IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 2, 2019, accepted April 26, 2019, date of publication May 1, 2019, date of current version June 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2914311

Predicting the Impact of Android Malicious Samples via Machine Learning

JUNYANG QIU[1], WEI LUO[1], LEI PAN[1], (Member, IEEE), YONGHANG TAI[2], JUN ZHANG[3], (Senior Member, IEEE), AND YANG XIANG[3], (Senior Member, IEEE)

[1]School of Information Technology, Deakin University, Geelong, VIC 3216, Australia
[2]School of Physics and Electronic Information, Yunnan Normal University, Kunming 650500, China
[3]School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia

Corresponding author: Yonghang Tai (taiyonghang@ynnu.edu.cn)

ABSTRACT Recently, Android malicious samples threaten billions of mobile end users' security or privacy. The community researchers have designed many methods to automatically and accurately identify Android malware samples. However, the rapid increase of Android malicious samples outpowers the capabilities of traditional Android malware detectors and classifiers with respect to the cyber security risk management needs. It is important to identify the small proportion of Android malicious samples that may produce high cyber-security or privacy impact. In this paper, we propose a light-weight solution to automatically identify the Android malicious samples with high security and privacy impact. We manually check a number of Android malware families and corresponding security incidents and define two impact metrics for Android malicious samples. Our investigation results in a new Android malware dataset with impact ground truth (low impact or high impact). This new dataset is employed to empirically investigate the intrinsic characteristics of low-impact as well as high-impact malicious samples. To characterize and capture Android malicious samples' pattern, reverse engineering is performed to extract semantic features to represent malicious samples. The leveraged features are parsed from both the *AndroidManifest.xml* files as well as the disassembled binary *classes.dex* codes. Then, the extracted features are embedded into numerical vectors. Furthermore, we train highly accurate support vector machine and deep neural network classifiers to categorize the candidate Android malicious samples into low impact or high impact. The empirical results validate the effectiveness of our designed light-weight solution. This method can be further utilized for identifying those high-impact Android malicious samples in the wild.

INDEX TERMS Android malware, deep neural network, high impact malicious samples, low impact malicious samples, static analysis, SVM.

I. INTRODUCTION

Lately Android OS is the number one system globally, with approximately 86.1% of the whole global mobile phones market share in the Q1 of 2017 [38]. Meanwhile, the prevalence of Android operating system, combining with its openness trait, has caused the number of Android malware skyrocketed in both the official markets (e.g., Google Play[1]) and third-party (WanDouJia[2]) Android application markets. In the first quarter of 2017 alone, over 750,000 new Android malware samples were identified by experts in G DATA security [27]. That is, approximately a new Android malware was identified in about 10 seconds. The growing momentum Android malware has brought great security and privacy risks to end mobile users and mobile service providers.

To maintain a healthy and clean ecosystem for Android applications, the research communities and security vendors have designed many techniques to identify and prevent Android malicious samples, e.g., software engineering analysis techniques [3], [6] and machine learning based techniques [1], [2]. Generally, the current malware analysis techniques can be divided into 3 categories: dynamic analysis, static analysis, and hybrid analysis (combining dynamic analysis and static analysis). Static analysis is performed without actually executing the applications, in contrast with dynamic

The associate editor coordinating the review of this manuscript and approving it for publication was Khalid Aamir.

[1]https://play.google.com/store
[2]https://www.wandoujia.com/

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

analysis which is performed during the run-time of the applications. Hybrid analysis will perform both static analysis and dynamic analysis for the candidate applications. The majority of the existing works focused on determining whether the candidate Android applications are benign or malicious. However, along with the tremendous number of Android malware being detected every day, only a small fraction of them will incur serious damage to mobile users or organizations. Based on this observation, we infer the importance of predicting the impact of malware (low impact or high impact). If the malware is with high impact, we must defend it with high priority to be aware of its potential damage as much as possible. In this paper, we propose a new research problem: How to automatically predict the impact of the detected Android mawlare?

However, there are few work on predicting the impact of the detected Android malware so far. Most of the existing works concerned only the identification of Android malware or classification of malware into the specific families. To investigate the solution to the proposed research question, we should address three critical challenges. Firstly we need to define the impact metrics to validate the impact of malicious Android samples. Besides, another challenge is the lack of an Android malware dataset with impact ground truth in terms of impact. The third challenge is about how to effectively and efficiently make the impact prediction for the newly encountered Android malware. To address the listed challenges, we make four main contributions in this work:

1. We raise a new research problem: how to automatically and accurately make the impact predict for the identified Android malicious samples?
2. We define two impact metrics (the size of affected Android end devices, in short for ***SOD***, and the size of infected countries, also called ***SOC***) to characterize the impact of Android malicious samples.
3. We construct a new Android malware dataset (also called Mal_Impact Dataset) containing low impact and high impact malicious samples. To facilitate the following similar works related to the impact prediction of Android malicious samples, we will release our constructed malware impact dataset to the public.
4. We design a light-weight solution to automatically and accurately make the impact prediction for the identified Android malicious samples. Semantic features are decoded from *AndroidManifest.xml* files and disassembled binary *classes.dex* codes to represent the intrinsic characteristics of both low and high impact Android malicious samples. Then we train the Support Vector Machine and Deep Neural Network models to categorize the candidate Android malicious samples into low or high impact. Our empirical studies validate the effectiveness and feasibility of our designed solution.

The rest of the paper is organized as follows: Section 2 reviews the related work of Android malware analysis. Our designed methodology is introduced in Section 3. In Section 4, we give the experimental settings, evaluation metrics and detailed results. In Section 5, we present the limitations of this work. Section 6 concludes the paper and gives the following research directions.

II. RELATED WORK

The analysis and identification of Android malware has been an active research area in the recent years. The research communities have proposed many techniques to deal with the growing number and evolution of Android malware. Android malware detection is surveyed in [40].

A. STATIC ANALYSIS TECHNIQUES

Static analysis is conducted through reverse engineering the binary codes of the candidate Android applications without actual execution. We can further divided static analysis into three categories: The first category is called signature related methods. These methods focus on producing a robust malware signature based on the particular strings or semantic code patterns in the applications' code packages [16]. Then in the malware detection process, if the candidate application's signature is similar to one of the signatures in the malware's signature database, then it is regarded as a malware. DroidAnalytics [50], AndroSimilar [15] and ASTROID [17] are three signature related examples. The second category is permission related methods which collect the requested permissions declared in *AndroidManifest.xml* files to detect potential malware. The permission related methods include [36], [43] and [24]. The final category is related to binary Dalvik bytecode *classes.dex* analysis. This category of methods will obtain the source code through disassembling the binary *classes.dex* bytecode, such as [4], [12], [14], [29]. When the binary *classes.dex* codes have been disassembled into the corresponding source codes, the suspicious API calls, CFG (control flow graphs), or FCG (function call graphs) can be revealed to represent the discriminate patterns between malicious Android samples and benign samples [1], [42], [46].

B. DYNAMIC ANALYSIS TECHNIQUES

Dynamic analysis traces and analyses the malicious behaviors and system calls of Android applications during execution in the virtual environments. TaintDroid [13] was the pioneer dynamic analysis system. It was efficient in dynamic taint tracking. Another popular dynamic analysis system was DroidScope. DroidScope can rebuild semantic information seamlessly and simultaneously on the Java Dalvik level as well as Linux OS level [45]. Zhang *et al.* [49] created a system called VetDroid, which can rebuild malicious and sensitive behaviors in Android applications. Another dynamic framework AppsPlayground was implemented to analyze Android applications automatically. AppsPlayground combined many automatic exploration and detection strategies (e.g., kernel level system call monitoring, taint tracing tool [13]) for large-scale Android applications analysis [34].

An alternative dynamic solution was proposed by Georgios Portokalidis *et al.* to perform the detection task on remote

TABLE 1. The differences between the state-of-the-art works and our work.

Research work	Research problem	Feature sources	Research approach
DroidAPIMiner [1]	Malware detection	Frequency of API calls	KNN, SVM, Decision Tree
Drebin [2]	Malware detection	*AndroidManifest.xml* and disassembled code based features	Linear SVM
HinDroid [23]	Malware detection	Relationships between APIs and Apps	Multi-Kernel Learning
MaMaDroid [28]	Malware detection	Markov chains of sequences of API calls	KNN, SVM, Random Forest
DroidSIFT [48]	Family classification	Weighted contextual API dependency graphs	Multi-label classifier
ASTROID [17]	Family classification	Maximally suspicious common subgraph	Subgraph matching
This work	Malware impact prediction	*AndroidManifest.xml* and disassembled code based features	SVM, Deep Neural Network

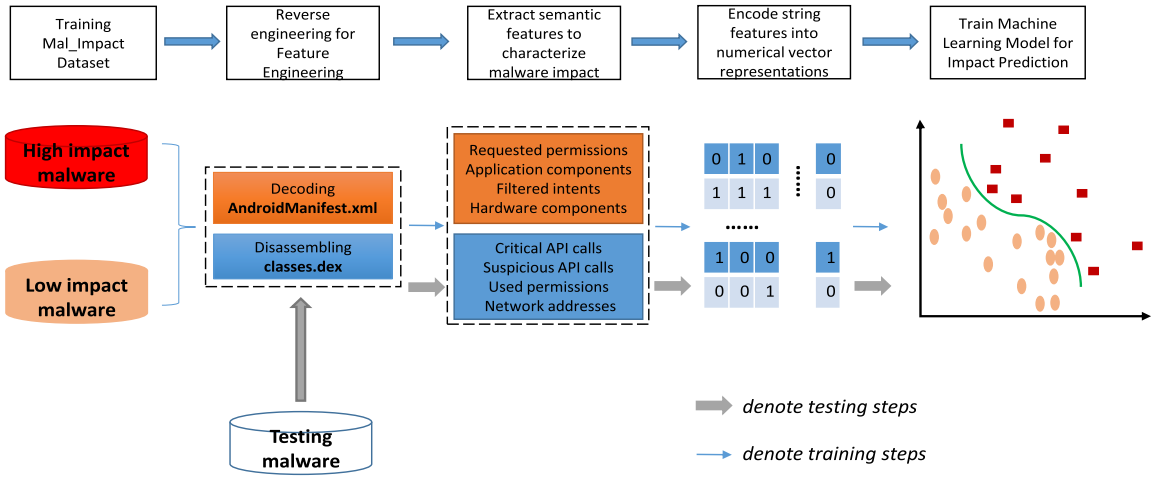


FIGURE 1. The mainly flowchart of the proposed solution for automatically Android malware impact prediction.

security servers in the cloud where execution of the applications on the device is mirrored in virtual machine environments. The remote servers are not subject to the same constraints with the devices, thus it is possible to employ multiple detection techniques in parallel [31].

C. MACHINE LEARNING BASED TECHNIQUES

As we know, machine learning technique has been used in many cyber security related areas [25], [26], [39], [44], [47], [41]. In the malware analysis area, it is expensive and difficult for security experts to manually generating and updating the malware detection patterns (or signatures) for Android malware. Inspired by the efficiency and generalization ability of machine learning technique, many machine learning involved methods have been designed to automatically detect or further classify Android malware samples. Drebin is a traditional and notable machine learning method which can detect Android malicious samples directly on the mobile devices [2]. Drebin employed semantic features parsed from both *AndroidManifest.xml* files and disassembled binary *classes.dex* files. In addition, the papers [1], [7], [8], [18], [43], and [32] all used machine learning related methods. In summary, the key challenge of machine learning related methods is how to obtain the robust and informative features to represent Android malware samples. The semantic features may be directly derived from statistical features (for example, API calls, requested or used permissions [2]), features of a tree structure (for example, abstract syntax tree) or features of a graph structure (for example, API call graphs (ACG) [23], data flow graphs (DFG) or control flow graphs (CFG)).

Table 1 compares our research work with three similar state-of-the-art works, including Drebin [2], DroidAPIMiner [1], HinDroid [23] and MaMaDroid [28].

III. METHODOLOGY

In this part, we give the detailed introduction of our proposed framework in Fig. 1.

A. OVERVIEW

The proposed framework consists of three phases. Firstly, the training Android malware samples will be collected with low impact or high impact ground truth. Secondly, based on the collected malware applications, reverse engineering will be conducted to extract features from the disassembled binary *classes.dex* codes and from *AndroidManifest.xml*. These extracted semantic features will be employed to represent the impact characteristics of each Android malware sample. Then the extracted string features will be encoded into numerical vector representations. Finally, with the vector representations, we train the impact prediction model to categorize the candidate Android malware samples into low impact or high impact. The details of each step will be presented in the following sections.

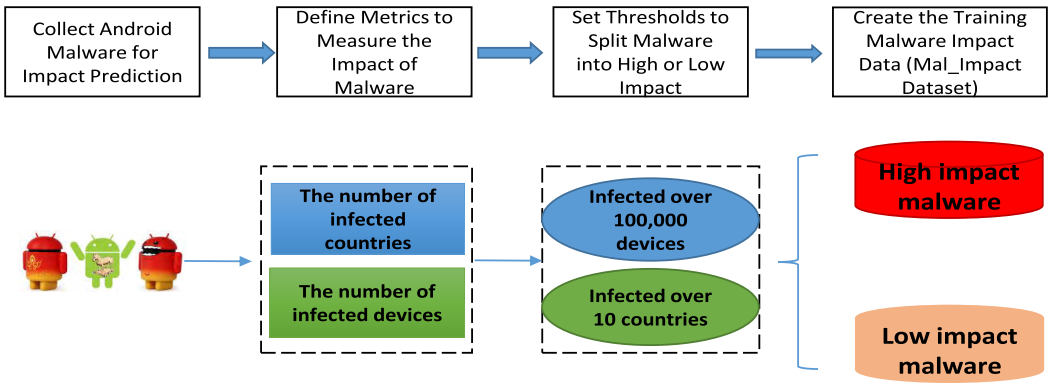


FIGURE 2. The steps of how to construct the Android malware dataset with impact ground truth (Mal_Impact dataset).

B. DATA COLLECTION

The critical challenge of the proposed research problem origins from the absence of an Android malware dataset with impact ground truth available. Thus, to further validate our proposed solution to the research problem, we have to construct a dataset with high impact or low impact ground truth. Fig. 2 presents the steps of how to construct the Android malware dataset together with impact ground truth (called Mal_Impact Dataset, denoted as D).

In this work, the constructed Mal_Impact Dataset is built on three open sourced Android malware datasets with family labels:

Source_1: contagion mobile[3]

The contagion mobile website is actively open for sharing the malware dataset. Anyone can upload the mobile malware samples to Contagio mobile mini-dump through dropbox. And people can download any malware samples. Currently, contagion website contains malware samples with family labels from 2011 till now.

Source_2: Android malware samples[4]

The repository contains live Android malware applications. Anyone can upload and share the Android malware samples in this repository. At the time when this paper is written, this site contains about 144 Android malicious samples within 26 families.

Source_3: Android Malware Dataset[5]

Android Malware Dataset, also known as AMD. It is a well-labeled and deep-studied dataset. For each malware family, it provides detailed profile information. Lately, it contains approximately 24,553 samples from 71 Android malware families between 2010 and 2016. AMD provides a snapshot of the current Android malware landscape along with the detailed descriptions, results and reports of the malware's behaviors.

If an instance of malware appears in two or three sources, then we aggregate these malware samples together and remove the duplicates. The final malware dataset will be the union of the three malware sources as shown in Eq.(1).

$$Source\_1 \cup Source\_2 \cup Source\_3 \tag{1}$$

C. METRICS FOR MALWARE IMPACT

The next challenge is on how to define metrics to evaluate the Android malware impact. Theoretically, there are many metrics available that we can employ to scale the malware impact, for instances, the economic loss or the number of compromised devices resulted by the Android malicious samples. But in reality, it is difficult to collect all the precise metrics values due to various reasons, e.g., it is infeasible for individuals to access the accurate value of global economic loss. Thus, as a proof-of-concept, we define two metrics to evaluate the Android malware impact in this paper:

Impact Metrics_1: *The size of affected Android end devices that the malware family has resulted in, also named* ***SOD*** *for short.*

Impact Metrics_2: *The size of countries the candidate malware family has infected, also called* ***SOC*** *for short.*

Having defined the Android malware impact metrics, we should set the impact metrics threshold value to segregate low and high impact Android malicious samples. In this work, all the malicious samples within the candidate Android malware family are labeled as high impact malicious samples once one of the two setting conditions is met. Otherwise the malware samples within this Android malware family are then categorized to the low impact samples.

Condition 1: $\boldsymbol{SOD} \geq 1e5$, the specific malware family has affected over $1e5$ end devices.

Condition 2: $\boldsymbol{SOC} \geq 10$, the specific malware family has infected more than 10 countries.

D. SETUP THE GROUND TRUTH

To setup the ground truth, the best and most reliable way is the manual annotation at individual malware sample level. However, this is difficult, infeasible and resources consuming for us to setup the impact ground truth for many malicious samples. In addition, the following impact prediction model

[3] http://contagiominidump.blogspot.com.au/
[4] https://github.com/ashishb/android-malware
[5] http://amd.arguslab.org/

TABLE 2. The detailed introduction of the labeled Android malware data with high impact ground truth (*High_Mal*).

Time	Family class	Sample size	The size of infected countries	The size of compromised devices
2012	Plankton	25	Null	Over 5 million
2012	Carberp	3	Over 10	Null
2012	MMarketPay	14	Over 1	Over 100,000
2012	Counterclank	5	Null	5 million
2013	Opfake	10	97	over 1 million
2013	Zitmo	24	Over 2	Over 3.5 million
2014	Koler	69	30	Over 200,000
2015	Fakeinst	2172	Over 130	Over 5 million
2015	Simple Locker	173	Over 1	Over 150,000
2016	Copycat	9	Over 5	Over 14 million
2016	Fusob	1277	Over 100	Null
2016	Hummingbad	590	Over 100	85 million
2016	Pokemongo	1	3	Over 7.5 million
2016	Svpeng	34	Over 23	Over 318,000
2016	Ztorg	28	Null	Over 1 million
2016	Viking Horde	7	13	Over 100,000
2017	Chrysaor	3	Over 11	Null
2017	Judy	14	Null	Over 36 million
2017	Xavier	6	Over 6	Over 100 million

is trained without considering the family-boundary of the data. As a proof-of-concept, the ground truth (low impact or high impact) of Android malware samples are annotated on Android malware family-grained. The malicious samples within the high impact families are all annotated as high impact (denoted as *High_Mal*), while Android malicious samples within other Android malware families will be labeled with low impact ground truth (denoted as *Low_Mal*).

The baseline Android malware families listi (denoted as *Fam_list*) can be accessed from this website.[6] For each malware family f, we retrieve the values of the defined two impact metrics ***SOD*** and ***SOC*** from open-source information from various sources. In this work, we collect the related information from various sources, e.g., cyber-security news, white papers, or other related reports published by security vendors, or security-labs (for example, *CheckPoint*, *Symantec*, *Kaspersky* and *McAfee*).

Let's explain the ground truth annotation process through a case study. According to a published research report of *CheckPoint* in 2016, the Android malware family called *Copycat* has affected more than 14 million mobile devices, rooted about 8 million infected devices, earned about $1.5 million in false advertisement income for the hackers behind the malware campaign.[7] Therefore, all the malware samples within *Copycat* are annotated with high impact ground truth.

E. CONSTRUCT THE ANDROID MALWARE IMPACT DATASET

With the available Android malware data and the corresponding impact ground truth, we construct the final Android malware dataset with impact ground truth, called the Mal_Impact Dataset (D). The information about the collected high impact malware samples (*High_Mal*) is listed in Table 2, and the details of Android malicious samples (*Low_Mal*) with low impact ground truth can be found in Table 3.

TABLE 3. The detailed introduction of the annotated Android malware data with low impact (*Low_Mal*).

Time	Family class	Sample size	Time	Family class	Sample size
2010	FakePlayer	21	2011	DroidKungfu	546
2011	GingerMaster	128	2011	GoldDream	53
2012	Boxer	44	2012	FakeAngry	10
2012	FakeDoc	21	2012	FakeTimer	12
2012	Fjcon	16	2012	Lotoor	329
2012	MobileTX	17	2012	Nandrobox	76
2012	Penetho	18	2012	SmsZombie	9
2012	SpyBubble	10	2012	Steek	12
2012	UpdtKiller	24	2013	AndroidRat	45
2013	Bankun	70	2013	Boqx	215
2013	FakeUpdates	5	2013	Gumen	145
2013	Ksapp	36	2013	Kyview	175
2013	Lnk	5	2013	Monimob	203
2013	Mseg	235	2013	Mtk	67
2013	Obad	9	2013	SmsKey	165
2013	Spambot	15	2013	Stealer	25
2013	Tesbo	5	2013	Vidro	23
2013	Vmvol	13	2013	Winge	19
2014	Airpush	7840	2014	Andup	45
2014	Aples	20	2014	Cova	17
2014	Erop	46	2014	FakeAV	5
2014	Finspy	9	2014	Jiust	560
2014	Ramnit	8	2014	Univert	10
2014	Utchi	12	2015	BankBot	724
2015	Dowgin	3377	2015	Fobus	4
2015	Gorpo	37	2015	Kemoge	14
2015	Kuguo	1199	2015	Leech	127
2015	Mecor	1820	2015	Ogel	6
2015	Roop	48	2015	SlemBunk	174
2015	Youmi	1301	2016	RuMMS	311
2016	Triada	210			

F. REVERSE ENGINEERING, FEATURE EXTRACTION AND EMBEDDING

After the Mal_Impact Dataset D is constructed, the robust and informative semantic feature set F are decoded to capture the patterns for high impact malware data (*High_Mal*) and

[6] https://forensics.spreitzenbarth.de/android-malware/

[7] https://www.checkpoint.com/downloads/resources/copycat-research-report.pdf

TABLE 4. The detailed description of the extracted feature set *F* used to represent malicious samples.

Feature subset	Detailed information about the feature subset
Requested permissions (f_rp)	Android apps will request permissons for accessing critical resources during installation
Application components (f_ac)	Android apps can declare many components, for example, Service, Activity, Broadcastreceiver, ContentProvider
Filtered intents (f_fi)	The intent filters are used to appoints the operations it can perform and the data type it can manipulate
Hardware components (f_hc)	Apply specific hardware or a series of particular hardwares may imply potential security or privacy risks
Critical API calls (f_ca)	The critical calls reveal the critical capability of Android applications
Sensitive API calls (f_sa)	The sensitive API calls represent the potential malicious actions of malware
Used permissions (f_up)	The critical API calls will be used to decide and match the requested or indeed used permissions
Network addresses (f_na)	The network addresses appeared in the source codes can be related to botnet attacks or suspicious websites

low impact malware data (*Low_Mal*) in order to accurately separate them.

In this paper, we perform reverse engineering of malware samples to parse the features. Generally, the semantic features are parsed from the disassembled binary *classes.dex* codes and the *AndroidManifest.xml* files. Each Android application package must contain the *AndroidManifest.xml* file. This *AndroidManifest.xml* file contains the essential information of the application, such as the components, the requested permissions, hardwares and filtered intents of the application. We will extract these information to characterize the impact of malware samples. In the following experiment setup, such information are parsed as features to represent each Android malware sample.

Android application packages are usually implemented using Java programming language and then compiled into *classes.dex* bytecodes for its execution in the Dalvik virtual machine. The *classes.dex* bytecode contains the comprehensive semantic knowledge about the critical API calls and data access within an application. Besides, the *classes.dex* bytecode can be efficiently disassembled and parsed.

Table 4 presents the detailed information of the extracted features. The first 4 feature sets are captured based on the *AndroidManifest.xml* files, and the rest 4 are extracted based on the disassembled *classes.dex* codes.

In order to feed the training malware samples to the machine learning models (denoted as M), we encode the string features of each malware sample as a numerical vector. After the string features for the training malware samples have been extracted, then we map all these features to a joint feature vector space. In this feature vector space, each malware x is embedded into vector representations. We employ the one-hot-encoding strategy to represent each malicious sample as a vector. For each feature f_i, if it is presented in x, then the i-th dimension feature value is set to 1, otherwise the corresponding dimension feature value is 0. Iteratively, we map each Android malware into the joint feature vector space. In this space, the malware sharing similar impact characteristics (low impact or high impact) will lie close to each other, but low and high impact malicious samples are expected to be separated with long distances. In addition, due to the large number of API calls, permissions and other features, the one-hot-encoding embedding method will generate extremely high dimensional vector. In order to reduce the storage memory and improve the computation speed, the produced feature matrices of malware samples are stored in the Compressed Sparse Column (CSC) format.

G. IMPACT MODEL TRAIN AND PREDICTION

This phase is to learn the Android malware impact predict model M. The collected Android malware impact dataset Mal_Impact Dataset (D) is class-imbalanced, however, the imbalanced rate is not too high, less than 5.0. Thus we expect the traditional machine learning classifiers can work well in this scenario. In this work, we employ the classical Linear Support Vector Machine (SVM) [20] and Deep Neural Network (DNN) [5] to train the malware impact prediction models.

The Linear Support Vector Machine has been used in many cyber security related classification tasks. In addition, it is an inherent interpretable classifier which is suitable for feature explanation. We can easily obtain the core features which are helpful for us to figure out the core features in predicting the Android malware impact. The Deep Neural Network has been used to achieve many great successes in various applications, e.g., image recognition, machine translation or cyber security related areas [9], [22], [33], thus we design a specific Deep Neural Network to predict the impact of Android malware samples. As a nonlinear model, Deep Neural Network complements the Linear SVM to provide a comprehensive empirical evaluation.

According to previous academic papers and engineering applications, Linear SVM model is suitable for the small dataset classification scenarios, while DNNs can learn the complex and accurate patterns from large-scale dataset. In our work, we construct an initial and relatively small Android malicious samples impact dataset with impact ground truth. Thus, SVM is expected to perform better than DNN. However, if we can collect more malicious samples together with impact ground truth, then DNN will achieve the best performance in Android malicious samples' impact prediction task.

The SVM aims to learn an optimal hyperplane which separates the high impact and low impact malware with maximal margin as presented in Fig. 3. To determine the optimal hyperplane, the following objective function Eq.(2) should be minimized [10].

$$min\ \frac{1}{2}||w||^2 \quad s.t., y_i(w^T x_i + b) \geq 1,\ i = 1, \ldots, n \qquad (2)$$

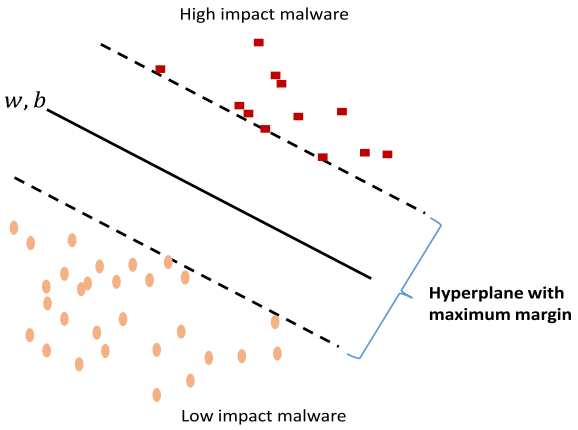


FIGURE 3. The schematic depiction of the SVM malware impact prediction model.

Once the SVM based impact prediction model M has been trained, it will be used to predict the testing malware into low or high impact by computing their distances to the separating hyperplane as shown in Eq.(3).

$$\begin{aligned} &< w, \phi(x_i + b) \geq +1 >, \quad \text{for } x_i \text{ is high impact} \\ &< w, \phi(x_i + b) \leq -1 >, \quad \text{for } x_i \text{ is low impact} \end{aligned} \tag{3}$$

The learned weights w can also be used to evaluate the importance of features to the construction of the classifier. In this work, we will use the learned feature weights to measure the contributions of features to the Android malware impact prediction model.

Deep Neural Networks have become an increasingly prevalent technique for malware analysis related applications. Their power of learning complex patterns and behaviors make Deep Neural Networks an appropriate technique for malware detection or classification. In this paper, we also create a Deep Neural Network structure to conduct the malware impact prediction task. To determine the structure and hyperparameters of Deep Neural Network, we have tried several combinations of the layer number and the related neurons number. The activation functions are selected from sigmoid, ReLU and tanh. The optimizer is selected from SGD, Adadelta, RMSprop and Adamax. We have also tried several loss functions, e.g., Mean_Squared_Error, Mean_Absolute_Error and Squared_Hinge. Based on the malware impact performance, we finally determined the structure of the network as presented in Fig. 4. The details of our designed Deep Neural Network can be found in Table 5.

For our designed Deep Neural Network, suppose that the $(l-1)$-th layer has m neurons, then the output of the l-th layer's j-th neuron can be formulated as the following equation [19]:

$$a_j^l = \sigma(z_j^l) = \sigma(\sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l) \tag{4}$$

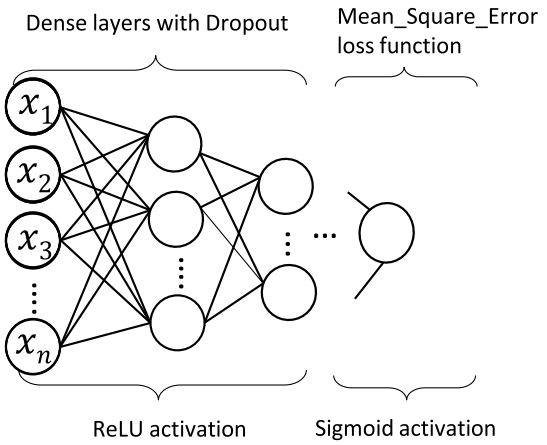


FIGURE 4. The architecture of the Deep Neural Network based malware impact prediction model.

TABLE 5. The architecture information of the deep neural network designed in our work.

#	Layer type	# of neuron	Activation
1	Dense	128	ReLU
2	Dense	64	ReLU
3	Dense	32	ReLU
4	Dense	16	ReLU
5	Dense	8	ReLU
6	Dense	1	Sigmoid

Here σ is the activation function, a_k^1 is the k-th input feature x_k for $l = 2$. The matrix formulation of Eq.(4) is:

$$a^l = \sigma(z^l) = \sigma(W^l a^{l-1} + b^l) \tag{5}$$

Then the final loss function can be represented as Eq.(6).

$$J(W, b, x, y) = \frac{1}{2} ||\sigma(W^L a^{L-1} + b^L) - y||_2^2 \tag{6}$$

During the learning of W and b, we can compute their gradient with respect to J, and then apply the Back Propagation [35] optimization algorithm to train the network.

Overfitting is a common problem of Deep Neural Networks [19]. Once a Deep Neural Network is overfitted with the specific malware impact training set, the derived network becomes less useful. To address the overfitting problem, the dropout regularization strategy [37] is used in our designed Deep Neural Network impact prediction model. With a fixed probability, the Dropout strategy randomly removes a number of incoming and outgoing connections of some neurons. Using the Dropout strategy makes the network relatively independent on a particular set of neurons and the associated weights as well as the biases [19]. In this paper, the dropout rate of the deep neural network is 0.5.

IV. EXPERIMENTAL ANALYSIS

To validate the effectiveness of our approach, this section presents an empirical validation of our proposed method.

To fully evaluate the effectiveness of our proposed solution, we will address the 3 research questions:

- **Research Question 1:** How effective is it to represent both low and high impact Android malicious samples using *AndroidManifest.xml* based information (for example, requested permissions, application components or hardware components)?
- **Research Question 2:** How good is the performance if we represent the low and high impact Android malicious samples exploiting the features decoded from the disassembled binary *classes.dex* codes (for example, suspicious API calls, critical API calls)?
- **Research Question 3:** How can we make impact prediction for the newly identified Android malicious samples in the wild?

A. EXPERIMENTAL SETUPS

The following experiments were implemented on a workstation equipped using two E5-2690 v3 2.60GHz CPUs, in total of 48 logical CPU cores. The workstation offers 3.5 TB storage and 64 GB memory which meets the required experiment conditions.

In the reverse engineering part, we use the open sourced tool *Androguard* [11]. *Androguard* is a tool developed using python language to reverse engineer Android application packages, e.g., parse *AndroidManifest.xml* files or disassemble *classes.dex* bytecodes. Using *Androguard* we can decode the semantic features to represent Android malicious samples.

During the impact prediction stage, we employ the open-source python Machine Learning package *scikit-learn* [30] to perform the training and predicting task of SVM based impact model. We employ the grid search strategy to search the optimal parameters C and *gamma* for the impact prediction model. 50% of the constructed Mal_Impact Dataset is splited as training dataset while the other 50% serves as the test data.

To implement the Deep Neural Network based malware impact prediction model, we use the *Tensorflow*[8] and *Keras*[9] packages to perform the malware impact prediction task. When training the Deep Neural Network, 20% of the training dataset is divided as the validation dataset, then we track the accuracy trends of both the training and validation data. Meanwhile we keep checking whether the model training is finished successfully to prevent overfitting. Once the accuracy growing trend of training and validation data are not consistent, then the training process will be immediately stopped, and network weight parameters are then revised to retrain again to prevent the overfitting issue.

B. EVALUATION METRICS

The proposed Android malware impact prediction research problem is a class-imbalance problem due to the fact that the size of low impact malicious samples (*Low_Mal*) is significantly larger than that of high impact malicious samples (*High_Mal*). Thus to comprehensively validate the effectiveness of our proposed solution, in this work, we employ the next 7 performance metrics:

- *TP*: the size of low impact Android malicious samples being correctly predicted as low impact malware.
- *TN*: the size of high impact Android malicious data being correctly predicted as high impact malware.
- *FP*: the size of low impact Android malicious samples being incorrectly predicted as high impact malware.
- *FN*: the size of high impact Android malicious data being incorrectly predicted as low impact malware.
- *Precision*: as shown by Equation 7, the size of correctly identified high impact malware over the size of correctly identified high impact malicious samples and incorrectly identified high impact malicious samples.

$$P = TP/(TP + FP) \tag{7}$$

- *Recall*: as defined in Equation 8, the size of correctly identified high impact malicious samples divided by the size of correctly identified high impact malicious samples and incorrectly identified low impact malicious samples.

$$R = TP/(TP + FN) \tag{8}$$

- *F1-score*: F1-score is a combination of *TN*, *TP*, *FN* and *FP*. It can reflect the prediction effectiveness of the classifiers in a more comprehensive way. Equation 9 provides the formula for the computation of F1-score.

$$F1 = 2\frac{P \times R}{P + R} \tag{9}$$

C. REPRESENT THE LOW AND HIGH IMPACT MALICIOUS SAMPLES USING ANDROIDMANIFEST.XML BASED FEATURES

In this part, we address our first research question, we perform the impact prediction task based on information from *AndroidManifest.xml* files. Both low impact malware (*Low_Mal*) samples and high impact samples (*High_Mal*) are characterized using *AndroidManifest.xml* based features.

To investigate the discriminative ability of *AndroidManifest.xml* based features, we visualize the malware data using t-SNE (t-Distributed Stochastic Neighbor Embedding) algorithm [21]. t-SNE is a popular algorithm for reducing dimensions that is suited for high dimensional data visualization.

Fig. 5 shows the visualization result of the training malware samples. We can observe the low and high impact malicious samples are clearly distinct in the reduced 2-D feature space. That is, the *AndroidManifest.xml* based features imply potential discriminant power in separating low and high impact malicious samples. Fig.6 shows the visualization result of the testing malware samples. From Fig. 6 and Fig.5, we observe that the testing malware distribution is similar to that of the training malware, which implies a good malware impact prediction performance.

[8] https://www.tensorflow.org/
[9] https://keras.io/

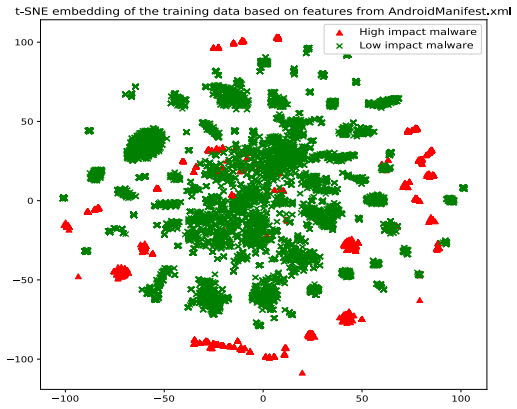


FIGURE 5. Visualization result of the training Android malware samples using features from *AndroidManifest.xml*.

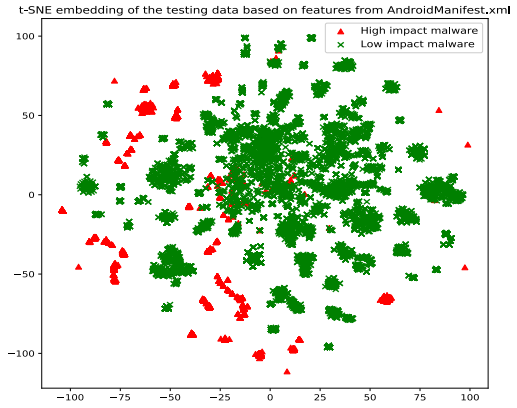


FIGURE 6. Visualization result of the testing Android malware samples using *AndroidManifest.xml* based features.

TABLE 6. The malware impact prediction results of SVM employing semantic features decoded from *AndroidManifest.xml* files.

	Precision	Recall	F1-score
Low Impact malware	1.00	1.00	1.00
High Impact malware	0.99	0.99	0.99
average	1.00	1.00	1.00

TABLE 7. The malware impact prediction results of Deep Neural Network employing features parsed from *AndroidManifest.xml* files.

	Precision	Recall	F1-score
Low Impact malware	0.97	0.96	0.96
High Impact malware	0.94	0.93	0.92
average	0.96	0.95	0.95

Table 6 and Table 7 list the detailed impact prediction performance of SVM and Deep Neural Network, it can be seen that the semantic features decoded from *AndroidManifest.xml* files perform better in characterizing those low impact malware samples. In addition, we can see that the impact prediction performance of SVM is superior to that of Deep Neural Network. The reason is that Deep Learning is hungry for the data volume and is good at learning complex pattern from a large amount of data. However, in our work, the size of the malware, especially the high impact malware is too few for training robust Deep Neural Networks.

TABLE 8. The most top ten significant features parsed from *AndroidManifest.xml* files.

Feature name	Score
service_com.apperhand.device.android.androidsdkprovider	2.15
requestedpermission_android.permission.receive_sms	1.84
intentfilter_android.intent.action.phone_state	1.72
requestedpermission_android.permission.camera	1.65
intentfilter_android.intent.action.reboot	1.59
intentfilter_android.intent.action.quickboot_poweron	1.50
requestedpermission_android.permission.read_settings	1.40
requestedpermission_android.permission.write_settings	1.40
broadcastreceiver_com.ads.smsreceiver	1.36
requestedpermission_android.permission.read_call_log	1.35

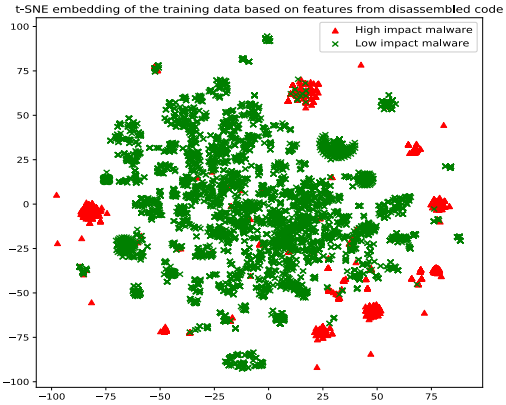


FIGURE 7. Visualization of the training malware data characterized by features from disassembled code.

In order to measure the contributions of different individual features in separating low impact and high impact malware, based on the SVM malware impact prediction model (each weight is assigned to a certain feature), we list the most top 10 features in the descending order according to the absolute values of the weight scores (denoted as $||w||$) in Table 8. From Table 8, among the top 10 features, 5 features are permission involved and 3 features are intent filters involved. We conclude that the intent filters and requested permissions involved features have better discriminative power in separating low impact and high impact malicious samples. In addition, the common sensitive and suspicious features all appear in the top 10 feature list, e.g., the SMS related permissions, reboot related permissions or camera related permissions.

D. REPRESENT THE LOW AND HIGH IMPACT MALICIOUS SAMPLES EMPLOYING THE INFORMATION PARSED FROM DISASSEMBLED CLASSES.DEX CODE

In this part, to answer the third research question, we represent the Android malicious samples using features decoded from the disassembled *classes.dex* codes, and then we vectorize the malicious samples for the following impact prediction task.

Fig. 7 shows the visualization result of the training malware dataset, and Fig. 8 is the visualization result for the

FIGURE 8. Visualization of the testing malware data characterized by features from disassembled code.

TABLE 9. The malware impact prediction outcomes of SVM exploiting features decoded from disassembled *classes.dex* codes.

	Precision	Recall	F1-score
Low Impact malware	0.99	0.99	0.99
High Impact malware	0.95	0.96	0.96
average	0.99	0.99	0.99

TABLE 10. The malware impact prediction results of deep neural network employing features decoded from disassembled *classes.dex* codes.

	Precision	Recall	F1-score
Low Impact malware	0.95	0.95	0.95
High Impact malware	0.90	0.91	0.90
average	0.93	0.94	0.93

testing malware samples. There is a clear boundary between low impact and high impact malicious samples. Therefore, the disassembled *classes.dex* involved features can also discriminate the low and high impact Android malicious samples with satisfactory performance.

We represent all the malware samples D with features from disassembled *classes.dex* code, then we perform the malware impact prediction task. Table 9 and Table 10 show the malware impact prediction results of SVM and Deep Neural Network. Compared with the performance using *AndroidManifest.xml* based features, we find that the prediction performance is slightly worse but also satisfactory. Besides, the performance of SVM is better than that of Deep Neural Network due to the data size limitation.

E. PREDICT THE IMPACT OF NEWLY IDENTIFIED ANDROID MALWARE IN THE WILD

As we know, Android malware evolve over time due to many factors, e.g., the employment of obfuscation or encryption techniques. Meanwhile, the properties of both low and high impact malware will also change over time. Thus the impact prediction accuracy might significantly decrease if we ignore such evolution of newly Android malware in the wild. Therefore it is essential to take into account such evolution of impact characteristics while validating the effectiveness of our proposed solution. In this section, our goal is to address

TABLE 11. The most top ten significant features from disassembled *classes.dex* code.

Feature name	Score
usedpermissions_android.permission.access_fine_location	3.87
usedpermissions_android.permission.read_phone_state	3.15
usedpermissions_android.permission.camera	2.78
suspiciousapi_landroid/telephony/telephonymanager.getsubscriberid	2.77
restrictedapi_android.net.connectivitymanager.getnetworkinfo	2.60
urldomain_119.147.23.195	2.16
urldomain_52.220.234.108	2.00
restrictedapi_android.location.locationmanager.getlastknownlocation	1.80
urldomain_54.149.205.221	1.79
usedpermissions_android.permission.restart_packages	1.77

the third research question: Is it possible to accurately and automatically make the impact prediction for those newly identified malware in the wild?

In order to investigate this research question, we simulate the malware evolution scenario by training the impact prediction models with 'older' malware samples (the low impact and high impact malicious samples were all detected between 2011 to 2015), while testing the model with 'newer' samples (these malware were detected in 2016 and 2017). In the experiments, the number of 'older' training low impact and high impact malicious samples is 15912 and 3409, respectively, and the 'newer' testing dataset includes 4537 low impact malicious samples and 928 high impact malicious samples.

In this work, we define a new evaluation metric *prediction power* to evaluate the contributions of different semantic feature subsets (e.g., *requested permissions* f_rp, *filtered intents* f_fi or *suspicious API calls* $f\_sa\ldots$) in malware impact prediction. Since the collected Android malware impact dataset is class-imbalanced, to make a comprehensive evaluation, we selected F1-score rather than accuracy or precision as the evaluation metric of *prediction power*. The higher prediction F1-score is, the corresponding feature subset is considered to have greater *prediction power*. Different feature subsets are individually employed to represent the malware samples for the following impact prediction. The *prediction power* of each feature set is shown in Fig.9. Firstly, among all the semantic feature set extracted from *AndroidManifest.xml* files, *application components* set f_ac, *requested permissions* set f_rp and *filtered intents* set f_fi have the optimal *prediction power* for low impact malware samples (*Low_Mal*) as well as high impact malware samples (*High_Mal*), while *hardware components* feature set f_hc has the worst prediction power, particularly for those high impact malicious samples (*High_Mal*). In addition, for feature sets extracted from disassembled *classes.dex* codes, the *prediction power* of *suspicious API calls* feature set f_sa and *Used permissions* feature set f_up are superior to that of *network addresses* set f_na and *critical API calls* set f_ca. In conclusion, in the task of Android malicious samples impact prediction, features decoded from *AndroidManifest.xml* files have more strong *prediction power* than features parsed from disassembled *classes.dex* codes.

Table 12 and Table 13 shows the impact prediction performance of SVM and Deep Neural Network using features from

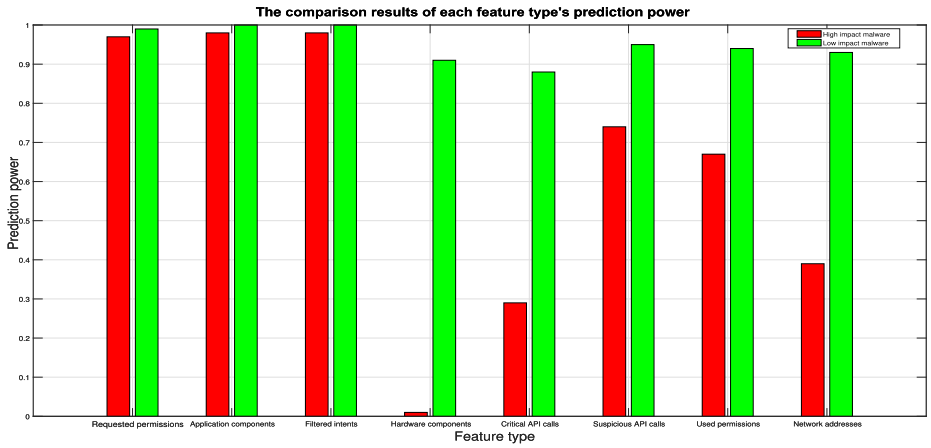


FIGURE 9. The comparison results of every feature type's prediction power.

TABLE 12. The malware impact prediction results of SVM using features from both disassembled binary *classes.dex* codes and *AndroidManifest.xml* files to represent malware samples.

	Precision	Recall	F1-score
Low Impact malware	1.00	1.00	1.00
High Impact malware	0.99	0.99	0.99
average	1.00	1.00	1.00

TABLE 13. The malware impact prediction results of deep neural network using features from both disassembled binary *classes.dex* codes and *AndroidManifest.xml* files to represent malware samples.

	Precision	Recall	F1-score
Low Impact malware	0.97	0.97	0.97
High Impact malware	0.95	0.93	0.94
average	0.97	0.96	0.96

both disassembled binary *classes.dex* codes and *AndroidManifest.xml*. Aggregating more features should produce better prediction performance. SVM also performs better than Deep Neural Network in the impact prediction of those newly identified Android malware. However, employing more features also implies it cost more expensive resources (e.g., time consumption, memory consumption) in reverse engineering. In reality, the security practitioner should balance the tradeoff between accuracy and efficiency.

V. CONCLUSIONS AND FUTURE DIRECTIONS

Android malware cause great security and privacy damage to our society. Despite huge research efforts, with the tremendous growth in Android malware applications, detecting the malware or classifying malware into specific families has no longer sufficient for security risk management. In this work, we raise a novel research problem: Is it possible to accurately and automatically make the impact prediction for the newly identified Android malicious samples? We attempt to design a light-weight solution to address this research question. Firstly, we construct a new Android malware dataset with low impact or high impact ground truth. Then, we perform feature engineering to parse semantic features to represent the malware samples. Thirdly, we build up the SVM and Deep Neural Network models to predict the impact of Android malicious samples. In the end, the empirical experiments validate the effectiveness of our proposed solution in the Android malware impact prediction.

In the future, our research work will focus on the following directions: On the one hand, we will investigate more impact related metrics, for example, the economic loss caused by the malware, to directly reflect the impact of Android malware. Besides, we may collaborate with security corporations or labs to obtain more accurate and trusted metrics values. On the other hand, we will collect more Android malware samples from other sources to update the created dataset in this paper. What's more, more advanced and informative features, e.g., graph based features (FCG, CFG), tree based features (abstract syntax tree), will be introduced to accurately capture the impact characteristics of Android malware.

REFERENCES

[1] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *9th Int. ICST Conf. Secur. Privacy Commun. Netw. (SecureComm)*, Sydney, NSW, Australia, Sep. 2013, pp. 86–103. doi: 10.1007/978-3-319-04283-1_6.

[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. 21st Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, 2014, pp. 23–26. [Online]. Available: https://www.ndss-symposium.org/ndss2014/drebin-effective-and-explainable-detection-android-malware-your-pocket

[3] S. Arzt *et al.*, "FlowDroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," in *Proc. 35th ACM SIGPLAN Conf. Program. Lang. Des. Implement. (PLDI)*, Edinburgh, U.K., 2014, pp. 259–269. doi: 10.1145/2594291.2594299.

[4] A. Bartel, J. Klein, Y. L. Traon, and M. Monperrus, "Dexpler: Converting Android dalvik bytecode to jimple for static analysis with soot," in *Proc. ACM SIGPLAN Int. Workshop State Art Java Program Anal. (SOAP)*, Beijing, China, 2012, pp. 27–38. doi: 10.1145/2259051.2259056.

[5] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Vancouver, BC, Canada, Dec. 2006, pp. 153–160. [Online]. Available: http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks

[6] T. Bläsing, L. Batyuk, A.-D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *Proc. 5th Int. Conf. Malicious Unwanted Softw.*, Lorraine, France, Oct. 2010, pp. 55–62. doi: 10.1109/MALWARE.2010.5665792.

[7] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices (SPSM)*, Chicago, IL, USA, Oct. 2011, pp. 15–26. doi: 10.1145/2046614.2046619.

[8] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "MAST: Triage for market-scale mobile malware analysis," in *Proc. 6th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Budapest, Hungary, Apr. 2013, pp. 13–24. doi: 10.1145/2462096.2462100.

[9] M. R. Costa-Jussà, A. Allauzen, L. Barrault, K. Cho, and H. Schwenk, "Introduction to the special issue on deep learning approaches for machine translation," *Comput. Speech Lang.*, vol. 46, pp. 367–373, Nov. 2017.

[10] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

[11] A. Desnos. (2011). *Androguard*. [Online]. Available: https://github.com/androguard/androguard

[12] A. Desnos and G. Gueguen, "Android: From reversing to decompilation," in *Proc. Black Hat*, Abu Dhabi, United Arab Emirates, 2011, pp. 77–101.

[13] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, p. 5, 2010. doi: 10.1145/2619091.

[14] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri, "A study of Android application security," in *Proc. 20th USENIX Conf. Secur.*, San Francisco, CA, USA, Aug. 2011, p. 21. [Online]. Available: http://static.usenix.org/events/sec11/tech/full_papers/Enck.pdf

[15] P. Faruki, V. Ganmoor, V. Laxmi, M. S. Gaur, and A. Bharmal, "Androsimilar: Robust statistical feature signature for android malware detection," in *Proc. 6th Int. Conf. Secur. Inf. Netw.*, Aksaray, Turkey, Nov. 2013, pp. 152–159. doi: 10.1145/2523514.2523539.

[16] Y. Feng, S. Anand, I. Dillig, and A. Aiken, "Apposcopy: Semantics-based detection of android malware through static analysis," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE)*, Hong Kong, Nov. 2014, pp. 576–587. doi: 10.1145/2635868.2635869.

[17] Y. Feng, O. Bastani, R. Martins, I. Dillig, and S. Anand, "Automated synthesis of semantic malware signatures using maximum satisfiability," in *Proc. 24th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, Feb./Mar. 2017, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/automated-synthesis-semantic-malware-signatures-using-maximum-satisfiability/

[18] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of Android malware using embedded call graphs," in *Proc. ACM workshop Artif. Intell. Secur. (CCS)*, Berlin, Germany, Nov. 2013, pp. 45–54. doi: 10.1145/2517312.2517315.

[19] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learn* (Adaptive Computation and Machine Learning). Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/

[20] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst.*, vol. 13, no. 4, pp. 18–28, Jul./Aug. 1998. doi: 10.1109/5254.708428.

[21] L. van der Maaten and G. Hinton, "Visualizing high-dimensional data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.

[22] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.

[23] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Hindroid: An intelligent android malware detection system based on structured heterogeneous information network," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Halifax, NS, Canada, Aug. 2017, pp. 1507–1515. doi: 10.1145/3097983.3098026.

[24] C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, "Performance evaluation on permission-based detection for android malware," in *Advances in Intelligent Systems and Applications*, vol. 2. Berlin, Germany: Springer, 2013, pp. 111–120.

[25] J. Jiang, S. Wen, S. Yu, Y. Xiang, and W. Zhou, "Identifying propagation sources in networks: State-of-the-art and comparative studies," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 1, pp. 465–481, 1st Quart., 2017. doi: 10.1109/COMST.2016.2615098.

[26] L. Liu, O. de Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1397–1417, 2nd Quart., 2018. doi: 10.1109/COMST.2018.2800740.

[27] C. Lueg. (Apr. 2017). 8,400 New Android Malware Samples Every Day. G DATA Software AG. Accessed: Jun. 28, 2017. [Online]. Available: https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day

[28] E. Mariconti, L. Onwuzurike, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building markov chains of behavioral models," in *Proc. 24th Annu. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, San Diego, CA, USA, Feb./Mar. 2017, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/mamadroid-detecting-android-malware-building-markov-chains-behavioral-models/

[29] D. Octeau, S. Jha, and P. McDaniel, "Retargeting android applications to java bytecode," in *Proc. ACM SIGSOFT 20th Int. Symp. Found. Softw. Eng.*, Cary, NC, USA, Nov. 2012, p. 6. doi: 10.1145/2393596.2393600.

[30] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2078195

[31] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: Versatile protection for smartphones," in *Proc. 26th Annu. Comput. Secur. Appl. Conf.*, Dec. 2010, pp. 347–356. doi: 10.1145/1920261.1920313.

[32] J. Qiu, W. Luo, S. Nepal, J. Zhang, Y. Xiang, and L. Pan, "Keep calm and know where to focus: Measuring and predicting the impact of android malware," in *Proc. 14th Int. Conf. Adv. Data Mining Appl. (ADMA)*, Nanjing, China, Nov. 2018, pp. 238–254. doi: 10.1007/978-3-030-05090-0_21.

[33] H. B. R. Mohammed, R. Vinayakumar, K. P. Soman. (2018). "A short review on applications of deep learning for cyber security." [Online]. Available: https://arxiv.org/abs/1812.06292

[34] V. Rastogi, Y. Chen, and W. Enck, "Appsplayground: Automatic security analysis of smartphone applications," in *Proc. 3rd ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, Feb. 2013, pp. 209–220. doi: 10.1145/2435349.2435379.

[35] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognit. Model.*, vol. 5, no. 3, p. 1, 1988.

[36] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, G. Álvarez, "PUMA: Permission usage to detect malware in Android," in *Proc. Int. Joint Conf. (CISIS'ICEUTE'SOCO')*, Ostrava, Czech Republic, Sep. 2012, pp. 289–298. doi: 10.1007/978-3-642-33018-6_30.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2670313

[38] (2017). *Global Market Share Held by the Leading Smartphone Operating Systems in Sales to End Users From 1st Quarter 2009 to 1st Quarter 2017*. Accessed: Jun. 28, 2017. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/, The Statistics Portal.

[39] N. Sun, J. Zhang, P. Rimba, S. Gao, Y. Xiang, and L. Y. Zhang, "Data-driven cybersecurity incident prediction: A survey," *IEEE Commun. Surveys Tuts.*, to be published.

[40] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of Android malware and Android analysis techniques," *ACM Comput. Surv.*, vol. 49, no. 4, p. 76, 2017. doi: 10.1145/3017427.

[41] S. Wen, M. S. Haghighi, C. Chen, Y. Xiang, W. Zhou, and W. Jia, "A sword with two edges: Propagation studies on both positive and negative information in online social networks," *IEEE Trans. Comput.*, vol. 64, no. 3, pp. 640–653, Mar. 2015. doi: 10.1109/TC.2013.2295802.

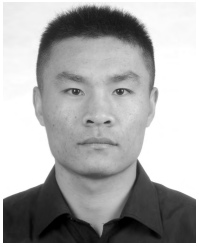
[42] E. R. Wognsen, H. S. Karlsen, M. C. Olesen, and R. R. Hansen, "Formalisation and analysis of dalvik bytecode," *Sci. Comput. Program.*, vol. 92, pp. 25–55, Oct. 2014. doi: 10.1016/j.scico.2013.11.037.

[43] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and API calls tracing," in *Proc. 7th Asia Joint Conf. Inf. Secur.*,Tokyo, Japan, Aug. 2012, pp. 62–69. doi: 10.1109/AsiaJCIS.2012.18.

[44] T. Wu, S. Wen, Y. Xiang, and W. Zhou, "Twitter spam detection: Survey of new approaches and comparative study," *Comput. Secur.*, vol. 76, pp. 265–284, Jul. 2018. doi: 10.1016/j.cose.2017.11.013.

[45] L. K. Yan and H. Yin, "Droidscope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis," in *Proc. 21th USENIX Secur. Symp.*, Bellevue, WA, USA, Aug. 2012, pp. 569–584. [Online]. Available: https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/yan

[46] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck, "Appcontext: Differentiating malicious and benign mobile app behaviors using context," in *Proc. 37th Int. Conf. Softw. Eng. (ICSE)*, Florence, Italy, vol. 1, May 2015, pp. 303–313. doi: 10.1109/ICSE.2015.50.
[47] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 104–117, Jan. 2013.
[48] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Scottsdale, AZ, USA, Nov 2014, pp. 1105–1116. doi: 10.1145/2660267.2660359.
[49] Y. Zhang *et al.*, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Berlin, Germany, Nov. 2013, pp. 611–622. doi: 10.1145/2508859.2516689.
[50] M. Zheng, M. Sun, and J. C. S. Lui, "Droid analytics: A signature based analytic system to collect, extract, analyze and associate Android malware," in *Proc. 12th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom), 11th IEEE Int. Symp. Parallel Distrib. Process. Appl. (ISPA), 12th IEEE Int. Conf. Ubiquitous Comput. Commun. (IUCC)*, Melbourne, VIC, Australia, Jul. 2013, pp. 163–171. doi: 10.1109/TrustCom.2013.25.



JUNYANG QIU is currently pursuing the Ph.D. degree with the School of Information Technology, Deakin University. His research interests include cyber security and machine learning.

WEI LUO received the Ph.D. degree in computer science from Simon Fraser University, Canada. He is currently a Senior Lecturer with the School of Information Technology, Deakin University. He has published more than 40 research papers in refereed international journals and conferences. His research interests include data mining and machine learning.

LEI PAN received the Ph.D. degree in computer forensics from Deakin University, Australia, in 2008, where he is currently a Senior Lecturer with the School of Information Technology. His research interest includes cyber security and privacy. He has published more than 30 research papers in refereed international journals and conferences, such as the IEEE Security & Privacy, *Journal of Multimedia*, and Digital Investigation. He is a member of the IEEE.

YONGHANG TAI received the M.Sc. degree in optic engineering from Yunnan Normal University, in 2012, and the Ph.D. degree in computer science from Deakin University, Australia, in 2019. His current research interests include cybersecurity and applied machine learning. In addition, he focuses on developing intelligent defense systems against sophisticated cyber attacks.

JUN ZHANG received the Ph.D. degree in computer science from the University of Wollongong, Australia, in 2011. He is currently an Associate Professor with the School of Software and Electrical Engineering, and the Deputy Director of the Swinburne Cybersecurity Lab, Swinburne University of Technology, Australia. His research interests include cybersecurity and applied machine learning. He has published more than 100 research papers in refereed international journals and conferences, such as the IEEE Communications Surveys and Tutorials, the IEEE/ACM Transactions on Networking, the IEEE Transactions on Image Processing, the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Information Forensics and Security, the ACM Conference on Computer and Communications Security, and the ACM Asia Conference on Computer and Communications Security. His publications have been widely cited in the area of cybersecurity. He is an IEEE Senior Member. He has been internationally recognized as an Active Researcher in cybersecurity, evidenced by his chairing of eight international conferences, from 2013, and presenting of invited keynote addresses in two conferences, and an Invited Lecturer in the IEEE SMC Victorian Chapter.

YANG XIANG received the Ph.D. degree in computer science from Deakin University, Australia. He is currently the Dean of the Digital Research and Innovation Capability Platform, Swinburne University of Technology, Australia. His research interest includes cyber security, which covers network and system security, data analytics, distributed systems, and networking. In the past 20 years, he has been leading the team developing active defense systems against large-scale distributed network attacks. His translational research has made significant impact on the real-world applications, such as AI-driven cyber security applications, malware applications, cloud and the IoT security applications, and blockchain applications. His research was funded by the Australian Research Council (ARC) and industry partners. He has published more than 200 research papers in many international journals and conferences, such as the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Information Security and Forensics, and the IEEE Transactions on Dependable and Secure Computing. He has published three books: *Honeypot Frameworks and Their Applications: A New Framework* (Springer), *Software Similarity and Classification* (Springer), and *Dynamic and Advanced Data Mining for Progressing Technological Development* (IGI-Global). He is a Senior Member of the IEEE. He is the foundation Editor-in-Chief of the *SpringerBriefs on Cyber Security Systems and Networks*. He is the Co-Founder and the Steering Committee Chair of the NSS, ICA3PP, CSS, SocialSec conference series. He served as the Associate Editor for the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, and *Security and Communication Networks* (Wiley), and the Editor of the *Journal of Network and Computer Applications*. He is the Coordinator for the IEEE Computer Society Technical Committee on Distributed Processing (TCDP).