Received September 14, 2019, accepted September 28, 2019, date of publication October 9, 2019, date of current version October 22, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2946392

A3CM: Automatic Capability Annotation for Android Malware

JUNYANG QIU[1], JUN ZHANG[2], WEI LUO[1], LEI PAN[1], SURYA NEPAL[3], YU WANG[4], AND YANG XIANG[2], (Senior Member, IEEE)

[1]School of Information Technology, Deakin University, Geelong, VIC 3216, Australia
[2]School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia
[3]CSIRO, Data61, Sydney, NSW 1710, Australia
[4]School of Computer Science, Guangzhou University, Guangzhou 510006, China

Corresponding author: Yu Wang (yuwang@gzhu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61802080 and Grant 61802077.

ABSTRACT Android malware poses serious security and privacy threats to the mobile users. Traditional malware detection and family classification technologies are becoming less effective due to the rapid evolution of the malware landscape, with the emerging of so-called zero-day-family malware families. To address this issue, our paper presents a novel research problem on automatically identifying the security/privacy-related capabilities of any detected malware, which we refer to as *Malware Capability Annotation* (MCA). Motivated by the observation that known and zero-day-family malware families share the security/privacy-related capabilities, MCA opens a new alternative way to effectively analyze zero-day-family malware (the malware that do not belong to any existing families) through exploring the related information and knowledge from known malware families. To address the MCA problem, we design a new MCA hunger solution, Automatic Capability Annotation for Android Malware (A3CM). A3CM works in the following four steps: 1) A3CM automatically extracts a set of semantic features such as permissions, API calls, network addresses from raw binary APKs to characterize malware samples; 2) A3CM applies a statistical embedding method to map the features into a joint feature space, so that malware samples can be represented as numerical vectors; 3) A3CM infers the malicious capabilities by using the multi-label classification model; 4) The trained multi-label model is used to annotate the malicious capabilities of the candidate malware samples. To facilitate the new research of MCA, we create a new ground truth dataset that consists of 6,899 annotated Android malware samples from 72 families. We carry out a large number of experiments based on the four representative security/privacy-related capabilities to evaluate the effectiveness of A3CM. Our results show that A3CM can achieve promising accuracy of 1.00, 0.98 and 0.63 in inferring multiple capabilities of known Android malware, small size-families' malware and zero-day-families' Android malware, respectively.

INDEX TERMS Android malware, security/privacy-related capability, multi-label learning, malicious capability prediction, zero-day-family malware.

I. INTRODUCTION

Currently Android has become the most popular mobile operating system, with 74.82% market share in February 2018 [1]. Unfortunately, the popularity of Android together with its openness causes the number of Android malware skyrocketed in both official and third-party Android app markets. It is estimated that almost 12,000 new Android malware samples being detected per day in 2018 [2]–[4]. This growth in Android malware has significantly compromised the functionality of the devices and even the financial security of to both mobile phone users as well as the service providers.

To preserve a healthy ecosystem for Android users, the research communities and security vendors have proposed various techniques to analyze malware [5]–[10]. Among these efforts, Android malware family classification is a key step for further analyzing and better understanding of malware, e.g., finding new risks or threat patterns, designing new signatures, updating old signatures or locating malicious codes [11]–[14]. To accelerate the family attribution process, machine learning based malware family classification techniques are designed to assign the most likely family class label to the detected malware, which guides the

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

subsequent forensic analysis and threat assessment. However, in reality, the machine learning based family classification techniques have to be frequently retrained to deal with the zero-day-family malware. Otherwise the latest zero-day-family malware will be misclassified into an existing malware family. In addition, there exist other inevitable challenges and limitations of multi-class family classification, e.g., the number of Android malware families is increasing rapidly, but a large proportion of families just have one or few samples. The vanish of old families and the emerge of zero-day-families poses another challenge for the multi-class family classification. The existing family division is inaccurate and typically carries little semantic information [15].

To address the limitations faced by malware family classification, we propose a new research problem called Android Malware Capability Annotation (MCA). To solve the MCA research problem, we design a solution employing the multi-label classification model to annotate the capabilities of Android malware. In order to realize automatic capability annotation for Android malware, the cornerstone is to define the security/privacy-related capability and to create a corresponding dataset with capability ground truth for validation. Thus we need to address the following two challenges (denoted as **C1** and **C2**):

C1: Data collection and ground truth. We are the first to propose the MCA research problem. There is no dataset publicly available with capability ground truth. To create such a dataset, a reliable and trusted ground truth is indispensable. Last but not the least, a cross-checking process is required to verify the dataset.

C2: Annotation for the zero-day-family malware. A challenging but crucial task of the real-world malware analysis is detecting and classifying zero-day-family malware. The zero-day-family malware is difficult to be effectively annotated in terms of the capability vectors due to the lack of knowledge.

In summary, we make the following contributions to the Android malware analysis in this paper:

- We present a novel research problem on automatically identifying the security/privacy-related capabilities of any detected malware, which we refer to as *Malware Capability Annotation* (MCA).
- To facilitate the MCA research problem, based on the existing open sourced Android malware datasets and ground truth, we firstly create a well security/privacy-related capability annotated dataset. The annotated dataset will be released to the public in the hope of stimulating future research.
- To address MCA research problem, we design a solution named A3CM which employs the semantic features through reverse engineering of malware samples. And then multi-label learning scheme is employed for the automatic capabilities annotation.

A. ORGANIZATION

The rest of this paper is organized as follows. In Section II, we review the related work about Android malware analysis. Section III states the background information, the challenges and limitations and the proposed research problem. Section IV introduces the creation process of the ground truth dataset. We present the detailed A3CM technique in Section V, followed by the experimental evaluation and analysis of A3CM in Section VI. Section VII presents the limitation analysis. Finally, Section VIII gives the conclusions of this paper.

II. RELATED WORK

Machine learning techniques have been widely used in cyber security in recent years [4], [16]–[22]. Most of the machine learning based malware analysis works fall into the category of malware detection or family classification. In the following, we briefly review the current research status about Android malware detection and family classification.

A. ANDROID MALWARE DETECTION

In 2013, a robust and lightweight classifier DroidAPIMiner was designed for Android malware detection in [23]. Drebin [24] is a notable classic machine learning based static analysis method that enables detecting Android malware directly on the device. In [25], FeatureSmith was designed to automatically engineer features from scientific papers for Android malware detection. In 2017, a structured heterogeneous information network (HIN) was used for Android malware detection [7], [26]. Zhang *et al.* [27] proposed a Dalvik opcode graph based Android malware variants detection method using global topology features. Mariconti *et al.* presented MaMaDroid [8], an Android malware detection system based on modeling the sequences of API calls as Markov chains. In [28], a novel Android malware detection system with dynamic features was proposed facilitated with ensemble learning. DroidEnsemble was proposed to extract both string features and structural features to systematically and comprehensively characterize the static behaviors of Android apps and thus build a more effective Android malware detector [29]. Sun *et al.* [30] showed that it is possible to reduce the number of permissions to be analyzed for Android malware detection, while maintaining high effectiveness and accuracy. A novel Android malware detector named DroidFusion [31] based on a multilevel architecture that enables effective fusion of machine learning classifiers for higher accuracy was proposed. DroidFusion firstly generated a model through training base classifiers at a lower level and then applied a series of ranking-based algorithms on their predictive accuracies at the higher level to obtain a final classifier. In 2019, an efficient Android malware detection system was designed based on the method-level correlation relationship of application's abstracted API calls [32]. A combination method for Android malware detection based on Control

Flow Graphs and machine learning algorithms was presented in [33]. Kim *et al.* [34] proposed a multimodal deep learning method for Android malware detection using various features (extracted from *AndroidManifest.xml*, *classes.dex* and shared library function files). A new concept of Complex Flows was proposed to derive Android application behavior on device sensitive data. Then an automated system was designed to detect the malware using app behavior and app information flows [35]. In 2019, a dynamic Android application classification technique named DroidCat was proposed to effectively detect and categorize Android malware. The diverse and novel dynamic features enabled the superior robustness of DroidCat against several challenges, e.g., the complex use of reflection, code obfuscation, run-time permissions, and other evasion strategies [36].

B. ANDROID MALWARE FAMILY CLASSIFICATION

There exists many works on multi-class Android malware family classification [13], [37]–[40]. In [38], a novel semantic-based approach was proposed to classify Android malware via dependency graphs. To fight against transformation attacks, a weighted contextual API dependency graph was extracted as program semantics to construct feature sets. In 2017, Feng *et al.* proposed a technique for automatically inferring semantic malware signatures for Android from a small number of a malware family [13]. The key idea underlying the technique is to look for a maximally suspicious common subgraph that is shared between all known samples within a malware family. The work [39] addressed the concept drifting problem in malware detection, which bridged a fundamental research gap when dealing with evolving malicious software. Alswaina and Elleithy [41] adopted machine learning to analyze and identify the permissions requested by malware. The Extremely Randomized Trees were used to identify a small number of permissions that could be used to attribute the malware into the malware families. In [42], a set of semi-supervised techniques were introduced with the ultimate goal of facilitating security experts to generate the malware family signatures. A scalable framework was proposed to mine massive of Android applications with the main goal of detecting new malware samples, while reducing false positive rate. The proposed framework was capable of automatically clustering the Android applications into families and generating formal rules for detecting them with 100% recall and high precision. Fan *et al.* [43] proposed a novel method that constructs the frequent subgraphs shared by malware belonging to the same family. Then a system named FalDroid was implemented to automatically classify the Android malware samples and identify the representative malware samples. The identified malware samples greatly accelerated the malware inspecting process. The experimantal results demonstrated that FalDroid outperformed the state-of-the-art approaches. It offered considerable knowledge for the detection and deep investigation of Android malware and thus raised the level for malware to avoid analysis.

There are several key distinctions between our work and these related works:

1) The MCA research problem can be seen as a further step of binary Android malware detection. Once a malware is detected, then MCA will perform further analysis, e.g., annotating its security/privacy-related capabilities, which will facilitate the security analysts to the subsequent malicious code localization or signature generation.

2) Compared with multi-class family classification techniques, MCA research problem can be regarded as an alternative way of family classification. Given a malware, A3CM will annotate its security/privacy-related capabilities rather than classify it into specific family.

3) Our work is formulated as neither a binary malware detection problem nor a multi-class malware family classification problem. A detected Android malware may possess several security/privacy-related capabilities, thus we address a multi-label classification problem, which is more complicated than binary detection or multi-class classification.

III. MALWARE CAPABILITY ANNOTATION

For deep analysis and further understanding of the detected Android malware, many Android malware family classification techniques are proposed to assign the corresponding family class to the detected malware. However, there exist inevitable challenges and limitations of multi-class family classification:

1. The number of malware families has reached several thousands until now [44], but a large proportion of families just have one or few samples (58% of malware families in Drebin dataset [24] have fewer than 5 samples). The sparsity of malware samples provides limited knowledge to characterize the small-size-families.

2. The vanishing of old families and the emerging of zero-day-families poses another challenge for the multi-class family classification; the classification models need to retrain regularly to keep up with the latest malware families landscape. Unfortunately, most of the existing works were exhausted in dealing with the evolution of malware family.

3. The existing family division is inaccurate and typically carries little semantic information. In other word, the existing family division is outdated and no longer represents the current Android malware landscape. It cannot provide detailed information on malware's security/privacy-related behaviors [15].

4. Given a candidate malware sample, different malware family classification methods or tools produce inconsistent results, thus is hard to categorize it into a specific family. Currently, the mainstream method has to use the 'majority vote' strategy among all the classification results to determine the final family class [15].

Since the current malware family classification methods have inevitable challenges and limitations, we may ask a question: Can we annotate the malicious capabilities of a malware directly rather than classify it into specific family? In this way, for each Android malware sample, its capability

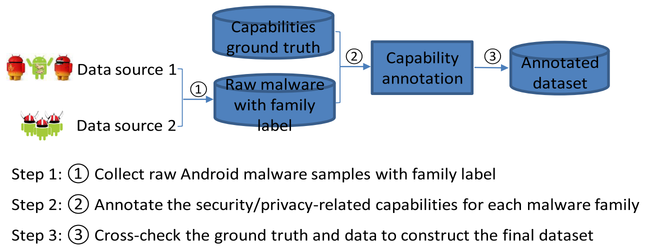

Capabilities ground truth
Data source 1
Data source 2
①
Raw malware with family label
②
Capability annotation
③
Annotated dataset
Step 1: ① Collect raw Android malware samples with family label
Step 2: ② Annotate the security/privacy-related capabilities for each malware family
Step 3: ③ Cross-check the ground truth and data to construct the final dataset

FIGURE 1. The flow diagram of the dataset creation.

vector can be annotated automatically with one or several security/privacy-related capabilities.

Observations. Previous research witnesses that the malware family classes evolve over the time, e.g., the vanish of old families and the birth of zero-day-families [39], [45]. However, during this evolving process, the significant security/privacy-related capabilities of malware samples change far less frequently than that of the families. Furthermore, the number of malware families (reach several thousand until now [44]) is far more than that of security/privacy-related capabilities (at most a dozen at present).

Based on the observations, we pursue the ability to identify the malicious capabilities of the detected malware rather than classify them into specific families. Suppose that a malware sample collects users' private information and sends premium a SMS message to subscribe services secretly. We annotate it with *Malicious SMS charge* and *Information stealing* labels. And this multi-label will be converted to a binary label indicator vector. Such a vector is defined as the malware's **Capability Vector**.

IV. THE CREATION OF NEW MALWARE CAPABILITY DATASET

Currently, many open source Android malware datasets available for the research community, e.g., *Drebin dataset* [24], *AMD dataset* [15], *VirusShare*.[1] We can also label the malicious apps based on the feedbacks of online service *VirusTotal*.[2] However, to the best of our knowledge, there is no annotated capability malware dataset available for the research community. Fig. 1 shows the flow diagram of our dataset creation. The three steps for creating the dataset are explained in details as follows.

A. RAW DATA COLLECTION

In this step, we collect the raw Android malware samples with the corresponding family labels. In this paper, the collected malware samples come from two publicly available sources: 1) **Drebin Dataset.**[3] Drebin contains 5,560 Android malware samples from 179 different malware families in the period of August 2010 to October 2012. 2) **AMD Dataset.**[4] AMD is a carefully-labeled and well-studied dataset that includes comprehensive profile information of malware. Currently, AMD contains 24,553 samples, categorized in 71 malware families ranging from 2010 to 2016.

For the **Drebin** and **AMD** dataset used in our paper, they have been widely used in many papers [30], [33], [46]–[48]. The most important is that these two datasets provided fine-grained family attribution ground truth information, which is indispensable for the creation of our dataset with malicious capability annotation.

B. GROUND TRUTH ANNOTATION

After the malware samples with family labels have been collected, the key building block is to annotate the security/privacy-related capabilities for each malware sample. However, it is very difficult and infeasible to annotate the capabilities for every malware sample.

In this work, the security/privacy-related capabilities are annotated according to the families. In other words, we hypothesize that the Android malware samples within the same families share the same capabilities. The referenced ground truth capabilities for Android malware families come from the report[5] [49]. We will label the training malware samples with the security/privacy-related capabilities as shown in Table 1.

TABLE 1. The list of the security/privacy-related capabilities.

Capability type	Detailed definition
Botnet attack	Infect and control the device, then look for more vulnerable devices across the Internet.
Unauthorized root access	Gain root access or at least try to convince the end user to root his/her phone.
Malicious SMS charge	Send paid or malicious SMS messages without users' awareness.
Information stealing	Steal users' private information and send to remote servers.
Location leakage	Track and steal location information or post the location to a web service.
Secretly installation	Install other apps or packages or update installed binaries.
Other unwanted functions	Potentially unwanted application, e.g., perform hacker or advertisement functions.
Banking Trojan	Intercept and modify banking authentication codes.
Windows infections	Infect a connected Windows PC, then gather a bunch of information from the computer.
Encryption Trojan	Encrypt all personal and private data on the infected device.

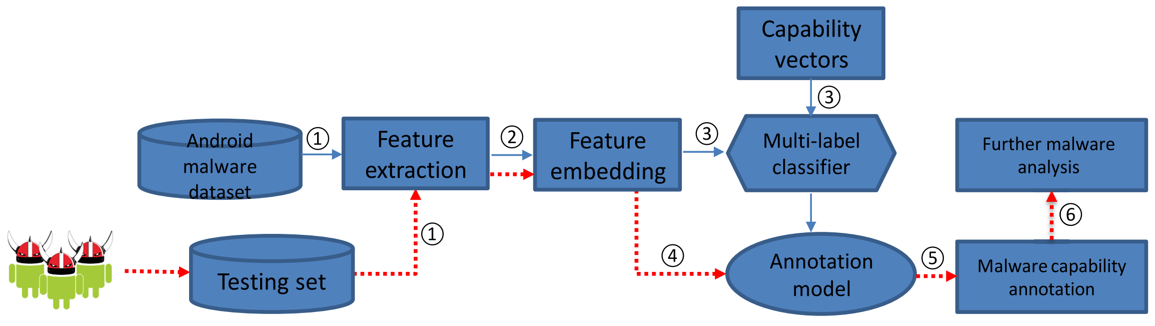
In the capability annotation process, some collected malware families are not annotated in the referenced ground truth report, and sometimes there are no malware samples available for certain annotated families in the ground truth report. Taking a conservative approach, we only keep the Android malware families with well capability annotation in the ground truth report.

C. DATA CROSS-CHECK

To form the final annotated dataset, we cross check the reliability of the ground truth and validate the data samples accordingly. Two Android malware professionals helped to verify and validate our dataset. The profile information

[1]https://virusshare.com/
[2]https://www.virustotal.com/#/home/upload
[3]https://www.sec.cs.tu-bs.de/ danarp/drebin/

[4]http://amd.arguslab.org/
[5]https://forensics.spreitzenbarth.de/android-malware/



Step 1: ① Extract semantic features to characterize the capabilities of each Android malware sample

Step 2: ② Embed the features into a joint feature space so as to fed into the classifier

Step 3: ③ Input the feature vectors and corresponding capability vectors to train the multi-label classification model

Step 4: ④ Input the feature vectors of the testing samples to the trained annotation model

Step 5: ⑤ Annotate the capability vectors of the testing malware samples

Step 6: ⑥ Locate the malicious codes using the capability vectors and feature weights

FIGURE 2. The overview of our proposed A3CM framework.

provided in the AMD dataset[6] can be used to validate the correctness of the ground truth for some malware families. The ground truth for the remaining malware families can be verified through the manual analysis according to [15].

V. THE PROPOSED ACAAM TECHNIQUE

To address the MCA research problem, we propose a solution, Automatic Capability Annotation for Android Malware (A3CM). Fig. 2 illustrates the overview of A3CM. In the following subsections, we will elaborate on the technical details about each part of A3CM.

A. FEATURE EXTRACTION

In this work, to maintain the capability annotation efficiency, we employ static analysis technique rather than the dynamic analysis to analyze the Android malware. Reverse engineering is performed to disassemble the raw binary Android APK files, and then the static features are parsed to characterize the Android malware. We extract semantic features as suggested in Drebin [24] to represent the Android malware samples. The semantic information about the extracted features is presented in Table 2. The feature sets in the top 4 rows are extracted from the *AndroidManifest.xml* file within the APK (Android application package), while the bottom 4 rows are from the disassembled *classes.dex* file. To extract the above semantic features, we need to disassemble the raw binary APK samples. In this work, *Androguard* is used to parse *AndroidManifest.xml* and to disassemble *classes.dex* bytecodes [50].

B. FEATURE EMBEDDING

Having extracted the string features for each malware, we construct the joint vector space and then embed the string features of each malware to obtain the numerical vectors.

[6]http://amd.arguslab.org/behaviors

TABLE 2. The detailed description of the extracted features.

Features	Brief information about the features
Requested permissions	An app will request permissons from users to access sensitive resources at installation
Application components	Android app can state several components, e.g., Activity, Service, ContentProvider, Broadcastreceiver
Filtered intents	The intent filter of a component specifies the action it can perform and the data type it is able to operate
Hardware components	Requesting certain hardware or a combination of specific hardwares have potentially security capabilities
Critical API calls	The critical calls reveal the sensitive capabilities of an app
Suspicious API calls	The suspicious API calls are extracted to reflect suspicious capabilities of malware
Used permissions	The critical API calls are used to determine and match both requested and actually used permissions
Network addresses	Some of network addresses may be involved in botnets or other malicious sites

We utilize the TF-IDF [51] algorithm to construct the feature vectors. More specifically, we use **TfidfVectorizer** in the open source package *scikit-learn* [52] to construct the feature vectors of each malware.

C. MULTI-LABEL CLASSIFICATION

Given the representation vector and the annotated capability vector, the annotated capability dataset can be denoted as $X = [x_1, x_2, ..., x_n]$, where $x_i \in \mathcal{R}^p$ represents the feature vector for each malware sample, n is the number of samples, and p is the feature dimensions. Let $Y = [Y_i^k]$ be the capability vectors for malware samples, and $Y_i^k \in \{0, 1\}$ with $Y_i^k = 1$ indicating the presence of the k-th capability for malware i.

In the above context, more than one capabilities can be simultaneously assigned to one malware sample. This is known as a multi-label classification [53] problem in the machine learning community. Currently, various methods have been proposed to address the multi-label classification problem. And these methods can be roughly divided into two categories: **Problem Transformation Methods** and

Algorithms Adaptation Methods [53]. In this paper, we use Linear Support Vector Machine and Decision Tree to perform the multi-label classification task for Problem Transformation Method and Algorithms Adaptation Method, respectively. Meanwhile we also design a Deep Neural Network (DNN) [54], [55] to perform the multi-label capability annotation task.

D. CAPABILITY ANNOTATION FOR THE CANDIDATE MALWARE

When the multi-label annotation model has been trained, it can be used to annotate the security/privacy-related capabilities of the testing samples. For the given testing malware sample, it will be disassembled, and the semantic features will be firstly extracted to characterize the malware, then it will be mapped into a vectorized representation. Then the testing sample's vector is fed to the trained annotation model to produce the Capability Vector indicating the annotation results. Each dimension of the output Capability Vector denotes the presence of the corresponding capability. For instance, the malware possesses the i^{th} malicious capability if the value of i^{th} dimension of Capability Vector is 1. If the value of j^{th} dimension of Capability Vector is 0, then we consider the malware doesn't have the j^{th} malicious capability. The annotation results can further assist the following deep analysis of the malware, e.g., malicious code localization.

VI. EXPERIMENTS AND ANALYSIS

A. EXPERIMENTAL SETTINGS

In this section, we present the experimental results and discussions of A3CM. To evaluate the effectiveness of A3CM, we measure the performance of A3CM, and compare A3CM with the malware family classification baseline. To be more specific, we design the experiments towards answering the following research questions:

- **RQ1:** How effective is A3CM at annotating the capability vectors of zero-day-family malware?
- **RQ2:** What is the annotation performance of A3CM on the malware samples belonging to large-size-families?
- **RQ3:** What is the A3CM's annotation performance on those malware samples belonging to small-size-families?
- **RQ4:** What information we can obtain from the annotation results of A3CM?
- **RQ5:** How about the performance if combining the static analysis and the dynamic behavioral features?

B. ANNOTATED DATASET AND EVALUATION MODELS

Dataset. In our collected security/privacy-related capability dataset, the samples belonging to some capabilities are limited. To validate the effectiveness of our proposed framework, as a proof-of-concept, we select four common and representative security/privacy-related capabilities (including *Botnet attack*, *Unauthorized root access*, *Malicious SMS charge* and *Information stealing*) with sufficient malware samples to perform the following experiments. The distribution of the focused four capabilities of the dataset is summarized in Table 3.

TABLE 3. The distribution of the focused four security/privacy-related capabilities.

Capability type	# malware samples
Botnet attack	3066
Unauthorized root access	1537
Malicious SMS charge	3701
Information stealing	3763

TABLE 4. The architecture information of the neural network.

#	Layer type	# of neuron	Activation
1	Dense	512	ReLU
2	Dense	256	ReLU
3	Dense	128	ReLU
4	Dense	32	ReLU
5	Dense	4	Sigmoid

Classifiers. To perform the classification task of A3CM, we employ Decision Tree (DT) and Support Vector Machine (SVM) classifiers implemented in scikit-learn [52], [56]. The optimal parameters of DT and SVM are determined using the grid search strategy. The DNN is implemented using keras[7] library and the detailed structural information of DNN is presented in Table 4.

Metrics. In this work, we employ the following metrics [57], [58] to evaluate the performance of A3CM:

- *Hamming loss*: It evaluates how many times a malware-capability is misclassified, i.e. a capability not belonging to the malware is annotated or a capability belonging to the malware is not annotated. The smaller the value of hamming loss means better annotation performance.

$$\frac{1}{K}\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}(Y_i^k \neq \widehat{Y}_i^k) \tag{1}$$

- *Accuracy score*: It is also called classification accuracy or exact match ratio. It computes the percentage of malware whole predicted capability vectors is exactly the same as their corresponding ground truth capability vectors. This metric tends to be overly strict when the size of label vector is large.

$$\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}(Y_i = \widehat{Y}_i) \tag{2}$$

- *Precision score*: A standard metric as shown in Eq.3, where TP is the number of true positives and FP the number of false positives.

$$P = TP/(TP + FP) \tag{3}$$

- *Recall score*: Another standard metric as shown in Eq.4, where FN is the number of false negatives. The best value is 1 and the worst value is 0.

$$R = TP/(TP + FN) \tag{4}$$

[7]https://keras.io/

TABLE 5. An example about how to compute the capability annotation performance of the baseline malware family classification results. The final annotation performance (e.g., accuracy score, hamming loss, precision score, etc.) will be calculated according to the 3rd column (Ground truth capability vector) and the 5th column (Capability vector corresponding to the predicted family label).

Testing samples	Ground truth family label	Ground truth capability vector	Predicted family label	Capability vector corresponding to the predicted family label
app1	Mobilespy	[0, 0, 0, 1, ...]	SpyBubble	[0, 0, 0, 1, ...]
app2	Fjcon	[1, 0, 1, 1, ...]	Obad	[1, 1, 1, 1, ...]
app3	Steek	[0, 0, 1, 1, ...]	Mania	[0, 0, 1, 0, ...]
app4	Basebridge	[1, 0, 0, 1, ...]	BankBot	[1, 0, 0, 1, ...]

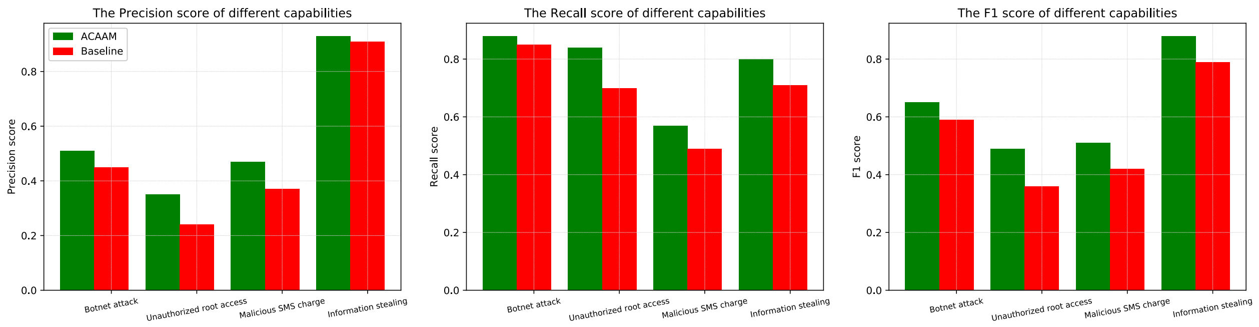


FIGURE 3. The Precision score, Recall score and F1 score comparisons between A3CM and the baseline malware multi-class family classification on zero-day-family malware using SVM.

- *F1 score*: F1 score is a combination of TP, TN, FP and FN. It can reflect the classification effectiveness of the model in a more comprehensive way. Equation 5 provides the formula for the computation of F1 score.

$$F1 = 2 * P * R/(P + R) \tag{5}$$

C. ANNOTATION PERFORMANCE ON ZERO-DAY-FAMILY MALWARE SAMPLES

To simulate the performance of A3CM on zero-day-family malware samples, we train the annotation model using malware samples belonging to specific families while we test the model on malware samples belonging to other families. Specifically, we use 5,993 malware samples from 62 families as training set and other unseen 906 samples from the rest 10 families as the testing set. In this way, we can calculate the capability annotation performance metrics. To obtain reliable results, the annotation results are averaged over 100 runs of the annotation model.

Currently, there is no similar work addressing the MCA research problem. To fully evaluate the effectiveness of our proposed A3CM, we employ the malware family classification method as the baseline. (Specifically, we select SVM and Decision Tree as the classifiers to perform the multi-class malware family classification). This is due to the fact that different malware families share some security/privacy related capabilities. Given an unknown malware sample, malware family classifiers will classify it into the family that has the most similar capability vector with it. Then we compute the performance based on the joint capabilities between predicted families and the ground truth families, as shown in Table 5.

In Table 6, we present the hamming loss and accuracy score of A3CM on zero-day-family malware samples using SVM and DT. Our proposed A3CM achieve 30% and 12% accuracy improvement using DT and SVM, respectively. The reason is that the trained baseline multi-class malware family classification model can only learn knowledge from the known families' training data, and generalize marginally well on the zero-day-families' data. In addition, the DNN achieves the highest 56% accuracy score. Thus it is reasonable to state that DNN has greater power in annotating the malicious capabilities of the zero-day-family malware samples.

Fig. 3 reports the comparison results of precision score, recall score and F1 score between A3CM and the baseline malware family classification method using SVM. Our proposed A3CM beats the baseline method on all the metrics. We further explore the reason of the performance gap

TABLE 6. The hamming loss and accuracy score of A3CM and the baseline malware family classification method on zero-day-family malware samples using SVM, DT and DNN.

Method \ Metrics	Hamming loss	Accuracy score
Baseline-SVM	0.34	0.25
A3CM-SVM	0.25	0.37
Baseline-DT	0.28	0.29
A3CM-DT	0.15	0.56
A3CM-DNN	**0.11**	**0.63**

TABLE 7. The hamming loss and accuracy score of A3CM on known malware samples using SVM, DT and DNN.

Classifier \ Metrics	Hamming loss	Accuracy score
Support Vector Machine	0.005	0.99
Decision Tree	0.02	0.96
Deep Neural Network	**0.00**	**1.00**

TABLE 8. The precision score (PS), recall score (RS) and F1 score of A3CM on known malware samples using SVM, DT and DNN.

Capability \ Metrics	DT-PS	SVM-PS	DNN-PS	DT-RS	SVM-RS	DNN-RS	DT-F1	SVM-F1	DNN-F1
Botnet attack	0.98	0.99	**1.00**	0.97	0.99	**1.00**	0.97	0.99	**1.00**
Unauthorized root access	0.98	0.98	**1.00**	0.96	0.99	**1.00**	0.97	0.99	**1.00**
Malicious SMS charge	0.98	**1.00**	**1.00**	0.99	**1.00**	**1.00**	0.99	1.00	**1.00**
Information stealing	0.99	0.99	**1.00**	0.99	**1.00**	**1.00**	0.99	**1.00**	**1.00**

TABLE 9. The precision score (PS), recall score (RS) and F1 score (F1) of A3CM and the baseline malware multi-class family classification on small-size-families' malware using Decision Tree.

Capability \ Metrics	Baseline-PS	A3CM-PS	Baseline-RS	A3CM-RS	Baseline-F1	A3CM-F1
Botnet attack	0.80	**0.95**	0.81	**0.97**	0.83	**0.96**
Unauthorized root access	0.77	**0.95**	0.83	**0.98**	0.81	**0.97**
Malicious SMS charge	0.83	**0.99**	0.80	**0.95**	0.81	**0.97**
Information stealing	0.81	**0.98**	0.85	**0.99**	0.85	**0.99**

between A3CM and the baseline multi-class malware family classification method on the zero-day-family scenario. Firstly, due to the baseline model is trained on the known families' malware sample while tested on zero-day-families' data, the model cannot learn valid information to annotate the unseen malware samples. Secondly, for our proposed A3CM, although the training data and testing data are with different family labels, the training families and testing families' malware data share certain capabilities. Thus the capability annotation model can learn effective knowledge from training data to identify different capabilities. In summary, the performance of A3CM is promising in annotating the security/privacy-related capabilities on the zero-day-families' malware.

D. ANNOTATION PERFORMANCE ON KNOWN MALWARE SAMPLES

To answer the second research question, we evaluate the performance of our proposed capability annotation on known malware samples. The created dataset with capability ground truth is randomly selected into the training set (contains 4,823 samples) and the testing set (contains 2,076 samples) with the ratio of 7 to 3. In this way, we can calculate the capability annotation performance metrics. From the results reported in Tables 7 and 8, we can see that the performance is inspiring with the F1 score greater or equal to 0.96. The proposed security/privacy-related capability annotation solution A3CM is effective in annotating the known malware samples. The annotation performance of Linear SVM is superior to that of DT. A plausible reason is that our feature representations for malware samples are very sparse and high-dimensional, while Linear SVM is powerful in dealing with such sparse data.

Meanwhile, as a baseline method, we perform the malware multi-class family classification experiment using the same training and testing sets, while the multi-label capability vectors are replaced with single family labels. We employ DT classifier to deliver the final classification results as shown in Fig. 4. As we can see, the classification results of some

TABLE 10. The hamming loss and accuracy score of A3CM and the baseline multi-class family classification on small-size-families' malware using DT, SVM and DNN.

Method \ Metrics	Hamming loss	Accuracy score
Baseline	0.09	0.81
A3CM-DT	0.02	0.95
A3CM-SVM	0.05	0.90
A3CM-DNN	**0.01**	**0.98**

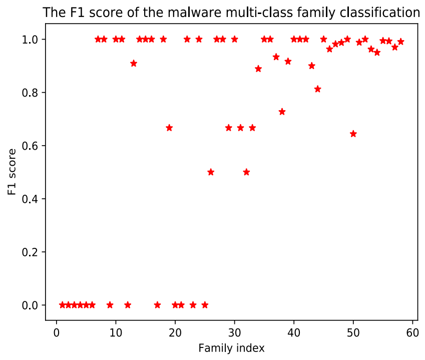


FIGURE 4. The F1 score of the multi-class family classification on known malware samples using Decision Tree.

families are satisfactory (the F1 score is close to 1.0). However, it is clear to see that the classification performance of a large proportion of family classes is extremely poor. The reason could be that these families are uncommon, thus it is hard to collect enough training samples. Thus the performance will further decline when more families are added.

As mentioned in the previous section, multi-class malware family classification cannot work effectively on those family classes with little samples. In this paper, we define those families with less than 100 malware samples as *small-size malware families*. Next we will compare the annotation performance of A3CM and the baseline multi-class malware family classification on small-size-families.

E. DEEP ANALYSIS OF THE IDENTIFIED CAPABILITIES

Table 10 and Table 9 present the comparative results on small-size-families' samples between A3CM and the baseline

TABLE 11. The top 10 significant features contributed to the security/privacy-related capabilities annotation of the first Case, the left column is the top features and the corresponding feature scores for the *Botnet attack* capability, while the right column reports the top features and the corresponding feature scores for the *Information stealing* capability.

Feature name	Score	Feature name	Score
RequestedPermission_android.permission.WRITE_CONTACTS	0.18	UsedPermissions_android.permission.RESTART_PACKAGES	0.30
IntentFilter_android.intent.action.USER_PRESENT	0.12	RequestedPermission_android.permission.CALL_PHONE	0.13
UsedPermissions_android.permission.DISABLE_KEYGUARD	0.10	UsedPermissions_android.permission.DISABLE_KEYGUARD	0.13
UsedPermissions_android.permission.INTERNET	0.10	UsedPermissions_android.permission.ACCESS_NETWORK_STATE	0.08
IntentFilter_android.intent.action.BATTERY_OKAY	0.07	RequestedPermission_android.permission.WRITE_SMS	0.06
IntentFilter_android.intent.action.BATTERY_LOW	0.07	IntentFilter_android.provider.Telephony.WAP_PUSH_RECEIVED	0.05
RequestedPermission_android.permission.CALL_PHONE	0.07	IntentFilter_android.intent.action.USER_PRESENT	0.05
RequestedPermission_android.permission.ACCESS_WIFI_STATE	0.06	IntentFilter_android.intent.action.BATTERY_OKAY	0.05
SuspiciousApi_Landroid/telephony/TelephonyManager.getSubscriberId	0.06	IntentFilter_android.intent.action.BATTERY_LOW	0.05
RequestedPermission_android.permission.WRITE_SMS	0.05	IntentFilter_android.intent.action.ACTION_POWER_CONNECTED	0.04

TABLE 12. The top 10 significant features contributed to the security/privacy-related capabilities annotation of Case 2, the left column is the top features and the corresponding feature scores for the correctly annotated *Malicious SMS charge* capability, while the right column reports the top features and the corresponding feature scores for the missing annotated *Information stealing* capability.

Feature name	Score	Feature name	Score
RestrictedApi_android.telephony.gsm.SmsManager.sendTextMessage	0.15	SuspiciousApi_Lorg/apache/http/client/methods/HttpPost	0.08
SuspiciousApi_Landroid/telephony/gsm/SmsManager.sendTextMessage	0.13	RestrictedApi_android.telephony.gsm.SmsManager.sendTextMessage	0.08
Activity_com.google.devtools.simple.runtime.components.android.WebViewActivity	0.08	RestrictedApi_android.media.MediaPlayer.start	0.06
Activity_com.google.devtools.simple.runtime.components.android.ListPickerActivity	0.08	RestrictedApi_android.media.MediaPlayer.stop	0.06
Activity_.Screen1	0.08	SuspiciousApi_Ljava/lang/Runtime;->exec	0.06
RestrictedApi_android.location.LocationManager.getProviders	0.06	RestrictedApi_android.media.MediaPlayer.release	0.04
RestrictedApi_android.location.LocationManager.requestLocationUpdates	0.06	RestrictedApi_android.os.Vibrator.vibrate	0.03
SuspiciousApi_Landroid/app/Activity.getSystemService	0.04	SuspiciousApiList_Landroid/content/Context.getSystemService	0.01
RestrictedApi_android.media.MediaPlayer.start	0.04	RestrictedApi_android.widget.VideoView.setVideoURI	0.01
RestrictedApi_android.media.MediaPlayer.release	0.04	RequestedPermission_android.permission.INTERNET	0.01

malware family classification method. We can see that the annotation performance of our proposed A3CM is superior to that of the baseline malware family classification on small-size-families' malware. To investigate the reason of the performance gap, we check those malware samples which were incorrectly annotated. We find that almost all the malware samples within families containing less than 10 samples were incorrectly annotated by the baseline malware family classification. It implies that a sufficient number of samples per family is required to guarantee the performance of multi-class malware family classification. However, due to different malware families share certain security/privacy-related capabilities, the impact of **small-size-family** can be mitigated by the sharing capabilities. So A3CM can work more effectively. In summary, on the **small-size-family** scenario, the annotation performance of A3CM with DNN is better than that of the baseline multi-class malware family classification, with an approximately 17% accuracy improvement.

From the above mentioned experimental results, we can draw the following conclusions: firstly, the malware family classification technique is ineffective when dealing with those uncommon families with few malware samples. With the rapid increase of malware family class, the malware family classification will be far from practical. Secondly, the proposed malware capability annotation solution A3CM is effective in annotating the capabilities for the known malware samples. Thus, our proposed framework MCA has more potentials than the malware family classification technique.

In this part, we present that how the above annotation results assist the further analysis of the malware. We report the top 10 influential features, sorted by their absolute values of the product of feature values and the corresponding feature weights, for 3 distinct malware samples annotated by the Linear SVM.

Case 1. The first example is a malware sample (hash name 3c6dfa528d0e26d35bd96fd92f41fca3eb95d6a56cb9f631b68831a965b5e4b6) belonging to the *BaseBridge* family. In the feature engineering part, we extract 54 string features (including Used permissions, Suspicious API calls, Requested permissions) through static analysis. This sample has *Botnet attack* and *Information stealing* capabilities, which is also correctly annotated by A3CM using Linear SVM. In Table 11, we report the top 10 significant features for each predicted capability. There exist 6 identical top features for both two capabilities. Thus A3CM searches and locates the 16 potential malicious codes in the disassembled code project, the security analysts then give priority to audit these 16 potential localization instead of the total 54 localization.

Case 2. In this case, we select an example with the *Malicious SMS charge* capability being correctly annotated, while *Information stealing* capability being missed annotated. The sample belongs to *Steek* family and its hash name is 6d0ef0a20210eba44135968b044bec5221282d2f03a55216c4f1feb7813e4fa8. The left column of Table 12 demonstrates the top 10 features of the truly annotated *Malicious SMS charge* capability. It is noted that the two sensitive SMS involved APIs are listed as top 2 features. The right column in Table 12 presents the top 10 important features of the missing annotated *Information stealing* capability. We can identify 16 unique features from Table 12, which indicates 16 suspect malicious codes localization in the disassembled code project. Therefore the security analysts can manually

TABLE 13. The top 10 significant features contributed to the security/privacy-related capabilities annotation of Case 3, the left column is the top features and the corresponding feature scores for the *Unauthorized root access* capability, while the right column reports the top features and the corresponding feature scores for the *Information stealing* capability.

Feature name	Score	Feature name	Score
SuspiciousApi_Ljava/lang/Runtime;->exec	0.23	IntentFilter_android.intent.action.PACKAGE_REMOVED	0.15
SuspiciousApi_Landroid/net/wifi/WifiManager.setWifiEnabled	0.12	RequestedPermission_com.android.launcher.permission.UNINSTALL_SHORTCUT	0.14
SuspiciousApi_Landroid/telephony/TelephonyManager.getSubscriberId	0.11	SuspiciousApi_Lorg/apache/http/client/methods/HttpPost	0.10
RequestedPermission_android.permission.WRITE_APN_SETTINGS	0.11	IntentFilter_android.intent.action.PACKAGE_ADDED	0.09
SuspiciousApi_Landroid/content/pm/PackageManager.getPackageInfo	0.10	RequestedPermission_android.permission.KILL_BACKGROUND_PROCESSES	0.09
RequestedPermission_android.permission.CHANGE_NETWORK_STATE	0.10	UsedPermissions_android.permission.ACCESS_NETWORK_STATE	0.07
UsedPermissions_android.permission.CHANGE_WIFI_STATE	0.09	SuspiciousApi_Ljava/lang/Runtime;->exec	0.07
RequestedPermission_android.permission.CHANGE_WIFI_STATE	0.07	HardwareComponents_android.hardware.telephony	0.06
UsedPermissions_android.permission.INTERNET	0.07	UsedPermissions_android.permission.GET_TASKS	0.06
IntentFilter_android.intent.action.PACKAGE_ADDED	0.07	IntentFilter_com.android.vending.INSTALL_REFERRER	0.05

TABLE 14. The top 10 significant features contributed to the wrongly annotated *Botnet attack* capabilities of Case 3.

Feature name	Score
IntentFilter_android.intent.action.PACKAGE_ADDED	0.12
RequestedPermission_android.permission.KILL_BACKGROUND_PROCESSES	0.12
UsedPermissions_android.permission.GET_TASKS	0.12
IntentFilter_android.intent.action.USER_PRESENT	0.11
UsedPermissions_android.permission.INTERNET	0.09
RequestedPermission_com.android.launcher.permission.UNINSTALL_SHORTCUT	0.09
SuspiciousApi_Ljava/lang/Runtime;->exec	0.09
SuspiciousApi_Landroid/net/wifi/WifiManager.setWifiEnabled	0.09
IntentFilter_android.intent.action.PACKAGE_REMOVED	0.08
RequestedPermission_android.permission.GET_TASKS	0.08

prioritize to check these indicated codes localization for locating the truly malicious codes.

Case 3. The third example is a malware sample (hash name e34f4d6e0b6c3ca9afdcdbf918db5263) with the family class label *GingerMaster*. Through the static analysis, 103 features, such as, security permissions, sensitive API calls, etc. were extracted to characterize the capabilities of this sample. This sample is correctly annotated by A3CM to have *Unauthorized root access* and *Information stealing* capabilities, while misclassified to have *Botnet attack* capability. In Table 13, we present the top 10 features essential to the correctly predicted *Unauthorized root access* and *Information stealing* capabilities. Several suspicious APIs and permissions are listed which indicate the *authorized root access*, e.g., *setWifiEnabled*, *WRITE_APN_SETTINGS*, *CHANGE_NETWORK_STATE or WIFI_STATE*. Table 14 shows the top 10 key features which account for the incorrectly annotation of the *Botnet attack* capability. Totally there are 20 unique top ranked features, which means 20 indicated potential malicious code localization. Thus the security analysts can focus on these 20 code snippets to confirm whether they are malicious or not.

F. ANNOTATION PERFORMANCE COMBINING THE STATIC ANALYSIS AND DYNAMIC BEHAVIORAL FEATURES

To cover more characteristics of malware, in the experimental part, as a comparison approach, we combine both the static features and the dynamic behavioral features with the aim to address the malicious capability annotation of malware with native code or dynamic loading library. In the experiment, we added a Subsection to present the performance of security/privacy-related capability annotation using both the static and the dynamic behavioral features. Specifically, each Android malware is firstly executed in the DroidBox[8] dynamic analysis Sandbox for 3 minutes, and the dynamic behavioral information (e.g., network data, file read and write operations, started services and loaded classes, information leaks via network, circumvented permissions, cryptographic operations, listing broadcast receivers and sent SMS and phone calls) is tracked and logged during the execution process [28]. Besides, the Strace is used to log the system calls while the Android application is executed. These extracted behavioral features are vectorized and then fused with the static analysis feature vectors. The fused vectors are input to train and test the SVM, Decision Tree and DNN classifiers. The security/privacy related capability annotation performance on three scenarios (annotation on zero-day-family malware, annotation on known malware and annotation on small-size-families' malware) is presented in Table 15, Table 16 and Table 17. It can be seen that combining both the static analysis and dynamic behavioral features do facilitate the security/privacy-related capability annotation performance on both scenarios. Compared with

TABLE 15. The hamming loss and accuracy score of A3CM on zero-day-family malware samples using SVM, DT and DNN using both the static analysis and dynamic behavioral features.

Metrics / Method	Hamming loss	Accuracy score
A3CM-SVM	0.22	0.41
A3CM-DT	0.11	0.62
A3CM-DNN	**0.09**	**0.67**

[8]https://github.com/pjlantz/droidbox

TABLE 16. The hamming loss and accuracy score of A3CM on known malware samples using SVM, DT and DNN using both the static analysis and dynamic behavioral features.

Method \ Metrics	Hamming loss	Accuracy score
A3CM-SVM	0.002	1.00
A3CM-DT	0.01	0.97
A3CM-DNN	**0.00**	**1.00**

TABLE 17. The hamming loss and accuracy score of A3CM on small-size-families' malware samples using SVM, DT and DNN using both the static analysis and dynamic behavioral features.

Method \ Metrics	Hamming loss	Accuracy score
A3CM-SVM	0.02	0.94
A3CM-DT	0.01	0.97
A3CM-DNN	**0.005**	**0.99**

the annotation performance merely using the static analysis features in Subsection C and Subsection D, adding the dynamic behavioral features will derive 1% to 6% accuracy improvement. However, since the dynamic analysis is time consuming (in this paper, each Android application is executed 3 minutes in the Sandbox). From the efficiency perspective, the static analysis features are used in real-deployment.

G. WHY A3CM WORK

In this subsection, we briefly present why A3CM works based on the experimental results. Firstly, A3CM predicts the security/privacy-related capabilities of the detected malware instead of categorizing it to any families. Thus A3CM is able to deal with the unknown or zero-day-family malware.

Secondly, A3CM extracts semantic features, such as suspicious API calls, requested permissions, to predict the capabilities. The security/privacy-related capabilities are related to the malicious behaviors of malware samples, while malicious behaviors are performed through certain suspicious APIs, permissions, and systems calls. Thus it is possible to capture the patterns of security/privacy-related capabilities using the semantic features.

VII. LIMITATIONS ANALYSIS

In this paper, we attempt to address a novel and challenging research question: How to automatically annotate the security/privacy-related capabilities for the detected Android malware. However, due to the lack of sufficient data with ground truth. Our solutions to the research issue the following limitation: In the experiments, our dataset is aggregated based on two open source datasets, and the ground truth is captured from a security blog released by security expert Michael Spreitzenbarth. Our limitations lie that the ground truth is on family granularity. The latest malware with ground truth should be included to cover the current landscape of the Android malware.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we propose a new technique to automatically identifying the security/privacy related capabilities of any detected malware, which named *Malware Capability Annotation* (MCA). Then we propose a new MCA solution, Automatic Capability Annotation for Android Malware (A3CM) to automatically predict the capabilities of newly detected Android malware. A3CM can annotate the capabilities of malware using the semantic features extracted from the raw software package. The multi-label classification techniques are employed to capture the relations between semantic features of malware and capability vectors. To the best of our knowledge, A3CM is the first work that can infer the Android malware's capability vectors. We also created the first Android malware dataset with the capability ground truth. Our method achieves satisfactory performance in inferring the capability vectors of known Android malware, small-size-families' malware and zero-day-families' Android malware, respectively. Compared with traditional malware family class classification methods, our work is able to annotate the security/privacy-related capabilities for those zero-day-families' malware.

In the future, we will extend the malware dataset to cover the latest landscape of the malware and investigate more reliable security/privacy-related capability ground truth.

REFERENCES

[1] StatCounter. (2018). *Mobile Operating System Market Share Worldwide*. Accessed: Mar. 19, 2018. [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide

[2] G. Suarez-Tangil and G. Stringhini, "Eight years of rider measurement in the Android malware ecosystem: Evolution and lessons learned," *CoRR*, vol. abs/1801.08115, pp. 1–18, Jan. 2018. [Online]. Available: http://arxiv.org/abs/1801.08115

[3] 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.*, vol. 21, no. 2, pp. 1744–1772, 2nd Quart., 2018.

[4] L. Ma, X. Liu, Q. Pei, and Y. Xiang, "Privacy-preserving reputation management for edge computing enhanced mobile crowdsensing," *IEEE Trans. Services Comput.*, vol. 12, no. 5, pp. 786–799, Sep./Oct. 2019.

[5] J. Qiu, W. Luo, L. Pan, Y. Tai, J. Zhang, and Y. Xiang, "Predicting the impact of Android malicious samples via machine learning," *IEEE Access*, vol. 7, pp. 66304–66316, 2019. doi: 10.1109/ACCESS.2019.2914311.

[6] 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.

[7] 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 (KDD)*, 2017, pp. 1507–1515.

[8] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De 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/. doi: 10.14722/ndss.2017.23353.

[9] 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.

[10] S. Wen, M. Sayad 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.

[11] 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)*, 2014, pp. 576–587.

[12] G. Meng, Y. Xue, C. Mahinthan, A. Narayanan, Y. Liu, J. Zhang, and T. Chen, "Mystique: Evolving Android malware for auditing anti-malware tools," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur. (AsiaCCS)*, 2016, pp. 365–376.

[13] 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/. doi: 10.14722/ndss.2017.23379.

[14] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "A multi-view context-aware approach to Android malware detection and malicious code localization," *Empirical Softw. Eng.*, vol. 23, no. 3, pp. 1222–1274, 2017.

[15] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2017, pp. 252–276.

[16] 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.

[17] 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.

[18] Y. Wang, W. Meng, W. Li, Z. Liu, Y. Liu, and H. Xue, "Adaptive machine learning-based alarm reduction via edge computing for distributed intrusion detection systems," *Concurrency Comput., Pract. Exper.*, vol. 31, no. 19, 2019, Art. no. e5101.

[19] L. Tang, W. Ma, M. Grobler, W. Meng, Y. Wang, and S. Wen, "Faces are protected as privacy: An automatic tagging framework against unpermitted photo sharing in social media," *IEEE Access*, vol. 7, pp. 75556–75567, 2019.

[20] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, "A multimodal malware detection technique for Android IoT devices using various features," *IEEE Access*, vol. 7, pp. 64411–64430, 2019.

[21] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, and M. Conti, "Detecting Android malware leveraging text semantics of network flows," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1096–1109, May 2018.

[22] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 355–369, Mar. 2018.

[23] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.* Cham, Switzerland: Springer, 2013, pp. 86–103.

[24] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, vol. 14, 2014, pp. 23–26.

[25] Z. Zhu and T. Dumitraş, "FeatureSmith: Automatically engineering features for malware detection by mining the security literature," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2016, pp. 767–778.

[26] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "Make evasion harder: An intelligent Android malware detection system," in *Proc. IJCAI*, 2018, pp. 5279–5283.

[27] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik opcode graph based Android malware variants detection using global topology features," *IEEE Access*, vol. 6, pp. 51964–51974, 2018.

[28] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic Android malware detection system with ensemble learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018.

[29] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "DroidEnsemble: Detecting Android malicious applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.

[30] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant permission identification for machine-learning-based Android malware detection," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[31] S. Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for Android malware detection," *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 453–466, Feb. 2019.

[32] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient Android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, pp. 69246–69256, 2019.

[33] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for Android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.

[34] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 3, pp. 773–788, Mar. 2019.

[35] F. Shen, J. Del Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android malware detection using complex-flows," *IEEE Trans. Mobile Comput.*, vol. 18, no. 6, pp. 1231–1245, Jun. 2019.

[36] H. Cai, N. Meng, B. G. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, Jun. 2019.

[37] L. Deshotels, V. Notani, and A. Lakhotia, "DroidLegacy: Automated familial classification of Android malware," in *Proc. ACM SIGPLAN Program Protection Reverse Eng. Workshop*, 2014, Art. no. 3.

[38] 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. (CCS)*, 2014, pp. 1105–1116.

[39] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proc. 26th Usenix Secur. Symp. (USENIX Secur.)*, 2017, pp. 625–642.

[40] W. Wang, M. Zhao, Z. Gao, G. Xu, H. Xian, Y. Li, and X. Zhang, "Constructing features for detecting Android malicious applications: Issues, taxonomy and directions," *IEEE Access*, vol. 7, pp. 67602–67631, 2019.

[41] F. Alswaina and K. Elleithy, "Android malware permission-based multi-class classification using extremely randomized trees," *IEEE Access*, vol. 6, pp. 76217–76227, 2018.

[42] A. Atzeni, F. Díaz, A. Marcelli, A. Sánchez, G. Squillero, and A. Tonda, "Countering Android malware: A scalable semi-supervised approach for family-signature generation," *IEEE Access*, vol. 6, pp. 59540–59556, 2018.

[43] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 8, pp. 1890–1905, Aug. 2018.

[44] G. Suarez-Tangil and G. Stringhini, "Eight years of rider measurement in the Android malware ecosystem: Evolution and lessons learned," 2018, *arXiv:1801.08115*. [Online]. Available: https://arxiv.org/abs/1801.08115

[45] P. M. Comar, L. Liu, S. Saha, P.-N. Tan, and A. Nucci, "Combining supervised and unsupervised learning for zero-day malware detection," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2013, pp. 2022–2030.

[46] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android HIV: A study of repackaging malware for evading machine-learning detection," *IEEE Trans. Inf. Forensics Security*, to be published.

[47] W. Li, Z. Wang, J. Cai, and S. Cheng, "An Android malware detection approach using weight-adjusted deep learning," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Maui, HI, USA, Mar. 2018, pp. 437–441.

[48] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digit. Invest.*, vol. 24, pp. S48–S59, Mar. 2018.

[49] M. Spreitzenbarth. (2016). *Current Android Malware*. Spreitzenbarth Consultants. Accessed: Mar. 22, 2018. [Online]. Available: https://forensics.spreitzenbarth.de/android-malware/

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

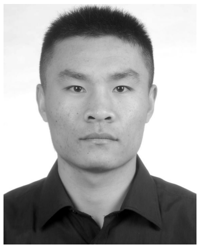
[51] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Proc. 1st Instructional Conf. Mach. Learn.*, vol. 242, Dec. 2003, pp. 133–142.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[53] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Int. J. Data Warehousing Mining*, vol. 3, no. 3, p. 13, 2006.

[54] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2006, pp. 153–160.

[55] W.-X. Liu, J. Cai, Y. Wang, Q. C. Chen, and D. Tang, ''Mix-flow scheduling using deep reinforcement learning for software-defined data-center networks,'' *Internet Technol. Lett.*, vol. 2, no. 3, p. e99, 2019.
[56] P. Szymański and T. Kajdanowicz, ''A scikit-based Python environment for performing multi-label classification,'' 2017, *arXiv:1702.01460*. [Online]. Available: https://arxiv.org/abs/1702.01460
[57] M.-L. Zhang and Z.-H. Zhou, ''A review on multi-label learning algorithms,'' *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1819–1837, Aug. 2014.
[58] E. Gibaja and S. Ventura, ''A tutorial on multilabel learning,'' *ACM Comput. Surv.*, vol. 47, no. 3, p. 52, 2015.



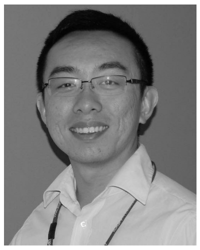
JUNYANG QIU is currently pursuing the Ph.D. degree with the School of Information Technology, Deakin University. His current research interests include cyber security and machine learning.

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 also the Deputy Director of Swinburne Cybersecurity Lab, Swinburne University of Technology, Australia. He has published more than 100 research articles 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. His current research interests include cybersecurity and applied machine learning. He has been internationally recognized as an Active Researcher in cybersecurity, evidenced by his chairing of eight international conferences, since 2013, and presenting of invited keynote addresses in two conferences, and an Invited Lecture of the IEEE SMC Victorian Chapter.

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 articles in refereed international journals and conferences. His current 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. He has published more than 30 research articles in refereed international journals and conferences, such as the IEEE Security & Privacy, the *Journal of Multimedia*, and *Digital Investigation*. His current research interests include cyber security and privacy.

SURYA NEPAL received the B.E. degree from the National Institute of Technology at Surat, India, the M.E. degree from the Asian Institute of Technology, Bangkok, Thailand, and the Ph.D. degree from RMIT University, Australia. He is currently a Principal Research Scientist with CSIRO Data61. His current research interests include the development and implementation of technologies in distributed systems including web services, cloud computing, and the Internet-of-Things (ioT), especially on security, privacy, and trust.



YU WANG received the Ph.D. degree in computer science from Deakin University, VIC, Australia. He is currently an Associate Professor with the School of Computer Science, Guangzhou University, China. His current research interests include network traffic analysis, mobile networks, social networks, and cyber security.

YANG XIANG received the Ph.D. degree in computer science from Deakin University, Australia. He is currently the Dean of the Digital Research Innovation Capability Platform, Swinburne University of Technology, Australia. He has published more than 200 research articles 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). His current research interests include 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 to 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 is a Co-Founder and the Steering Committee Chair of the NSS, ICA3PP, CSS, and SocialSec conference series. He served as an Associate Editor for the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, Security and Communication Networks (Wiley), and an Editor of the *Journal of Network and Computer Applications*. He is the Foundation Editor-in-Chief of the *SpringerBriefs on Cyber Security Systems and Networks*. He is the Coordinator at Asia for the IEEE Computer Society Technical Committee on Distributed Processing (TCDP).