Array 15 (2022) 100209

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/locate/array

Towards automatic discovery and assessment of vulnerability severity in cyber–physical systems

Yuning Jiang, Yacine Atif [*]

University of Skövde, Sweden

ARTICLE INFO

Keywords:
Cybersecurity
Text-mining
Cyber–physical system
Vulnerability analysis
CVSS

ABSTRACT

Despite their wide proliferation, complex cyber–physical systems (CPSs) are subject to cybersecurity vulnerabilities and potential attacks. Vulnerability assessment for such complex systems are challenging, partly due to the discrepancy among mechanisms used to evaluate their cyber-security weakness levels. Several sources do report these weaknesses like the National Vulnerability Database (NVD), as well as manufacturer websites besides other security scanning advisories such as Cyber Emergency Response Team (CERT) and Shodan databases. However, these multiple sources are found to face inconsistency issues, especially in terms of vulnerability severity scores. We advocate an artificial intelligence based approach to streamline the computation of vulnerability severity magnitudes. This approach decreases the error rate induced by manual calculation processes, that are traditionally used in cybersecurity analysis. Popular repositories such as NVD and SecurityFocus are employed to validate the proposed approach, assisted with a query method to retrieve vulnerability instances. In doing so, we report discovered correlations among reported vulnerability scores to infer consistent magnitude values of vulnerability instances. The method is applied to a case study featuring a CPS application to illustrate the automation of the proposed vulnerability scoring mechanism, used to mitigate cybersecurity weaknesses.

1. Introduction

Modern breakthroughs in information and communication technology facilitate the integration of digital and physical environments to improve the degree of automation in industrial processes enabled by cyber–physical systems (CPS). Nonetheless, CPS components are subject to vulnerabilities across the multitude of firmware versions [1]. Unwanted vulnerability occurrences are expected to be discovered. Meanwhile, their magnitude will be graded to determine a mechanism for patching prioritization. This analysis supports cybersecurity operators to anticipate cyber attacks from emerging threats and to prevent intrusion opportunities [2].

New measurements make it possible to quantify cybersecurity issues to support vulnerability-mitigation decisions. These measurements are captured from a range of cybersecurity repositories available online. The Common Vulnerabilities and Exposures (CVE) [3] repository is a prime database cumulating vulnerability reports that are further augmented with the Common Vulnerability Scoring System (CVSS) [4] scores. Other analytical measurements are provided by the National Vulnerability Database (NVD) [5]. However, vulnerability-mitigation decisions that rely on CVE or NVD records as primary data sources, can be biased and discriminating other sources of data [6,7]. For instance, BugTraq from SecurityFocus [8] contains vulnerabilities that are yet to be reported in CVE. Thus, inferred decisions based on cybersecurity measurements need to include a wide range of cybersecurity data repositories. A comprehensive knowledge base that combines multiple sources through some artificial intelligence (AI)-based rules is shown in this research to provide grounds for a required decision-support level.

Enterprises are increasingly confronted with cybersecurity issues resulting from sporadic vulnerabilities, with reported data supporting decision-making criteria. The enormous quantity of system data and reported vulnerabilities increases the workload of security analysts, which is both time-consuming and error-prone when performed manually. This data-driven evolution streamlines previous risk analysis frameworks while still taking into consideration human-expert judgements. Some preliminary works towards combining emerging cybersecurity metrics led to the standard mechanism CVSS. CVSS is widely adopted to assess vulnerability-severities across enterprises and academic research [9,10] [11]. However, CVSS exhibits some challenges when used in practice [12,13]. CVSS-scores are essentially influenced by individual experts, who may spend some time to rank the severity of a vulnerability since disclosed in CVE. This incurred time delay

* Corresponding author.
E-mail addresses: yuning.jiang@his.se (Y. Jiang), yacine.atif@his.se (Y. Atif).

https://doi.org/10.1016/j.array.2022.100209
Received 25 January 2022; Received in revised form 11 June 2022; Accepted 11 June 2022
Available online 18 June 2022

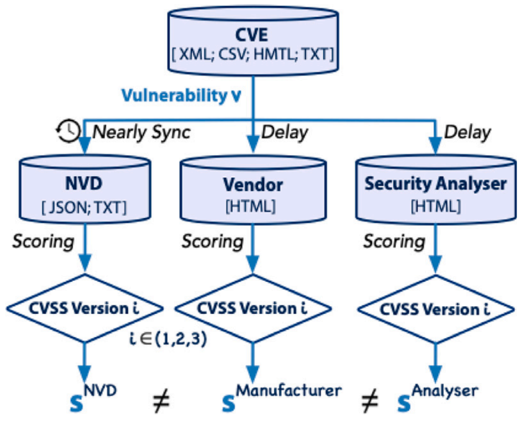


Fig. 1. Potential time delay of scoring and inconsistent scores.

in evaluating vulnerabilities increases the chances of threats to materialize into actual cyberattacks [14]. Automation of vulnerability scoring is therefore anticipated to narrow the gap for zero-day attacks. To design such an autonomous scoring system, several deficiencies must be examined, such as how to infer important measurements used to control vulnerability metrics at an appropriate scale for reported vulnerabilities. In addition, differences between existing CVSS versions produce incompatible metric measurements. Previous study did not adequately address these difficulties. Diverse businesses utilize distinct CVSS versions to evaluate instances of vulnerability [12], resulting in conflicting outcomes. For example, NVD uses CVSS version 3 scores to rate vulnerability instances reported only from 2015 onwards. These challenges of applying CVSS scores to support vulnerability analysis and management are illustrated in Fig. 1. Considering a random vulnerability instance v, NVD, the corresponding manufacturer, and a third-party analyser provide their severity scores as S^{NVD}, $S^{manufacturer}$ and $S^{Analyser}$ which can be inconsistent. Despite CVSS popularity [12, 13], inconsistency among reported scores for the same vulnerability instance does occur. Particularly, when considering other CVSS temporal and environmental metrics, whereby vulnerability properties evolve across time and deployment environments. Hence, additional sources of relevant data, including manufacturer-provided data, online reviews from relevant security sources and forums, are expected to consolidate further existing CVSS scores [10].

Data inconsistence also increases the difficulty of vulnerability retrieval. For example, using *MTU* as a single keyword in the NVD search engine returns vulnerabilities that are relevant to two diverse categories of devices, namely *Maximum Transfer Unit* (e.g., vulnerability instance *CVE-2005-0065*) and *Master Terminal Unit* (e.g., vulnerability instance *CVE-2015-0990*). Therefore, to retrieve Master Terminal Unit vulnerabilities only, we need to refine further the query with keywords like *SCADA-server* or vendor-specific modules. Meanwhile, vendor names in Common Platform Enumeration (CPE) [15] metadata may appear with variations. For example, the vendor *Schneider Electric SE* has variant forms like *'schneider-electric'*, *'chneider-electric'*, and *'schneider-electic'* in CPE database.

We propose a vulnerability scoring system to quantify the severity of a reported incidence of vulnerability. The computed scores facilitate situational awareness through quantitative indicators that are transformed into actionable intelligence. This method automates vulnerability investigation while addressing compatibility concerns between multiple CVSS versions. Standard CVSS criteria are used as a scoring basis to evaluate the exploitability of vulnerabilities and the consequences of maliciously exploits. In pursuit of these aims, we correlate vulnerability scores published by several online cybersecurity data sources, including NVD, vendor websites, and technical reports from third party reviewers (e.g., Cyber Emergency Response Team (CERT) [16] and Microsoft Security Response Center (MSRC) [17], to consolidate severity scores of vulnerability instances. Accordingly, we produce ground facts for our Machine Learning (ML)-based vulnerability-severity computation algorithm. These instances are then used to train our ML model, which we evaluate using vulnerabilities reported in vulnerability repositories such as NVD and SecurityFocus. In addition to NVD and SecurityFocus, our suggested approaches can incorporate additional other data sources such as CERT. We also propose a new query logic to identify relevant vulnerability instances, while excluding possible false positives based on other keywords. The evaluation study of CPS vulnerability and related factors shows an enhanced level of automation in cybersecurity assessments [14].

The main contributions of this paper are outlined as follows:

- A novel machine-learning based structure for vulnerability assessment that infers CVSS severity scores of reported vulnerability instances. This proposed technique addresses compatibility issues of CVSS scores using a majority voting system, as part of the proposed machine-learning model. The approach can be customized to accommodate a preferred CVSS version, in order to allow a common computational semantic that improves consistency in vulnerability assessment.
- A query generation method that takes system configuration information as input and exports the best matching query tags in the format similar to *CPE* metadata.
- A CPS vulnerability analysis case study that validates the proposed machine-learning based vulnerability-assessment approach.

The rest of this paper is organized as follows: In Section 2, we provide some background and formally state the problem addressed in this paper, followed by Section 3 which discusses vulnerability data sources, standard vulnerability-severity metrics and related vulnerability-assessment processes used in the CVSS mechanism. In Section 4, we reveal our vulnerability assessment prototype, which correlates existing CVSS scores against other security-alert indicators, as well as reconciles different CVSS versions using some text-mining techniques on a corpus of vulnerability reports. In Section 5, we evaluate our vulnerability-discovery and assessment methodology in CPS contexts using mainly NVD and *Shodan* [18] through some analysis. In Section 6, we provide some concluding remarks and discuss some future research directions.

2. Related works

Correlation studies between multiple cybersecurity data sources can combine various perspectives from different stakeholders, to connect multifaceted analysis into broader statistical associations. CVE, NVD, CERT and SecurityFocus are widely used vulnerability-analytics databases for uniquely identified vulnerability recordings. These databases are further correlated to data sources like ExploitDB [19]. An example of this is the study carried out by Allodi and Massacci [20] correlates. They correlate NVD to ExploitDB, Symantec AttackSignature and ThreatExplorer. By doing so, they enhance CVSS scoring practice by computing the temporal attributes based on the existence of public proof-of-concept (PoC) exploits. Geer and Roytman [21] also correlate NVD database to ExploitDB and Metasploit [22] to support penetration testers. Fang et al. employ SecurityFocus and NVD to predict the exploitability and exploitation of vulnerabilities, while taking PoCs extracted from ExploitDB as ground truth [13]. Rodriguez et al. [23] compare the original release dates of multiple data sources, including NVD, SecurityFocus, ExploitDB and three vendors Cisco, Wireshark and Microsoft. They observe that the vulnerability instances published in NVD are 1–7 days delayed compared to other data sources.

A variety of approaches apply text-mining techniques in industrial blogs like Twitter [24] and security papers. This is exemplified in the work undertaken by Zhu and Dumitras who apply NLP to extract malware detection features from research papers automatically [25]. Chen et al. [24], Bullough et al. [26] and Sabottke et al. [27] extract vulnerability-related data by crawling Twitter and extracting tweets that contain *CVE* as a keyword. These works highlight that statistical interpretations of CVE and NVD datasets need to be combined with other live security-related data sources, such as Twitter, the dark web and product vendors across deployed infrastructures, to raise indicators' reliability and precision. However, these works contribute to the wider field of software vulnerability analysis. Correlation studies considering different terminology used in cybersecurity addressing specifically CPS domains are limited. In our method, we extract relevant entities of vulnerable components and vendor information in CVE vulnerability reports, which we map against the Common Platform Enumeration (CPE) [15] as well as vendor websites, in order to generate a dictionary for CPS components and vendors.

The retrieved information from cybersecurity data sources supports further pattern recognition and trend analysis. Using AI techniques, large amounts of such open-source vulnerability data [11] can be analyzed. More specifically, machine-learning techniques like text-mining are applied to automatically classify disclosed vulnerabilities and guide predictive analytics of the security gap. The effectiveness of AI techniques has been illustrated in a study by Bozorgi et al. [28]. They employ SVM (referring to Support Vector Machine) to predict time-to-exploit indicators of reported vulnerabilities on the Open Source Vulnerability Database (OSVDB) and CVE. Targeting CVSS base score generation, Gawron et al. [9] apply Neural Networks and Naive Bayes algorithms, while Yamamoto et al. [29] deploy supervised LDA (referring to Latent Dirichlet Allocation) for CVSS metrics classification. Nevertheless, using correlated cybersecurity data sources also raises potential inconsistencies, such as the disparity between scores for the same vulnerability instances [10]. One drawback of previous AI-based CVSS computing approaches is that they directly adopt the vulnerability reports and CVSS scores from NVD as training grounds, which may induce a bias in their model. Instead, we correlate vulnerability instances in NVD to corresponding vendor reports and third-party cybersecurity analyzers such as CERT reports to consolidate data sources. Then we integrate relevant information into a unified structure as our training grounds. To be more specific, we use the vulnerability descriptions in NVD as training input. We then apply majority voting on inconsistent scores before using them as training grounds. In doing so, our approach streamlines the computation of vulnerability severity to address such inconsistencies upstream, in order to optimize security investments and to shorten the potential risk window. Based on our previous work [30], we extended CVSS base-score computation experiments to include more vulnerability data sources. This additional experimental study illustrates the capacity of our mechanism to extend to new data sources such as SecurityFocus. By including *Shodan* database, we also extract more vulnerability instances for our CPS cybersecurity case study.

3. Background

In this section, we introduce the data sources, the metrics used to calculate the vulnerability severity, and the severity score computing process.

3.1. Vulnerability data sources

MITRE Corporation publishes the CVE industry-standard to assign an identifier to each discovered vulnerability. In addition, it maintains a publicly accessible database of all identifiers through CVE Numbering Authorities (CNA) [31]. A typical CVE entry includes the following fields: a unique identifier, a brief description of the reported vulnerability, and any pertinent references about the vulnerability. The unique CVE identifier, or CVE ID, is the key that differentiates one security vulnerability from another. In doing so, CVE IDs provide a reliable way of communicating across these different databases to get more information about the reported security flaws.

NVD builds upon the information included in CVE entries to provide an enhanced information for each entry, such as severity scores (calculated based on CVSS standard) and impact ratings. NVD converts the unstructured CVE data into structured JSON (or JavaScript Object Notation) or XML (or Extensible Markup Language) formats [6]. As part of its enhanced information, NVD also provides advanced searching features such as by OS, by vendor name, by product name, by version number, and by vulnerability type and severity. Among these extra features, affected product names and versions have matching string entries in CPE entries. Vulnerability category features are provided in Common Weakness Enumeration (CWE) [32] repository, which abstracts the observed faults and flaws into common groups of vulnerabilities with additional information about expected effects, behaviors, and further implementation details. The vulnerability severity score is calculated following the CVSS version 3 and version 2 standards.

SecurityFocus is a widely used vulnerability database and also features a security news portal [13]. Even though this database is shut down in January 2021, still its historical reports are applicable to validate our experimental analysis. Besides vulnerability descriptions, SecurityFocus also addresses whether a vulnerability has a PoC exploit. Note that SecurityFocus is not dependent upon CVE data sources [23]. Actually, a BugTraq vulnerability report may refer to several CVE vulnerability instances. A statistic analysis by Fang et al. highlight that although the amount of vulnerabilities reported in SecurityFocus is less than the number of vulnerabilities found in NVD, the fraction of exploited vulnerabilities in SecurityFocus (37.008%) is much higher than the proportion in NVD (6.676%) [13]. They also observe that the vulnerability reports in SecurityFocus contain higher coverage and more reference significance in predictive cybersecurity analysis, leading to their experiment results where SecurityFocus performs well than NVD under an actual environment.

Industrial Control System CERT (ICS-CERT) [33] is a branch in US-CERT that focuses on control systems' security. The ICS-CERT advisories add further analysis on reported vulnerabilities in CVE, particularly on risk evaluation, affected products, and mitigations such as workarounds or official patches.

In the following example, we present the differences between the aforementioned vulnerability data sources, especially between NVD, SecurityFocus and ICS-CERT. The SecurityFocus historical reports were downloaded in December 2020 before shutting down. The vulnerability report under *BugTraq ID 108727* refers to three CVE reports, namely *CVE-2019-6580*, *CVE-2019-6581*, and *CVE-2019-6582*, respectively. NVD assigns CVSS V3 base scores 9.8, 8.8, and 7.7 to these three disclosed vulnerabilities. Yet, *ICS-CERT* and the vendor *Siemens* [34] assign the same CVSS V3 base scores to *CVE-2019-6581* and *CVE-2019-6582*, but a different score 8.8 to *CVE-2019-6580*. The inconsistency of *CVE-2019-6580* scores is due to different views on the metric of whether privileges are required to exploit this vulnerability. Here we compare the discussion section in SecurityFocus and the description section of one vulnerability instance *CVE-2019-6580* in NVD or CVE. We observe that the summary given by SecurityFocus highlights vulnerability types and potential threats targeting the vulnerability, while NVD emphasizes the affected products and the impact of the vulnerability.

- SecurityFocus discussion: "*Siemens Siveillance VMS is prone to multiple authorization-bypass vulnerabilities. Attackers can exploit these issues to bypass certain security restrictions and perform certain unauthorized actions. This may aid in further attacks. These issues have been fixed in Siveillance VMS 2017 R2 v11.2a, 2018 R1 v12.1a, 2018 R2 v12.2a, 2018 R3 v12.3a, and 2019 R1 v13.1a.*"
- NVD description (the same as CVE description): "*A vulnerability has been identified in Siveillance VMS 2017 R2 (All versions < V11.2a), Siveillance VMS 2018 R1 (All versions < V12.1a), Siveillance VMS 2018 R2 (All versions < V12.2a), Siveillance VMS 2018 R3 (All versions < V12.3a), Siveillance VMS 2019 R1 (All versions*

< V13.1a). An attacker with network access to port 80/TCP could change device properties without authorization. No user interaction is required to exploit this security vulnerability. Successful exploitation compromises confidentiality, integrity and availability of the targeted system. At the time of advisory publication no public exploitation of this security vulnerability was known."

Finally, *Shodan* is a data source mainly targeting CPS or IoT (referring to Internet of Things) security, including SCADA (referring to Supervisory Control and Data Acquisition) [35]. CPS and IoT systems include devices like webcams, routers, and servers. Relevant information like ports and vulnerabilities of these devices can be fetched through *Shodan* website or *Shodan API* (referring to Application Programming Interface). Interestingly, these are currently internet-connected devices, sending (public) live data from different locations across the World. Unlike NVD, where vulnerability reports are published, *Shodan* crawls IP addresses, made available on device respective websites and APIs. Returned data from *Shodan* can be cross-referenced with NVD for vulnerability analysis [36].

3.2. Vulnerability severity metrics

The Forum for Incident Response and Security Teams or FIRST initiated the development of the CVSS calculator while reporting cybersecurity incidents. The current CVSS Version 3 follows a sequence of three versions of the CVSS index calculator. Vulnerabilities are first assigned a unique identifier and their severity is rated by combining CVSS property metrics. CVSS score involves three groups of properties. The *Base* group describe static properties that are not subject to temporal or deployment environments. In contrast, the *Temporal* and *Environmental* groups of properties are respectively describing score variations across time or deployment contexts. However, we emphasize base score properties in this research and consider the latest version of CVSS base properties, that is Version 3 or V3. These base properties are further grouped under three classifications, namely exploitability $P^{v}_{Exploit}$, scope P^{v}_{Scope}, and impact P^{v}_{Impact} properties, which we discuss next.

3.2.1. CVSS exploitability property:

This property quantifies the likelihood as well as the effort and intricacy to be invested for exploiting a component that would be exposed to a given vulnerability. Hence, this property combines the following metrics: *AttackVector (AV)*, *AttackComplexity (AC)*, *PrivilegesRequired (PR)* and *UserInteraction (UI)*. The Attack Vector metric measures the likelihood for an attack scenario targeting the component to occur through this vulnerability. The effort that needs to be invested may vary across these scenarios, quantified as part of the Attack Complexity metric. Along the path of an attack scenario, some credentials or privileges may be required. The level of these requirements is measured by the Privileges Required metric, for an agent with authority to be granted access to the component. And, the level of participation that is expected in order to exploit and compromised the vulnerable component is measured by the User Interaction metric.

3.2.2. CVSS scope property:

The propagation of a vulnerability from a targeted component to eventually grant access to others within an asset configuration is measured by *ScopeChange* (or S). The Scope metric is used to measure the extent to which other components than the vulnerable one, can be accessed.

3.2.3. CVSS impact property:

This property groups metrics along the (CIA) triad, to quantify the magnitude of potential losses of *Confidentiality* (C), *Integrity* (I) and/or *Availability* (A). Measurements along these metrics categorize the severity levels impacted by the vulnerability as none- (N), low- (L) or high- (H).

3.3. Severity score computing process

CVSS combines the above properties to infer vulnerability level rating its severity based on a rule-based algorithm, which is further depicted in Eq. (1) that use measurements of *Exploitability, Scope and Impact* property metrics to generate the *Base* score of a vulnerability. Considering a component c of a CPS asset C, exploitability and impact property measurements are extracted as illustrated in Eqs. (2) and (3), separately. To infer a score of a vulnerability v for a component c, a function measures the corresponding base properties: $f_{Exploit}, f_{Scope}$ and f_{Impact}, as illustrated by the f_{Base} function illustrated by Eq. (1).

$$f_{Base} = (P_{Exploit}, P_{Scope}, P_{Impact}) \tag{1}$$

In Eq. (1), vulnerability measurements vector v_i are collected for component c_i by Eqs. (2) and (3). $f : A \rightarrow B$ means a functional association.

$$f_{Exploit} : C \rightarrow P_{Exploit} \tag{2}$$

$$f_{Impact} : C \rightarrow P_{Impact} \tag{3}$$

This can be illustrated briefly by the vulnerability instance *CVE-2021-37172* for example. This vulnerability instance affects *Siemens* PLC (or Programmable Logic-Controller) product running SIMATIC S7-1200 CPU family with firmware version number 4.5.0 (or the vulnerable component), by allowing a threat agent to bypass authentication and download arbitrary programs to this PLC (or the vulnerable CPS asset). This vulnerability has a CVSS version 3 base score of 7.5, which is further composed of an exploitability score of 3.9 as well as an impact score of 3.6.

4. Discovering vulnerability severity

In this section, we present a ML-based method to discover CVSS scores of reported vulnerability instances with no assigned score. Our proposed approach automatically generates severity scores for vulnerability instances, which decreases the potential of manual errors and requires less effort from human experts. We start with a brief overview of the system structure, followed by a detailed introduction of each element of the vulnerability-severity computing system.

4.1. System overview

As illustrated in Fig. 2, we first collect vulnerability data from open-source cybersecurity repositories. Simultaneously, we adopt majority voting techniques [37] to deal with inconsistent scores retrieved from different CVSS scored reports across multiple repository sources. We employ the reconciled scores as the training ground for our proposed ML models, together with vulnerability reports. Then we streamline score prediction by using a ML pipeline that classifies these instances considering various CVSS-metric labels. Meanwhile, CVSS metrics from different CVSS versions are stored in a knowledge base. Thus, the corresponding metric-set is retrieved through the user's query. The same goes for measurements and severity scales. In doing so, one can select any CVSS version to compute the corresponding score and vector for vulnerability instances. Our proposed vulnerability severity computing system contains a series of steps chained together through ML computational cycles. Each integrated ML cycle involves mainly three steps. Step 1 refers to obtaining the data. Step 2 performs data pre-processing to prepare the data for training/testing processes on a machine-learning algorithm. And finally, in Step 3 we output a predicted severity score. By preprocessed data, we mean both the training/testing instances and the classification measurements. Note that training and testing processes are not differentiated in Fig. 2 to facilitate readability.

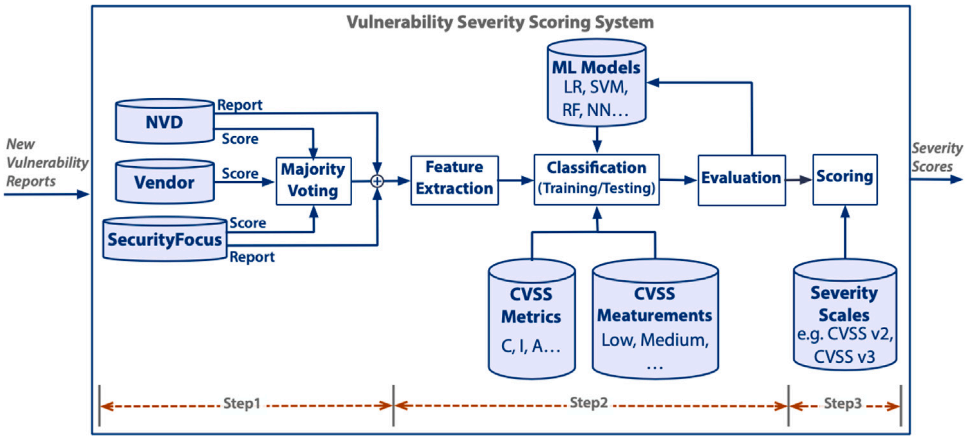

Vulnerability Severity Scoring System
New Vulnerability Reports
NVD
Vendor
SecurityFocus
Report
Score
Majority Voting
Feature Extraction
ML Models LR, SVM, RF, NN…
Classification (Training/Testing)
Evaluation
Scoring
Severity Scores
CVSS Metrics C, I, A…
CVSS Meaturements Low, Medium, …
Severity Scales e.g. CVSS v2, CVSS v3
Step1
Step2
Step3

Fig. 2. High-level structure of vulnerability severity computing system.

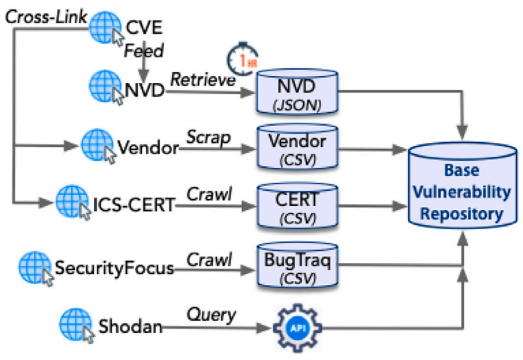


Fig. 3. Vulnerability severity data collection.

4.2. Data collection

We employ multiple ways to collect vulnerability data from online public repositories, as depicted in Fig. 3. More specifically, NVD data feeds are directly downloaded and stored in a local database in JSON format. The JSON format is an open standard file format used for interchanging data, consisting of human-readable text (i.e., not binary) attributes' value pairs. JSON objects can be nested inside other JSON objects, while each nested object has a unique access path across the tree-like structure. To ensure that the local files and online NVD data feeds are synchronized, we set up a scheduler to perform hourly data retrieval and update the data through an existing Python library *APScheduler* (referring to Advanced Python Scheduler). We choose the hourly schedule to mirror NVD data considering that the "*recent*" and "*modified*" feeds in NVD are updated every two hours, while the rest are updated nightly. Besides NVD, we apply web crawling and web scraping techniques [38] to grasp vulnerability information published in SecurityFocus, ICS CERT and vendor websites. Web crawling refers to the process of browsing and indexing contents from web pages. Examples of relevant built-in Python functions include *urllib.request* that downloads *html* pages and *urllib.error* that handle exceptions. Web scraping means locating and collecting certain information and are supported by tools like HTML parser *Beautiful Soup* [39]. After fetching, parsing and extracting targeted information, we store the retrieved data in a proper format tailored to the data usage. For instance, we store the data extracted from SecurityFocus in local files with fields like Bugtraq-ID, CVE-ID, title, publish date, affected product, etc., in CSV (or comma-separated values) format. Finally, we query *Shodan* API to get CPS relevant vulnerabilities.

4.3. Majority voting for inconsistent scores

Relying upon NVD scores alone as the model training ground can bring bias in vulnerability assessment [6,7]. This is because a small percentage of score records in NVD is assumed to have errors due to the manual scoring process [12]. Besides statistical vulnerability patterns mined from CVE reports, other data sources like vendors and third-party security analyzers (e.g., ICS CERT and MSRC) provide different perspectives for vulnerability scoring. We set up a majority voting [37] module using Python whereby the score that the majority of data sources ($[V_1, \dots, V_d, \dots, V_D]$ where $0 \langle d \leq D, D \rangle 2$) in the pipeline agree on is delivered as *true score* or ground truth score. In the cases where only two score sources are found, or $[V_1, V_2]$, and these two scores are inconsistent, we take the average of these scores.

We give an example using the vulnerability instance *CVE-2018-7791* for which a CVSS V3 base-score of 9.8 is assigned by NVD and vendor *Schneider Electric* with the vector *AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H*. Nevertheless, ICS CERT assigns this vulnerability with a score of 7.7 with the vector *AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:L*. Similarly, the inconsistence comes as a result of different measurement for attack complexity. Using our majority voting approach, we choose a final score of 9.8 as the true score. Another example of score inconsistencies is the vulnerability instance *CVE-2014-0754* which is assigned a CVSS V2 base-score 10.0 by NVD, VulDB, as well as ICS CERT with a vector *AV:N/AC:L/Au:N/C:C/I:C/A:C*. Note that the first three CVSS V2 metrics differ from CVSS V3 metrics. *AV* means Access Vector, which is equivalent to the Attack Vector in CVSS V3. *AC* refers to Access Complexity, which is equivalent to the Attack Complexity in CVSS V3. *Au* denotes Authentication which measures the number of times an attacker needs to authenticate oneself to the targeted component in order to exploit the vulnerability. Yet, a different score 9.3 is assigned by the vendor *Schneider Electric* with a vector *AV:N/AC:M/Au:N/C:C/I:C/A:C*. The inconsistency occurs due to different measurements for *Access Complexity* of this instance, whereby *Schneider Electric* assigns medium complexity, while the other three parties assign low complexity. Using our majority voting approach, we choose a final score of 10.0 as a true score.

4.4. Vulnerability severity computing

Cybersecurity data is classified using a pipeline of ML algorithms, in order to fill CVSS score gaps. Using text-mining approaches [11], retrieved vulnerability reports from existing cybersecurity repositories are contrasted against vulnerability descriptors. Subsequently, new vulnerability reports are classified along CVSS-metric property groups, using a ML algorithm, which is trained from a set of historical instances of reported data V. Considering N vulnerability instances from this

Algorithm 1 Vulnerability CVSS Base-Score Computing.

```
1: procedure SeverityComputing($\mathcal{ML}, m, M, V, V'$)
     ▷ $\mathcal{ML}$ is a machine learning model $f()$.
     ▷ $f_{Base}()$ is the CVSS calculator, as shown in Equation (1)
     ▷ $[V_1, \dots, V_d, \dots, V_D]$ $(0 < d \le D, D > 2)$ is a list of data sources, each of which has $N$ vulnerability instances. Each vulnerability instance $v_i$ $(0 < i \le N)$ is assigned a list of severity scores as $[s_{i,1}, \dots, s_{i,d}, \dots, s_{i,D}]$ and a list of CVSS vectors as $[Y_{i,1}, \dots, Y_{i,d}, \dots, Y_{i,D}]$.
     ▷ $V'$ is a set of vulnerability instances $v_p$ $(0 < p \le N')$ that have no severity score or CVSS measurements.
     ▷ $m$ is a set of CVSS metrics $m_j$ $(0 < j \le M)$ where each metric $m_j$ has a set of $K^{m_j}$ classes as maps to a value $Y_i^{(m_j)} \in \{c_{1^{(m_j)}}, \dots, c_{k^{(m_j)}}, \dots, c_{K^{(m_j)}}\}$ $(0 < k^{(m_j)} \le K^{(m_j)})$.
2:   $D = |[V_1, \dots, V_d, \dots, V_D]|, N = |V_d|, N' = |V'|, M = |m|, K^{(m_j)} = |\{c_{1^{(m_j)}}, \dots, c_{k^{(m_j)}}, \dots, c_{K^{(m_j)}}\}|$
3:   For vulnerability instance $v_i$ $(i = 1, \dots, N)$ do
4:     For CVSS metric $m_j$ $(j = 1, \dots, M)$ do
5:       Set $Y_i^{(m_j)} = \arg\max_{K^{(m_j)}} [card(\{c_{1^{(m_j)}}, \dots, c_{k^{(m_j)}}, \dots, c_{K^{(m_j)}}\} \| Y_{i,d}^{(m_j)})] (0 < d \le D)$ as ground truth for CVSS measurement
6:     End For
7:     $Y_i = [Y_i^{(m_1)}, \dots, Y_i^{(m_j)}, \dots, Y_i^{(m_M)}]$ $(j = 1, \dots, M)$
8:     Set $s_i = f_{Base}(Y_i)$ as ground truth for severity score
9:   End For
10:  For $j = 1, \dots, M$ CVSS metric $m_j$ $(j = 1, \dots, M)$ do
11:    Train($\mathcal{ML}$)          ▷ $\mathcal{ML}$ model training and testing for historical dataset
12:    $f^{(m_j)}(v_i) = \arg\max_{k^{(m_j)}} f_{k^{(m_j)}}^{(m_j)}(v_i)$
13:  End For
14:  For vulnerability instance $v_p$ $(p = 1, \dots, N')$ do
15:    For CVSS metric $m_j$ $(j = 1, \dots, M)$ do
16:      $Z_p^{(m_j)} = f^{(m_j)}(v_p)$          ▷ Get the resulting $\mathcal{ML}$ predicted CVSS measurement
17:    End For
18:    $Z_p = [Z_p^{(m_1)}, \dots, Z_p^{(m_j)}, \dots, Z_p^{(M)}]$ $(j = 1, \dots, M)$
19:  End For
20:  The resulting predicted score $z_p = f_{Base}(Z_p)$
21: End procedure
```

data set, (v_i, Y_i) $(0 < i \le N)$ represents a mapping between a vulnerability report v_i and a vector Y_i describing the ground truth employed by the ML algorithm.

CVSS metrics $m = [m_1, \dots, m_j, \dots, m_M]$ $(0 < j \le M)$, determine the M classes $Y_i = [Y_i^{(m_1)}, \dots, Y_i^{(m_j)}, \dots, Y_i^{(m_M)}]$ where each metric class $Y_i^{(m_j)}$ has a set of measurements $Y_i^{(m_j)} \in \{c_{1^{(m_j)}}, \dots, c_{k^{(m_j)}}, \dots, c_{K^{(m_j)}}\}$ $(0 < k^{(m_j)} \le K^{(m_j)})$. For example, CVSS V3 is employed with the set of metrics $m = [AV, AC, PR, UI, S, C, I, A]$ (where $M = 8$), with the corresponding measurements $Y_i = [Y_i^{(AV)}, Y_i^{(AC)}, Y_i^{(PR)}, Y_i^{(UI)}, Y_i^{(S)}, Y_i^{(C)}, Y_i^{(I)}, Y_i^{(A)}]$ such as, $Y_i^{(AV)} \in \{N, A, L, P\}$ (where $K^{(AV)} = 4$), $Y_i^{(AC)} \in \{L, H\}$ (where $K^{(AC)} = 2$), $Y_i^{(PR)} \in \{N, L, H\}$ (where $K^{(PR)} = 3$), $Y_i^{(UI)} \in \{N, R\}$ (where $K^{(UI)} = 2$), $Y_i^{(S)} \in \{U, C\}$ (where $K^{(S)} = 2$), $Y_i^{(C)} \in \{H, L, N\}$ (where $K^{(C)} = 3$), $Y_i^{(I)} \in \{H, L, N\}$ (where $K^{(I)} = 3$), $Y_i^{(A)} \in \{H, L, N\}$ (where $K^{(A)} = 3$). This definition is illustrated with the vulnerability instance we introduced earlier, namely *CVE-2021-37172*, which can be written as *(CVE-2021-37172, [N, L, N, N, U, N, H, N])*.

Algorithm 1 shows the base score computation of vulnerability severity. Considering the vulnerability computing illustration shown in Fig. 2, Lines 3–9 in Algorithm 1 represent the procedure for Step1. Lines 10–13 show the process for Step2. And finally, Lines 14–20 unfold the procedure for Step 3. The classes m_j with $K^{(m_j)}(K^{(m_j)} > 2)$ amount of measurements, such as AV, is simplified into multiple binary classification problems, to differentiate between classes. Assume the employed ML model (e.g. SVM) is $f()$, multi-class categorization is achieved through a "one-against-all" method whereby $f^{(m_j)}(v_i) = \arg\max_{k^{(m_j)}} f_{k^{(m_j)}}^{(m_j)}(v_i)$.

The classification of CVSS measurements into class labels calibrates severity scores from property attributes. A *high* label of *AttackComplexity (AC)* for example, pertains to the attribute value of 0.44, and 0.77 attribute score pertains to *low* label. These numerical values are use in the CVSS calculation process.

4.5. Evaluation metrics

The contrast between severity predictions and originally labeled ones is used for training and testing the classification performance. We employ Accuracy, Balanced Accuracy metrics [40] as well as F1-score [41] to assess this contrast. The performance implication accounts for unbalanced classes, such as *AccessVector(AV)* classes for example, where *Network* category has much larger sample size than *Physical* category, as depicted in Fig. 4. This observation is emphasized in Table 2 and Table 3. *AccessVector(AV)* classification may involve multi-class associations, whereby micro-average is used to compute the mean of value across class associations. Micro-average differs from macro-average in the sense that micro-average aggregates the weighted contributions of all classes, while macro-average take the average contributions of all classes. And therefore, micro-average is preferable for multi-class categorization problems with class imbalance. The same approach is employed for other multi-class occurrences. Binary classifiers like the one employed for *UserInteraction (UI)* uses a confusion matrix to infer the balanced-accuracy and F1-score values of the classification.

4.6. Performance evaluation

We validate our approach following two evaluation experiments, that use respectively data for retrieved reports in existing repositories and data from crawled websites. We also published codes for ML algorithm implementation and vulnerability data correlation in two Github projects [42,43], to enhance further the reproducibility [44,45] of our proposed methods.

4.6.1. Vulnerability databases

156 040 vulnerability records corresponding to 2002 to 2020 range are retrieved from NVD (November 3, 2021 release). The reports that

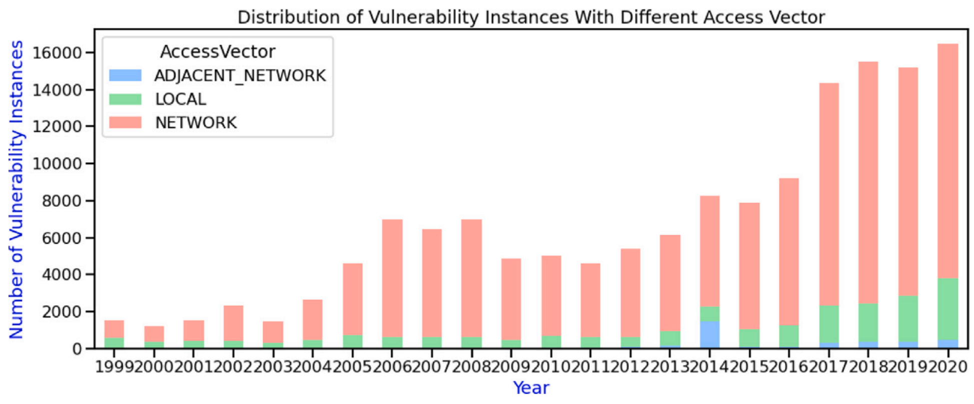


Fig. 4. Imbalance classes of access vector.

Table 1
Evaluation of CVSS categorization.

CVSS-Metric	NVD and Securityfocus text features			NVD text features only		
	Micro F1-score	Balanced accuracy	Accuracy	Micro F1-score	Balanced accuracy	Accuracy
V2 AccessVector(AV)	84.97%	81.05%	95.76%	80.87%	79.53%	95.09%
V2 AccessComplexity(AC)	71.18%	64.01%	83.63%	63.68%	63.64%	84.02%
V2 Authentication(Au)	56.34%	56.21%	95.00%	57.47%	55.34%	93.92%
V2 ConfidentialityImpact(C)	81.03%	80.42%	82.98%	80.66%	79.88%	82.45%
V2 IntegrityImpact(I)	82.40%	82.04%	84.60%	82.34%	81.85%	84.43%
V2 AvailabilityImpact(A)	80.12%	80.09%	81.08%	79.44%	79.19%	80.53%
V3 AttackVector(AV)	75.92%	68.33%	93.68%	75.86%	67.93%	90.36%
V3 AttackComplexity(AC)	81.94%	75.53%	95.58%	78.78%	74.83%	95.31%
V3 PrivilegesRequired(PR)	78.79%	73.25%	90.71%	77.40%	72.50%	85.77%
V3 UserInteraction(UI)	93.45%	93.05%	94.13%	91.41%	91.00%	92.11%
V3 Scope(S)	93.65%	92.64%	97.48%	93.08%	90.66%	96.29%
V3 ConfidentialityImpact(C)	88.36%	87.74%	91.46%	84.37%	82.33%	86.67%
V3 IntegrityImpact(I)	90.58%	90.33%	92.02%	86.91%	85.79%	87.45%
V3 AvailabilityImpact(A)	75.75%	71.55%	93.01%	77.84%	70.41%	89.18%

are marked as *REJECT* are removed from further consideration. A corpus of CVSS V2 reports is set up by excluding reports that are not scored under CVSS V2. 148 803 vulnerability reports are subsequently filtered out, which are then correlated against trusted sources of data like ICS-CERT(asserted by cybersecurity experts). Manufacturer data sources are also used to resolve disparate scores. The proposed ML model uses these scores as ground truths for training purposes. Following the same approach, reports that are not rated under CVSS V3 are taken out to set up 75 265 instances of CVSS V3 corpus data. CVSS V3 scored reports are fewer from 2015 and earlier, with a total of 4 958 reports.

Python package *pipeline* in *Scikit-learn* library has been used to implement the machine-learning pipeline including features extraction and other data processes. Severity scores from different CVSS versions are thus transformed in a streamlined way. Processing NVD vulnerability reports' data starts from tokenisation and subsequent feature extractions using *CountVectorizer* [46] and *TdidfTransforer* [47] utilities. Subsequently, TF–IDF (referring to Term Frequency–Inverse Document Frequency) values are calculated, to generate a TF–IDF matrix from word features. *Train_test_split* procedure is used to randomly divide data records into training (75%) and testing (25%) datasets, following a random distribution.

Machine learning classifiers classify new vulnerability reports within predicted severity patterns. The results obtained from our case studies use *LogisticRegression* (LR) classifier, besides a 5-fold stratified cross-validation applied to the CVSS training dataset to reduce overfitting occurrences. CVSS classifier prediction performances for the testing datasets are illustrated in Table 1. CVSS V3 metric classifications reach an overall higher performance than CVSS V2 counterparts. However, the larger set of metrics offsets the CVSS V3 error rate.

The outcomes assure satisfactory performances when contrasted to closely related CVSS classification researches from Gawron et al. [9] as well as Yamamoto et al. [29]. Gawron et al. apply Naive Bayes and Neural Networks algorithms onto CVE vulnerability reports published before and within 2016 to train CVSS version 3 classifiers. Their training dataset is adjusted to uneven the influence from data imbalance. The performance of the model proposed by Gawron et al. uses only an accuracy metric, that may not adequately capture unbalanced classification instances. Nevertheless, our accuracy is higher on average. For example, the accuracy for Attack Vector classifier is 90.36% when using only NVD vulnerability entries, or 93.68% when using both NVD and SecurityFocus entries. In comparison, Attack Vector classifier based on Neural Network in [9] has an accuracy of 88.9% on testing data and 80.3% on validation data. The other Attack Vector classifier based on Naive Bayes in [9] achieves an accuracy of 90.8% on testing data and 92.3% on validation data. Yamamoto et al. train their CVSS version 2 classifiers on vulnerability instances disclosed in NVD from 1999 till 2014. They employed several ML algorithms, including Naive Bayes, LDA, SLDA (referring to supervised LDA), and Latent Semantic Indexing.

4.6.2. Security news website

In this validation experiment, we crawl vulnerability reports from SecurityFocus and map the reports to the corresponding CVE indexes. This step is done in December 2020, before SecurityFocus's shut down in January 2021. Yet, our proposed methods are still valid in the aspect that utilizing multiple vulnerability data sources enriches the features and may enhance the performance of the classification models. These external descriptive reports are added as text features together with NVD reports for model training. The results are also listed in Table 1. We observe that by adding more text features, the performance of our CVSS scorer improves.

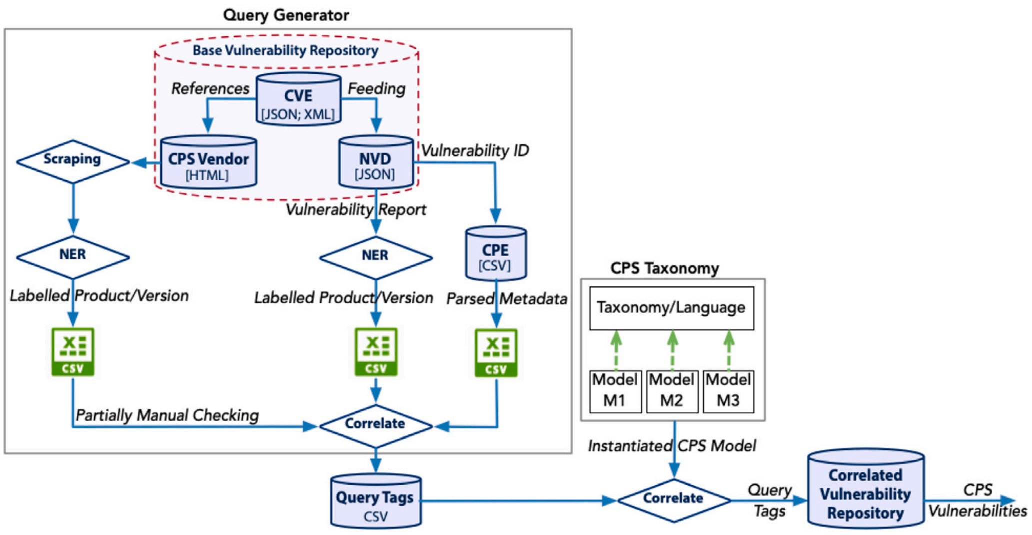


Fig. 5. CPS vulnerability filter.

5. Cyber–physical systems vulnerability assessment

Vulnerabilities in CPS infrastructure are assessed to enumerate and rank their severity to prevent threat-induced anomalies, or intrusion attempts [1,2]. Here we present a vulnerability analysis case study of several prominent CPS components. This case study is composed of three main steps. First, we query and filter CPS relevant vulnerabilities from online cybersecurity data sources. Then, we compute the CVSS V3 base scores and corresponding vectors for retrieved vulnerability instances. Finally, we perform an analysis to explore the statistical patterns of existing CPS vulnerabilities.

5.1. CPS taxonomy

Jang-Jaccard, Julian, and Surya Nepal propose three categories of vulnerabilities, namely hardware, software, and network infrastructure and protocol vulnerabilities [48]. Hardware derived vulnerabilities are mostly seen in the form of unauthentic or illegal hardware clones. One example of hardware vulnerability is no physical-access protection which an attacker might exploit to gain unauthorized physical access. And hence, exploiting hardware base vulnerabilities enables the threat agents to access or alter physical elements of a computer server (e.g., a hard drive) or a network (e.g., a router). Software oriented vulnerabilities exist in system firmware or application software. An outdated software with flaws in source code might be exploited by a bypass threat that is further materialized by a code-injection attack triggered by malicious actors. Network infrastructure and protocol vulnerabilities frequently appear in network protocols such as TCP (referring to transmission control protocol).

Considering the nature of CPS, we define a CPS as composed of software (e.g., firmware, toolset, software library, etc.) and hardware (e.g., a hard drive). Note that software further subsumes operating system (OS) (e.g., a Windows system). OS functionally manages software components and acts as an interface between application software and hardware. For example, a buffer overflow attack might exploit an OS that contains resource management error. It may trigger further Denial of Service (DoS) and result in loss of control of this OS. Once an OS is shut down, the application software components are deactivated. Hardware, software, and OS are assembled and used in different ways within CPS fabrics, creating various binaries with potential backdoors [1,2]. Software is embedded in hardware and thus relies on this hardware component's electricity supply and CPU. Hardware-dependent software is one such example [49]. Meanwhile, software monitors, controls, and actuates hardware components. Furthermore, CPS relies on a proper network connection to transfer data and complete feedback loops.

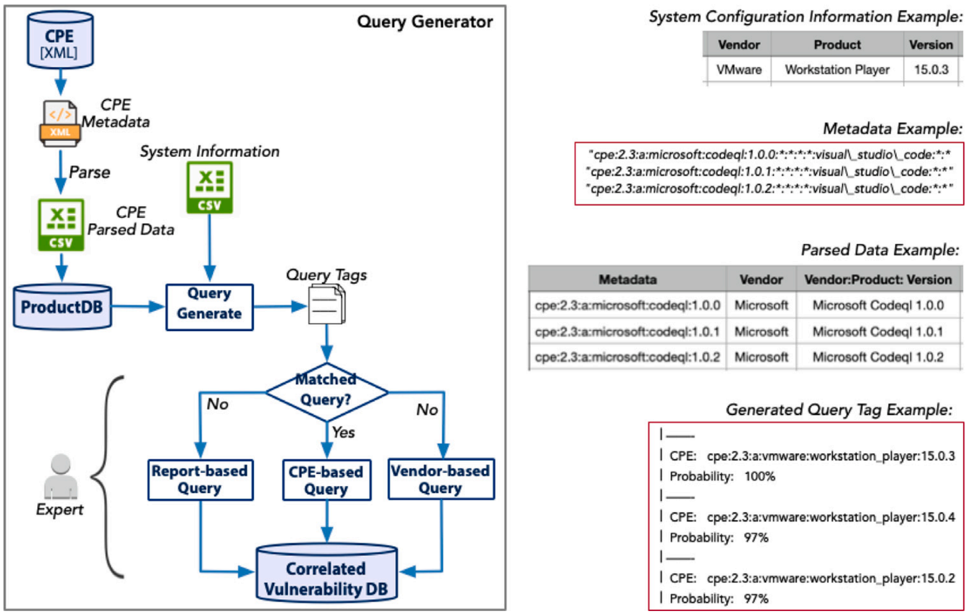
We evaluate our streamlined vulnerability-severity scoring mechanism through vulnerability analysis practices on several prominent CPSs such as PLCs, RTUs (or Remote Terminal Units), MTUs (or Master Terminal Units) and HMIs (or Human Machine Interfaces). A PLC is a crucial CPS asset that controls industrial devices to keep production processes in order. A RTU transmits telemetry data from sensing devices that are associated with physical power components to a MTU system. Finally, a HMI is either a standalone device or embedded communication interface to visualize and monitor MTU activities and RTU information flow [2].

5.2. CPS vulnerability filter

Using a Python script, we retrieve CPS-relevant vulnerability instances from multiple online cybersecurity data sources, including NVD, vendor websites, ICS CERT, and SecurityFocus. In addition, we employ *Shodan* query APIs to obtain vulnerability instances. The retrieval workflow is illustrated in Fig. 3. Following this vulnerability retrieval workflow, we further added a query-keywords generator and a CPS vulnerability filter, as illustrated in Fig. 5.

Using the proposed retrieval and filter workflow, we extracted vulnerable component entities from CVE vulnerability reports using an open-source NER (referring to Named Entity Recognition) model [50], as illustrated in Fig. 5. We then map these retrieved entities with the CPE as well as vendor websites to generate a list of terms related to these components. We retrieved vendor information for each extracted CPS vulnerability instance using NER and correlated against the CPE database. To do so, we obtained vendor *HTML* links from CVE reference maps, based on which we crawl the vendor websites fetching CPS vulnerability related data. This step aims at reconciling potential inconsistent product names. Finally, these terms are combined with the corresponding component versions of interest, that are then used as tags to query vulnerability instances from public vulnerability data sources. We also conducted manual checks on CPS-related vendor metadata, and optimized our search engine outcomes to detect hidden metadata for each vendor, in order to decrease possible false negatives.

More specifically, the query generating process is composed of three major steps and presented in Fig. 6. Note that we only show the processing details for CPE to ensure readability, although we process data extracted from CPE, as well as NVD and vendor reports.



System Configuration Information Example:

Vendor	Product	Version
VMware	Workstation Player	15.0.3

Metadata Example:

```
"cpe:2.3:a:microsoft:codeql:1.0.0:*:*:*:*:visual\_studio\_code:*:*
"cpe:2.3:a:microsoft:codeql:1.0.1:*:*:*:*:visual\_studio\_code:*:*"
"cpe:2.3:a:microsoft:codeql:1.0.2:*:*:*:*:visual\_studio\_code:*:*"
```

Parsed Data Example:

Metadata	Vendor	Vendor:Product: Version
cpe:2.3:a:microsoft:codeql:1.0.0	Microsoft	Microsoft Codeql 1.0.0
cpe:2.3:a:microsoft:codeql:1.0.1	Microsoft	Microsoft Codeql 1.0.1
cpe:2.3:a:microsoft:codeql:1.0.2	Microsoft	Microsoft Codeql 1.0.2

Generated Query Tag Example:

```
|——
| CPE:   cpe:2.3:a:vmware:workstation_player:15.0.3
| Probability:   100%
|——
| CPE:   cpe:2.3:a:vmware:workstation_player:15.0.4
| Probability:   97%
|——
| CPE:   cpe:2.3:a:vmware:workstation_player:15.0.2
| Probability:   97%
```

Fig. 6. Generate query tags.

- In the first step, we parse the metadata from *CPE*, NVD vulnerability report and vendor reports, and extract vendor (e.g., "*microsoft*"), product (e.g., "*codeql*") and version (e.g., "*1.0.0*") information from these metadata. We canonize these extracted items into one entity as "*Vendor Product Version*"(e.g., "*microsoft codeql 1.0.0*"), which results in a dictionary of 816875 such entities. We further generate a dictionary using a shortened metadata as key, and then using our generated entity as value, which is stored in *ProductDB* shown in Fig. 5.
- In the second step, we generate a list of query tags ranked by matching similarities. To start with, we canonize system configuration information that is usually a result of system scan into "*Vendor Product Version*"(e.g., "*vmware woskstation player 15.0.3*") value pairs. We use the vendor information (e.g., "*vmware*") to filter out entities in *ProductDB* from other vendors, then we generate an initial query tag list selected from the remaining entities if they partially share the tokens of software and version information (e.g. "*woskstation player 15.0.3*"). Subsequently, we measure the similarities between the system information string (e.g., "*vmware woskstation player 15.0.3*") and strings in the initialized query tag list. By doing so, we generate a new dictionary using *CPE* metadata as key and similarity as value (e.g., *'cpe:2.3:a:vmware:workstation_player:15.0.3': 100*). We rank this query tag list from higher similarity to lower similarity, and send out the first five (can be customized to other numbers) query tags as results. We summarize this query generation process in Algorithm 2. In our approach, we compute the *Levenshtein* distance to calculate the difference between two strings, and instantiated our method by utilizing the python package *fuzzy.ratio* from [51]. The *Levenshtein* distance refers to the minimum number of the required single-character editing to change one string into the other [52].
- In the last step, we allow manual check and query selection to decrease possible false positives based on other keywords that distinguish them from CPS-related concepts. If we adopt one of the query tags and use *CPE*-based query, the correlated database would return vulnerability instances that share the same *CPE* metadata. If we find that all the generated query tags are not correct, we switch to report- or vendor-based query and retrieve reports that contain the system configuration information string.

5.3. CPS vulnerabilities

We first investigate in *Shodan* database to extract product names, versions and vendors of industrial PLC, RTU, MTU and HMI equipments. The reason we started with *Shodan* investigation is that *Shodan* contains open ports of connected ICS devices nearly in real time. It also covers the most commonly used CPS-based CI equipments, and therefore provides actual device names, versions and vendors for our case study analysis. For example, using *PLC* as the query tag, we gather products like *Mitsubishi Q PLC*. We use these 4 lists of CI product features as input for our query generator to generate queries for our correlated database.

By querying NVD, we obtain respectively 257, 445, 107, and 258 vulnerability reports related to PLC, RTU, MTU and HMI. These retrieved 1067 CPS related vulnerabilities extend till November 3, 2021. Note that some vulnerabilities appear in more than one type of CPS components. One example is the vulnerability instance *CVE-2019-0708* appearing in both PLC and HMI vulnerability groups. We removed duplicated vulnerabilities and kept 870 instances in the analysis corpus when we need to assess general CPS vulnerability features.

We further analyze these identified CPS vulnerabilities to get their CVSS V2 and V3 scores assigned by *NVD*. All of these CPS vulnerabilities are assigned CVSS V2 scores and relevant labels like V2 access vector. In contrast, 319 (71.69%) RTU vulnerabilities, 121 (47.08%) PLC vulnerabilities, 47 (43.93%) MTU vulnerabilities, and 121 (46.90%) HMI vulnerabilities are not assigned CVSS V3 scores. We conduct an investigation of the CVSS V2 labels assigned by *NVD* to our retrieved CPS vulnerabilities. Table 2 lists the exploitability and impact distributions of these vulnerability instances under CVSS V2 metrics. Vulnerabilities exist in the four types of CPS show similar distributions in terms of access vector, access complexity, authentication, and availability impact. There is a higher probability that exploiting PLC vulnerabilities may bring lower confidentiality impact, but higher availability impact. Generally, CPS vulnerabilities have higher exploitability compared to the overall reported vulnerabilities, especially in terms of required authentication and complexity of such exploits.

Vulnerability in endpoint communication of CPE devices may expose the critical data to unauthorized threat actors, and can be exploited by attacks like MiTM attacks. One such example is unencrypted protocol. Vulnerability instance *CVE-2021-22779* shows such improper network segmentation weakness that has been identified in *Modicon*

Algorithm 2 Query tag generation with system configuration information.

```
procedure VulnerabilityQueryGenerator($D, S_1, S_2, S_3, m$)
▷ $D$ is a size $n$ dictionary that has one key $K_1$ and one value $V_1$, whereby $K_1$ is a string representing metadata, and $V_1$ is a string in the format of "vendor product version".
▷ Three imported strings: $S_1$ for vendor: $S_2$ for product; $S_3$ for version.
▷ $m$ is an imported number.
    New query lists $Qlist = []$
    Search term $S_4 = S_2 + S_3$, $S_5 = S_1 + S_2 + S_3$
  For all key-value pairs $(K_i, V_i)$ in D, (i=1,…,n) do
      if ($S_1$ in $V_i$) And ($S_4$ in $V_i$) then
    $Qlist.append(K_i, V_i)$
      End if
  End ForNew query dictionary $Qdict = \{\}$
  For all key-value pairs $(K_j, V_j)$ in Qlist, (j=1,…, len(Qlist)) do
    Compute strings similarity $Similarity_j = fuzzy.ratio(S_5, V_j)$
    $Qdict[K_j] = int(Similarity_j)$
  End For
    Sort dictionary $Qdict(K_j, Similarity_j)$ by value $Similarity_j$
    Export the first $m$ key-value pairs in $Qdict$
End procedure
```

Table 2
CPS vulnerability measurements distribution of CVSS version 2 base metrics.

Metric	Measurement	PLC	RTU	MTU	HMI	CPS	CVE
AccessVector	Network	91.44%	93.03%	90.65%	87.21%	91.00%	83.20%
	AdjacentNetwork	1.95%	0.22%	0.93%	3.10%	1.41%	2.43%
	Local	6.61%	6.74%	8.41%	9.69%	7.59%	14.37%
AccessComplexity	Low	64.20%	64.27%	65.42%	56.59%	62.51%	58.39%
	Medium	32.30%	30.11%	28.04%	39.92%	32.80%	38.62%
	High	3.50%	5.62%	6.54%	3.49%	4.69%	2.98%
Authentication	None	90.27%	95.06%	90.65%	92.25%	92.78%	85.20%
	Single	9.73%	4.94%	9.34%	7.75%	7.22%	14.76%
	Multiple	0%	0%	0%	0%	0%	0.04%
ConfidentialityImpact	None	50.58%	46.29%	42.05%	40.31%	45.45%	32.80%
	Partial	43.58%	42.47%	50.47%	42.64%	43.58%	49.05%
	Complete	5.84%	11.24%	7.48%	17.06%	10.97%	18.15%
IntegrityImpact	None	49.42%	48.76%	54.21%	43.02%	48.08%	29.76%
	Partial	44.74%	40.90%	38.32%	41.09%	41.61%	52.64%
	Complete	5.84%	10.34%	7.48%	15.89%	10.31%	17.60%
AvailabilityImpact	None	28.02%	30.34%	37.38%	33.33%	31.21%	36.23%
	Partial	54.09%	55.73%	52.34%	44.57%	52.30%	42.67%
	Complete	17.89%	13.93%	10.28%	22.09%	16.49%	21.10%

M580 PLCs from *Schneider Electric*. Vulnerable *Modicon* PLCs employ *Schneider Electric* UMAS protocol that operates over the Modbus protocol which lacks encryption and proper authentication mechanisms. This vulnerability allows spoofing attacks to happen against the Modbus communication between the PLC controller and the *EcoStruxure* software in the engineering workstation.

Another common weakness in CPS devices is improper memory access control that allow read or write operations of memory locations, which may cause out-of-bounds read and/or write. One example of such vulnerabilities is *CVE-2020-15782* that exemplifies weakness of improper operation restrictions within the bounds of a memory buffer (or *cwe119*). This vulnerability has been identified in a list of *Siemens SIMATIC* firmware, which allows attackers with network access and download rights to a PLC to bypass existing protections in the PLC, such as PLC sandbox, and obtain read–write memory access remotely while staying undetected. A PLC sandbox refers to a protected area of memory where engineering code could run.

We distinguish some specific CPS manufacturers or vendor vulnerabilities reported by *Schneider Electric SE*, *Siemens AG*, and *Mitsubishi*. More specifically, we discovered 29 vulnerabilities from *Schneider Electric SE* products and 39 vulnerabilities from *Siemens AG* products. We also identified 12 vulnerabilities from *Mitsubishi*. We also observe some frequently published products that are affected by CI vulnerabilities. One typical example is *OpenSSL* that appears in 120 CPS vulnerability instances. *OpenSSL* [53] is a library implementing the SSL/TLS protocol. SSL (referring to secure sockets layer) is the old name of TLS (referring to transport layer security). We also found 51 vulnerability instances related to *Simatic* PLCs and HMIs developed by *Siemens AG*.

5.4. Characteristics analysis of CPS vulnerabilities

As we discussed earlier, more than 57% of extracted CPS vulnerability instances are not scored under the CVSS V3 mechanism. The scoring system shown in Section 4 is used to compute scores for these vulnerabilities, in order to bridge the gap of missing CVSS V3 information. We also calculate CVSS V3 scores for the vulnerabilities with inconsistent scores assigned. We design this re-computation step considering two factors, (i) CVSS V3 is only applied to vulnerabilities disclosed within and after 2015 in some data sources like NVD, and (ii) inconsistent scores are provided by multiple score sources. Subsequently, the diversity of their sub-scores is inspected to reflect CVSS V3 metric scores through property vectors evaluations. Exploitability, Scope and Impact base metric attributes for CPS vulnerabilities are contrasted against actual values and illustrated in Table 3.

CPS component attributes are evaluated individually in Columns 3–6 (or Columns PLC, RTU, MTU, HMI), and averaged in Column 7 (or

Table 3
CPS vulnerability measurements distribution of CVSS version 3 base metrics.

Metric	Measurement	PLC	RTU	MTU	HMI	CPS	CVE
AttackVector	Network	90.27%	94.38%	93.46%	84.11%	90.48%	74.35%
	AdjacentNetwork	1.17%	0.45%	0.93%	3.10%	1.43%	22.57%
	Local	8.17%	5.17%	5.61%	12.79%	7.96%	2.01%
	Physical	0.39%	0%	0%	0%	1.30%	1.06%
AttackComplexity	Low	87.16%	85.39%	80.37%	91.09%	88.79%	91.21%
	High	12.84%	14.61%	19.63%	8.91%	11.21%	8.79%
PrivilegesRequired	None	85.60%	89.44%	84.11%	85.66%	88.01%	69.55%
	Low	12.84%	9.89%	15.89%	13.18%	10.69%	25.18%
	High	1.56%	0.67%	0%	1.16%	1.30%	5.28%
UserInteraction	None	84.82%	91.46%	91.59%	76.36%	84.49%	62.80%
	Required	15.18%	8.54%	8.41%	23.64%	15.51%	37.20%
ScopeChange	Unchanged	90.27%	96.18%	96.26%	87.21%	92.31%	83.64%
	Changed	9.73%	3.82%	3.74%	12.79%	7.69%	16.36%
ConfidentialityImpact	None	44.36%	41.12%	41.12%	30.62%	36.77%	22.15%
	Low	11.67%	5.62%	12.15%	13.95%	9.39%	19.10%
	High	43.97%	53.26%	46.73%	55.43%	53.85%	58.75%
IntegrityImpact	None	52.92%	50.56%	54.21%	44.96%	46.81%	31.14%
	Low	10.89%	5.62%	5.61%	13.18%	8.60%	17.20%
	High	36.19%	43.82%	40.19%	41.86%	44.59%	51.66%
AvailabilityImpact	None	29.18%	32.58%	36.45%	30.23%	29.86%	38.22%
	Low	3.11%	1.57%	3.74%	4.26%	2.22%	2.30%
	High	67.70%	65.84%	59.81%	65.50%	67.93%	61.19%

Column CPS). Column 8 (or Column CVE) shows the overall rate of published CVE reports that have assigned CVSS V3 scores, by dividing the vulnerabilities with certain labeled measurement (e.g., Network) against all the disclosed vulnerabilities till November 3, 2021. By doing so, we show how the significant characteristics of CPS vulnerabilities diverge when considering different cybersecurity data sources.

Exploitability property attributes of CPS vulnerability contrast with CVE counterparts in average, showing that a significant amount (90.48%) of attacks originate from Network-based sources, particularly for RTU vulnerabilities. There are limited occurrences of adjacent network-based attacks against CPS. However, local attacks occur more frequently in CPS. A large amount of CPS vulnerabilities (98.72%) are prone to exploitability by malicious actors without privilege or user interaction. Change of scope is observed in 7.69% instances of CPS vulnerabilities, resulting in severe consequences. A higher diversity among possible impact values is observed compared to exploitability and scope property attributes. Confidentiality and availability are more impacted than the integrity of vulnerable CPS components. Nevertheless, impact of CPS vulnerabilities show polarization distributions when using CVSS V3 as assessment metrics, which is the opposite when using CVSS V2 as metrics. Impact of CPS vulnerabilities are mostly none or partial under CVSS V2 mechanisms. In contrast, CVSS V3 suggests that CPS vulnerabilities exploitation result in either low or high compact.

CVSS V3 is used to rate CVSS severity base scores of retrieved CPS vulnerability reports. HMI, RTU and PLC vulnerability instances show high base scores at 6.5 and 8.5, respectively. MTU vulnerabilities vary within the range [4.5–8.5]. Average scores of 7.54, 8.00, 6.88 and 7.51 in CVSS V3 Base-Scores are observed respectively for PLC, RTU, MTU and HMI. Considering CVSS qualitative scales [54] from quantitative CVSS V3 scores, CPS vulnerability severities are rated *medium* ([4.0–6.9]) to *high* ([7.0–8.9]).

6. Conclusion and future works

Discovering and evaluating vulnerabilities in CPS networks are both crucial and challenging processes. We proposed to raise the efficiency of vulnerability-severity scoring systems following CVSS standards, to rate the severity of a reported vulnerability instance. Our approach reconciles inconsistent vulnerability severity scores that are contributed from different cybersecurity analysers, and also decrease potential conflicts resulting from various CVSS mechanisms. We employed majority voting technique to decide the score of inconsistent reports for the same vulnerabilities in different cybersecurity repositories. We then used these compatible vulnerability instances as ground truth to train a machine-learning model as a scoring basis. The performance of the proposed model is shown to obtain high accuracy and micro F1-score thresholds compared to similar studies. A case study involving CPS vulnerability reports from multiple repositories is illustrated to validate the proposed vulnerability assessment model. A query-filter logic is used to customize retrieved vulnerability instances. The outcomes are contrasted against reported CVE instances to further analyze the characteristics of CPS vulnerabilities. The results of our case study also indicate that vulnerability patterns are diverse when relying on different cybersecurity data sources, which may mislead cybersecurity decision making in the perspective of patch prioritization or budget allocation. And hence, a vulnerability analysis approach that correlates multiple data sources is necessary to enhance further cybersecurity awareness.

The proposed research can be further extended by adjusting the majority voting tie while involving experts' supervision in the assessment loop. This approach includes security experts to provide some startup settings, along with computational intelligence techniques to adjust these settings dynamically. Another possible future direction involves arithmetic means of different scores to weigh several sources, to evaluate the reliability of scores provided from these sources. Finally, we plan to investigate the correlations between vulnerability severity with the attack surface of the system on which the vulnerability assessment is applied. This last planned work is also closely related to the environmental property of CVSS metrics.

CRediT authorship contribution statement

Yuning Jiang: Conceptualization, Methodology, Simulation, Data collection, Writing – original draft. **Yacine Atif:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research has been supported in part by EU ISF (Internal Security Fund) in the context of Project Grant #A431.678/2016.

References

[1] Ashibani Y, Mahmoud QH. Cyber physical systems security: Analysis, challenges and solutions. Comput Secur 2017;68:81–97.

[2] Humayed A, Lin J, Li F, Luo B. Cyber-physical systems security—A survey. IEEE Internet Things J 2017;4(6):1802–31.

[3] Common vulnerability enumeration (CVE). 2022, MITRE, https://www.cve.org/. [Accessed 01 June 2022].

[4] Common vulnerability scoring system (CVSS). 2022, Forum of Incident Response and Security Teams, https://www.first.org/cvss/. [Accessed 01 June 2022].

[5] National vulnerability database (NVD). 2022, National Institute of Standards and Technology (NIST), https://nvd.nist.gov/vuln. [Accessed 01 June 2022].

[6] Anwar A, Abusnaina A, Chen S, Li F, Mohaisen D. Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses. 2020, arXiv preprint arXiv:2006.15074.

[7] Jo H, Kim J, Porras P, Yegneswaran V, Shin S. GapFinder: Finding inconsistency of security information from unstructured text. IEEE Trans Inf Forensics Secur 2020;16:86–99.

[8] Securityfocus Forum. 2021, Accenture, https://www.securityfocus.com/. [Accessed 01 June 2022].

[9] Gawron M, Cheng F, Meinel C. Automatic vulnerability classification using machine learning. In: International conference on risks and security of internet and systems. Springer; 2017, p. 3–17.

[10] Johnson P, Lagerström R, Ekstedt M, Franke U. Can the common vulnerability scoring system be trusted? a bayesian analysis. IEEE Trans Dependable Secure Comput 2016;15(6):1002–15, IEEE.

[11] Spanos G, Angelis L, Toloudis D. Assessment of vulnerability severity using text mining. In: Proceedings of the 21st Pan-Hellenic conference on informatics. 2017, p. 1–6.

[12] Scarfone K, Mell P. An analysis of CVSS version 2 vulnerability scoring. In: 2009 3rd international symposium on empirical software engineering and measurement. IEEE; 2009, p. 516–25.

[13] Fang Y, Liu Y, Huang C, Liu L. FastEmbed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. Plos One 2020;15(2):e0228439, Public Library of Science San Francisco, CA USA.

[14] Ruohonen J. A look at the time delays in CVSS vulnerability scoring. Appl Comput Inf 2019;15(2):129–35, Elsevier.

[15] Common platform enumeration (CPE). 2022, MITRE, https://cpe.mitre.org/. [Accessed 01 June 2022].

[16] Computer emergency response team. 2022, Cybersecurity & Infrastructure Security Agency, https://www.cisa.gov/uscert. [Accessed 01 June 2022].

[17] Microsoft Security Response Center (MSRC). 2022, Microsoft, https://msrc.microsoft.com/update-guide/vulnerability. [Accessed 01 June 2022].

[18] Shodan database. 2022, Shodan, https://www.shodan.io/dashboard. [Accessed 01 June 2022].

[19] Exploit database. 2022, Offensive Security, https://github.com/offensive-security/exploitdb. [Accessed 01 June 2022].

[20] Allodi L, Massacci F. Comparing vulnerability severity and exploits using case-control studies. ACM Trans Inf Syst Secur 2014;17(1):1–20, ACM New York, NY, USA.

[21] Geer D, Roytman M. Measuring vs. modeling. Login:: The Magazine of USENIX & SAGE 2013;38(6):64–7, USENIX Association.

[22] Metasploit. 2022, Rapid 7, https://www.metasploit.com/. [Accessed 01 June 2022].

[23] Rodriguez LGA, Trazzi JS, Fossaluza V, Campiolo R, Batista DM. Analysis of vulnerability disclosure delays from the national vulnerability database. In: Anais do i workshop de seguranÇa cibernética em dispositivos conectados. SBC; 2018.

[24] Chen H, Liu R, Park N, Subrahmanian V. Using twitter to predict when vulnerabilities will be exploited. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. 2019, p. 3143–52.

[25] Zhu Z, Dumitraş T. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In: Proceedings of the 2016 ACM SIGSAC Conference on computer and communications security. 2016, p. 767–78.

[26] Bullough BL, Yanchenko AK, Smith CL, Zipkin JR. Predicting exploitation of disclosed software vulnerabilities using open-source data. In: Proceedings of the 3rd ACM on international workshop on security and privacy analytics. 2017, p. 45–53.

[27] Sabottke C, Suciu O, Dumitra? T. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In: 24th $\{USENIX\}$ security symposium ($\{USENIX\}$ security 15). 2015, p. 1041–56.

[28] Bozorgi M, Saul LK, Savage S, Voelker GM. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. 2010, p. 105–14.

[29] Yamamoto Y, Miyamoto D, Nakayama M. Text-mining approach for estimating vulnerability score. In: 2015 4th international workshop on building analysis datasets and gathering experience returns for security. IEEE; 2015, p. 67–73.

[30] Jiang Y, Atif Y. An approach to discover and assess vulnerability severity automatically in cyber-physical systems. In: 13th international conference on security of information and networks. 2020, p. 1–8.

[31] CVE numbering authorities. 2022, MITRE, https://cve.mitre.org/cve/cna.html. [Accessed 01 June 2022].

[32] Common weakness enumeration (CWE). 2022, MITRE, https://cwe.mitre.org/index.html. [Accessed 01 June 2022].

[33] ICS-CERT advisories. 2022, Cybersecurity & Infrastructure Security Agency, https://us-cert.cisa.gov/ics/advisories. [Accessed 01 June 2022].

[34] Siemens Report for CVE-2019-6580, CVE-2019-6581 and CVE-2019-6582. 2021, https://cert-portal.siemens.com/productcert/pdf/ssa-212009.pdf. [Accessed 01 June 2022].

[35] Bodenheim R, Butts J, Dunlap S, Mullins B. Evaluation of the ability of the shodan search engine to identify internet-facing industrial control devices. Int J Crit Infrastruct Prot 2014;7(2):114–23, Elsevier.

[36] Fagroud FZ, Ajallouda L, Toumi H, Achtaich K, El Filali S, et al. IOT search engines: Exploratory data analysis. Procedia Comput Sci 2020;175:572–7, Elsevier.

[37] Tao D, Cheng J, Yu Z, Yue K, Wang L. Domain-weighted majority voting for crowdsourcing. IEEE Trans Neural Netw Learn Syst 2018;30(1):163–74, IEEE.

[38] Mahto DK, Singh L. A dive into web scraper world. In: 2016 3rd international conference on computing for sustainable global development. IEEE; 2016, p. 689–93.

[39] Beautiful soup. 2022, Leonard Richardson, https://www.crummy.com/software/BeautifulSoup/bs4/doc/. [Accessed 01 June 2022].

[40] Brodersen KH, Ong CS, Stephan KE, Buhmann JM. The balanced accuracy and its posterior distribution. In: 2010 20th international conference on pattern recognition. IEEE; 2010, p. 3121–4.

[41] Powers DM. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. 2011, Bioinfo Publications.

[42] NVD feature analysis. 2021, Yuning Jiang, https://github.com/Yuning-J/NVDFeatureAnalysis. [Accessed 01 June 2022].

[43] Vulnerability classifier. 2021, Yuning Jiang, https://github.com/Yuning-J/VulnerabilityClassifier. [Accessed 01 June 2022].

[44] Stodden VC. Reproducible research: Addressing the need for data and code sharing in computational science. 2010.

[45] González-Barahona JM, Robles G. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. Empir Softw Eng 2012;17(1):75–89, Springer.

[46] Countvectorizer. 2022, scikit-learn, https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. [Accessed 01 June 2022].

[47] Tdidftransforer. 2022, scikit-learn, https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html. [Accessed 01 June 2022].

[48] Jang-Jaccard J, Nepal S. A survey of emerging threats in cybersecurity. J Comput System Sci 2014;80(5):973–93, Elsevier.

[49] Ecker W, Müller W, Dömer R. Hardware-dependent software. In: Hardware-dependent software. Springer; 2009, p. 1–13.

[50] Yang Z, Salakhutdinov R, Cohen WW. Transfer learning for sequence tagging with hierarchical recurrent networks. 2017, arXiv preprint arXiv:1703.06345.

[51] Fuzzywuzzy: Fuzzy string matching in python. 2021, https://github.com/seatgeek/fuzzywuzzy. [Accessed 01 June 2022].

[52] Haldar R, Mukhopadhyay D. Levenshtein distance technique in dictionary lookup methods: An improved approach. 2011, arXiv preprint arXiv:1101.1232.

[53] OpenSSL. 2022, https://www.openssl.org/. [Accessed 01 June 2022].

[54] CVSS V3 documentation. 2022, Forum of incident response and security teams, https://www.first.org/cvss/v3.1/specification-document. [Accessed 01 June 2022].