used for the Five Identified SV Assessment and Prioritization Themes

Source/Technique/Metric	Strengths	Weaknesses
Element: Data Source		
NVD/CVE/CVE Details (deprecated OSVDB)	• Report expert-verified information (with CVE-ID) • Contain CWE and CVSS entries for each SV • Link to external sources (official fixes or vendors' info)	• Missing/incomplete links to vulnerable code/fixes • Inconsistencies due to human errors • Delayed SV reporting and assignment of CVSS metrics
ExploitDB	• Report PoC exploits of SVs (with links to CVE-ID)	• May not lead to real exploits in the wild
Other security advisories (e.g., SecurityFocus, Symantec or X-Force)	• Report real-world exploits of SVs • Cover a wide range of SVs (including ones w/o CVE-ID)	• Some exploits may not have links to CVE entries for mapping with other assessment metrics
Informal sources (e.g., Twitter and darkweb)	• Early reporting of SVs (maybe even earlier than NVD) • Contain non-technical SV information (e.g., financial damage or socio-technical challenges in addressing SVs)	• Contain non-verified and even misleading information • May cause adversarial attacks to assessment models
Element: Model Feature		
BoW/tf-idf/n-grams	• Simple to implement • Strong baseline for text-based inputs (e.g., SV descriptions in security databases/advisories)	• May suffer from vocabulary explosion (e.g., many new description words for new SVs) • No consideration of word context/order (may be needed for code-based SV analysis) • Cannot handle Out-of-Vocabulary (OoV) words (can be resolved with subwords [113])
Word2vec	• Capture nearby context of each word • Can reuse existing pre-trained model(s)	• Cannot handle OoV words (can be resolved with fastText [18]) • No consideration of word order
DL model end-to-end trainable features	• Produce SV task-specific features	• May not produce high-quality representation for tasks with limited data (e.g., real-world exploit prediction)
Bidirectional Encoder Representations from Transformers (BERT)	• Capture contextual representation of text (i.e., the feature vector of a word is specific to each input) • Capture word order in an input • Can handle OoV words	• May require GPU to speed up feature inference • May be too computationally expensive and require too much data to train a strong model from scratch • May require fine-tuning to work well for a source task
Source/expert-defined metadata features	• Lightweight • Human interpretable for a task of interest	• Require SV expertise to define relevant features • Hard to generalize to new tasks
Element: Prediction Model		
Single ML models (e.g., Linear SVM, Logistic regression, NaÃŕve Bayes)	• Simple to implement • Efficient to (re-)train on large data (e.g., using the entire NVD database)	• May be prone to overfitting • Usually do not perform as well as ensemble/DL models
Ensemble ML models (e.g., Random forest, XGBoost, LGBM)	• Strong baseline (usually stronger than single models) • Less prone to overfitting	• Take longer to train than single models
Latent Dirichlet Allocation (LDA – topic modeling)	• Require no labeled data for training • Can provide features for supervised learning models	• Require SV expertise to manually label generated topics • May generate human non-interpretable topics
Deep Multi-Layer Perceptron (MLP)	• Work readily with tabular data (e.g., manually defined features or BoW/tf-idf/n-grams)	• Perform comparably yet are more costly compared to ensemble ML models • Less effective for unstructured data (e.g., SV descriptions)
Deep Convolutional Neural Networks (CNN)	• Capture local and hierarchical patterns of inputs • Usually perform better than MLP for text-based data	• Cannot effectively capture sequential order of inputs (maybe needed for code-based SV analysis)
Deep recurrent neural networks (e.g., LSTM or Bi-LSTM)	• Capture short-/long-term dependencies from inputs • Usually perform better than MLP for text-based data	• May suffer from the information bottleneck issue (can be resolved with attention mechanism [9]) • Usually take longer to train than CNNs
Deep graph neural networks (e.g., Graph convolutional network)	• Capture directed relationships among multiple SV entities and sources	• Require graph-based inputs to work • More computationally expensive than other DL models
Deep transfer learning with fine-tuning (e.g., BERT with task-specific classification layer(s))	• Can improve the performance for tasks with small data (e.g., real-world exploit prediction)	• Require target task to have similar nature as source task
Deep constrastive learning (e.g., Siamese neural networks)	• Can improve performance for tasks with small data • Robust to class imbalance (e.g., CWE classes)	• Computationally expensive (two inputs instead of one) • Do not directly produce class-wise probabilities
Deep multi-task learning	• Can share features for predicting multiple tasks (e.g., CVSS metrics) simultaneously • Reduce training/maintenance cost	• Require predicted tasks to be related • Hard to tune the prediction/performance of individual tasks
Element: Evaluation Technique		
Single k-CV without test	• Easy to implement • Reduce the randomness of results with multiple folds	• Do not have a separate test set for validating optimized models (can be resolved with separate test set(s)) • May be infeasible for expensive DL models • Use future data/SVs for training, maybe leading to biased results
Single/multiple random train/test with/without val (using val to tune hyperparameters)	• Easy to implement • Reduce the randomness of results (the multiple version)	• May produce unstable results (the single version) • May be infeasible for expensive DL models (the multiple version) • Use future data/SVs for training, maybe leading to biased results
Single/multiple time-based train/test with/without val (using val to tune hyperparameters)	• Consider the temporal properties of SVs, simulating the realistic evaluation of ever-increasing SVs in practice • Reduce the randomness of results (the multiple version)	• Similar drawbacks for the single & multiple versions as the random counterparts • May result in uneven/small splits (e.g., many SVs in a year)
Element: Evaluation Metric		
F1-score/Precision/Recall (classification)	• Suitable for imbalanced data (common in SV assessment and prioritization tasks)	• Do not consider True Negatives in a confusion matrix (can be resolved with Matthews Correlation Coefficient (MCC))
Accuracy (classification)	• Consider all the cells in a confusion matrix	• Unsuitable for imbalanced data (can be resolved with MCC)
Area Under the Curve (AUC) (classification)	• Independent of prediction thresholds	• May not represent real-world settings (i.e., as models in practice mostly use fixed classification thresholds) • ROC-AUC may not be suitable for imbalanced data (can be resolved with Precision-Recall-AUC)
Mean absolute (percentage) error/ Root mean squared error (regression)	• Show absolute performance of models	• May be hard to interpret a value on its own without domain knowledge (i.e., whether an error of x is sufficiently effective)
Correlation coefficient (r)/ Coef. of determination (R^2) (regression)	• Show relative performance of models (0 – 1), where 0 is worst & 1 is best	• R^2 always increases when adding any new feature (can be resolved with adjusted R^2)

Notes: The values are organized based on their overall frequency across the five themes. For the Prediction Model and Evaluation Metric elements, the values are first organized by categories (ML then DL for Prediction Model and classification then regression for Evaluation Metric) and then by frequencies. k-CV stands for k-fold cross-validation. The full list of values and their appearance frequencies for the five elements in the five themes can be found at Reference [108].

(ExploitDB and OSVDB) and/or real-world (Symantec and Security Focus) exploits. However, real-world exploits are much rarer and different compared to PoC ones [86, 165]. It is recommended that future work should explore more data sources (other than the ones in Table 2) and better methods to retrieve real-world exploits (see Section 9.1). Additionally, CVE Details and OSVDB are SV databases like NVD yet with a few key differences. CVE Details explicitly monitors Exploit-DB entries that may be missed on NVD and provides a more user-friendly interface to view/search SVs. OSVDB also reports SVs that do not appear on NVD (without CVE-ID), but this site was discontinued in [2016].

Besides official/expert-verified data sources, we have seen an increasing interest in mining SV information from informal sources that also contain non-expert generated content such as social media (e.g., Twitter) and darkweb. Especially, Twitter has been widely used for predicting exploits, as this platform has been shown to contain many SV disclosures even before official databases like NVD [29, 165]. Recently, darkweb forums/sites/markets have also gained traction, as SV mentions on these sites have a strong correlation with their exploits in the wild [5, 6]. However, SV-related data on these informal sources are much noisier, because they neither follow any pre-defined structure nor have any verification and they are even prone to fake news [165]. Thus, the data integrity of these sources should be checked, potentially by checking the reputation of posters, to avoid inputting unreliable data to prediction models and potentially producing misleading findings.

8.2 Model Features

Collected raw data need to be represented by suitable features for training prediction models. There are three key types of feature representation methods in this area: term frequency (e.g., BoW, tf-idf, and n-grams), DL-learned features (e.g., BERT and word2vec), and source/expert-defined metadata (e.g., CVSS metrics and CPE on NVD or tweet properties on Twitter), as summarized in Table 11.

Regarding the term-frequency-based methods, BoW has been the most popular one. Its popularity is probably because it is one of the simplest ways to extract features from natural language descriptions of SVs and directly compatible with popular ML models (e.g., Linear SVM, Logistic regression, and Random forest) in Section 8.3. Besides plain term count/frequency, other studies have also considered different weighting mechanisms such as **inverse document frequency weighting (tf-idf)** or **inverse gravity moment weighting (tf-igm)** [33]. Tf-igm has been shown to work better than BoW and tf-idf at classifying severity [32, 102]. Future work is still needed to evaluate the applicability and generalizability of tf-igm for other SV assessment and prioritization tasks.

Recently, **Neural Network (NN)** or DL-based features such as word2vec [129] and BERT [43] have been increasingly used to improve the performance of predicting CVSS exploitation/impact/-severity metrics [67, 75], CWE types [40], and SV information retrieval [71, 196]. Compared to BoW and its variants, NN and DL can extract more efficient and context-aware features from vast SV data [109]. NN/DL techniques rely on distributed representation to encode SV-related words using fixed-length vectors much smaller than a vocabulary size. Moreover, these techniques capture the sequential order and context (nearby words) to enable better SV-related text comprehension (e.g., SV vs. general *exploit*). Importantly, these NN/DL-learned features can be first trained in a non-SV domain with abundant data (e.g., Wikipedia pages [62]) and then transferred/fine-tuned in the SV domain to address limited/imbalanced SV data [208]. The main concern with these sophisticated NN/DL features is their limited interpretability, which is an exciting research area (see Section 9.2.2).

The metadata about SVs can also complement the missing information in descriptions or code for SV assessment and prioritization. For example, prediction of exploits and their characteristics have been enhanced using CVSS metrics [6], CPE [3], and SV types [13] on NVD.

Additionally, Twitter-related statistics (e.g., number of followers, likes, and retweets) have been shown to increase the performance of predicting SV exploitation, impact, and severity [30, 165]. Recently, alongside features extracted from vulnerable code, the information about a software development process and involved developers have also been extracted to predict SV fixing effort [147]. Currently, metadata-based and text-based features have been mainly integrated by concatenating their respective feature vectors (e.g., References [5, 6, 29, 31]). An alternative yet unexplored way is to build separate models for each feature type and then combine these models using meta-learning (e.g., model stacking [49]).

8.3 Prediction Models

The extracted features enter a wide variety of ML/DL-based prediction models shown in Table 11 to automate various SV assessment and prioritization tasks. Classification techniques have the largest proportion, while regression and unsupervised techniques are less common.

Linear SVM [37] has been the most frequently used classifier, especially in the Exploitation, Impact, and Severity themes. This popularity is expected as Linear SVM works well with the commonly used features, i.e., BoW and tf-idf. Besides Linear SVM, Random forest, Naïve Bayes, and Logistic regression have also been common classification models. In recent years, advanced boosting models (e.g., XGBoost [34] and LGBM [91]), and more lately, DL techniques (e.g., CNN [96] and (Bi-)LSTM with attention [9]) have been increasingly utilized and shown better results than simple ML models like Linear SVM or Logistic regression. In this area, some DL models are essential for certain tasks, e.g., building SV knowledge graph from multiple sources with graph neural networks [97]. DL models also offer solutions to data-related issues such as addressing class imbalance (e.g., deep Siamese network [158]) or improving data efficiency (e.g., deep multi-task learning [217]). Whenever applicable, it is recommended that future work should still consider simple baselines alongside sophisticated ones, as simple methods can perform on par with advanced ones [123].

Besides classification, various prediction models have also been investigated for regression (e.g., predicting exploit time, severity score, and fixing time). Linear SVM has again been the most commonly used regressor, as SV descriptions have usually been the regression input. Notably, many studies in the Severity theme did not build regression models to directly obtain the severity score (e.g., References [52, 88, 144, 177, 199]). Instead, they used the formulas defined by assessment frameworks (e.g., CVSS versions 2/3 [58, 59] or WIVSS [179]) to compute the severity score from the base metrics predicted by respective classification models. We argue that more effort should be invested in determining the severity score directly from SV data, as these severity formulas can be subjective [180]. We also observe that there is still limited use of DL models for regression compared to classification.

In addition to supervised (classification/regression) techniques, unsupervised learning has also been considered for extracting underlying patterns of SV data, especially in the Type theme. **Latent Dirichlet Allocation (LDA)** [17] has been the most commonly used topic model to identify latent topics/types of SVs without relying on a labeled taxonomy. The identified topics were mapped to the existing SV taxonomies such as CWE [139] and OWASP [134, 189]. The topics generated by topic models like LDA can also be used as features for classification/regression models [160] or building topic-wise models to capture local SV patterns [125]. However, definite interpretations for unsupervised outputs are challenging to obtain, as they usually rely on human judgement [150].

8.4 Evaluation Techniques

It is important to evaluate a trained model to ensure the model meets certain requirements (e.g., advancing the state-of-the-art). The evaluation generally needs to be conducted on a different set

of data other than the training set to avoid overfitting and objectively estimate model generalizability [76]. The commonly used evaluation techniques are summarized in Table 11.

The reviewed studies have mostly used one or multiple validation and/or test sets[18] to evaluate their models, in which each validation/test set has been either randomly or time-based selected. Specifically, k-fold cross-validation has been one of the most commonly used techniques. The number of folds has usually been 5 or 10, but less standard values like 4 [207] have also been used. However, k-fold cross-validation uses all parts of data at least once for training; thus, there is no hidden test set to evaluate the optimal model with the highest (cross-)validation performance.

To address the lack of hidden test set(s), a common practice in the studied papers has been to split a dataset into single training and test sets, sometimes with an additional validation set for tuning hyperparameters to obtain an optimal model. Recently, data has been increasingly split based on the published time of SVs to better reflect the changing nature of ever-increasing SVs [23, 113]. There have been various ratios for random (e.g., 80:20, 75:25, or 67:33) and time-based (e.g., week/month/year-wise) splits. However, the results reported using single validation/test sets may be unstable (i.e., unreproducible results using different set(s)) [157].

To ensure both the time order and reduce the result randomness, we recommend using multiple splits of training and test sets in combination with time-based validation in each training set. Statistical analyses (e.g., hypothesis testing and effect size) should also be conducted to confirm the reliability of findings with respect to the randomization of models/data in multiple runs [41].

8.5 Evaluation Metrics

Evaluating different aspects of a model requires respective proper metrics. The popular metrics for evaluating the tasks in each theme are given in Table 11.

Across the five themes, Accuracy, Precision, Recall, and F1-score [89] have been the most commonly used metrics because of a large number of classification tasks in the five themes. However, Accuracy is not a suitable measure for SV assessment and prioritization tasks with imbalanced data (e.g., SVs with real-world exploits vs. non-exploited SVs). The sample size of one class is much smaller than the others, and thus the overall Accuracy would be dominated by the majority classes. Besides these four commonly used metrics, **AUC based on the ROC curve (ROC-AUC)** [89] has also been considered, as it is threshold-independent. However, we suggest that ROC-AUC should be used with caution in practice, as most deployed models would have a fixed decision threshold (e.g., 0.5). Instead of ROC-AUC, we suggest **Matthews Correlation Coefficient (MCC)** [89] as a more meaningful evaluation metric to be considered, as it explicitly captures all values in a confusion matrix, and thus has less bias in results.

For regression tasks, various metrics have been used such as Mean absolute error, Mean absolute percentage error, Root mean squared error [177], as well as Correlation coefficient (r) and Coefficient of determination (R^2) [147]. Note that *adjusted* R^2 should be preferred over R^2, as R^2 would always increase when adding a new (even irrelevant) feature.

A model can have a higher value of one metric yet lower values of others.[19] Therefore, we suggest using a combination of suitable metrics for a task of interest to avoid result bias towards a specific metric. Currently, most studies have focused on evaluating model effectiveness, i.e., how well the predicted outputs match the ground-truth values. Besides effectiveness, other aspects

[18]Validation set(s) helps optimize/tune a model (finding the best task/data-specific hyperparameters), and test set(s) evaluates the optimized/tuned model. Using only validation set(s) means evaluating a model with default/pre-defined hyperparameters.

[19]https://stackoverflow.com/questions/34698161.

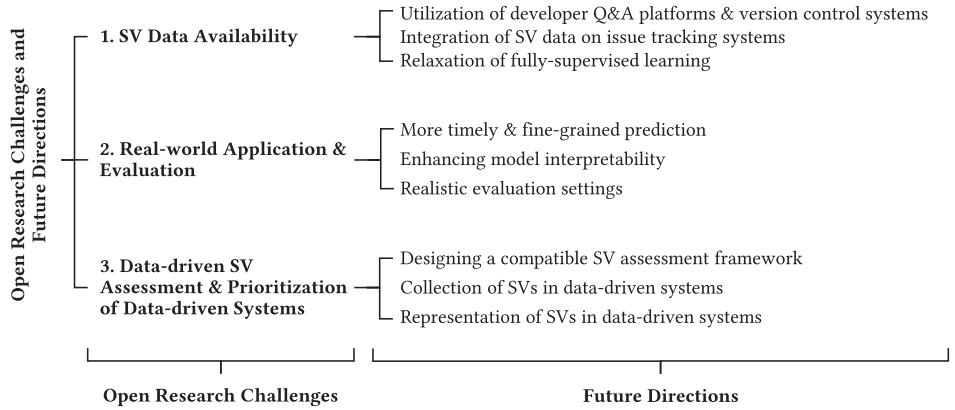

Fig. 2. List of challenges and future directions for data-driven SV assessment and prioritization.

(e.g., efficiency in training/deployment and robustness to input changes) of models should also be evaluated to provide a complete picture of model applicability in practice (see Section 9.2.3).

9 OPEN RESEARCH CHALLENGES AND FUTURE DIRECTIONS OF DATA-DRIVEN SOFTWARE VULNERABILITY ASSESSMENT AND PRIORITIZATION

We discuss three main open challenges with the reviewed studies of data-driven SV assessment and prioritization and then present nine potential directions to address such challenges (see Figure 2).

9.1 SV Data Availability

This section focuses on three key issues with the currently used data sources and potential solutions. First, the current data hardly contain specific developer's concerns and practices when addressing real-world SVs (Section 9.1.1). Second, the data sources still miss many SV-related bugs reported in issue-tracking systems, limiting the amount of data for training prediction models (Section 9.1.2). Third, some tasks/outputs (e.g., real-world exploit prediction) suffer from limited and/or imbalanced labeled data, potentially leading to unreliable performance of fully supervised models (Section 9.1.3).

9.1.1 Utilization of Developer Q&A Platforms and Version Control Systems. *Developer Question & Answer (Q&A) platforms* such as Stack Overflow and Security StackExchange[20] contain tens of thousands of posts about challenges and solutions shared by millions of developers when tackling known SVs in real-world scenarios [110]. One of the key insights of Le et al. [110]'s study is that the top SV types that developers usually struggle with are not always the same as those reported on SV databases (CWE [133] or OWASP [149]). Thus, future work should also consider real-world development-related issues discussed on developer Q&A platforms for automatically assessing and prioritizing SVs. For example, the fixing effort of SVs may depend on the technical difficulty of implementing the respective mitigation strategies in a language or system of interest.

Version control systems like GitHub[21] provide details about how developers addressed past SVs in real-world projects. Shrestha et al. [173] found developers sometimes discuss/disclose SV-related information on GitHub discussions even before the studied social media such as Twitter or Reddit. These findings show the potential of using GitHub discussions to complement the current sources

[20] Stack Overflow: https://stackoverflow.com and Security StackExchange: https://security.stackexchange.com.
[21] https://github.com.

for earlier SV assessment and prioritization. GitHub can also provide vulnerable code for performing assessment and prioritization for SVs rooted in source code [107], which is important yet has received limited attention from the community so far. Moreover, Walden [191] demonstrated the impact of a major SV (i.e., Heartbleed) on the characteristics (e.g., code complexity/style, contributors and development practices) of a single project (i.e., OpenSSL). Based on Walden's findings, future work can study whether the impact of an SV would be similar or different in multiple affected projects. Such investigation would give insights into the possibility of leveraging data from large projects to perform SV assessment and prioritization in smaller projects with the same/similar SVs.

9.1.2 Integration of SV Data on Issue Tracking Systems. *Issue/bug tracking systems* such as JIRA, Bugzilla, or GitHub issues[22] have been reporting numerous security-related bugs, many of which are SVs, but they have been underexplored for data-driven SV assessment and prioritization. Besides providing SV descriptions like CVE/NVD, these bug reports also contain other artifacts such as steps to reproduce, stack traces, and test cases that give extra information about SVs [219]. However, it is not trivial to obtain and integrate these SV-related bug reports with the ones on SV databases.

One way to retrieve SVs on issue tracking systems is to use security bug reports [14]. Much research work has been put into developing effective models to automatically retrieve security bug reports (e.g., References [65, 153, 203]). Among these studies, Wu et al. [203] manually verified and cleaned the security bug reports to provide a clean dataset for automated security bug report identification. However, more of such manual effort is still required to obtain up-to-date data, because the original security bug reports in Reference [203] were actually a part of the dataset collected back in 2014 [146].

It is worth noting that *not* all security bug reports are related to SVs such as issues/improvements in implementing security features.[23] Thus, future studies need to filter out these cases before using security bug reports for SV assessment and prioritization. We also emphasize that some SV-related bug reports are overlapping with the ones on NVD (e.g., the SV report AMBARI-14780[24] on JIRA refers to CVE-2016-0731 on CVE/NVD). Such overlaps would require data cleaning during the integration of reports on issue tracking systems and SV databases to avoid data duplication (e.g., similar SV descriptions) when developing SV assessment and prioritization models.

9.1.3 Relaxation of Fully Supervised Learning. Supervised learning models of many tasks in the five themes (see Section 8.3) require fully labeled data, but the data of some tasks are quite limited. To address the data-hungriness of these fully supervised learning models, future studies can approach the SV assessment and prioritization tasks with *low-shot learning* and/or *semi-supervised learning*.

Low-shot learning a.k.a. *few-shot learning* is designed to perform supervised learning using only a few examples per class, significantly reducing the labeling effort [195]. So far, only one study utilized low-shot learning with a deep Siamese network [40] (i.e., a shared feature model with similarity learning) to effectively predict SV types (CWE) and even generalize to unseen classes (i.e., zero-shot learning). There are still many opportunities for investigating different few-shot learning techniques for other SV assessment and prioritization tasks. Note that the shared features

[22] JIRA: https://www.atlassian.com/software/jira, BugZilla: https://www.bugzilla.org/, and GitHub issues: https://docs.github.com/en/issues.

[23] The security bug report AMBARI-1373 on JIRA (https://issues.apache.org/jira/browse/AMBARI-1373) was about improving the front-end of AMBARI Web by displaying the current logged in user.

[24] https://issues.apache.org/jira/browse/AMBARI-14780.

in few-shot learning can also be enhanced with pretrained models (e.g., BERT [43]) on another domain/task/project with more labeled data than the current task/project in the SV domain.

Semi-supervised learning enables training models with limited labeled data yet a large amount of unlabeled data [188], potentially leveraging hidden/unlabeled SVs in the wild. Recently, we have seen an increasing interest in using different techniques of this learning paradigm in the SV domain such as collecting SV patches using multi-view co-training [167] or retrieving SV discussions on developer Q&A sites using positive-unlabeled learning [111]. However, it is still little known about the effectiveness of semi-supervised learning for SV assessment and prioritization.

9.2 Real-world Application and Evaluation

The *experimental* performance of some SV assessment and prioritization models is promising, but the *real-world* applicability of such models is still questionable. First, these models may not be useful in practice due to delayed inputs and coarse-grained outputs (Section 9.2.1). Second, many models are black-box, limiting the understanding of the model predictions (Section 9.2.2). Third, some models are evaluated in over-optimistic conditions far from real-world scenarios (Section 9.2.3).

9.2.1 More Timely and Fine-grained Prediction. Although SV descriptions have been commonly used as model inputs (see Section 8.1), these descriptions are usually published long after SVs introduced/discovered in codebases [124]. One potential solution to this issue is to perform assessment and prioritization of SVs in code commits. Code commits contain changes made by developers to fix a bug/SV, implement a new feature or refactor code, and new SVs may be introduced in such changes [19]. Commit-level prediction would allow just-in-time SV assessment and prioritization as soon as SVs are introduced, reducing the waiting time for SV information to be verified and published on security advisories/databases [112]. It should be noted that report-level prediction is still important for assessing and prioritizing third-party libraries/software, especially the ones without available code (commits), and/or SVs missed by commit-level prediction.

CVSS [57] has been most frequently used for assessing the exploitability, impact, and severity levels/score of SVs (see Sections 3, 4, and 5), but there are increasing concerns that CVSS outputs are still generic. Specifically, Spring et al. [180] argued that CVSS tends to provide one-size-fits-all assessment metrics regardless of the context of SVs; i.e., the same SVs in different domains/environments are assigned the same metric values. For instance, banking systems may consider the confidentiality and integrity of databases more important than the availability of web/app interfaces. In the future, alongside CVSS, prediction models should also incorporate the domain/business knowledge to customize the assessment of SVs to a system of interest (e.g., the impact of SVs on critical component(s) and/or the readiness of developers/solutions for mitigating such SVs in the current system). Future case studies with practitioners are also desired to correlate the quantitative performance of models and their usability/usefulness in real-world systems (e.g., reducing more critical SVs yet using fewer resources).

9.2.2 Enhancing Model Interpretability. Model interpretability is important to increase the transparency of the predictions made by a model, allowing practitioners to adjust the model/-data to meet certain requirements [216]. Unfortunately, very few reviewed papers (e.g., References [75, 186]) explicitly discussed important features and/or explained why/when their models worked/failed for a task.

SV assessment and prioritization can draw inspiration from the related SV detection area where the interpretability of (DL-based) prediction models has been actively explored mainly by using (*i*) specific model architectures/parameters or (*ii*) external interpretation models/techniques [216]. In the first approach, prior studies successfully used the feature activation maps in a CNN model [162] or leveraged attention-based neural network [48] to highlight and visualize the

important code tokens that contribute to SVs. The second approach uses separate interpretation models on top of trained SV detectors. The interpretation models are either domain/model-agnostic [197], domain-agnostic yet specific to a model type (graph neural network [114]), or SV-specific [221]. The aforementioned approaches produce local/sample-wise interpretation, which can be aggregated to obtain global/task-wise interpretation. The global interpretation is similar to the feature importance of traditional ML models [28] such as the weights of linear models (e.g., Logistic regression) or the (im)purity of nodes split by each feature in tree-based models (e.g., Random forest). However, it is still unclear about the applicability/effectiveness of these approaches for interpreting ML/DL-based SV assessment and prioritization models, requiring further investigations.

9.2.3 Realistic Evaluation Settings. Most of the reviewed studies have evaluated their prediction models without capturing many factors encountered during the deployment of such models to production. Specifically, the models used in practice would require to handle new data and be robust against adversarial data from informal sources such as social media or darkweb.

There are concerns with both predicting and integrating new SV data. Regarding the prediction, Out-of-Vocabulary words in new data need to be properly accommodated to avoid performance degradation of prediction models [113]. Regarding the new data integration, online/incremental training on new data can be considered instead of batch training on the whole dataset to reduce computational cost [24]. The time-based splits should be used rather than random splits for evaluating online training to avoid leaking unseen (future) patterns to the model training (see Section 8.4).

Regarding the model robustness, only three reviewed studies considered adversarial attacks as part of their evaluation [6, 165, 204]. However, a recent survey shows the prevalence of adversarial attacks targeted models in cybersecurity [159]. Thus, there is certainly a need for more evaluation of adversarial robustness for SV assessment and prioritization models, especially DL-based ones.

9.3 Data-driven SV Assessment and Prioritization of Data-driven Systems

Compared to other systems, reporting/analyzing SVs of data-driven/**Artificial Intelligence (AI)**-based systems is still in its infancy [103]. Data-driven systems (e.g., smart recommender systems, chatbots, robots, and autonomous cars) are an emerging breed of systems whose cores are powered by AI technologies, e.g., ML and DL models built on data, rather than human-defined instructions as in traditional systems. We discuss three key challenges of SV assessment and prioritization of data-driven systems compared to traditional systems and suggest potential solutions. First, the current SV assessment frameworks need customizations to better reflect the nature of SVs in data-driven systems (Section 9.3.1). Second, there is a lack of SVs collected from real-world data-driven systems, limiting the potential of data-driven SV assessment and prioritization (Section 9.3.2). Third, the current models require redesign, especially in the SV representation, to capture unique characteristics and artifacts of data-driven systems (Section 9.3.3).

9.3.1 Designing a Compatible SV Assessment Framework. CVSS [57] is currently the most popular SV assessment framework for traditional systems, but its compatibility with data-driven systems still requires more investigation. The current CVSS documentation lacks instructions on how to assign metrics/score for SVs in data-driven systems. For example, it is unclear how to assign static CVSS metrics to systems with automatically updated data-driven models [35], because adversarial examples for exploitation would likely change after the models are updated. Such ambiguities should be clarified/resolved in future CVSS versions, as data-driven systems become more prevalent.

The types of SVs in ML/DL models in data-driven systems are also mostly different from the ones provided by CWE [133]. The difference is mainly because these new SVs do not only emerge

from configurations/code as in traditional systems, but also from training data and/or trained models [159]. Thus, we recommend that a new category of these SVs should be studied and potentially incorporated into CWE, similar to the newly added category for architectural SVs.[25]

9.3.2 Collection of SVs in Data-driven Systems. To the best of our knowledge, there has been no existing large-scale dataset of SVs in ML/DL models deployed in real-world data-driven systems. Very few of such SVs have been reported in the wild, one of which is CVE-2019-20634.[26] More of these SVs are required to help develop sufficiently effective SV assessment and prioritization models. One potential way to build such a dataset is to first match the (pre-trained) ML/DL models proposed in the literature or released on model repositories (e.g., Tensorflow Hub[27]) with the ones used in real-world systems either on version control systems or in mobile apps [83]. The matched models would then be tested against known adversarial attacks to identify corresponding SVs. Notably, significant effort is still required to define/label assessment outputs of these SVs (see Section 9.3.1).

9.3.3 Representation of SVs in Data-driven Systems. Existing SV assessment and prioritization models for traditional systems have not considered unique data/model-related characteristics/features of data-driven systems [192]. Specifically, data-driven systems also encompass information about data (e.g., format, type, size, and distribution) and ML/DL model(s) (e.g., configurations, parameters and performance). It is worth noting that SVs of ML/DL models in data-driven systems can also come from the frameworks used to develop such models (e.g., Tensorflow or Keras).[28] However, developers of data-driven systems may not be aware of the (security) issues in the used ML/DL frameworks [118]. Thus, besides currently used features, future work should also consider the information about underlying data/models and used ML/DL development frameworks to improve the SV representation for building models to assess and prioritize SVs in data-driven systems.

10 CONCLUSIONS

Assessment and prioritization are crucial phases to optimize resource utilization in addressing SVs at scale. The two phases have witnessed radical transformations following the increasing availability of SV data from multiple sources and advances in data-driven techniques. We presented a taxonomy to summarize the five main directions of the research work so far in this area. We identified and analyzed the key practices to develop data-driven models in the reviewed studies. We also highlighted the open challenges and suggested respective solutions to advance the field.

We envision the field will largely continue to improve the effectiveness of the presented tasks by leveraging more enriched data sources and sophisticated data-driven models, especially DL-based ones. Besides the improved performance, we also see many open opportunities/concerns in under-explored aspects of developing such advanced models. Overall, a deeper understanding of practitioners' concerns and real-world usage scenarios is the key to bridging the current gap between model development in academia and model deployment in production.

REFERENCES

[1] Ehsan Aghaei, Waseem Shadid, and Ehab Al-Shaer. 2020. ThreatZoom: CVE2CWE using hierarchical neural network. *arXiv preprint arXiv:2009.11501* (2020).

[2] Shirin Akbarinasaji, Bora Caglayan, and Ayse Bener. 2018. Predicting bug-fixing time: A replication study using an open source software project. *J. Syst. Softw.* 136 (2018), 173–186.

[25] https://cwe.mitre.org/data/definitions/1008.html.
[26] https://nvd.nist.gov/vuln/detail/CVE-2019-20634.
[27] https://www.tensorflow.org/hub.
[28] Tensorflow: https://github.com/tensorflow/tensorflow & Keras: https://github.com/keras-team/keras.

[3] M. Ugur Aksu, Kemal Bicakci, M. Hadi Dilek, A. Murat Ozbayoglu, and E. Islam Tatli. 2018. Automated generation of attack graphs using NVD. In *Proceedings of the 8th Conference on Data and Application Security and Privacy*. 135–142.
[4] Wajdi Aljedaani, Yasir Javed, and Mamdouh Alenezi. 2020. LDA categorization of security bug reports in chromium projects. In *Proceedings of the European Symposium on Software Engineering*. 154–161.
[5] Mohammed Almukaynizi, Eric Nunes, Krishna Dharaiya, Manoj Senguttuvan, Jana Shakarian, and Paulo Shakarian. 2017. Proactive identification of exploits in the wild through vulnerability mentions online. In *Proceedings of the International Conference on Cyber Conflict (CyCon US)*. IEEE, 82–88.
[6] Mohammed Almukaynizi, Eric Nunes, Krishna Dharaiya, Manoj Senguttuvan, Jana Shakarian, and Paulo Shakarian. 2019. Patch before exploited: An approach to identify targeted software vulnerabilities. In *AI in Cybersecurity*. Springer, 81–113.
[7] Afsah Anwar, Ahmed Abusnaina, Songqing Chen, Frank Li, and David Mohaisen. 2020. Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses. *arXiv preprint arXiv:2006.15074* (2020).
[8] Masaki Aota, Hideaki Kanehara, Masaki Kubo, Noboru Murata, Bo Sun, and Takeshi Takahashi. 2020. Automation of vulnerability classification from its description using machine learning. In *Proceedings of the IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 1–7.
[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).
[10] Roberto Camacho Barranco, Arnold P. Boedihardjo, and M. Shahriar Hossain. 2019. Analyzing evolving stories in news articles. *Int. J. Data Sci. Analyt.* 8, 3 (2019), 241–256.
[11] Lotfi ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, Achim D. Brucker, and Philip Miseldine. 2015. Factors impacting the effort required to fix security vulnerabilities. In *Proceedings of the International Conference on Information Security*. Springer, 102–119.
[12] Massimo Bernaschi, Emanuele Gabrielli, and Luigi V. Mancini. 2002. REMUS: A security-enhanced operating system. *ACM Trans. Inf. Syst. Secur.* 5, 1 (2002), 36–61.
[13] Navneet Bhatt, Adarsh Anand, and V. S. S. Yadavalli. 2021. Exploitability prediction of software vulnerabilities. *Qual. Reliab. Eng. Int.* 37, 2 (2021), 648–663.
[14] Farzana Ahamed Bhuiyan, Md Bulbul Sharif, and Akond Rahman. 2021. Security bug report usage for software vulnerability research: A systematic mapping study. *IEEE Access* 9 (2021), 28471–28495.
[15] Hodaya Binyamini, Ron Bitton, Masaki Inokuchi, Tomohiko Yagyu, Yuval Elovici, and Asaf Shabtai. 2021. A framework for modeling cyber attack techniques from security vulnerability descriptions. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2574–2583.
[16] David M. Blei and Jon D. McAuliffe. 2007. Supervised topic models. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*. 121–128.
[17] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *J. Mach. Learn. Res.* 3, Jan (2003), 993–1022.
[18] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Trans. Assoc. Computat. Ling.* 5 (2017), 135–146.
[19] Amiangshu Bosu, Jeffrey C. Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. 2014. Identifying the characteristics of vulnerable code changes: An empirical study. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 257–268.
[20] Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. 2010. Beyond heuristics: Learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 105–114.
[21] Broadcom. [n. d.]. Symantec attack signatures. Retrieved from https://bit.ly/symantec_att_sign.
[22] Broadcom. [n. d.]. Symantec threat explorer. Retrieved from https://bit.ly/symantec_threats.
[23] Benjamin L. Bullough, Anna K. Yanchenko, Christopher L. Smith, and Joseph R. Zipkin. 2017. Predicting exploitation of disclosed software vulnerabilities using open-source data. In *Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics*. 45–53.
[24] George G. Cabral, Leandro L. Minku, Emad Shihab, and Suhaib Mujahid. 2019. Class imbalance evolution and verification latency in just-in-time software defect prediction. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 666–676.
[25] CERT. [n. d.]. Basic fuzzing framework. Retrieved from https://bit.ly/basic_fuzzing_framework.
[26] Sang Kil Cha. [n. d.]. OFuzz. Retrieved from https://github.com/sangkilc/ofuzz.
[27] Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert, and David Brumley. 2012. Unleashing mayhem on binary code. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 380–394.
[28] Girish Chandrashekar and Ferat Sahin. 2014. A survey on feature selection methods. *Comput. Electric. Eng.* 40, 1 (2014), 16–28.

[29] Haipeng Chen, Jing Liu, Rui Liu, Noseong Park, and V. S. Subrahmanian. 2019. VASE: A Twitter-based vulnerability analysis and score engine. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. IEEE, 976–981.
[30] Haipeng Chen, Jing Liu, Rui Liu, Noseong Park, and V. S. Subrahmanian. 2019. VEST: A system for vulnerability exploit scoring & timing. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 6503–6505.
[31] Haipeng Chen, Rui Liu, Noseong Park, and V. S. Subrahmanian. 2019. Using Twitter to predict when vulnerabilities will be exploited. In *Proceedings of the 25th International Conference on Knowledge Discovery & Data Mining*. 3143–3152.
[32] Jinfu Chen, Patrick Kwaku Kudjo, Solomon Mensah, Selasie Aformaley Brown, and George Akorfu. 2020. An automatic software vulnerability classification framework using term frequency-inverse gravity moment and feature selection. *J. Syst. Softw.* 167 (2020), 110616.
[33] Kewen Chen, Zuping Zhang, Jun Long, and Hao Zhang. 2016. Turning from TF-IDF to TF-IGM for term weighting in text classification. *Exp. Syst. Applic.* 66 (2016), 245–260.
[34] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*. 785–794.
[35] Yang Chen, Andrew E. Santosa, Ang Ming Yi, Abhishek Sharma, Asankhaya Sharma, and David Lo. 2020. A machine learning approach for vulnerability curation. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 32–42.
[36] Zhongqiang Chen, Yuan Zhang, and Zhongrong Chen. 2010. A categorization framework for common computer vulnerabilities and exposures. *Comput. J.* 53, 5 (2010), 551–580.
[37] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Mach. Learn.* 20, 3 (1995), 273–297.
[38] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive aggressive algorithms. (2006).
[39] Daniela S. Cruzes and Tore Dybå. 2011. Research synthesis in software engineering: A tertiary study. *Inf. Softw. Technol.* 53, 5 (2011), 440–455.
[40] Siddhartha Shankar Das, Edoardo Serra, Mahantesh Halappanavar, Alex Pothen, and Ehab Al-Shaer. 2021. V2W-BERT: A framework for effective hierarchical multiclass classification of software vulnerabilities. *arXiv preprint arXiv:2102.11498* (2021).
[41] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, Lucas Gren, Carlo A. Furia, and Ziwei Huang. 2019. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. *J. Syst. Softw.* 156 (2019), 246–267.
[42] Daniel Alves de Sousa, Elaine Ribeiro de Faria, and Rodrigo Sanches Miani. 2020. Evaluating the performance of Twitter-based exploit detectors. *arXiv preprint arXiv:2011.03113* (2020).
[43] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
[44] Nesara Dissanayake, Asangi Jayatilaka, Mansooreh Zahedi, and M. Ali Babar. 2020. Software security patch management—a systematic literature review of challenges, approaches, tools and practices. *arXiv preprint arXiv:2012.00544* (2020).
[45] Brendan Dolan-Gavitt, Patrick Hulin, Engin Kirda, Tim Leek, Andrea Mambretti, Wil Robertson, Frederick Ulrich, and Ryan Whelan. 2016. LAVA: Large-scale automated vulnerability addition. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*. IEEE, 110–121.
[46] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the detection of inconsistencies in public security vulnerability reports. In *Proceedings of the 28th USENIX Security Symposium*. 869–885.
[47] Xuanyu Duan, Mengmeng Ge, Triet Huynh Minh Le, Faheem Ullah, Shang Gao, Xuequan Lu, and M. Ali Babar. 2021. Automated security assessment for the internet of things. In *Proceedings of the IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 47–56.
[48] Xu Duan, Jingzheng Wu, Shouling Ji, Zhiqing Rui, Tianyue Luo, Mutian Yang, and Yanjun Wu. 2019. VulSniper: Focus your attention to shoot fine-grained vulnerabilities. In *Proceedings of the International Joint Conference on Artificial Intelligence*. 4665–4671.
[49] Saso Dzeroski and Bernard Zenko. 2002. Is combining classifiers better than selecting the best one? In *Proceedings of the International Conference on Machine Learning*. Citeseer, 123e30.
[50] Michel Edkrantz. 2015. *Predicting Exploit Likelihood for Cyber Vulnerabilities with Machine Learning*. Master's thesis.
[51] Michel Edkrantz, Staffan Truvé, and Alan Said. 2015. Predicting vulnerability exploits in the wild. In *Proceedings of the IEEE 2nd International Conference on Cyber Security and Cloud Computing*. IEEE, 513–514.
[52] Clément Elbaz, Louis Rilling, and Christine Morin. 2020. Fighting N-day vulnerabilities with automated CVSS vector prediction at disclosure. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 1–10.

[53] ESET. [n. d.]. ESET security advisories. Retrieved from https://bit.ly/eset_virus.
[54] João Rafael Gonçalves Evangelista, Renato José Sassi, Márcio Romero, and Domingos Napolitano. 2020. Systematic literature review to investigate the application of open source intelligence (OSINT) with artificial intelligence. *J. Appl. Secur. Res.* (2020), 1–25.
[55] Yong Fang, Yongcheng Liu, Cheng Huang, and Liang Liu. 2020. FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. *PLoS One* 15, 2 (2020), e0228439.
[56] Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. 363–370.
[57] FIRST. [n. d.]. Common Vulnerability Scoring System. Retrieved from https://www.first.org/cvss.
[58] FIRST. [n. d.]. CVSS version 2. Retrieved from https://www.first.org/cvss/v2/guide.
[59] FIRST. [n. d.]. CVSS version 3. Retrieved from https://www.first.org/cvss/v3.0/specification-document.
[60] FIRST. [n. d.]. CVSS version 3.1. Retrieved from https://www.first.org/cvss/v3.1/specification-document.
[61] Park Foreman. 2019. *Vulnerability Management*. CRC Press.
[62] Wikimedia Foundation. [n. d.]. Wikipedia pages. Retrieved from https://www.wikipedia.org.
[63] Recorded Future. [n. d.]. Recorded Future security advisories. Retrieved from https://bit.ly/rf_sec.
[64] Marian Gawron, Feng Cheng, and Christoph Meinel. 2017. Automatic vulnerability classification using machine learning. In *Proceedings of the International Conference on Risks and Security of Internet and Systems*. Springer, 3–17.
[65] Michael Gegick, Pete Rotella, and Tao Xie. 2010. Identifying security bug reports via text mining: An industrial case study. In *Proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR'10)*. IEEE, 11–20.
[66] Seyed Mohammad Ghaffarian and Hamid Reza Shahriari. 2017. Software vulnerability analysis and discovery using machine-learning and data-mining techniques: A survey. *ACM Comput. Surv.* 50, 4 (2017), 1–36.
[67] Xi Gong, Zhenchang Xing, Xiaohong Li, Zhiyong Feng, and Zhuobing Han. 2019. Joint prediction of multiple vulnerability characteristics through multi-task learning. In *Proceedings of the 24th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 31–40.
[68] Danielle Gonzalez, Holly Hastings, and Mehdi Mirakhorli. 2019. Automated characterization of software vulnerabilities. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 135–139.
[69] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.
[70] Gustavo Grieco, Guillermo Luis Grinblat, Lucas Uzal, Sanjay Rawat, Josselin Feist, and Laurent Mounier. 2016. Toward large-scale vulnerability discovery using machine learning. In *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy*. 85–96.
[71] Hao Guo, Zhenchang Xing, and Xiaohong Li. 2020. Predicting missing information of key aspects in vulnerability reports. *arXiv preprint arXiv:2008.02456* (2020).
[72] Jiawei Han, Micheline Kamber, and Jian Pei. 2011. Data mining concepts and techniques third edition. *Morg. Kauf. Series Data Manag. Syst.* 5, 4 (2011), 83–124.
[73] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. *ACM SIGMOD Rec.* 29, 2 (2000), 1–12.
[74] Zhuobing Han, Xiaohong Li, Hongtao Liu, Zhenchang Xing, and Zhiyong Feng. 2018. DeepWeak: Reasoning common software weaknesses via knowledge graph embedding. In *Proceedings of the IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 456–466.
[75] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. 2017. Learning to predict severity of software vulnerability using only vulnerability description. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 125–136.
[76] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.
[77] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computat.* 9, 8 (1997), 1735–1780.
[78] Daan Hommersom, Antonino Sabetta, Bonaventura Coppola, and Damian A. Tamburri. 2021. Automated mapping of vulnerability advisories onto their fix commits in open source repositories. *arXiv preprint arXiv:2103.13375* (2021).
[79] Sameera Horawalavithana, Abhishek Bhattacharjee, Renhao Liu, Nazim Choudhury, Lawrence O. Hall, and Adriana Iamnitchi. 2019. Mentions of security vulnerabilities on Reddit, Twitter and Github. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*. 200–207.
[80] Susan Horwitz, Thomas Reps, and David Binkley. 1990. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.* 12, 1 (1990), 26–60.
[81] Guoyan Huang, Yazhou Li, Qian Wang, Jiadong Ren, Yongqiang Cheng, and Xiaolin Zhao. 2019. Automatic classification method for software vulnerability based on deep neural network. *IEEE Access* 7 (2019), 28291–28298.

[82] Shin-Ying Huang and Yiju Wu. 2020. Dynamic software vulnerabilities threat prediction through social media contextual analysis. In *Proceedings of the 15th Asia Conference on Computer and Communications Security*. 892–894.
[83] Yujin Huang, Han Hu, and Chunyang Chen. 2021. Robustness of on-device models: Adversarial attack to deep learning models on android apps. In *Proceedings of the IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 101–110.
[84] SecurityFocus Inc. [n. d.]. BugTraq vulnerability database. Retrieved from http://www.securityfocus.com.
[85] Secunia Inc. [n. d.]. Secunia vulnerability advisories. Retrieved from http://secunia.com.
[86] Jay Jacobs, Sasha Romanosky, Idris Adjerid, and Wade Baker. 2020. Improving vulnerability remediation through better exploit prediction. *J. Cybersecur.* 6, 1 (2020), tyaa015.
[87] Jay Jacobs, Sasha Romanosky, Benjamin Edwards, Michael Roytman, and Idris Adjerid. 2019. Exploit prediction scoring system (EPSS). *arXiv preprint arXiv:1908.04856* (2019).
[88] Yuning Jiang and Yacine Atif. 2020. An approach to discover and assess vulnerability severity automatically in cyber-physical systems. In *Proceedings of the 13th International Conference on Security of Information and Networks*. 1–8.
[89] Yasen Jiao and Pufeng Du. 2016. Performance measures in evaluating machine learning based bioinformatics predictors for classifications. *Quantitat. Biol.* 4, 4 (2016), 320–330.
[90] Kenta Kanakogi, Hironori Washizaki, Yoshiaki Fukazawa, Shinpei Ogata, Takao Okubo, Takehisa Kato, Hideyuki Kanuka, Atsuo Hazeyama, and Nobukazu Yoshioka. 2021. Tracing CAPEC attack patterns from CVE vulnerability information using natural language processing technique. In *Proceedings of the 54th Hawaii International Conference on System Sciences*. 6996.
[91] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A highly efficient gradient boosting decision tree. *Adv. Neural Inf. Process. Syst.* 30 (2017), 3146–3154.
[92] Staffs Keele et al. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report. Technical report, Ver. 2.3 EBSE Technical Report. EBSE.
[93] Inc. Kenna Security. [n. d.]. Kenna Security. Retrieved from http://www.kennasecurity.com.
[94] Saad Khan and Simon Parkinson. 2018. Review into state of the art of vulnerability assessment using artificial intelligence. In *Guide to Vulnerability Analysis for Computer Networks and Systems*. Springer, 3–32.
[95] Atefeh Khazaei, Mohammad Ghasemzadeh, and Vali Derhami. 2016. An automatic method for CVSS score prediction using vulnerabilities description. *J. Intell. Fuzzy Syst.* 30, 1 (2016), 89–96.
[96] Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1746–1751.
[97] Thomas N. Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).
[98] Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. 2014. Multilayer networks. *J. Complex Netw.* 2, 3 (2014), 203–271.
[99] Teuvo Kohonen. 1990. The self-organizing map. *Proc. IEEE* 78, 9 (1990), 1464–1480.
[100] Kyriakos Kritikos, Kostas Magoutis, Manos Papoutsakis, and Sotiris Ioannidis. 2019. A survey on vulnerability assessment tools and databases for cloud-based web applications. *Array* 3 (2019), 100011.
[101] Patrick Kwaku Kudjo, Jinfu Chen, Solomon Mensah, Richard Amankwah, and Christopher Kudjo. 2020. The effect of bellwether analysis on software vulnerability severity prediction models. *Softw. Qual. J.* (2020), 1–34.
[102] Patrick Kwaku Kudjo, Jinfu Chen, Minmin Zhou, Solomon Mensah, and Rubing Huang. 2019. Improving the accuracy of vulnerability report classification using term frequency-inverse gravity moment. In *Proceedings of the IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 248–259.
[103] Ram Shankar Siva Kumar, Jonathon Penney, Bruce Schneier, and Kendra Albert. 2020. Legal risks of adversarial machine learning research. *arXiv preprint arXiv:2006.16179* (2020).
[104] Miron B. Kursa, Aleksander Jankowski, and Witold R. Rudnicki. 2010. Boruta—A system for feature selection. *Fundamenta Informaticae* 101, 4 (2010), 271–285.
[105] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[106] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1188–1196.
[107] Triet H. M. Le and M. Ali Babar. 2022. On the use of fine-grained vulnerable code statements for software vulnerability assessment models. *arXiv preprint arXiv:2203.08417* (2022).
[108] Triet H. M. Le, Huaming Chen, and M. Ali Babar. [n. d.]. Supplementary materials. Retrieved from https://figshare.com/s/da4d238ecdf9123dc0b8.
[109] Triet H. M. Le, Hao Chen, and M. Ali Babar. 2020. Deep learning for source code modeling and generation: Models, applications, and challenges. *ACM Comput. Surv.* 53, 3 (2020), 1–38.

[110] Triet H. M. Le, Roland Croft, David Hin, and M. Ali Babar. 2021. A large-scale study of security vulnerability support on developer Q&A websites. In *Evaluation and Assessment in Software Engineering*. 109–118.

[111] Triet H. M. Le, David Hin, Roland Croft, and M. Ali Babar. 2020. PUMiner: Mining security posts from developer question and answer websites with PU learning. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 350–361.

[112] Triet H. M. Le, David Hin, Roland Croft, and M. Ali Babar. 2021. DeepCVA: Automated commit-level vulnerability assessment with deep multi-task learning. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 717–729.

[113] Triet H. M. Le, Bushra Sabir, and Muhammad Ali Babar. 2019. Automated software vulnerability assessment with concept drift. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 371–382.

[114] Yi Li, Shaohua Wang, and Tien N. Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*.

[115] Guanjun Lin, Sheng Wen, Qing-Long Han, Jun Zhang, and Yang Xiang. 2020. Software vulnerability detection using deep neural networks: A survey. *Proc. IEEE* 108, 10 (2020), 1825–1848.

[116] Zhechao Lin, Xiang Li, and Xiaohui Kuang. 2017. Machine learning in vulnerability databases. In *Proceedings of the 10th International Symposium on Computational Intelligence and Design (ISCID)*. IEEE, 108–113.

[117] Hailong Liu and Bo Li. 2019. Automated classification of attacker privileges based on deep neural network. In *Proceedings of the International Conference on Smart Computing and Communication*. Springer, 180–189.

[118] Jiakun Liu, Qiao Huang, Xin Xia, Emad Shihab, David Lo, and Shanping Li. 2020. Is using deep learning frameworks free? Characterizing technical debt in deep learning frameworks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society*. 1–10.

[119] Kai Liu, Yun Zhou, Qingyong Wang, and Xianqiang Zhu. 2019. Vulnerability severity prediction with deep neural network. In *Proceedings of the 5th International Conference on Big Data and Information Analytics (BigDIA)*. IEEE, 114–119.

[120] Ruchika Malhotra et al. 2021. Severity prediction of software vulnerabilities using textual data. In *Proceedings of the International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications*. Springer, 453–464.

[121] Pratyusa K. Manadhata and Jeannette M. Wing. 2010. An attack surface metric. *IEEE Trans. Softw. Eng.* 37, 3 (2010), 371–386.

[122] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. (1993).

[123] A. Mazuera-Rozo, A. Mojica-Hanke, M. Linares-Vasquez, and G. Bavota. 2021. Shallow or deep? An empirical study on detecting vulnerabilities using deep learning. In *Proceedings of the IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. 276–287.

[124] Andrew Meneely, Harshavardhan Srinivasan, Ayemi Musa, Alberto Rodriguez Tejeda, Matthew Mokary, and Brian Spates. 2013. When a patch goes bad: Exploring the properties of vulnerability-contributing commits. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 65–74.

[125] Tim Menzies, Suvodeep Majumder, Nikhila Balaji, Katie Brey, and Wei Fu. 2018. 500+ times faster than deep learning: A case study exploring faster methods for text mining stackoverflow. In *Proceedings of the IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE, 554–563.

[126] Trend Micro. [n. d.]. Trend Micro security advisories. Retrieved from https://bit.ly/trend_micro_sec.

[127] Trend Micro. [n. d.]. ZeroDay Initiative security advisories. Retrieved from https://bit.ly/zeroday_sec.

[128] Microsoft. [n. d.]. Microsoft security advisories. Retrieved from https://bit.ly/ms_sec_advisories.

[129] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546* (2013).

[130] MITRE. [n. d.]. Common Attack Pattern Enumeration and Classification. Retrieved from https://capec.mitre.org.

[131] MITRE. [n. d.]. Common Platform Enumeration. Retrieved from https://cpe.mitre.org.

[132] MITRE. [n. d.]. Common Vulnerabilities and Exposures. Retrieved from https://cve.mitre.org/.

[133] MITRE. [n. d.]. Common Weakness Enumeration. Retrieved from https://cwe.mitre.org.

[134] Vanamala Mounika, Xiaohong Yuan, and Kanishka Bandaru. 2019. Analyzing CVE database using unsupervised topic modelling. In *Proceedings of the International Conference on Computational Science and Computational Intelligence*. 72–77.

[135] Syed Shariyar Murtaza, Wael Khreich, Abdelwahab Hamou-Lhadj, and Ayse Basar Bener. 2016. Mining trends and patterns of software vulnerabilities. *J. Syst. Softw.* 117 (2016), 218–228.

[136] Sarang Na, Taeeun Kim, and Hwankuk Kim. 2016. A study on the classification of common vulnerabilities and exposures using naïve Bayes. In *Proceedings of the International Conference on Broadband and Wireless Computing, Communication and Applications*. Springer, 657–662.

[137] Sheikh Motahar Naim, Arnold P. Boedihardjo, and M. Shahriar Hossain. 2017. A scalable model for tracking topical evolution in large document collections. In *Proceedings of the IEEE International Conference on Big Data (Big Data)*. IEEE, 726–735.
[138] Shunta Nakagawa, Tatsuya Nagai, Hideaki Kanehara, Keisuke Furumoto, Makoto Takita, Yoshiaki Shiraishi, Takeshi Takahashi, Masami Mohri, Yasuhiro Takano, and Masakatu Morii. 2019. Character-level convolutional neural network for predicting severity of software vulnerability from vulnerability description. *IEICE Trans. Inf. Syst.* 102, 9 (2019), 1679–1682.
[139] Stephan Neuhaus and Thomas Zimmermann. 2010. Security trend analysis with CVE topic models. In *Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 111–120.
[140] NIST. [n. d.]. National Vulnerability Database. Retrieved from https://nvd.nist.gov.
[141] NIST. [n. d.]. Software Assurance Reference Dataset (SARD). Retrieved from https://samate.nist.gov/SRD.
[142] NIST. [n. d.]. Vulnerability description ontology. Retrieved from https://bit.ly/nist_vdo.
[143] Eric Nunes, Ahmad Diab, Andrew Gunn, Ericsson Marin, Vineet Mishra, Vivin Paliath, John Robertson, Jana Shakarian, Amanda Thart, and Paulo Shakarian. 2016. Darknet and Deepnet mining for proactive cybersecurity threat intelligence. In *Proceedings of the IEEE Conference on Intelligence and Security Informatics (ISI)*. IEEE, 7–12.
[144] Saahil Ognawala, Ricardo Nales Amato, Alexander Pretschner, and Pooja Kulkarni. 2018. Automatically assessing vulnerabilities discovered by compositional analysis. In *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis*. 16–25.
[145] Saahil Ognawala, Martín Ochoa, Alexander Pretschner, and Tobias Limmer. 2016. MACKE: Compositional analysis of low-level vulnerabilities with symbolic execution. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 780–785.
[146] Masao Ohira, Yutaro Kashiwa, Yosuke Yamatani, Hayato Yoshiyuki, Yoshiya Maeda, Nachai Limsettho, Keisuke Fujino, Hideaki Hata, Akinori Ihara, and Kenichi Matsumoto. 2015. A dataset of high impact bugs: Manually-classified issue reports. In *Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 518–521.
[147] Lotfi Ben Othmane, Golriz Chehrazi, Eric Bodden, Petar Tsalovski, and Achim D. Brucker. 2017. Time for addressing software security issues: Prediction models and impacting factors. *Data Sci. Eng.* 2, 2 (2017), 107–124.
[148] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. 2005. MulVAL: A logic-based network security analyzer. In *Proceedings of the USENIX Security Symposium*. 113–128.
[149] OWASP. [n. d.]. Open Web Application Security Project. Retrieved from https://bit.ly/owasp_main.
[150] Julio-Omar Palacio-Niño and Fernando Berzal. 2019. Evaluation metrics for unsupervised learning algorithms. *arXiv preprint arXiv:1905.05667* (2019).
[151] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* 22, 10 (2009), 1345–1359.
[152] Javier Pastor-Galindo, Pantaleone Nespoli, Félix Gómez Mármol, and Gregorio Martínez Pérez. 2020. The not yet exploited goldmine of OSINT: Opportunities, open challenges and future trends. *IEEE Access* 8 (2020), 10282–10304.
[153] Fayola Peters, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. 2017. Text filtering and ranking for security bug report prediction. *IEEE Trans. Softw. Eng.* 45, 6 (2017), 615–631.
[154] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365* (2018).
[155] Openwall Project. [n. d.]. Openwall security advisories. Retrieved from https://bit.ly/sec_openwall.
[156] Rapid7. [n. d.]. Metasploit security advisories. Retrieved from https://www.rapid7.com/db/modules.
[157] Sebastian Raschka. 2018. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808* (2018).
[158] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese BERT-networks. *arXiv preprint arXiv:1908.10084* (2019).
[159] Ishai Rosenberg, Asaf Shabtai, Yuval Elovici, and Lior Rokach. 2021. Adversarial machine learning attacks and defense methods in the cyber security domain. *ACM Comput. Surv.* 54, 5 (2021), 1–36.
[160] Jukka Ruohonen. 2017. Classifying web exploits with topic modeling. In *Proceedings of the 28th International Workshop on Database and Expert Systems Applications (DEXA)*. IEEE, 93–97.
[161] Jukka Ruohonen and Ville Leppänen. 2018. Toward validation of textual information retrieval techniques for software weaknesses. In *Proceedings of the International Conference on Database and Expert Systems Applications*. Springer, 265–277.
[162] Rebecca Russell, Louis Kim, Lei Hamilton, Tomo Lazovich, Jacob Harer, Onur Ozdemir, Paul Ellingwood, and Marc McConley. 2018. Automated vulnerability detection in source code using deep representation learning. In *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 757–762.
[163] Ernesto R. Russo, Andrea D. Sorbo, Corrado A. Visaggio, and Gerardo Canfora. 2019. Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities. *J. Syst. Softw.* 156 (2019), 84–99.

[164] Bushra Sabir, Faheem Ullah, M. Ali Babar, and Raj Gaire. 2021. Machine learning for detecting data exfiltration: A review. *ACM Comput. Surv.* 54, 3 (2021), 1–47.
[165] Carl Sabottke, Octavian Suciu, and Tudor Dumitraş. 2015. Vulnerability disclosure in the age of social media: Exploiting Twitter for predicting real-world exploits. In *Proceedings of the 24th USENIX Security Symposium*. 1041–1056.
[166] Sefa Eren Sahin and Ayse Tosun. 2019. A conceptual replication on predicting the severity of software vulnerabilities. In *Proceedings of the Evaluation and Assessment on Software Engineering*. 244–250.
[167] Arthur D. Sawadogo, Tegawendé F. Bissyandé, Naouel Moha, Kevin Allix, Jacques Klein, Li Li, and Yves Le Traon. 2020. Learning to catch security patches. *arXiv preprint arXiv:2001.09148* (2020).
[168] Offensive Security. [n. d.]. Exploit Database. Retrieved from https://www.exploit-db.com.
[169] SecurityTracker. [n. d.]. SecurityTracker vulnerability database. Retrieved from https://securitytracker.com.
[170] Abubakar Omari Abdallah Semasaba, Wei Zheng, Xiaoxue Wu, and Samuel Akwasi Agyemang. 2020. Literature survey of deep learning-based vulnerability analysis on source code. *IET Softw.* (2020).
[171] Internet Security Services. [n. d.]. Online database X-Force. Retrieved from http://www.iss.net/xforce.
[172] Ruchi Sharma, Ritu Sibal, and Sangeeta Sabharwal. 2021. Software vulnerability prioritization using vulnerability description. *Int. J. Syst. Assur. Eng. Manag.* 12, 1 (2021), 58–64.
[173] Prasha Shrestha, Arun Sathanur, Suraj Maharjan, Emily Saldanha, Dustin Arendt, and Svitlana Volkova. 2020. Multiple social platforms reveal actionable signals for software vulnerability awareness: A study of Github, Twitter and Reddit. *PLoS One* 15, 3 (2020), e0230250.
[174] Bo Shuai, Haifeng Li, Mengjun Li, Quan Zhang, and Chaojing Tang. 2013. Automatic classification for vulnerability based on machine learning. In *Proceedings of the IEEE International Conference on Information and Automation (ICIA)*. IEEE, 312–318.
[175] Shashank Kumar Singh and Amrita Chaturvedi. 2020. Applying deep learning for discovery and analysis of software vulnerabilities: A brief survey. *Soft Comput.: Theor. Applic.* (2020), 649–658.
[176] Vincent Smyth. 2017. Software vulnerability management: How intelligence helps reduce the risk. *Netw. Secur.* 2017, 3 (2017), 10–12.
[177] Georgios Spanos and Lefteris Angelis. 2018. A multi-target approach to estimate software vulnerability characteristics and severity scores. *J. Syst. Softw.* 146 (2018), 152–166.
[178] Georgios Spanos, Lefteris Angelis, and Dimitrios Toloudis. 2017. Assessment of vulnerability severity using text mining. In *Proceedings of the 21st Pan-Hellenic Conference on Informatics*. 1–6.
[179] Georgios Spanos, Angeliki Sioziou, and Lefteris Angelis. 2013. WIVSS: A new methodology for scoring information systems vulnerabilities. In *Proceedings of the 17th Pan-Hellenic Conference on Informatics*. 83–90.
[180] Jonathan Spring, Eric Hatleback, Allen Householder, Art Manion, and Deana Shick. 2021. Time to change the CVSS? *IEEE Secur. Priv.* 19, 2 (2021), 74–78.
[181] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolut. Computat.* 10, 2 (2002), 99–127.
[182] Octavian Suciu, Connor Nelson, Zhuoer Lyu, Tiffany Bao, and Tudor Dumitras. 2021. Expected exploitability: Predicting the development of functional vulnerability exploits. *arXiv preprint arXiv:2102.07869* (2021).
[183] Jiamou Sun, Zhenchang Xing, Hao Guo, Deheng Ye, Xiaohong Li, Xiwei Xu, and Liming Zhu. 2021. Generating informative CVE description from ExploitDB posts by extractive summarization. *arXiv preprint arXiv:2101.01431* (2021).
[184] Nan Sun, Jun Zhang, Paul Rimba, Shang Gao, Leo Yu Zhang, and Yang Xiang. 2018. Data-driven cybersecurity incident prediction: A survey. *IEEE Commun. Surv. Tutor.* 21, 2 (2018), 1744–1772.
[185] Nazgol Tavabi, Palash Goyal, Mohammed Almukaynizi, Paulo Shakarian, and Kristina Lerman. 2018. DarkEmbed: Exploit prediction with neural language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[186] Dimitrios Toloudis, Georgios Spanos, and Lefteris Angelis. 2016. Associating the severity of vulnerabilities with their description. In *Proceedings of the International Conference on Advanced Information Systems Engineering*. Springer, 231–242.
[187] Shubham Tripathi, Gustavo Grieco, and Sanjay Rawat. 2017. Exniffer: Learning to prioritize crashes by assessing the exploitability from memory dump. In *Proceedings of the 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 239–248.
[188] Jesper E. Van Engelen and Holger H. Hoos. 2020. A survey on semi-supervised learning. *Mach. Learn.* 109, 2 (2020), 373–440.
[189] Mounika Vanamala, Xiaohong Yuan, and Kaushik Roy. 2020. Topic modeling and classification of common vulnerabilities and exposures database. In *Proceedings of the International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*. IEEE, 1–5.
[190] Hein S. Venter, Jan H. P. Eloff, and Y. L. Li. 2008. Standardising vulnerability categories. *Comput. Secur.* 27, 3–4 (2008), 71–83.

[191] James Walden. 2020. The impact of a major security event on an open source project: The case of OpenSSL. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 409–419.
[192] Zhiyuan Wan, Xin Xia, David Lo, and Gail C. Murphy. 2019. How does machine learning change software development practices? *IEEE Trans. Softw. Eng.* (2019).
[193] Ju An Wang and Minzhe Guo. 2010. Vulnerability categorization using Bayesian networks. In *Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research*. 1–4.
[194] Peichao Wang, Yun Zhou, Baodan Sun, and Weiming Zhang. 2019. Intelligent prediction of vulnerability severity level based on text mining and XGBboost. In *Proceedings of the 11th International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 72–77.
[195] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM Comput. Surv.* 53, 3 (2020), 1–34.
[196] Emil Wåreus and Martin Hell. 2020. Automated CPE labeling of CVE summaries with machine learning. In *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–22.
[197] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. 2020. Evaluating explanation methods for deep learning in security. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 158–174.
[198] Sachini Weerawardhana, Subhojeet Mukherjee, Indrajit Ray, and Adele Howe. 2014. Automated extraction of vulnerability information for home computer security. In *Proceedings of the International Symposium on Foundations and Practice of Security*. Springer, 356–366.
[199] Tao Wen, Yuqing Zhang, Ying Dong, and Gang Yang. 2015. A novel automatic severity vulnerability assessment framework. *J. Commun.* 10, 5 (2015), 320–329.
[200] Mark A. Williams, Roberto Camacho Barranco, Sheikh Motahar Naim, Sumi Dey, M. Shahriar Hossain, and Monika Akbar. 2020. A vulnerability analysis and prediction framework. *Comput. Secur.* 92 (2020), 101751.
[201] Mark A. Williams, Sumi Dey, Roberto Camacho Barranco, Sheikh Motahar Naim, M. Shahriar Hossain, and Monika Akbar. 2018. Analyzing evolving trends of vulnerabilities in national vulnerability database. In *Proceedings of the IEEE International Conference on Big Data (Big Data)*. IEEE, 3011–3020.
[202] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemomet. Intell. Laborat. Syst.* 2, 1–3 (1987), 37–52.
[203] Xiaoxue Wu, Wei Zheng, Xin Xia, and David Lo. 2021. Data quality matters: A case study on data label correctness for security bug report prediction. *IEEE Trans. Softw. Eng.* (2021).
[204] Chaowei Xiao, Armin Sarabi, Yang Liu, Bo Li, Mingyan Liu, and Tudor Dumitras. 2018. From patching delays to infection symptoms: Using risk profiles for an early discovery of vulnerabilities exploited in the wild. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security'18)*. 903–918.
[205] Hongbo Xiao, Zhenchang Xing, Xiaohong Li, and Hao Guo. 2019. Embedding and predicting software security entity relationships: A knowledge graph based approach. In *Proceedings of the International Conference on Neural Information Processing*. Springer, 50–63.
[206] Yasuhiro Yamamoto, Daisuke Miyamoto, and Masaya Nakayama. 2015. Text-mining approach for estimating vulnerability score. In *Proceedings of the 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. IEEE, 67–73.
[207] Guanhua Yan, Junchen Lu, Zhan Shu, and Yunus Kucuk. 2017. ExploitMeter: Combining fuzzing with machine learning for automated evaluation of software exploitability. In *Proceedings of the IEEE Symposium on Privacy-Aware Computing (PAC)*. IEEE, 164–175.
[208] Jiao Yin, MingJian Tang, Jinli Cao, and Hua Wang. 2020. Apply transfer learning to cybersecurity: Predicting exploitability of vulnerabilities by description. *Knowl.-based Syst.* 210 (2020), 106529.
[209] Sofonias Yitagesu, Xiaowang Zhang, Zhiyong Feng, Xiaohong Li, and Zhenchang Xing. 2021. Automatic part-of-speech tagging for security vulnerability descriptions. In *Proceedings of the IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 29–40.
[210] Awad A. Younis and Yashwant K. Malaiya. 2014. Using software structure to predict vulnerability exploitation potential. In *Proceedings of the IEEE 8th International Conference on Software Security and Reliability*. IEEE, 13–18.
[211] Peng Zeng, Guanjun Lin, Lei Pan, Yonghang Tai, and Jun Zhang. 2020. Software vulnerability analysis and discovery using deep learning techniques: A survey. *IEEE Access* (2020).
[212] Hongyu Zhang, Liang Gong, and Steve Versteeg. 2013. Predicting bug-fixing time: An empirical study of commercial software projects. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)*. IEEE, 1042–1051.
[213] Li Zhang and Vrizlynn L. L. Thing. 2018. Assisting vulnerability detection by prioritizing crashes with incremental learning. In *Proceedings of the TENCON IEEE Region 10 Conference*. IEEE, 2080–2085.

[214] Xiong Zhang, Haoran Xie, Hao Yang, Hongkai Shao, and Minghao Zhu. 2020. A general framework to understand vulnerabilities in information systems. *IEEE Access* 8 (2020), 121858–121873.
[215] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *arXiv preprint arXiv:1509.01626* (2015).
[216] Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. 2020. A survey on neural network interpretability. *arXiv preprint arXiv:2012.14261* (2020).
[217] Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE Trans. Knowl. Data Eng.* (2021).
[218] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE International Conference on Computer Vision*. 19–27.
[219] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schroter, and Cathrin Weiss. 2010. What makes a good bug report? *IEEE Trans. Softw. Eng.* 36, 5 (2010), 618–643.
[220] Deqing Zou, Sujuan Wang, Shouhuai Xu, Zhen Li, and Hai Jin. 2019. μVulDeePecker: A deep learning-based system for multiclass vulnerability detection. *IEEE Trans. Depend. Secure Comput.* (2019).
[221] Deqing Zou, Yawei Zhu, Shouhuai Xu, Zhen Li, Hai Jin, and Hengkai Ye. 2021. Interpreting deep learning-based vulnerability detector predictions based on heuristic searching. *ACM Trans. Softw. Eng. Methodol.* 30, 2 (2021), 1–31.
[222] Serkan Özkan. [n. d.]. CVE Details. Retrieved from https://www.cvedetails.com.

Received 25 July 2021; revised 13 February 2022; accepted 30 March 2022