Predicting IoT Security Vulnerabilities from Device Specifications

Arslane Fawzi Halilou (✉) and Natalia Stakhanova (✉)

Department of Computer Science, University of Saskatchewan, Canada
kjc705@usask.ca, natalia@cs.usask.ca

Abstract. Technical documentation often encodes implicit indicators of device behavior and design choices that can correlate with security weaknesses. This paper introduces a specification-driven framework that automatically extracts such indicators from technical documentation to predict the presence of vulnerabilities in IoT devices. We convert textual specifications into features using TF-IDF and BM25, then evaluate XGBoost and Random Forest classifiers on a corpus of 1,521 documents spanning vulnerable IoT devices, non-vulnerable IoT devices, and non-IoT products. XGBoost achieved the best performance (accuracy 95.1%, precision 95.4%, recall 95.1%) with TF-IDF and BM25 yielding near-equivalent results, indicating stable, recurring specification patterns associated with vulnerable devices. These findings show that specification-derived signals can serve as an effective, scalable early-warning mechanism to prioritize devices for targeted security assessment, complementing vulnerability repositories such as the NVD.

1 Introduction

With the wide integration of IoT devices into daily life comes a pressing need for robust security, as compromised devices pose risks not only to individual users but also to large-scale systems. Despite this need, proactively identifying vulnerabilities in IoT devices remains a significant challenge. Existing methods present numerous challenges. *First*, they are reactive as developers often discover vulnerabilities only after security incidents occur. When a vulnerability is discovered, it is usually documented and reported in official repositories such as the National Vulnerability Database (NVD). These repositories provide standardized identifiers, descriptions, and affected components, enabling developers, researchers, and security teams to assess the impact and prioritize remediation. However, this process is inherently reactive and depends on vulnerabilities being discovered first, leaving a gap in early detection and risk mitigation.

Second, they are fragmented, since inconsistencies in development standards and the closed-market environment hinder the development of scalable, comprehensive solutions for IoT vulnerability detection [11]. *Third*, they are constrained by limited access to device internals. Firmware is commonly encrypted, obfuscated, and shipped with disabled debugging interfaces [11], which also prevents analysis of deeper security flaws that cannot be observed at the network or service layers [11,37]. Together, these challenges create a persistent gap between security expectations and real-world practices.

Addressing these gaps requires exploring alternative strategies that can proactively identify weaknesses before they are exploited. To date, research on proactive vulnerability identification has been dominated by static and dynamic software code analysis [24]. Beyond code analysis, a few studies have highlighted the significance of unofficial sources (such as mailing lists and news podcasts) [12], as well as dependency and code metrics [27], in predicting early signals of device vulnerabilities. Imtiaz et al. [17,15] advocated for the important role of early software development life-cycle (SDLC) stages, exploring requirements engineering and design phases to indicate typical vulnerabilities encountered in similar past implementations.

To complement these approaches, we take a step further and investigate the role of device technical documents in predicting implementation weaknesses that may lead to software vulnerabilities. Typically, system requirements are first defined in specifications, which are then translated into device technical documentation (e.g., datasheets) describing the detailed characteristics, features, and limitations of a specific component or device. These datasheets, long considered purely functional documents, can encode implicit indicators of potential weaknesses. Unlike firmware or source code, specifications are often available prior to deployment and describe device design choices (e.g., supported protocols, allowed features, authentication requirements). Extracting and modeling features from this technical documentation therefore offers a potentially scalable, low-cost method to complement vulnerability testing, enabling earlier detection and prioritization of high-risk devices.

Building on this premise, this study investigates the potential of leveraging IoT device datasheets to detect vulnerabilities and their associated weaknesses. This approach is particularly relevant given the widespread deployment of IoT devices in diverse contexts, including time-sensitive and safety-critical environments, where patching is often constrained by strict operational requirements.

We propose *a framework that explores the potential of IoT devices inherent specifications found in technical datasheets to identify indicators of vulnerabilities and weaknesses.*

To extract meaningful features from these technical documents, we employed two information retrieval techniques: TF-IDF and BM25. The framework was evaluated on a dataset of 1,521 documents of vulnerable IoT devices, non-vulnerable IoT devices, and non-IoT devices. The experimental results demonstrate high predictive performance, with XGBoost achieving the best results across all evaluation metrics (95.1% accuracy, 95.4% precision, 95.1% recall). The findings highlight the effectiveness of the proposed framework in identifying vulnerable IoT devices and their associated weaknesses solely from their specifications. Although BM25 is generally regarded as an improvement over the traditional TF-IDF technique, the performance differences in this study were negligible, with both techniques achieving results in the range of 94% to 95%

across all metrics. The results suggest that vulnerable IoT devices exhibit recurring specification patterns that are consistently reflected in their technical documentation. These patterns provide a valuable basis for early identification of potential vulnerabilities and weaknesses in IoT devices.

2 Background

Traditionally, the vulnerability discovery process focuses on uncovering security weaknesses in software systems. Once a vulnerability is detected, the initial step is confirming that the issue is a legitimate flaw. If vulnerability is validated, the next step determines the severity of the vulnerability and its potential risks. The findings are then reported according to the disclosure policy, whether internally, to the software vendor, or through public vulnerability databases. Each officially recognized vulnerability is assigned a unique CVE identifier, which serves as a standardized reference across security tools, researchers, and organizations. Officially recognized software vulnerabilities are registered in authoritative repositories such as the National Vulnerability Database (NVD) and the Common Vulnerabilities and Exposures (CVE) database.

CVE records commonly reference one or more Common Weakness Enumeration (CWE) identifiers to indicate the underlying weakness responsible for a reported vulnerability. The CWE taxonomy provides a structured vocabulary for software and hardware weaknesses, allowing systematic classification, analysis, and communication. In the CWE *research concept view*, weaknesses are further organized into a set of 10 higher-level "pillars" that group functionally related weakness types. For example, *Stack-based Buffer Overflow* and *Return of Pointer Value Outside of Expected Range* are both categorized under the pillar *Improper Control of a Resource Through its Lifetime.* The CWE taxonomy further classifies weaknesses under each pillar into 3 hierarchical categories based on their level of abstraction: ① Base weaknesses describe resource-independent flaws with sufficient technical detail to enable concrete detection and prevention strategies. ② Class weaknesses provide broader, technology-agnostic descriptions of vulnerability patterns. ③ Variant weaknesses occupy the most specific tier, describing flaws tied to particular programming languages, technologies, or device implementations.

3 Related work

Over the past decade, research on vulnerabilities has grown significantly. This increased interest stems mainly from the limitations of publicly available vulnerability databases, delays in the vulnerability registration process [30], and inconsistencies between NVD and CVE information [10].

Vulnerability prediction. Research on prediction of vulnerabilities, i.e., detection of vulnerabilities before their public release, has been limited. Existing efforts often investigate traditional fault prediction metric (e.g., complexity, code churn, fault history) [31], similarities between imports and function calls shared between different components [26] to identify vulnerable code. In similar context, Imtiaz and Bhowmik proposed a framework that leverages past implementations of similar requirements to predict common vulnerabilities encountered during design, and early implementation phases [17]. This framework was later refined in [16] to predict vulnerabilities for new requirements.

These solutions, however, might not be always applicable to IoT devices since vendors often encrypt firmware updates, obfuscate binaries, or disable JTAG/USB debugging [11]. To overcome this, a later work proposed leveraging unofficial source (i.e., mailing list, news website, and podcast) to detect early vulnerability signals for OT devices [12].

Vulnerability detection. Multiple studies have been elaborated in this context, especially in software security, to mitigate the risks of wide-spread vulnerabilities. Current software vulnerability detection methodologies involve static, dynamic, and hybrid analysis [24]. Static analysis leverages abstract interpretation of source code to analyze program attributes (e.g., function calls and control flow) and encompass code similarity detection [18,29], rule-based analysis [22], and symbolic execution [23]. Dynamic analysis methods, on the other hand, simulate the program's operation under various input data to detect vulnerabilities that emerge during execution. Methods for dynamic analysis include taint analysis [21] and fuzz testing [28]. For a more comprehensive and enhanced analysis, hybrid approaches combine static and dynamic approaches [1].

These solutions were also proposed to detect vulnerabilities in IoT firmware but proved to be more complicated. Challenges for detecting vulnerabilities in IoT devices firmware were not restricted on the code alone but also emulation and re-hosting the firmware, difficulty of dynamic analysis (computing power and operational constraints), and availability of the firmware (closed-source, JTAG/USB ports disabled) [11].

Vulnerability exploitation, The studies focusing on exploitation aim to understand how likely and when attackers would take advantage of detected vulnerabilities. One of the key challenges in this area is the limited availability of ground truth data. Existing studies have leveraged sources such as public exploits (e.g., ExploitDB) [3], security advisories [4], dark web forums [2], vulnerability databases [5,14], and social media [14]. Yet, Suciu et al. [32] showed that artifacts published after vulnerability disclosure are generally good predictors of vulnerability exploitability.

Beyond vulnerability exploitation prediction based on textual descriptions, several studies explored exploitability at the code level, e.g., analyzing vulnerable functions in the Apache HTTP Server [36], static features extracted from Linux application executables [35], and code crashes [33]. Other lines of research have investigated the timeframe of vulnerability exploitation after its official registration [8,7], and its severity [25,34].

Although efforts in the current state of the art show the potential of online public sources as threat intelligence sources, they often rely on known vulnerabil-

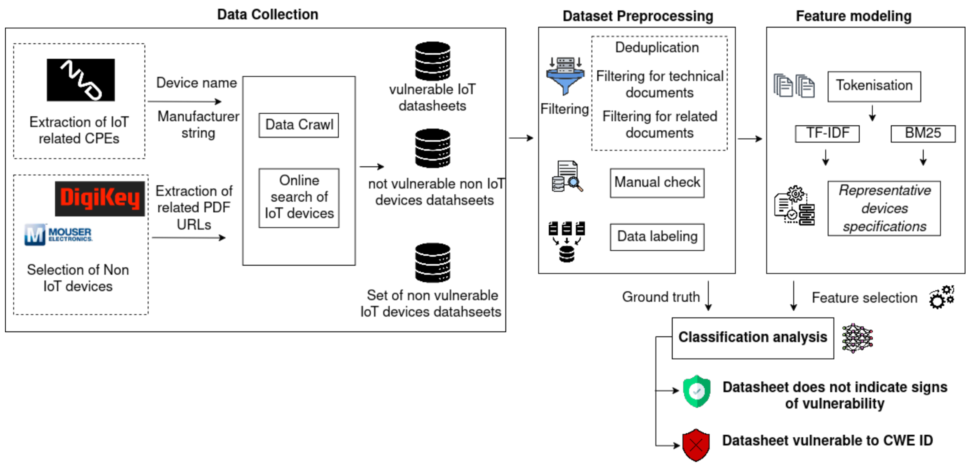
Data Collection
Extraction of IoT related CPEs
Selection of Non IoT devices
Device name
Manufacturer string
Extraction of related PDF URLs
Data Crawl
Online search of IoT devices
vulnerable IoT datasheets
not vulnerable non IoT devices datahseets
Set of non vulnerable IoT devices datahseets
Dataset Preprocessing
Filtering
Deduplication
Filtering for technical documents
Filtering for related documents
Manual check
Data labeling
Feature modeling
Tokenisation
TF-IDF
BM25
Representative devices specifications
Ground truth
Feature selection
Classification analysis
Datasheet does not indicate signs of vulnerability
Datasheet vulnerable to CWE ID

Fig. 1: Flow of the proposed framework.

ities for detection and prediction purposes. Furthermore, information extracted from these sources cannot be verified and can easily be misinterpreted, leading to imprecisions and performance issues. Even when information is extracted from known security experts, character limitations to posts in platforms like Twitter (now known as X) makes it difficult to extract detailed technical information [6].

Unlike previous works in the literature, we explore a different approach to vulnerability identification and propose a framework that leverages technical documents to extract early signs of weaknesses and vulnerabilities from device specifications alone.

4 Proposed approach

To facilitate early identification of vulnerabilities, we propose a framework that leverages devices datasheets to determine whether technical characteristics of device implementation are likely to lead to a security vulnerability. Figure 1 illustrates the general flow of the proposed framework that consists of four main stages: data collection and pre-processing, feature modelling, and classification analysis.

4.1 Data

To establish ground truth for our study, we required reliable information on both vulnerable and non-vulnerable devices. While details on vulnerable IoT devices are relatively easy to find in public sources, identifying devices that are not vulnerable is less straightforward. To address this, we built a dataset by collecting datasheets across three categories: vulnerable IoT devices, non-vulnerable IoT devices, and non-IoT devices. We intentionally included the third

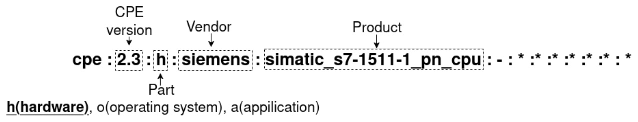
CPE version
Vendor
Product
cpe : 2.3 : h : siemens : simatic_s7-1511-1_pn_cpu : - : * :* :* :* :* : *
Part
h(hardware), o(operating system), a(appilication)

Fig. 2: An example of CPE field extracted from CVE-2014-5074 record.

category of non-IoT device documents as a challenging negative control group. The purpose of this category is to test the robustness of our model. It ensures that the model learns to identify meaningful signals specifically characteristic of vulnerable IoT devices, rather than simply recognizing common technical specifications or patterns found across all types of device technical documents.

Vulnerable IoT devices datasheets. Although several sources provide information on publicly disclosed vulnerabilities (e.g., NVD), they typically do not offer a way to exclusively filter CVEs related solely to IoT devices. To address this, we leveraged VARIoT, a database dedicated to IoT vulnerabilities [19]. VARIoT provides standardized and aggregated information on official IoT vulnerabilities collected from multiple vulnerability repositories. By extracting CVE identifiers from this database and checking them against the NVD, it is therefore possible to extract only NVD entries related to IoT.

Typically, NVD records include the description of the vulnerability and supplementary information (i.e., external links and Common Platform Enumeration fields) to specify the affected component. Although external links seem more promising to extract datasheets of affected devices, our analysis showed that only 2,653 unique IoT-related CVEs (7.2%) in the NVD contained links to PDF documents hosted on official company websites, typically company reports on the affected devices. These reports generally discuss the vulnerability or issue at hand, with only a brief mention of the affected device(s) and their versions rather than technical device specifications.

Common Platform Enumeration (CPE) scheme, on the other hand, provides more specific information on the affected device including product name, vendor, and version. Figure 2 shows an example of the CPE field extracted from NVD records for *Simatic s7-1511-1 pn cpu* device from *Siemens*.

In this work, we leverage the VARIoT database to extract CVE identifiers related to IoT devices. Using these CVE identifiers, we then retrieve the corresponding CPEs of vulnerable IoT devices from the NVD database. From these, we filter entries that indicate hardware devices (i.e., where the part subfield is set to "h"). The resulting CPE fields are parsed to extract vendor and product strings, which facilitate the automated collection of datasheets.

Data collection. For the collection process, we used Bing search, as it imposes fewer restrictions on automated queries compared to Google search. Datasheets were collected in PDF format using a custom crawler with the following query: *"vendor" "product" datasheet filetype:pdf.* To improve accuracy, we restricted downloads to PDFs hosted on domains containing the vendor name in the URL.

Data preprocessing. Once the collection of the datasheets was done, we preprocessed them to remove duplicate documents and only keep those relevant to our study. For the vulnerable set containing datasheets of vulnerable IoT devices, we applied a three layer automated filtering:

- *Layer 1:* We de-duplicated the set of documents using the SHA-256 hash function.
- *Layer 2:* Since the automatically obtained set occasionally included documents that were not actual device specifications, a second layer of filtering was applied to remove unrelated files. To achieve this, we used a hardcoded list of terms that consistently appear on the first page of technical documents describing device implementation details, such as guide, datasheet, manual, specification(s), brochure, and catalogue. Based on our analysis, we found most datasheets explicitly stating the vendor and product name on the first page of the document. The first page of each pdf document was tokenized by spaces and matched against this list. Only files containing at least one match were retained at this stage. Although, some documents might have an empty first page, this filter ensures that only related documents are maintained.
- *Layer 3:* The final filtering step aimed to exclude documents unrelated to the vulnerable devices in our IoT set. The token set generated during Layer 2 filtering was matched against the vendor and product strings extracted from the CPE fields of IoT-related CVE entries. Documents that did not contain these vendor or product strings were excluded from the analysis.

The final set was then manually validated.

To establish ground truth, we labelled the extracted datasheets. Vulnerable IoT devices datasheets were labelled according to the vulnerability information present in the CVE database. To provide a better understanding on the vulnerability and its probable cause, we leveraged CWE base pillars as these provide sufficient technical detail to enable concrete detection. For each vulnerability, we extracted the corresponding CWE field and its corresponding pillar that specified the high level type of problem. To avoid confusion, we only retained entries that had one pillar. We leverage these pillars to categorize devices' datasheets based on the pillar CWE.

Non vulnerable IoT devices datasheets. *Data collection.* For this set of datasheets, we manually downloaded 200 datasheets of 150 IoT devices commonly found in home, health, and fitness applications (e.g., smart lock, wearable devices, remote patient monitoring devices) online.

Data preprocessing. This set was preprocessed in the same way as the vulnerable set. We applied Layer 1 and Layer 2 filtering to remove duplicates and unrelated files. To ensure the validity of the non-vulnerable IoT set, we compared the vendor and product strings of all CPEs in the NVD against the tokens assembled during Layer 2 filtering and retained only those documents without matches. This filtering step is specific to the non-vulnerable IoT set, and we refer to it as *Layer 4* filtering. The resulting set of 168 datasheets was labeled as not vulnerable.

Non IoT devices datasheets. For non IoT devices, we focused on standalone devices that are not built for connected applications. Examples of these devices include Computer Numerical Control (CNC) machines (drilling and milling machines), Uninterruptible Power Supply (UPS), display modules (excluding human-machine interface devices), process control and temperature control devices, and industrial robots. Datasheets for these devices were collected from DigiKey and Mousers Electronics, one of the leading providers of electronic components, devices, and parts.

Data collection. As for non IoT devices datasheets, we targeted standalone devices that are not built specifically for the collection and exchange of data with other internet connected devices. These devices datasheets were manually downloaded from DigiKey (third party provider of electronic components and devices).

Data preprocessing. We applied Layer 1 and Layer 2 filtering for this set, similar to the other sets. The resulting set of 652 documents were labeled as non IoT.

4.2 Feature modelling

A crucial step in our analysis is constructing a set of features that capture potential indicators of vulnerability in IoT device specifications. One of the main challenges is the absence of direct references to vulnerabilities or attacks in device datasheets. While the language used to describe vulnerabilities can be found in official repositories such as the NVD, these descriptions are typically brief (averaging 42 words) and focus on the type of problem and the affected component or device, rather than explaining the underlying cause of the vulnerability.

For our feature set, we leveraged the collected datasets. The collected datasheets were converted to plain text. The resulting text was tokenized using whitespace as the delimiter, producing three sets of tokens for vulnerable IoT, not vulnerable IoT, and not IoT datasheets.

For efficient analysis, it was imperative to reduce the volume of information, as we initially extracted over 16 million tokens from all datasheets. To address this, we employed Term Frequency–Inverse Document Frequency (TF-IDF) and Best Matching 25 (BM25) to identify the most distinguishing tokens. These techniques, widely used in natural language processing [20], evaluate the significance of terms across a corpus, in our case, the collected datasheets. TF-IDF combines term frequency (TF) and inverse document frequency (IDF) to assign a score reflecting the uniqueness of terms across all documents. BM25 builds on this idea but introduces a saturation effect to limit the influence of document length on the scores. For instance, a relevant term may appear many times in a long document compared to a short one, which could otherwise distort its importance score. By normalizing the frequency component, BM25 mitigates this effect and provides a more balanced evaluation.

For TF-IDF, let $f_{t,d}$ represent the number of times a word t appears in a datasheet d, and $\sum_{t' \in d} f_{t',d}$ represent the total number of terms in d. As for BM25, let *avgl* indicate the average document length in the corpus of documents d of average length $|d|$, with parameters b and k to control the influence of the document length and the saturation effect respectively.

$$TF_{TF-IDF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

$$TF_{BM25}(t,d) = \frac{f(t,d) \cdot (k_1 + 1)}{f(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)} \tag{2}$$

Inverse Document Frequency (IDF): reflects the relevance of a term across datasheets by assessing its occurrence throughout the corpus. Let N be the total number of datasheets in the corpus and n_t the number of datasheets that contain the term t.

$$IDF_{TF-IDF}(t,D) = \log\left(\frac{N}{1+n_t}\right) \tag{3}$$

$$IDF_{BM25}(t) = \log\left(\frac{N - n_t + 0.5}{n_t + 0.5}\right) \tag{4}$$

Term Frequency–Inverse Document Frequency (TF-IDF): combines both TF and IDF in equations (1) and (3) to determine the significance of a word across a corpus.

$$TF - IDF(t,d,D) = \text{TF}_{TF-IDF}(t,d) \cdot \text{IDF}_{TF-IDF}(t,D) \tag{5}$$

Best Match 25 (BM25): the final score for a document d with respect to a term t is calculated by combining TF and IDF in (2) and (4).

$$BM25(t,d) = IDF(t)_{BM25} \cdot TF_{BM25}(t,d) \tag{6}$$

Our choice of using a syntactic feature extraction technique (i.e., TF-IDF and BM25) rather than semantic methods (e.g., KeyBERT) is motivated by the nature of the data. In our case, semantic approaches might interpret tokens according to their literal meaning such as 4MB (memory size) or 5V (voltage) which are important device specifications that could help identify potential vulnerabilities or weaknesses.

4.3 Identification of vulnerable IoT devices datasheets

The features derived from specifications of devices are used to train machine learning classifiers. In this work, we focus on two particular algorithms: Random Forest and XGBoost to validate the proposed framework. Random Forest [13] is an ensemble learning method that combines the decisions of numerous decision trees created during training and outputs the final decision by averaging the results during inference. The model is resistant to overfitting and works well with

Table 1: Model parameters for XGBoost and Random Forest algorithms

Parameter	**XGBoost**	**Random Forest**
n_estimators	100	150
max_depth	5	10
class_weight	–	`balanced_subsample`
scale_pos_weight	N/P	–
random_state	42	42
n_jobs	Default (None)	-1 (all processors)

N and P denote the number of negative and positive samples, respectively.

multidimensional feature spaces. XGBoost (Extreme Gradient Boosting) [9] is a scalable and efficient gradient boosting framework that generates decision trees consecutively. Each new tree tries to fix the mistakes produced by the previous ones, and the model is optimized using the gradient descent process. XGBoost is well-known for its outstanding classification performance, particularly in structured data contexts.

5 Classification analysis

5.1 Experimental setup

The framework was implemented using Python programming language with the following libraries: Pymupdf for conversion of datasheets to plain text, Scikit-learn for TF-IDF scores calculation, and finally Pandas and Pyarrow for data processing. Experiments were conducted using 5-fold cross validation for all experiments. Table 1 states the parameters used for the classification algorithms. The experiments were conducted on a Linux machine running on Fedora OS with 134 GB of RAM and 56 CPU cores.

5.2 Overview of the collected data

IoT datasheets. The VARIoT dataset [19] consists of 41,946 entries, of which 36,865 are unique records with a CVE identifier referencing the NVD (87.9%). These records were cross-checked against the NVD, and all were confirmed to be present. The IoT-related CVEs in the NVD corresponded to 11,882 unique hardware CPEs, which were used to collect datasheets and manuals online.

Table 2 provides an overview of the collected datasheets for IoT and non-IoT devices. As shown, vulnerable IoT devices had the highest number of available documents (2,875). However, after deduplication and filtering, the number of relevant datasheets dropped drastically to 701 (24.4%). This is comparable to the filtered non-IoT device documents, which is 652 (85%). The set of non-vulnerable IoT devices, in contrast, contained 168 documents (84%) after filtering.

Table 3 offers further details on the analyzed documents. Datasheets for vulnerable IoT devices were generally longer than those for non-IoT devices, with 46.6% of vulnerable IoT documents exceeding 15 pages, compared to 9.4% of non-IoT documents.

Table 2: Summary of the collected datasheets.

Set	# of Documents	# Layer 1 filter	# Layer 2 filter	# Layer 3 filter	# Layer 4 filter
Vulnerable IoT devices	2,875	1,912 (66.5%)	1,041 (36.2%)	701 (24.4%)	-
Non vulnerable IoT devices	200	200 (100.0%)	200 (100.0%)	-	168 (84.0%)
Non IoT devices	772	746 (96.6%)	652 (84.5%)	-	-
Total	**3,847**	**2,858 (74.3%)**	**1,893 (49.2%)**	**701 (18.2%)**	**168 (4.4%)**

Table 3: Overview of relevant IoT datasheets.

Set	# Docs	< 5 pages	5-15 pages	> 15 pages	# Tokens
Vulnerable IoT	701	155 (22.1%)	219 (31.2%)	327 (46.6%)	13,923,734
Non-vulnerable IoT	168	81 (48.2%)	35 (20.8%)	52 (31.0%)	539,266
Non-IoT	652	274 (42.0%)	317 (48.6%)	61 (9.4%)	1,568,166
Total	**1521**	**510 (33.5%)**	**571 (37.5%)**	**440 (28.9%)**	**16,031,166**

Feature sets. Tokenization of documents' text was performed prior to applying TF-IDF and BM25, which resulted in 16 million tokens (Table 3). After application of TF-IDF and BM25, we extracted the top 1,000, 2,000, 5,000, 10,000, and 20,000 features for each of the three sets: vulnerable IoT, non-vulnerable IoT, and non-IoT datasheets. We then merged the sets within each category and deduplicated the features to create the feature sets: 1,000, 2,000, 5,000, 10,000, and 20,000 features for both TF-IDF and BM25. We use different number of tokens to evaluate the performance of the classification models and determine whether the increase in the number of features lead to better results or simply introduce unnecessary noise.

Table 4 describes the extracted feature sets using TF-IDF and BM25. While alphabetic tokens dominate the smaller sets for both methods, the proportion of tokens containing special characters increases as the feature sets expand (from around 30% to 70% of the total). These tokens with special characters capture detailed technical characteristics, including measurement specifications (e.g., ±2% and ±0.5°C for tolerance and accuracy; -40°C to +85°C for operating temperature ranges; 802.11b/g/n for supported WiFi standards), model or part numbers (e.g., ESP32-WROOM-32 microcontroller), pin identifiers (e.g., GPIO_0 and TX/RX), and technical parameters (e.g., >100dBm and 100mA@3.3V for measurement).

CWE pillars. For our proposed framework, we focused on datasheets and manuals having one single pillar to evaluate whether our framework can help distinguish between device's datasheets having different weakness from specifications alone. The grouping of datasheets and manuals based on the CWE pillars is described in Table 5. In total, we counted 148 (out of 701) datasheets with only one CWE pillar. These documents were categorized into four pillars: *Improper adherence to coding standards*, *Improper control of a resource through its lifetime*, *Improper neutralization*, and *Improper access control.* Most of the datasheets (80%) fell under the *Improper control of a resource through its lifetime* pillar and were predominantly short (fewer than 5 pages). The prevalence of short documents in our subset contrasts with the general distribution of datasheets

Table 4: Feature sets extracted from IoT devices datasheets.

Technique	**Features**	**Total tokens**	**Alphabetic tokens**	**Alphanumerical tokens**	**Tokens with special chars**
TF-IDF	TF-IDF 1000	1,188	767 (64.56%)	15 (1.26%)	406 (34.18%)
	TF-IDF 2000	2,456	1,318 (53.66%)	64 (2.61%)	1,074 (43.73%)
	TF-IDF 5000	6,481	2,471 (38.13%)	231 (3.56%)	3,779 (58.31%)
	TF-IDF 10k	14,450	4,114 (28.47%)	672(4.65%)	9,664 (66.88%)
	TF-IDF 20k	34,108	7,471 (21.90%)	2,132 (6.25%)	24,505 (71.85%)
BM25	BM25 1000	1,268	859(67.74%)	20(1.58%)	389(30.68%)
	BM25 2000	2,437	1,398 (57.37%)	54(2.22%)	985(40.42%)
	BM25 5000	6,445	2,683 (41.63%)	239(3.71%)	3,523(54.66%)
	BM25 10k	14,614	4,870 (33.32%)	705(4.82%)	9,039 (61.85%)
	BM25 20k	34,610	8,389 (24.24%)	2,564(7.41%)	23,657 (68.35%)

Table 5: Datasheets and manuals with a single CWE pillar.

Device category	**# Docs**	**<5 pages**	**5-15 pages**	**>15 pages**	**# Tokens**
improper access control	1	0 (0.0%)	0 (0.0%)	1 (100.0%)	19,583
Improper adherence to coding standards	4	0 (0.0%)	3(75.0%)	1(25.0%)	16,982
Improper control of a resource through its lifetime	118	97 (82.2%)	10 (8.5%)	11 (9.3%)	191,427
Improper neutralization	25	22(88.0%)	2 (8.0%)	1(4.0%)	39,458
Total	**148**	**119 (80.4%)**	**15(10.1%)**	**14(9.5%)**	**267,450**

and manuals for vulnerable IoT devices. This trend can be attributed to the limited functionalities, and consequently fewer specifications, of vulnerable devices with a single-pillar weakness.

5.3 Vulnerability identification results

In this research, we conducted several sets of experiments to evaluate the proposed framework's ability: ① to detect vulnerable IoT datasheets, and ② to differentiate high-level weaknesses of vulnerable datasheets in a presence of non-vulnerable specifications.

Detection of vulnerable IoT datasheets. Table 6 describes evaluation results of the classifiers in detecting vulnerable IoT datasheets.

As the results show, XGBoost consistently outperforms Random Forest in this experiment, with the gap most apparent for accuracy and recall. XGBoost exhibits steady performance across different configurations for both TF-IDF and BM25, reaching its best result with BM25 at 20k features (accuracy 97.1%).

Random Forest achieves good overall results across TF-IDF and BM25 feature sets, although its performance gradually decreases as the number of TF-IDF features increases, suggesting that additional TF-IDF features introduce noise that reduces accurate detection of vulnerable IoT datasheets. Nevertheless, Random Forest show strong performance at TF-IDF 1000 with 94.7% accuracy, 97.4% precision, and 96% recall.

Table 6: Experimental results for vulnerable IoT datasheets.

Technique	**Features**	**Random Forest**			**XGBoost**		
		Accuracy	**Precision**	**Recall**	**Accuracy**	**Precision**	**Recall**
TF-IDF	1000	0.947	0.974	0.960	0.967	0.984	0.974
	2000	0.941	0.975	0.952	0.969	0.984	0.977
	5000	0.938	0.972	0.950	0.962	0.981	0.971
	10k	0.930	0.966	0.946	0.968	0.983	0.977
	20k	0.923	0.962	0.941	0.967	0.983	0.976
BM25	1000	0.936	0.966	0.953	0.969	0.984	0.977
	2000	0.937	0.969	0.951	0.957	0.981	0.966
	5000	0.941	0.970	0.957	0.961	0.984	0.967
	10k	0.919	0.957	0.943	0.965	0.991	0.966
	20k	0.916	0.955	0.940	0.971	0.989	0.976

Overall, these results indicate that XGBoost is more responsive to larger feature sets and offers superior detection of vulnerable IoT datasheets in this dataset.

Closer inspection of the features selected for analysis revealed that both models identified similar discriminative features, which clustered around several themes: authentication mechanisms (e.g., "factory-default," "password," "admin," "root," "credential"), communication protocols (e.g., "http," "telnet," "uart"), cryptographic implementations (e.g., "ssl," "tls," "md5"), and hardware debug interfaces (e.g., "debug," "console," "serial," "jtag," "i2c").

Further analysis of vulnerabilities in our dataset provided additional insights. For example, Korenix JetNet 5628G series switches were found to contain hard-coded credentials (CVE-2017-14027), as well as embedded certificates and private keys (CVE-2017-14021) that allowed unauthenticated remote access and arbitrary code execution. Examining the device datasheet revealed specific language patterns that correspond to these vulnerabilities. For instance, the datasheet listed "Admin password", "Reset to default" and "backup/restore" capabilities that cluster with the "factory-default" and "credential" features our models prioritized.

Similarly, Rockwell Automation EtherNet/IP controllers exhibited firmware update vulnerabilities (CVE-2012-6437) where authentication was not performed for Ethernet firmware updates, allowing remote code execution via malicious update images. The user manual documents multiple firmware update pathways ("FactoryTalk Linx", "RSLinx Classic", "USB port", "Product Compatibility and Download Center") with detailed procedural instructions but no mention of authentication requirements, signature verification, or code signing mechanisms. For example, the technical documentation states that USB driver allows "Update the device firmware" and that "USB port is intended for temporary local programming purposes only", suggesting that USB was a known less-secure channel but was still enabled for firmware updates. This documentation pattern, where update procedures are extensively described without corresponding security controls, directly aligns with the authentication bypass vulnerability and demonstrates how absence of security-related terminology in specifications can indicate exploitable weaknesses.

This alignment between feature importance and documented vulnerabilities validates that the model captures genuine security risks rather than spurious correlations.

The overall results of these experiments show that *the technical documentation of vulnerable IoT devices often contains language that reflects vulnerable code patterns*. This further suggests that developers may commonly interpret certain technical characteristic descriptions in ways that lead to vulnerable implementations.

Detection of high-level weaknesses of vulnerable datasheets in a presence of non-vulnerable specifications. This set of experiments focused on multiclass classification of datasheets.

We used 147 datasheets linked to vulnerabilities mapped to a single CWE pillar, along with 30 randomly selected non-IoT and non-vulnerable IoT datasheets to reduce class imbalance. Documents from two CWE pillars were excluded due to insufficient samples, i.e., *improper adherence to coding standards* (4 documents) and *improper access control* (1 document).

According to the results shown in Table 7, specifications in datasheets capture functional characteristic descriptions of weaknesses that later manifest as vulnerabilities. Similar to our previous set of experiments, the results demonstrate that device specifications provide robust signals for identifying vulnerable IoT devices and their associated weaknesses. As in the previous experiment, XGBoost generally outperforms Random Forest across most feature sets and metrics.

Random Forest tends to provide better detection of weaknesses for vulnerable IoT datasheets with smaller feature sets. Surprisingly, the model exhibits a non-monotonic relationship with vocabulary size despite its robustness to noise. This effect is most apparent for the TF-IDF sets where accuracy, precision, and recall drop from 91.6%, 92.2%, 91.6% (1,000 features) to 87.2%, 87.5%, 87.2% (20k features). This suggests that excessive TF-IDF features introduce noise that degrades the model's true-positive detection. Random Forest distinguishes vulnerable datasheets well overall but struggles with the non-vulnerable class (precision and recall as low as 84% and 87% for the best configuration). It also shows weakness with class 2 (improper neutralization), where true-positive recall drops to 68% in some configurations. Nonetheless, Random Forest remains competitive, with its best performance achieved using BM25 with 5,000 features.

Unlike Random Forest, XGBoost benefits better from larger feature sets. Its performance accross different classes, specifically with vulnerable datasheets, generally improves as features increase. The model, however, experiences modest declines observed at 2k and 5k in TF-IDF and BM25 sets. This pattern is visible in the TF-IDF series, where XGBoost drops from 91.1% accross all metrics at 10k to 95.1%, 95.4%, 95.1% (accuracy, precision, and recall) at 20k. XGBoost also shows better per-class performance results for vulnerable datasheets (class 2 and 3) at 20k TF-IDF set. For this configuration, XGBoost attains near-perfect detection for vulnerable IoT datasheets with improper neutralization class (96%

Table 7: Experimental results on detection of high-level weaknesses of vulnerable datasheets in a presence of non-vulnerable specifications.

Set	**Features**	**Class**	**Random Forest**					**XGBoost**				
			Accuracy	**Precision**		**Recall**		**Accuracy**	**Precision**		**Recall**	
				Avg	**Per class**	**Avg**	**Per class**		**Avg**	**Per class**	**Avg**	**Per class**
TF-IDF	1000	0	0.916	0.922	1.00	0.916	0.87	0.926	0.928	0.93	0.926	0.90
		1			0.79		0.87			0.78		0.83
		2			1.00		0.76			0.96		0.88
		3			0.92		0.97			0.96		0.97
	2000	0	0.897	0.906	1.00	0.897	0.80	0.911	0.911	0.93	0.911	0.93
		1			0.72		0.87			0.77		0.77
		2			0.95		0.72			0.88		0.84
		3			0.92		0.97			0.95		0.96
	5000	0	0.916	0.923	1.00	0.916	0.83	0.941	0.943	0.93	0.941	0.93
		1			0.79		0.87			0.81		0.87
		2			1.00		0.76			0.88		0.88
		3			0.92		0.98			0.99		0.97
	10k	0	0.887	0.890	0.96	0.887	0.83	0.936	0.939	0.85	0.936	0.97
		1			0.74		0.77			0.81		0.83
		2			0.95		0.76			0.96		0.88
		3			0.90		0.96			0.99		0.97
	20k	0	0.872	0.875	1.00	0.872	0.80	0.951	0.954	0.93	0.951	0.90
		1			0.71		0.67			0.83		0.97
		2			0.90		0.76			0.96		0.92
		3			0.88		0.97			0.99		0.97
BM25	1000	0	0.902	0.904	1.00	0.902	0.90	0.926	0.929	0.97	0.926	0.93
		1			0.85		0.73			0.79		0.87
		2			0.95		0.76			0.92		0.88
		3			0.88		0.97			0.96		0.95
	2000	0	0.911	0.917	1.00	0.911	0.87	0.921	0.923	0.84	0.921	0.90
		1			0.75		0.80			0.80		0.80
		2			1.00		0.80			1.00		0.92
		3			0.92		0.97			0.96		0.96
	5000	0	0.926	0.928	1.00	0.926	0.93	0.902	0.903	0.87	0.902	0.90
		1			0.84		0.87			0.75		0.80
		2			0.95		0.76			0.90		0.76
		3			0.93		0.97			0.95		0.96
	10k	0	0.892	0.896	0.96	0.892	0.87	0.931	0.936	0.88	0.931	0.93
		1			0.76		0.83			0.76		0.87
		2			0.94		0.68			0.95		0.84
		3			0.90		0.96			0.99		0.97
	20k	0	0.887	0.893	1.00	0.887	0.80	0.946	0.949	0.93	0.946	0.90
		1			0.78		0.83			0.82		0.93
		2			0.94		0.68			0.92		0.92
		3			0.88		0.97			0.99		0.97

0: Non IoT datasheets
1: Non vulnerable IoT datasheets
2: Vulnerable IoT datasheets with *improper neutralization*
3: Vulnerable IoT datasheets with *improper control of a resource through its lifetime*

precision and 92% recall) and improper control of a resource through its lifetime class (99% precision and 97% recall), outperforming Random Forest on both.

The difference in performance between XGBoost and Random Forest is important as it translates a better detection and coverage of different classes of weaknesses that led to the vulnerability. While both XGBoost and Random Forest exhibits extremely low false positives and negatives counts (with near perfect precision for XGBoost), XGBoost achieves noticeably better true-positive detection. This is critical in this context as it illustrates better coverage of vulnerabilities. The observed trends suggest that expanding feature sets may further enhance XGBoost's performance.

To gain deeper insight into classifier performance, we manually inspected the feature sets of vulnerable IoT device datasheets, which revealed two primary vulnerability patterns corresponding to fundamental security weaknesses. The *first* involves authentication-related tokens ("username," "password," "login") and network communication indicators ("http," "cookies"), which suggest exposed input surfaces that frequently correlate with improper neutralization vulnerabilities, particularly when combined with insecure default configurations.

The *second* pattern relates to hardware and resource management tokens, which demonstrate systemic resource control weaknesses. Memory-related features ("flash," "eeprom," "ddr3," "buffer," "cache") indicate potential issues with improper memory lifecycle management, while hardware interface tokens ("gpio," "i2c," "spi," "uart," "interrupt") and power management features ("battery," "voltage," "sleep") suggest inadequate resource allocation controls. These patterns reflect the inherent constraints of IoT architectures (e.g., limited memory, simplified OS, and resource, constrained hardware) that often lack sophisticated management mechanisms.

In summary, the results obtained in this experiment and the previous experiments show strong performance of the models in detecting vulnerable IoT datasheets and associated weaknesses using only device specifications, supporting our hypothesis that *inherent device specifications can be used as early signals to detect weaknesses associated with vulnerabilities.*

6 Conclusion

In this work, we demonstrate that technical specifications found in datasheets, manuals, and product guides contain reproducible signals that can be used to anticipate security weaknesses in IoT products. By transforming specification text into features with TF-IDF and BM25 then applying supervised classifiers (Random Forest and XGBoost), our specification-driven framework successfully distinguished vulnerable IoT devices from non-vulnerable and non-IoT products. Overall, XGBoost delivered the strongest performance (accuracy 95.1%, precision 95.4%, recall 95.1%) with comparable performance of TF-IDF and BM25, suggesting that vulnerability patterns are robustly expressed in device documentation. Our results show that device specifications offer a lightweight, scalable early-warning signal that can help organizations triage devices for deeper technical assessment before vulnerabilities are disclosed in reactive repositories such as the NVD. Recurring specification patterns identified by the model provide strong signals that security teams can use to prioritize testing and remediation under constrained operational conditions (e.g., safety-critical or time-sensitive deployments).

References

1. Alhuzali, A., Gjomemo, R., Eshete, B., Venkatakrishnan, V.: {NAVEX}: Precise and scalable exploit generation for dynamic web applications. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 377–392 (2018)
2. Almukaynizi, M., Grimm, A., Nunes, E., Shakarian, J., Shakarian, P.: Predicting cyber threats through hacker social networks in darkweb and deepweb forums. In: Proceedings of the 2017 International Conference of The Computational Social

Science Society of the Americas. CSS 2017, Association for Computing Machinery, New York, NY, USA (2017)

3. Almukaynizi, M., Nunes, E., Dharaiya, K., Senguttuvan, M., Shakarian, J., Shakarian, P.: Proactive identification of exploits in the wild through vulnerability mentions online. In: 2017 International Conference on Cyber Conflict (CyCon U.S.). pp. 82–88 (2017)
4. Bennouk, K., Mahouachi, D., Ait Aali, N., El Bouzekri El Idrissi, Y., Sebai, B., Faroukhi, A.Z.: Dynamic data updates and weight optimization for predicting vulnerability exploitability. IEEE Access **13**, 65266–65284 (2025)
5. Bozorgi, M., Saul, L.K., Savage, S., Voelker, G.M.: Beyond heuristics: learning to classify vulnerabilities and predict exploits. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 105–114. KDD '10, Association for Computing Machinery, New York, NY, USA (2010)
6. Burnap, P., Javed, A., Rana, O.F., Awan, M.S.: Real-time classification of malicious urls on twitter using machine activity data. In: 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). pp. 970–977 (2015)
7. Chen, H., Liu, J., Liu, R., Park, N., Subrahmanian, V.: Vest: A system for vulnerability exploit scoring & timing. In: Kraus, S. (ed.) Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI 2019. pp. 6503–6505. IJCAI International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence (2019)
8. 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. p. 3143–3152. KDD '19, Association for Computing Machinery, New York, NY, USA (2019)
9. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 785–794. KDD '16, Association for Computing Machinery, New York, NY, USA (2016)
10. Dong, Y., Guo, W., Chen, Y., Xing, X., Zhang, Y., Wang, G.: Towards the detection of inconsistencies in public security vulnerability reports. In: 28th USENIX Security Symposium (USENIX Security 19). pp. 869–885. USENIX Association, Santa Clara, CA (Aug 2019)
11. Feng, X., Zhu, X., Han, Q.L., Zhou, W., Wen, S., Xiang, Y.: Detecting vulnerability on iot device firmware: A survey. IEEE/CAA Journal of Automatica Sinica **10**(1), 25–41 (2023)
12. Halilou, A., Stakhanova, N.: Revealing unreported ot vulnerabilities from public discussions. In: The 20th International Conference on Risks and Security of Internet and Systems (CRiSIS). Springer in the Lecture Notes in Computer Science (LNCS) (2025)
13. Ho, T.K.: Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition. vol. 1, pp. 278–282. IEEE (1995)
14. Huang, S.Y., Ban, T.: Monitoring social media for vulnerability-threat prediction and topic analysis. In: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 1771–1776 (2020)
15. Imtiaz, S., Amin, M.R., Do, A.Q., Iannucci, S., Bhowmik, T.: Predicting vulnerability for requirements. In: 2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI). p. 160–167. IEEE Press (2021)

16. Imtiaz, S., Amin, M.R., Do, A.Q., Iannucci, S., Bhowmik, T.: Predicting vulnerability for requirements. In: 2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI). p. 160–167. IEEE Press (2021)
17. Imtiaz, S.M., Bhowmik, T.: Towards data-driven vulnerability prediction for requirements. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. p. 744–748. ESEC/FSE 2018, Association for Computing Machinery, New York, NY, USA (2018)
18. Jang, J., Agrawal, A., Brumley, D.: Redebug: Finding unpatched code clones in entire os distributions. In: 2012 IEEE Symposium on Security and Privacy. pp. 48–62 (2012)
19. Janiszewski, M., Rytel, M., Lewandowski, P., Romanowski, H.: Variot - vulnerability and attack repository for the internet of things. In: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid). pp. 752–755 (2022)
20. Kadhim, A.I.: Term weighting for feature extraction on twitter: A comparison between bm25 and tf-idf. In: 2019 International Conference on Advanced Science and Engineering (ICOASE). pp. 124–128 (2019)
21. Karim, R., Tip, F., Sochůrková, A., Sen, K.: Platform-independent dynamic taint analysis for javascript. IEEE Transactions on Software Engineering **46**(12), 1364–1379 (2020)
22. Lee, M., Cho, S., Jang, C., Park, H., Choi, E.: A rule-based security auditing tool for software vulnerability detection. In: 2006 International Conference on Hybrid Information Technology. vol. 2, pp. 505–512 (2006)
23. Li, H., Kim, T., Bat-Erdene, M., Lee, H.: Software vulnerability detection using backward trace analysis and symbolic execution. In: 2013 International Conference on Availability, Reliability and Security. pp. 446–454 (2013)
24. Liang, C., Wei, Q., Du, J., Wang, Y., Jiang, Z.: Survey of source code vulnerability analysis based on deep learning. Computers & Security **148**, 104098 (2025)
25. Manai, E., Mejri, M., Fattahi, J.: Helping cnas generate cvss scores faster and more confidently using xai. Applied Sciences **14**(20) (2024)
26. Neuhaus, S., Zimmermann, T., Holler, C., Zeller, A.: Predicting vulnerable software components. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. p. 529–540. CCS '07, Association for Computing Machinery, New York, NY, USA (2007)
27. Owolabi, S., Rosati, F., Abdellatif, A., Carli, L.D.: Characterizing packages for vulnerability prediction. In: 2025 IEEE/ACM 22nd International Conference on Mining Software Repositories (MSR). pp. 359–363 (2025)
28. Pham, V.T., Böhme, M., Santosa, A.E., Căciulescu, A.R., Roychoudhury, A.: Smart greybox fuzzing. IEEE Transactions on Software Engineering **47**(9), 1980–1997 (2021)
29. Sajnani, H., Saini, V., Svajlenko, J., Roy, C.K., Lopes, C.V.: Sourcerercc: scaling code clone detection to big-code. In: Proceedings of the 38th International Conference on Software Engineering. p. 1157–1168. ICSE '16, Association for Computing Machinery, New York, NY, USA (2016)
30. Sauerwein, C., Sillaber, C., Huber, M.M., Mussmann, A., Breu, R.: The tweet advantage: An empirical analysis of 0-day vulnerability information shared on twitter. In: Janczewski, L.J., Kutyłowski, M. (eds.) ICT Systems Security and Privacy Protection. pp. 201–215. Springer International Publishing, Cham (2018)
31. Shin, Y., Williams, L.: Can traditional fault prediction models be used for vulnerability prediction? Empirical Software Engineering **18**(1), 25–59 (Dec 2011)

32. Suciu, O., Nelson, C., Lyu, Z., Bao, T., Dumitras, T.: Expected exploitability: Predicting the development of functional vulnerability exploits. In: 31st USENIX Security Symposium (USENIX Security 22). pp. 377–394. USENIX Association, Boston, MA (Aug 2022)
33. Tripathi, S., Grieco, G., Rawat, S.: Exniffer: Learning to prioritize crashes by assessing the exploitability from memory dump. In: 2017 24th Asia-Pacific Software Engineering Conference (APSEC). pp. 239–248 (2017)
34. 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 (BADGERS). pp. 67–73 (2015)
35. Yan, G., Lu, J., Shu, Z., Kucuk, Y.: Exploitmeter: Combining fuzzing with machine learning for automated evaluation of software exploitability. In: 2017 IEEE Symposium on Privacy-Aware Computing (PAC). pp. 164–175 (2017)
36. Younis, A.A., Malaiya, Y.K.: Using software structure to predict vulnerability exploitation potential. In: 2014 IEEE Eighth International Conference on Software Security and Reliability-Companion. pp. 13–18 (2014)
37. Yu, M., Zhuge, J., Cao, M., Shi, Z., Jiang, L.: A survey of security vulnerability analysis, discovery, detection, and mitigation on iot devices. Future Internet **12**(2) (2020)