MalVulDroid: Tracing Vulnerabilities from Malware in Android using Natural Language Processing

Shivi Garg[1,*] and Niyati Baliyan[2]

[1]*Faculty of Informatics and Computing, J.C. Bose University of Science and Technology YMCA, Faridabad, India*
[2]*Department of Computer Engineering, National Institute of Technology Kurukshetra, Haryana, India*
E-mail: shivi1989@gmail.com; niyatibaliyan@nitkkr.ac.in
[*]*Corresponding Author*

Received 20 March 2022; Accepted 11 January 2023;
Publication 13 March 2023

Abstract

The Android operating system is often inflicted with mobile malware attacks, which occur due to some system loopholes or vulnerabilities. One malware can exploit numerous vulnerabilities and multiple malware can exploit a single vulnerability, thus, causing many-to-many $(X:Y)$ mapping between malware and vulnerability. Therefore, it is crucial to understand malware behaviour to reduce the vulnerabilities. This paper presents the concept of a "MalVulDroid" framework that maps malware to vulnerabilities using a two-dimensional matrix. The many-to-many $(X:Y)$ mapping matrix is obtained by using natural language processing techniques such as Bag-of-Words (BoW) leveraging n-gram probability generation and term frequency-inverse document frequency (TF-IDF), in addition to supervised machine learning classifiers such as multilayer perceptron (MLP), a support vector machine (SVM), a ripple down rule learner (RIDOR), and a pruning rule-based classification tree (PART). This study is the first of its kind where

Journal of Web Engineering, Vol. 21_8, 2339–2362.
doi: 10.13052/jwe1540-9589.2185

malware-to-vulnerability mapping can be leveraged to measure the rigorousness of unknown vulnerabilities and malware during the early phases of application development. The study considers extensive datasets such as Androzoo, AMD, and CICInvesAndMal2019 with 150 malware families and 48,907 malware samples, and nine major vulnerabilities affecting Android. MalVulDroid exhibits highly promising results with an accuracy of 98.04% for unigrams, and precision and F1-scores of over 90% using ensemble classifiers.

Keywords: Android, machine learning, malware, mapping, natural language processing, vulnerability.

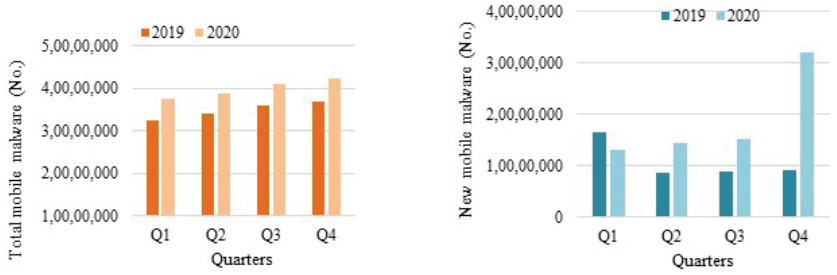
1 Introduction

The COVID-19 pandemic speeded up the transition of the global workforce to their homes. As a result, organizations were forced to change their infrastructure so that employees could work from homes comfortably and in a productive manner. This new paradigm led to the increased usage of mobile devices to perform critical tasks. Due to this, mobile attack expanded, making them more susceptible to cyber threats. Phishing attacks increased by 52% during COVID-19. People are lured to share sensitive information due to the wide interest in COVID-19 tracking and vaccination apps. Cybercriminals also targeted the oxygen shortage in India during COVID-19 in April 2021. The fake oximeter app, distributed through SMS and WhatsApp, executed banking trojans and captured the financial information of the user [1].

According to [2], new ransomware samples have increased by 106% and trojans by 128%. Figure 1 shows the detection of total mobile malware and new mobile malware [3].

Android is more susceptible to attacks since it is more popular among users with the global market share being $\sim$87% at the end of 2018 [4].

The rapid evolution of Android malware makes it difficult to be detected, which causes serious concern among the security investigators. To the aim of handling the tremendous increase in volume and diversity of Android malware, this paper presents the crucial relationship between malware and vulnerabilities. According to [5], applications from third party unofficial repositories are other major sources of Android malware. Past research has focused on detection and classification of Android malware. In [6], a one-dimensional convolutional neural network (1D CNN) is automated to detect Android malware, where the model automatically extracts the features from n-grams of opcode sequences to determine the malicious binary file;



(a) Total mobile malware detections (b) New mobile malware detections

Figure 1 Mobile malware detections by quarter during 2019–20. (a) Total mobile malware detections. (b) New mobile malware detections.

however, they did not attempt to explore malware-to-vulnerability mapping. This mapping can help identify correlated threats in an Android system.

1.1 Contributions

The following are the primary contributions of this work:

1. We propose a framework "MalVulDroid", which automatically maps malware to the vulnerabilities exploited.
2. MalVulDroid is a novel amalgamation of text processing techniques (BoW, *n*-gram, and TF-IDF) and machine learning (ML) methods such as multilayer perceptron (MLP), support vector machine (SVM), ripple down rule learner (RIDOR), and pruning rule-based classification tree (PART) in the domain of Android malware analysis.
3. We assess MalVulDriod on 150 malware family samples from Androzoo, CICInvesAndMal2019, and AMD with 48,907 malware samples. The accuracy rate for unigrams is 98.04%, and precision and F1-scores of over 90% with ensemble classifiers.
4. MalVulDroid allows the discovery of loopholes in Android OS stack during early design phases while mapping malware to known/ unknown vulnerabilities.

1.2 Research Objectives (ROs)

We aim to achieve the following research objectives (ROs) in this study:

1. To systematically map malware to the exploited vulnerabilities in Android. A malware can occur due to loopholes in the system.

For example DroidKungFu malware exploits the root privileges vulnerability (gain privileges) in the Android kernel.

2. To identify unknown malware and vulnerabilities. The new malware and vulnerabilities are continuously updated from different cybersecurity reports and bulletins periodically. We tend to create a rich knowledge base of malware and vulnerabilities for future use, which can be helpful to make useful predictions with high accuracy.
3. To take appropriate actions after successfully mapping malware to vulnerabilities at early stages of Android stack development.

We aim to achieve the ROs while addressing the following research questions (RQs):

RQ1. What are the various kinds of vulnerabilities in Android OS, and additionally which malware do these vulnerabilities relate to?
RQ2. How can machine learning models be used to fit words in the malware corpus?
RQ3. How effectively can the MalVulDroid framework detect malware families using a word-based model on behavioural reports?

The rest of the paper is arranged as follows: Section 2 presents the various studies related to this domain. Section 3 talks about the different vulnerabilities and malware prevalent in Android. Section 4 focuses on the proposed method. Evaluation of MalVulDroid is presented in Section 5. Section 6 draws the generalized insights. Finally, Section 7 concludes the research along with the future directions.

2 Related Works

This section primarily focusses on the recent studies based on Android malware and vulnerabilities. These works help to address the major concerns and to present the notion of malware-to-vulnerability mapping in Android.

2.1 Literature Review

The authors of [7] introduced a hybrid model using NLP and image recognition to detect the conventional obfuscation techniques in the Android environment. They targeted four obfuscation classes, namely string encryption, identifier renaming, reflection, and class encryption, and achieved an average F-measure of 0.985.

In another work [8], the authors proposed TC-Droid for detecting Android malware based on text classification. TextCNN was used to further

differentiate malicious and benign applications. TC-Droid achieved an accuracy of 96.72% and precision of 97.60%.

The authors of [9] proposed a MalDy framework for malware detection using advanced NLP and supervised ML. MalDy behavioural reports were transformed into word sequences to automatically engineer the relevant security features using NLP and ML approaches. MalDy could achieve good results, however, it could not measure the quality of the behavioural reports and that impacted the performance.

In another work [10], CoDroid was proposed, which is a sequence-based hybrid method for Android malware detection. It used the sequences of dynamic system calls and static opcodes. The authors then classified the hybrid sequences into malicious and benign using the CNN–BiLSTM–Attention model. CoDroid achieved 97.6% accuracy.

The authors of [11] detected Android malware using the text semantics of network traffic. They considered each HTTP flow generated by mobile apps as a text document, which was processed by using NLP to extract text-level features. Text semantic features of network traffic were used to develop an effective malware detection model. For assessment purpose, they used 31,706 benign flows and 5258 malicious flows. Although they achieved an accuracy of 99.15%, it suffered from major limitations. Firstly, some of the malicious activities were not fully triggered without effective user inputs. Secondly, the method was able to detect unknown samples that present some common characteristics using n-gram features with the malware samples in the training dataset.

In another work [12], the authors proposed a novel feature selection method along with an Android malware detection approach. The static features such as permissions, API calls, and strings were used in this study. The feature vector was then fed to different ML algorithms to detect Android malware. The authors used a document frequency-based approach called Delta IDF for feature selection. They obtained the highest accuracy and malware detection ratios ranging from 99.4% to 99.8% for a MalGenome dataset, 99.7% to 100% for an AndroZoo dataset, and 98.8% to 99.6% for a Drebin dataset.

In [13], the authors used CNN to detect malicious opcode sequence locations and compared them with the state-of-the-art local interpretable model-agnostic explanations (LIME) method. CNN achieved the highest accuracy of 98% on the Drebin dataset.

In another work [14], the authors proposed a DeepAMD to classify Android malware using a deep artificial neural network (ANN). The results

Table 1 Comparison of previous studies

Citation	Objective	Dataset	ML/DL Model	NLP Technique	Results
[7]	Obfuscation detection	PRAGuard	SVM, DNN, CNN	Bag of Words (Words Count, TF-IDF)	F-measure = 0.985
[8]	Malware detection	Anzhi, MalGenome,	TextCNN	–	Accuracy = 0.966
[9]	Malware detection	MalGenome, Drebin, Maldozer, AndroZoo, PlayDrone	CART, Etrees, KNN, RF, SVM, XGBoost	Bag of words (TF-IDF)	F-measure = 0.948
[10]	Malware detection	PlayDrone, Drebin	CNN–BiLSTM	Word2vec, GloVe	Accuracy = 0.97
[11]	Malware detection	VirusShare, hiapk	DT, Bayesian Network, AdaBoost, NN	Bag of words (TF-IDF)	Accuracy = 0.991
[12]	Malware detection	MalGenome, Drebin, AndroZoo	RBF, MLP, k-NN, SLogReg, SVM, MODLEM, ID3, J48, CART, RF	TF-IDF	Accuracy = 0.997
[13]	Malware detection	Drebin	CNN	–	Accuracy = 0.98
[14]	Malware detection + classification	CICInvesAndMal2019, CICAndMal2017	DT (J48), NB, SM, MLP, Deep ANN	–	Accuracy = 0.934 on the static layer to classify binary Malware. Accuracy = 0.931 on the static layer to classify malware families. Accuracy = 0.803 on the dynamic layer for malware category classification. Accuracy = 0.59 on the dynamic layer for malware family classification.

showed that DeepAMD achieved an accuracy of 93.4% for malware classification, 90% for malware family classification, and 92.5% for malware category classification.

Table 1 presents a comparison of previous studies.

2.2 Motivation

Previous studies in the domain of Android malware and vulnerabilities mainly focus on the detection and classification of malware, vulnerability patching, etc. Nevertheless, none of the works report the vulnerabilities exploited by malware. Given the continuous exploitation of vulnerabilities,

the necessity to understand Android malware properties is the need of the hour. The work presented in this study is novel in mapping malware to the exploited vulnerabilities in Android.

3 Android Malware and Vulnerabilities

This section answers RQ1. Cyber-criminals can launch malware attacks by exploiting the vulnerabilities in the mobile platforms. According to [15], malware can exploit 89% of reported vulnerabilities.

Mobile malware is categorized into different types [16] based on their intended functions, such as, Adware, Ransomware, BackDoor, Trojan, Trojan-SMS, Trojan-Banker, Trojan-Clicker, Trojan-Spy, Trojan-Dropper, and HackerTool [17]. Figure 2 reveals the percentage distribution of the various kinds of mobile malware collected from CICInvesAndMal2019, Androzoo, and AMD. There are various approaches to detect Android malware such as static, dynamic, and hybrid analysis [18].

Android is often exploited by several vulnerabilities types such as memory corruption, SQL injection, denial of service (DoS), gain privileges, overflow, gain information, code execution, bypass something, and directory traversal. A detailed snapshot of Android vulnerabilities and their impact on the Android platform during 2015–2019 is presented in [19].

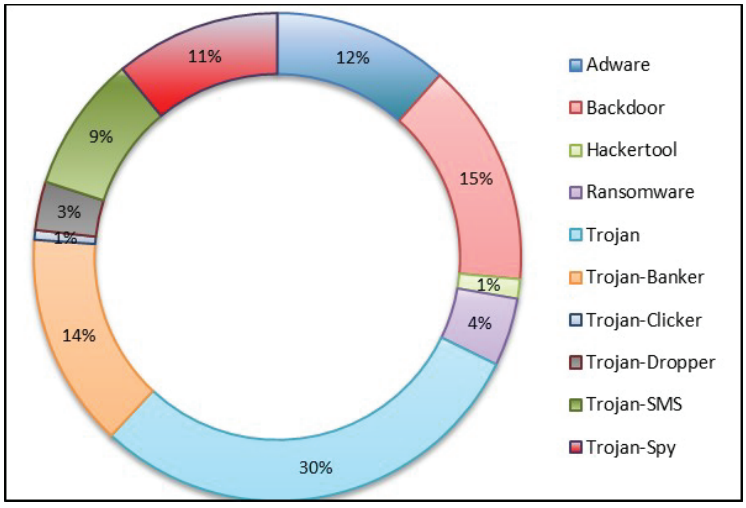


Figure 2 Mobile malware types.

Intrinsic properties of malware need to be explored for understanding malware characteristics. Android malware can exhibit different characteristics such as how the malware is installed and activated, its composition, its anti-analysis techniques, etc. [20].

4 Malware-to-vulnerabilities Mapping

Malware are introduced in the early phases of software development due to loopholes that are leveraged by the attackers. There are various techniques for detecting Android malware, namely, static, dynamic, and hybrid along with ML approaches [21]. A 2D many-to-many $(X : Y)$ mapping is constructed when a malware exploits many known or unknown vulnerabilities and one vulnerability is exploited by many malware, as shown in Figure 3. In the future, more comprehensive mapping between unknown malware and vulnerabilities can be established.

Set X denotes malware and represented as $X = \{x_1, x_2, \ldots, x_m\}$ and the set Y denotes vulnerability, represented as $Y = \{y_1, y_2, \ldots, y_n\}$. A mapping function $f(x) = y$ is represented as $f : x \rightarrow y$, where $x \in X$ and $y \in Y$, which draws malware in X to the vulnerabilities in Y. $f(x)$ is a multi-valued function, i.e., for every value of x it gives multiple values of y.

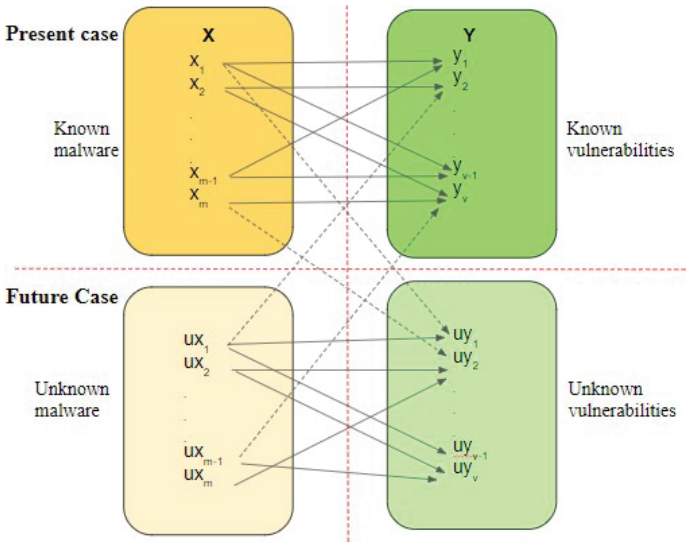


Figure 3 Malware-to-vulnerability mapping $(X : Y)$.

Malware-to-vulnerability mapping properties are defined as follows:

1. Malware-to-vulnerability mapping should be enhanced continuously, i.e., $\forall\, x\, \exists\, y$ s.t. $f(x) \in \{y_1, y_2, \ldots, y_n\}$.
2. Malware-to-vulnerability mapping should ensure soundness (produce only the mappings which actually exist) and completeness (produce all the mappings which actually exist), i.e., $f(x) \neq \emptyset$.
3. Assume there is no mutual exclusion in the vulnerability sets such that $y_a \in Y'$ and $y_b \in Y''$, where Y' and Y'' are two vulnerability datasets with a and b number of vulnerabilities such that $y_a \cap y_b \neq \emptyset$, then $Y' \cup Y'' = Y' + Y'' - (Y' \cap Y'')$. This condition is sufficiently met applicable for updating the vulnerability dataset to discourage any spurious mapping.

4.1 Proposed Method

The step-by-step method, shown in Figure 4, is as follows:

4.1.1 Report generation

We have collected a dataset with a total of 150 families of Android malware and 48,907 malware samples. The benchmarked datasets such as AMD [19] classified 24,553 malicious APK samples in 135 varieties among 71 malware families. The data description is presented in [21]. AndroZoo categorized 20,000 malware samples in 37 malware families [22]

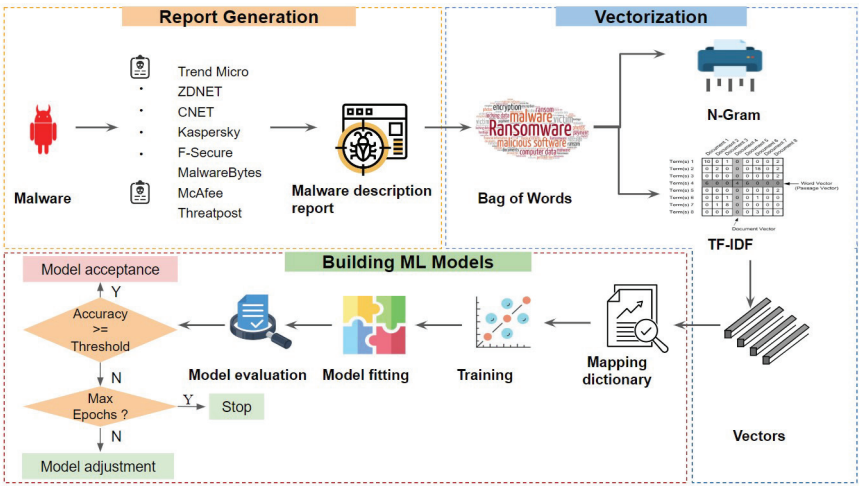


Figure 4 Proposed method of MalVulDroid.

and CICInvesAndMal2019 categorized the remaining 4,354 malware samples into 42 unique malware families [23]. These datasets are preferred over Malgenome and Drebin because these were gathered between 2010 and 2012 and is currently discontinued. For years it had been considered the de facto Android malware dataset and is included in newer datasets such as AMD.

The ground truth forms the training corpus consisting of malware and malware types and is obtained from cybersecurity blogs such as Trend micro,[1] Kaspersky,[2] F-secure,[3] Malware bytes,[4] CNet,[5] ZDNet,[6] Checkpoint,[7] and New Jersey Cybersecurity and Communications Integration Cell (NJCCIC).[8] We have listed nine different types of vulnerabilities from the National Vulnerability Database (NVD)[9] and CVEdetails[10] (c.f. Section 3) for mapping malware to exploited vulnerabilities. Algorithm 1 presents the generation and update of the malware description report.

Algorithm 1 Malware description report generation and update

Input: *mal_dataset*: known/detected malware information from cybersecurity solutions (CSS) as text file (.txt)
Output: *Malware description report* in CSV file (.csv)

```
while true, do
    if ∃ new_malware, then
        Add new_malware to mal_dataset;
        for M_i ∈ mal_dataset, do
            Report ← collectDescription (M);
            wordBag ← getwordBag (Report);
            saveReport (wordBag);
```

4.1.2 Vectorization

This section helps in answering RQ2. Previous techniques manually inspected the malware features from multiple reports. This process is not

[1] https://www.trendmicro.com/vinfo/dk/security/news/cybercrime-and-digital-threats
[2] https://www.kaspersky.co.in/resource-center/threats/mobile
[3] https://www.f-secure.com/en/home/products/mobile-security
[4] https://www.malwarebytes.com/mobile/
[5] https://www.cnet.com/news/android-malware-that-comes-preinstalled-are-a-massive-threat
[6] https://www.zdnet.com/topic/security/
[7] https://research.checkpoint.com/category/android-malware/
[8] https://www.cyber.nj.gov/threat-center/threat-profiles/android-malware-variants
[9] https://nvd.nist.gov/
[10] https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224

scalable when there are a large number of reports and is very time-consuming. Therefore, features (words) should be represented automatically for mapping vulnerabilities. MalVulDroid employs the BoW [24], NLP model leveraged with n-gram generation, probability [25], and TF-IDF [26]. BoW is the simplest approach to convert text into structured features and maintains the count statistics of an individual word appearing in the document. This technique generates fixed-length vectors from the corpus using words' frequencies. The reports are formulated into features vectors, which are then used to build classification models.

1. $\boldsymbol{n}$**-Gram**: n-gram is a technique where ordering of words is taken into consideration. It represents a contiguous sequence of *n* words of a text corpus (malware description) and predicts the next possible word(s) in a sequence that appear(s) in the malware description corpus.
    a. $\boldsymbol{n}$**-Gram generator**: The value for n is dependent on the dataset type and problem domain and can be any positive integer. The different values of *n* ranging from 1 to 4 are chosen to analyze malware-vulnerability mapping. Possible *n*-gram sequences can be generated based on the given value of n. We consider only the unique set of n-gram sequences.
    b. $\boldsymbol{n}$**-Gram probability scoring:** n-gram sequences generated in step 1 are fed to the probability scoring component. The Markov assumption is applied, which considers the word length of immediate $n-1$ words only.

    n-Gram sequences considered for this study are vulnerability classes such as steal information, gain privileges, DoS, bypass information, etc. The probability scores for each n-gram sequence are then stored in a feature database.
2. **TF-IDF**: TF-IDF is defined as the product of term frequency (TF) and inverse document frequency (IDF). Term frequency is the number of particular n-gram sequences occurring in a malware description.

Algorithm 2 presents the mapping of malware to vulnerabilities.

4.1.3 Building ML models

The MalVulDroid framework uses supervised ML [27] to build malware to vulnerability mapping models as explained in Algorithm 3. Supervised classification algorithms cannot be used directly on the multi-labelled dataset. Therefore, target variable is transformed using the binary relevance (BR)

Algorithm 2 Mapping algorithm

Input: *Report*: malware report
Output: M: Mapping Decision

$M_{malware} \leftarrow M^1_{malware}$
$V_{vulnerability} \leftarrow \{V^2_{vulnerabilityI}, \ldots, V^Y_{vulnerabilityK}\}$
$x \leftarrow$ vectorize (*Report*);
malware_result $\leftarrow M_{malware}(x)$
if *malware*$_{result} < 0$, **then**
  malware_result;
  for $V_i \in V_{vulnerability}$, **do**
    map_result $\leftarrow V_{vulnerabilityK}(x)$
malware_result, *map_result*;

approach (a multi-label binarizer in this case). Furthermore, each ML algorithm classification performance is tuned up under an array of hyper-parameters (different for each ML algorithm such as the learning rate and momentum of the neural network, training set size, choice of kernel in SVM, etc.), which is done automatically.

Different ML classifiers used in MalVulDroid are MLP, SVM, RIDOR, and PART. These ML classifiers are preferred over other classifiers because they have presented promising results in the past researches [28].

Notations

$Y = \{Y_{build}, Y_{test}\}$: Global dataset '$Y$' is used to build and report the performance of MalVulDroid in the various tasks. Y_{build} is used to train and tune the hyper-parameters of MalVulDroid models. Y_{test} is a test set, which is used to measure the final performance of MalVulDroid classifiers. Stratified random split on Y is used to divide Y_{build} (70%) and Y_{test} (30%).

$Y_{build} = \{Y_{build}, Y_{test}\}$: Y_{build} consists of training set and validation set and is used to build models of MalVulDroid.

$Y_{train} = \{(y_0, z_0), (y_1, z_1), \ldots, (y_t, z_t)\}$: Y_{train} is the training dataset of MalVulDroid ML models.

$Y_{valid} = \{(y_0, z_0), (y_1, z_1), \ldots, (y_v, z_v)\}$: Y_{valid} is the validation set used to tune the trained models. Hyper-parameters are tuned to achieve the best scores on validation set.

(y_i, z_i): A single record in Y is composed of y_i and its label $z_i \in \{V_1, V_2, \ldots, V_n\}$, where z_i is multi-label with the set of vulnerabilities.

$Y_{test} = \{(y_0, z_0), (y_1, z_1), \ldots, (y_x, z_x)\}$: Y_{test} is used to calculate the final performance.

Subsequently, optimum decision thresholds for each model are determined using its performance on Y_{valid}. Finally, tuples of optimum ML models are obtained as $Op = \{< ml_0, u_0, p_0 >, < ml_1, u_1, p_1 >, \ldots, < ml_a, u_a, p_a >$, where a is the number of classification algorithms explored. A tuple $< ml_i, u_i, p_i >$ specifies the optimum hyper-parameters p_i and decision threshold u_i for ML classification algorithm ml_i.

Algorithm 3 Building machine learning models

Input: Y_{build}: Building set
Output: Op: Optimal models

```
Y_train, Y_valid ← Y_build
V_vulnerability ← {V^2_vulnerabilityI, ..., V^Y_vulnerabilityK}
for ml ∈ MLalgorithms, do
    score ← 0
    for p in ml.p_array, do
        model ← train (algo, Y_train, p);
        t, u ← validate (model, Y_valid);
        if t > score, then
            mla ← < ml, u, p >;
    Op.add (mla);
Op
```

5 Evaluation and Results

We now answer RQ3 and exhibit the evaluation results of MalVulDroid. The malware types are mapped to numerous vulnerabilities in Table 2. Here, bit 1 denotes the corresponding vulnerability, whereas bit 0 otherwise. Table 3 depicts the mapping of malware families to vulnerabilities. Table 3 is only presented partially owing to space limitations. The full table can be accessed through the link.[11]

5.1 Accuracy Evaluation for n-gram Probabilities

A 10-fold cross-validation technique is used to evaluate ML classifiers in MalVulDroid. Accuracies of different ML classifiers are evaluated based on different n values ranging from 1 to 4. Table 4 presents the average accuracies of ML classifiers based on n-gram TF-IDF for mapping malware type to vulnerabilities and malware families to vulnerabilities.

[11] https://docs.google.com/spreadsheets/d/1gocjQfJ8Ukryif3rCZHUuiWa1ONkLtuOIxXVidtzKvo/edit?usp=sharing

Table 2 Mapping malware type to vulnerabilities

Malware Type	Code Execution	DoS	Overflow	Gain Information	Memory Corruption	Bypass Something	Gain Privileges	SQL Injection	Directory Traversal
Hackertool	0	0	0	1	1	1	1	0	0
Backdoor	1	1	1	1	0	1	1	0	0
Adware	1	1	0	1	0	1	1	0	0
Ransom-ware	1	1	0	1	1	1	1	0	0
Trojan	1	1	1	1	0	1	1	0	0
Trojan-Banker	0	0	0	1	0	1	1	1	0
Trojan-Dropper	1	1	0	1	0	0	1	0	1
Trojan-SMS	0	1	0	1	0	1	1	0	0
Trojan-Spy	1	0	1	1	0	1	1	0	0
Trojan-Clicker	1	0	0	1	0	1	0	0	0

Table 3 Mapping malware families to vulnerabilities

Malware Type	Malware Family	Code Execution	DoS	Overflow	Gain Information	Memory Corruption	Bypass Something	Gain Privileges	SQL Injection	Directory Traversal
Hackertool	Lotoor	0	0	0	0	1	1	0	0	0
Backdoor	Androrat	0	1	0	1	0	1	1	0	0
Backdoor	Dendroid	1	0	0	1	0	1	0	0	0
Adware	AdDown	1	0	0	1	0	1	1	0	0
Adware	Chamois	1	0	0	1	0	0	0	0	0
Ransom-ware	Fusob	1	0	0	1	0	0	1	1	0
Trojan-Spy	Triout	0	0	0	1	0	0	0	0	0
Trojan-Dropper	Boqx	0	0	0	0	0	0	0	0	1
Trojan-SMS	Gumen	0	0	0	1	0	0	1	0	0
Trojan	Xiny	0	1	0	1	0	0	1	0	0
.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	
.	.	.	.	.	.	.	.	.	.	.

Table 4 Accuracies of ML algorithms for n-gram probabilities

	n-Gram Class Intervals			
ML Algorithms	1	2	3	4
SVM	93.45	89.21	88.14	85.71
MLP	98.04	93.56	91.17	88.70
PART	95.72	91.14	89.26	86.81
RIDOR	96.81	92.28	91.52	87.13

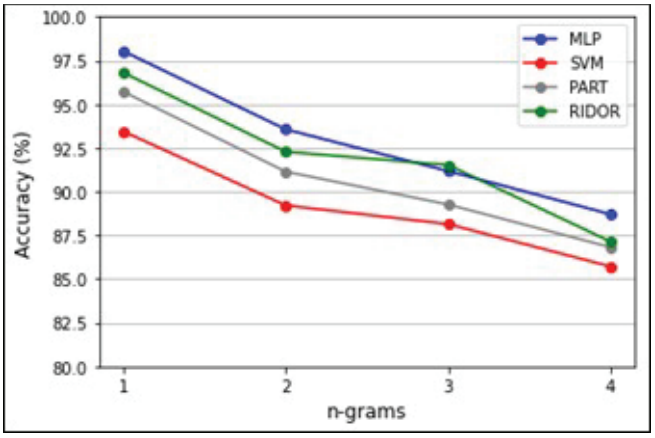


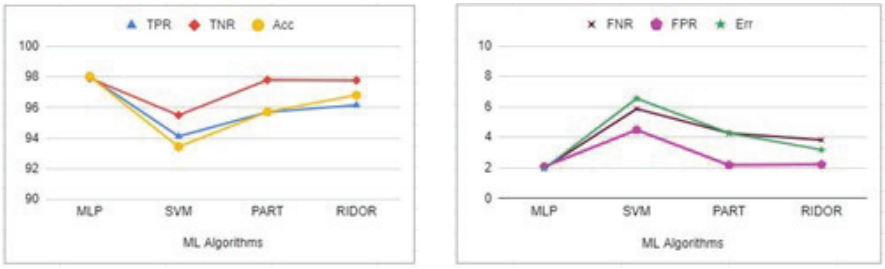
Figure 5 Accuracies of ML algorithms for n-gram probabilities.

MLP achieves the highest accuracy of 98.04% at $n = 1$. The second-best performance is shown by RIDOR with 96.81% accuracy at $n = 1$. It is seen that the accuracy of ML classifiers decreases with an increase in the value of n, as shown in Figure 5.

5.2 Comparison of ML Algorithms Based on Different Metrics

ML models used in MalVulDroid are compared on the basis of various metrics such as true positive rate (TPR), true negative rate (TNR), false positive rate (FPR), false negative rate (FNR), accuracy (Acc) and error (Err) [29] as shown in Figure 6.

Evidently, Figure 6(a) shows that MLP has a reasonable TPR, TNR, and accuracy in comparison to the other ML algorithms. However, SVM shows the highest error rate followed by PART and RIDOR, as shown in Figure 6(b).



(a) TPR, TNR, Accuracy (b) FNR, FPR, Error

Figure 6 Comparison of ML algorithms based on different metrics.

5.3 Performance of MalVulDroid

The effectiveness of MalVulDroid is evaluated for different ML algorithms with hyper-parameters for setting neural network such as the learning rate for training MLP (0.3 in this case), momentum (0.2), number of hidden layers (2) and number of epochs (450). These hyper-parameters are verified for multiple iterations where the reasonable results are achieved.

5.3.1 F1-score

In most cases, MalVulDroid achieves an F1-score of 90%, as shown in Table 5. A baseline is a basic model that does not require much expertise and provides reasonable results. A baseline model is based on the heuristics embedded in the model and, therefore, misses out a lot of structural information. A baseline model can be tuned with the hyper-parameters to improve the results. Ensemble models are built by combining multiple base learners using different strategies (averaging or voting) to reasonably improve the performance considerably.

It can be seen from Table 4 that all the ensemble models show reasonable F1-scores, and MLP shows the highest F1-scores.

5.3.2 Recall and precision

It is evident from Figure 7 that MLP has higher precision and recall than other ML models.

5.3.3 Accuracy

Accuracy of different ML models across the training-set size (number of behavioural reports) and test-size for MalVulDroid is also investigated. Figure 8 illustrates the results of our analysis for different ML models.

Table 5 F1-score (%) of ML algorithms after hyperparameter tuning

		Baseline	Tuned	Ensemble
SVM	Mean	86.666	90.697	94.428
	SD	7.354	7.555	6.867
	Min	69.629	73.854	77.685
	25%	83.807	89.075	91.375
	50%	85.723	89.801	96.726
	75%	92.164	96.835	92.164
	Max	93.209	98.225	93.209
MLP	Mean	89.761	94.091	97.664
	SD	5.978	5.719	5.200
	Min	78.006	82.341	85.354
	25%	85.930	89.960	98.135
	50%	91.971	97.209	99.932
	75%	92.823	97.036	92.823
	Max	93.512	97.983	93.512
PART	Mean	89.025	93.558	97.607
	SD	5.654	5.096	5.517
	Min	78.688	83.908	85.906
	25%	85.427	89.693	96.737
	50%	91.679	96.507	91.679
	75%	92.797	97.475	92.797
	Max	93.306	98.042	93.306
RIDOR	Mean	81.116	85.390	88.890
	SD	7.091	6.621	6.362
	Min	70.004	74.459	77.691
	25%	76.019	81.052	85.456
	50%	84.458	89.549	91.943
	75%	86.884	89.125	93.482
	Max	86.263	90.416	94.128

It is evident that MalVulDroid achieved high scores even with relatively small training samples; however, SVM achieved 55% accuracy (lowest) MLP achieved 72% (highest) accuracy with 100 training samples. Here, the problem of over-fitting is solved using principal component analysis (PCA) [30], a dimensionality reduction technique.

5.4 Efficiency of MalVulDroid

The efficiency of MalVulDroid can be assessed with respect to mean running time for examining behavioural reports. The runtime of MalVulDroid

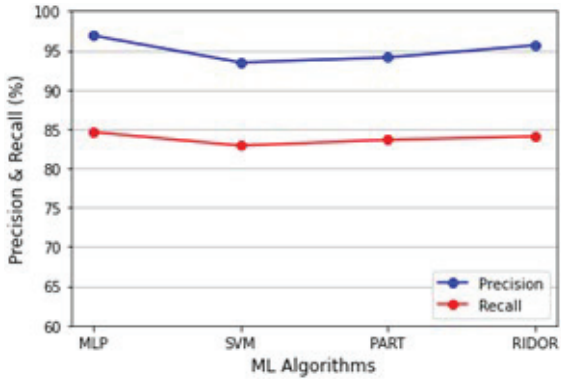
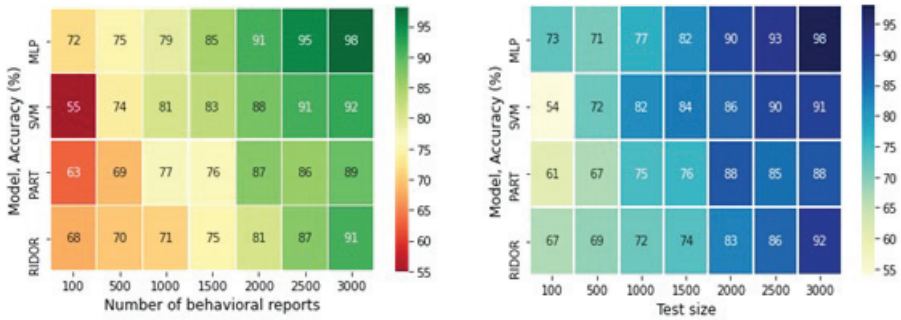


Figure 7 Recall and precision of MalVulDroid.



(a) Training sample (b) Testing sample

Figure 8 Accuracy of ML models for training and testing samples.

comprises pre-processing time and mapping time (training time + prediction time). Pre-processing time remains same across all ML models; however, mapping time varies. Figure 9 clearly shows that MLP is more expensive in terms of prediction time and training time. The efficiency of MalVulDroid is assessed on an Intel (R) Core(TM) i5-5300U CPU @ 2.30GHz (16.0 GB RAM) machine with quad-core processor.

6 Generalized Insights

It can be comprehended from the results (obtained in Section 5) that MalVulDroid shows encouraging results for mapping malware-to-vulnerabilities and single malware can be exploited by multiple vulnerabilities. However, there can be many variants in a single malware family. For example, AdDown

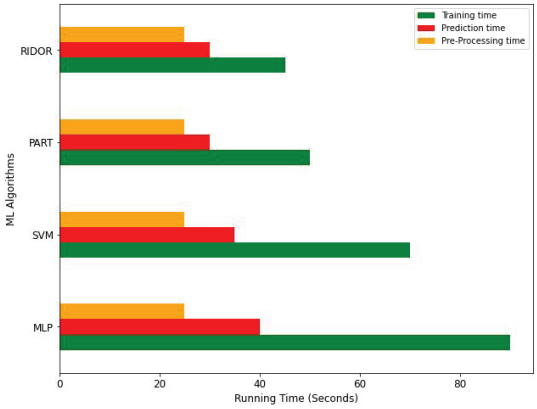


Figure 9 Average runtime of MalVulDroid.

adware can have multiple variants such as Joymobile, Nativemob, and Xavier. These malware variants are mapped to the majorly exploited vulnerabilities. In a broader view, malware types such as ransomware, adware, spyware, trojan, etc. are mapped to the exploited vulnerabilities.

There is a dearth of literature in this domain and, hence, this study is the first of its kind, and it cannot be compared with the existing literature. The closest work to our study is of [31], where the authors mapped unintentional Android bugs to security vulnerabilities. Our methodology is different from [31] in terms of text processing techniques using ML classifiers and malware dataset.

MalVulDroid is conceptualized on the set theory, and to a large extent it maps malware to vulnerabilities. This study presents novelty in the area of Android security where NLP along with ML techniques are leveraged for mapping malware and vulnerabilities. The MalVulDroid framework can be put to use by Android developers and researchers, where the security issues can be resolved during early phases of application development.

7 Conclusion

The spur in Android malware has caused serious concern among security investigators. The root cause of malware attack can be known by investigating the malware behaviour. This study provides a malware to vulnerability mapping using a novel "MalVulDroid" framework. MalVulDroid models the malware description reports using BoW and leverages NLP and ML

techniques to build ensemble ML models. MalVulDroid achieves an accuracy of 98.04% using MLP. The results seem encouraging and, thus, it is crucial to further explore this wide yet nascent domain.

The quality of malware description reports determines the performance of MalVulDroid. Currently, MalVulDroid is incompetent to assess the quality of the description report and hence investigators can choose non-standard (from unknown sources) malware reports. This issue can be taken for future work. Deep learning and transfer learning can be used in future to achieve more promising results.

References

[1] Check Point Software Technologies Ltd., Report on Insights on Emerging Mobile Threats, 2021.
[2] Skybox Security, Report on Vulnerability and Threat Trends, 2021.
[3] McAfee, Report on Mobile Threat, 2021.
[4] U. Ahmed, J.C.W. Lin, and G. Srivastava, G., 'Mitigating adversarial evasion attacks of ransomware using ensemble learning', *Computers and Electrical Engineering*, vol. 100, p. 107903, 2022.
[5] D. Ö. Şahın, S. Akleylek, and E. Kiliç, 'LinRegDroid: Detection of Android malware using multiple linear regression models-based classifiers', *IEEE Access*, vol. 10, pp. 14246–14259, 2022.
[6] P.N. Yeboah and H.B. Baz Musah, 'NLP technique for malware detection using 1D CNN fusion model', *Security and Communication Networks*, 2022.
[7] M. Conti, P. Vinod, and A. Vitella, 'Obfuscation detection in Android applications using deep learning', *Journal of Information Security and Applications*, vol. 70, p. 103311, 2022.
[8] N. Zhang, Y.A. Tan, C. Yang, and Y. Li, 'Deep learning feature exploration for android malware detection', *Applied Soft Computing*, vol. 102, p. 107069, 2021.
[9] E.B. Karbab and M. Debbabi, 'Maldy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports', *Digital Investigation*, vol. 28, pp. S77–S87, 2019.
[10] N. Zhang, J. Xue, Y. Ma, R. Zhang, T. Liang, and Y.A. Tan, 'Hybrid sequence-based Android malware detection using natural language processing', *International Journal of Intelligent Systems,* vol. 36, no. 10, pp. 5770–5784, 2021.

[11] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, M. Conti, and Shandong, 'Detecting Android malware leveraging text semantics of network flows', *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1096–1109, 2017.

[12] G. Peynirci, M. Eminağaoğlu, and K. Karabulut, 'Feature selection for malware detection on the Android platform based on differences of IDF values', *Journal of Computer Science and Technology,* vol. 35, no. 4, pp. 946–962, 2020.

[13] M. Kinkead, S. Millar, N. McLaughlin, and P. O'Kane, 'Towards explainable CNNs for Android malware detection', *Procedia Computer Science*, vol. 184, pp. 959–965, 2021.

[14] S. I. Imtiaz, S. ur Rehman, A.R. Javed, Z. Jalil, X. Liu, and W.S. Alnumay, 'DeepAMD: Detection and identification of Android malware using high-efficient deep artificial neural network', *Future Generation Computer Systems*, vol. 115, pp. 844–856, 2021.

[15] ZDNet, 'Three quarters of mobile apps have this security vulnerability that could put your personal data at risk', 2019.

[16] R. Surendran, T. Thomas, and S. Emmanuel, 'GSDroid: Graph signal based compact feature representation for Android malware detection', *Expert Systems with Applications*, vol. 159, p. 113581, 2020.

[17] S. Garg and N. Baliyan, 'Comparative analysis of Android and iOS from security viewpoint', *Computer Science Review*, vol. 40, p. 100372, 2021.

[18] D. Costa, F. Handrick, I. Medeiros, M. Thales, J. Victor da Silva, I. Lorraine da Silva, and M. Ribeiro, 'Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification', *Journal of Systems and Software*, vol. 183, p. 111092, 2022.

[19] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, 'Deep ground truth analysis of current Android malware', In *Int. Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 10327, pp. 252–276, 2017.

[20] S. Garg and N. Baliyan, 'Android security assessment: A review, taxonomy and research gap study', *Computers & Security*, vol. 100, p. 102087, 2020.

[21] S. Garg and N. Baliyan, 'Data on vulnerability detection in Android', *Data in Brief*, vol. 22, pp. 1081–1087, 2019.

[22] K. Allix, T.F. Bissyandé, J. Klein, and Y.L. Traon, 'AndroZoo: Collecting millions of Android apps for the research community', In *Working Conference on Mining Software Repositories (MSR)*, pp. 468–471, 2016.

[23] T. Taheri, A.F. Kadir, and A.H. Lashkari, 'Extensible Android malware detection and family classification using network-flows and API-Calls', In *Int. Conf. on Security Technology (ICCST)*, pp. 1–8, 2019.
[24] Z.S. Harris, 'Distributional structure', *Word*, vol. 10, no. 2–3, pp. 146–162, 1954.
[25] D. Jurafsky and J.H. Martin, *Speech and Language Processing*, Pearson Education India, 2000.
[26] H.P. Luhn, 'A statistical approach to mechanized encoding and searching of literary information', *IBM Journal of Research and Development*, vol. 1, no. 4, pp. 309–317, 1957.
[27] S. Noekhah, N.b. Salim, and N.H. Zakaria, 'Opinion spam detection: Using multi-iterative graph-based model', *Information Processing & Management*, vol. 57, no. 1, p. 102140, 2020.
[28] S. Garg and N. Baliyan, 'A novel parallel classifier scheme for vulnerability detection in Android', *Computers & Electrical Engineering*, vol. 77, pp. 12–26, 2019.
[29] T. Mitchell, *Machine Learning,* Pittsburgh: McGraw-Hill Education, 1997.
[30] T. Chai, S. Prasad and S. Wang, 'Boosting palmprint identification with gender information using DeepNet'. *Future Generation Computer Systems*, pp. 41–53, 2019.
[31] G. Bajwa, M, Fazeen, R. Dantu, and S. Tanpure, "Unintentional bugs to vulnerability mapping in android applications." In *2015 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 176–178, IEEE.

Biographies

Shivi Garg is an Assistant Professor, Department of Computer Engineering, J.C. Bose University of Science & Technology, YMCA, Faridabad. She has

attained Doctor of Philosophy from Information Technology Department, Indira Gandhi Delhi Technical University for Women, (IGDTUW), Delhi, India in December 2021. Her Thesis title is Design and Analysis of Mobile Application Vulnerabilities. She is also a post graduate in Information security from Delhi Technological University (DTU) Delhi, India. She has teaching and research experience since August 2016. Her research interests include-Information Security, mobile security, cyber security, and Machine learning. Her publication and other details can be found at: https://sites.google.com/view/shivigarg/home.

Niyati Baliyan is an Assistant Professor, Department of Computer Engineering, National Institute of Technology Kurukshetra, Haryana. She has attained Doctor of Philosophy from Computer Science Department, Indian Institute of Technology (IIT) Roorkee, India. Her thesis title was "Quality Assessment of Semantic Web based Applications". She also has a Post Graduate Certificate in Information Technology from Sheffield Hallam University, Sheffield, U.K. Dr. Niyati obtained Chancellor's Gold Medal for being University topper during post graduate studies at Gautam Buddha University. She is co-author of "Semantic Web Based Systems: Quality Assessment Models, Springer Briefs in Computer Science", 2018. Her research interests include-Knowledge Engineering, Machine Learning, Healthcare analytics, Recommender systems, Information Security, and Natural Language Processing. Her publication and other details can be found at: https://sites.google.com/site/niyatibaliyan.