Received November 30, 2017, accepted January 28, 2018, date of publication February 13, 2018, date of current version March 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2805783

Consortium Blockchain-Based Malware Detection in Mobile Devices

JINGJING GU[1], BINGLIN SUN[1], XIAOJIANG DU[2], (Senior Member, IEEE), JUN WANG[1], YI ZHUANG[1], AND ZIWANG WANG[1]

[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 200016, China
[2]Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA

Corresponding authors: Jingjing Gu (gujingjing@nuaa.edu.cn) and Xiaojiang Du (dxj@ieee.org)

This work was supported by the National Science Foundation of China under Grant 61572253.

ABSTRACT To address the problem of detecting malicious codes in malware and extracting the corresponding evidences in mobile devices, we construct a consortium blockchain framework, which is composed of a detecting consortium chain shared by test members and a public chain shared by users. Specifically, in view of different malware families in Android-based system, we perform feature modeling by utilizing statistical analysis method, so as to extract malware family features, including software package feature, permission and application feature, and function call feature. Moreover, for reducing false-positive rate and improving the detecting ability of malware variants, we design a multi-feature detection method of Android-based system for detecting and classifying malware. In addition, we establish a fact-base of distributed Android malicious codes by blockchain technology. The experimental results show that, compared with the previously published algorithms, the new proposed method can achieve higher detection accuracy in limited time with lower false-positive and false-negative rates.

INDEX TERMS Consortium Blockchain, malware detection, multi-feature.

I. INTRODUCTION

Malware detection in intelligent mobile devices has always been a challenging issue, especially on the efficient and open-source Android platform. Android-based system, as one of the most popular mobile operating systems, was a palpable target of malicious developers. In the 2016 Mobile Malware Evolution Report [1], Kaspersky lab detected 8,526,221 malicious installation packages, 128,886 mobile banking Trojans, and 261,214 mobile ransomware Trojans, which have been significantly increased compared with those of the previous year. During the whole 2016, Kaspersky's lab has recorded nearly 40 million mobile malware attacks and protected 4,018,234 users' Android devices. Although Google has developed Bouncer tool to perform malice detection on applications of Google Play, attackers could still bypass Bouncer to send malware into Google Play [2]. On the other hand, due to many third-party software and active software forums, the open source of Android-based system provides basis for the spread of malware and makes it difficult to monitor the quality of software.

Thus, the detection of malware and classification of malicious codes have become an important research work. Generally, the existing malware detecting technologies for Android devices could be divided into two categories: static-based analysis and dynamic-based analysis. The static-based analysis method used the decompiler technology or performance analysis of the control flow and data flow in the smali intermediate code, which was applicable for automatic analysis through a large amount of software samples. But it could not solve the problems of code obfuscation, encryption, and other issues, which could only be performed in dynamic execution [3], [4]. The dynamic-based analysis method stimulated the execution of software to avoid the code obfuscation and encryption problems. However, the coverage of its dynamic testing code always is not enough. Besides, some malicious programs might camouflage themselves when running under simulators [5], [6]. Moreover, there were a variety of malware in different families with various features, which also increased the difficulty of detection. At the same time, the extraction of some features in the existing methods required high time cost. Therefore, we need to continue to delve into the research of malicious code or malware detecting technology in Android-based devices.

Recently, the blockchain technology, as a new type of distributed computing paradigm, has gained much importance due to its high efficiency, high data security, high credibility and low cost. The core characteristic of the blockchain technology is "de-centration", which can effectively build programmable currency and energy [7]. On the basis of it, we can develop certificate store, digital property protection and many other applications. Thus, it is a key technology to lead the conversion from the information Internet to the value Internet [8]. The research of the blockchain technology can be divided into three categories: the public blockchain, the private blockchain and the consortium blockchain [9]. Differing from the former two, the consortium blockchain may allow each member to be read, be limited to participants, and take a mixed route. Thus, the consortium blockchain can be regarded as "partially de-centralized" [10].

A. RELATED WORKS

It is always a hot issue for the related security problem in mobile devices and wireless network, which has been studied by many researchers [11]–[15]. Nowadays, one of the typical methods in malware detection is the feature extraction like signature and permission information [16]–[18]. For instance, the Ensemble Learning (EL) extended the feature set of detection and proposed the classification way of 179 features, including API calls, instructions and authorities for detecting zero-day Android malicious code [16]. In literature [17], the selected benign software in Android system was first analyzed, and then used to obtain the API call flow and build the trigger metric of the API users. The mobile malware detection method ANNCMDroid [18] was based on co-occurrence matrices and artificial neural networks, which took into account the relations among sequences of system call. The other common methods in Android-based devices were some works based on Java code and Soot framework optimization [19]–[21]. For example, the automated reasoning tool StubDroid that studied data flow summarization information [19]. The static stain analysis tool—DidFail [21] was developed by combining the inter-component communication detection engines FlowDroid and Epicc [20].

Generally, the malware detecting technologies for Android devices can be divided into static-based analysis and dynamic-based analysis. The static-based analysis method can implement efficient and automatic analysis in some ways. However, it cannot detect code obfuscation and encryption, and it is insufficient to decrypt malicious code in dynamic execution. Also, it uses a coarse-grained detecting approach of information flows between applications, which is easy to produce false positive [21]. The dynamic-based analysis method of Android-based software collects applications' behavior information during its operation. They can solve the problems such as code obfuscation and encryption. Nevertheless, malware usually have a well-designed triggering mechanism when facing dynamic tests, while some malicious programs can detect their own operating environments and automatic crash behaviors when running under the simulator.

In addition, using a single feature to determine software's malice is far below satisfaction [22]. At the same time, the extraction of some features in the existing methods requires high time cost [23]. For example, the extraction of API context with Appcontext requires a lot of time and memory, and its experiment just dealt with samples whose software packages were less than 5 MB [24].

Recently, the blockchain technology has gained much more focus. Its key technology derives from the consensus mechanism, which is an example of a distributed computing system with high fault tolerance [7]. The common technologies are: Proof of Work (PoW), Proof of Stake (PoS), Practical Byzantine Fault Tolerance (PBFT), Delegated Proof of Stake (DPOS) and so on [25]. Generally, the research of the blockchain technology can be divided into three categories: the public blockchain, the private blockchain and the consortium blockchain [9]. From the public blockchain, anyone can read and send the transaction, which can be effectively recognized, and anyone can participate in the consensus process. Thus, in the public blockchain, all records are visible to the public and everyone can participate in the consensus process. In the private blockchain, only those users from specific organizations can be allowed to participate in the consensus process. In the consortium blockchain, the consensus process is controlled by preselected nodes, which maintain a copy of the distributed data store. That is, only a set of preselected users can participate in the consensus process. The consortium blockchain is a community of N member organizations, each of which runs a node. And in order for each block to take effect, it requires the confirmation of 2/3 of the organizations. Thanks to the flexibility, the consortium blockchain technology has been applied to both financial and non-financial systems.

B. CONTRIBUTION

- First, we propose a framework of Consortium Blockchain for Malware Detection and Evidence Extraction (CB-MDEE) in mobile devices. The framework is composed of two parts of mixed chains: detecting consortium chain by test members and public chain by users. As is well known, there was no relevant work using the consortium blockchain for security detection that has been published before.
- We analyze different malware families on the basis of Android-based systems and build a corresponding Multi-Feature Model (*MFM*) by adopting a fuzzy comparison method. In order to reduce false-positive rate and improve the detecting ability of malware variants, we propose multiple marking functions. From this model, we can extract the features to construct the feature database, and develop a multi-feature detection algorithm.
- We establish a fact-base of the Android malicious codes by the blockchain technology to detect malware information, which will be sent to the consortium blockchain for automatically generating new blocks.
- We perform tests to verify our method on 4486 malware samples and 2140 benign software samples collected from real scenario. The experimental results show that

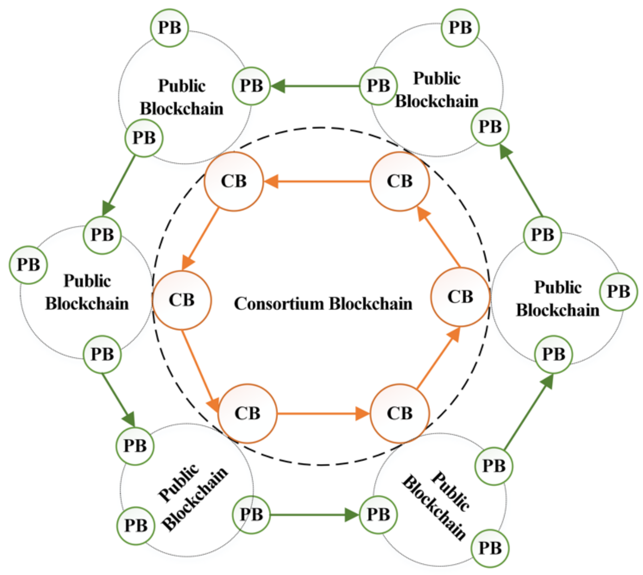


FIGURE 1. The organization structure of the CB-MDEE.

our method can be applied in malware multi-feature classification effectively. Compared with the previously reported algorithms, our CB-MDEE can have better performances on average time cost, detecting accuracy and recall rate.

II. THE ARCHITECTURE OF CONSORTIUM BLOCKCHAIN IN MALWARE DETECTION

Our CB-MDEE framework consists of consortium blockchain and public blockchain, as shown in Fig. 1, where CB represents the Consortium Blockchain and PB represents the Public Blockchain. The CB is the core chain, composed of the members in distributed malware detection organizations. These members build a fact-base of the distributed malicious codes. The PB is the application chain, open for any user who needs to provide detection and evidence services for joining as a member node.

The overall framework of our CB-MDEE is shown in Fig. 2, which includes four layers: the network layer, the storage layer, the support layer, and the application layer.

In the network layer, link nodes can communicate through a P2P network. The network has the characteristics of decentration and dynamic change, which involves ways of networking and mechanism of communication between nodes. The nodes are composed of servers, which are geographically dispersed. There is no central node, and every node can freely join or exit the network. When a testing task is completed, the detecting information and synchronizing block information are usually sent to all CB nodes by relay-repeater mode. Each node sends information to its neighbor node, and the neighbor node forwards the submitted information to its own neighbor node. In this way, it gradually spreads throughout the network. The synchronous block information uses a request-response pattern. The node first sends its own block height (similar to ID) to the neighbor node. If the neighbor node's height is less than this node's, the block requires to obtain the missing information. If the height of the neighbor node is higher, the neighbor node takes reverse block information. All nodes continuously exchange block information with their neighbor nodes.

In the storage layer, features of malicious codes are stored. Factual information is also provided in this layer to form a distributed malware fact-base in the Consortium Blockchain. Each block contains a page of malware features and other information, including block head and block data. Once these blocks are confirmed, they cannot be modified. The block head contains information such as timestamp, Pre-hash, Nonce, etc. Each block uses the hash encryption value from the previous block for validating information. The block data includes specific information related to malicious code features, such as the sensitive behavior set, the permission, and installation package.

In the support layer, the interface between the users and the fact-base of malicious codes are provided, including functions of consensus mechanism, data encryption, digital signature, key management, identity authentication, access control, synchronization management, etc. The consensus mechanism determines the legitimacy of nodes in the Consortium Blockchain, confirms the submitted information, and ensures the legality and validity of block information in the fact-base. The data encryption is the basic function of the security system. The digital signature makes the data submitted by users undeniable, and makes the blockchain capable of keeping the user who submits the illegal data. The key management is used to manage the key information safely and effectively. The access control prevents malicious users from illegally manipulating data. The synchronization management updates the fact-base.

In the application layer, programs and interfaces for various applications are provided. Users can interact with various applications without considering details about the bottom technology of the blockchain. Typical applications include the malware detection, the malicious event detection, the evidence tracking and extraction, and the digital storage applications. The detecting consortium members need to confirm legitimate identities of users. If the test's results, records or evidences are uploaded, a new block will be generated and the related information will be broadcasted.

III. MULTI-FEATURE MODEL OF MALWARE FAMILY

Features of Android-based software can be extracted from different features [3], such as package structure features, application and permission features, system call sequence features, and system call context features.

A. CRITICAL FEATURE REPRESENTATION OF SOFTWARE

The call graph of software function obtained by the static-based analysis method can reflect behavior features of the software. In this paper, we use FlowDroid [20] to build the call graph. The secure sensitive method concerned here has

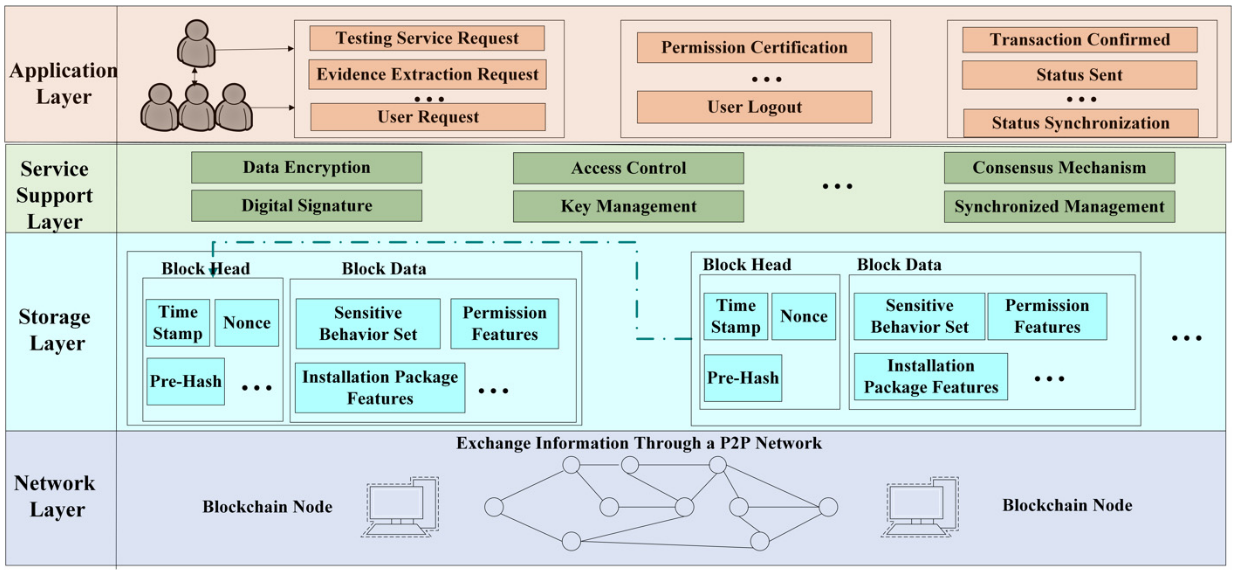


FIGURE 2. The overall framework of CB-MDEE.

been extended on the basis of AppContext [23], which is mainly divided into three parts:

1) Permission-protected: In Android-based systems, some APIs need to apply for corresponding permissions to visit secure sensitive resources in the system. For example, android.telephony.SmsManager.sendTextMessage as a typical call method needs the permission of android. permission.SEND_SMS [25].
2) Source/sink methods of data flows: These methods may generate or send sensitive information, such as getDeviceID(). For a detailed information list, please refer to SuSi [26].
3) Other suspicious methods: Typical methods are dynamic loading functions, reflection functions, encryption and decryption functions, execution of native codes, and call functions, such as invoke() [27]. These methods exist in Android malware and increase the difficulty of software security analysis.

Due to the Android-based system that is driven by events, the constructing call relations between different methods can not only be represented as direct calling codes, but also used by the intent mechanism to implement Inter-Component Communication (ICC). Therefore, it is necessary to analyze software's ICCs and use them to describe the behavior of software. In this paper, we make an extension and optimization based on the call graph of software functions so as to enrich the expression ability of nodes. More importantly, we add supports to ICCs. The definition of a Sensitive Behavior Graph (*SBG*) is given below. For easy of presentation, the basic notations used through out this article are presented in Table 1.

Definition 1: A *SBG* can be described as a quadruple as shown in (1).

$$SBG = (V_d, V_n, E, \mu) \tag{1}$$

TABLE 1. Basic notations used in CB-MDEE.

SYMBOL	DESCRIPTION
SBG	The Sensitive Behavior Graph
$V_{d,}$	The secure sensitive node set of *SBG*
V_n	The not secure sensitive node set of *SBG*
E	The set of edge of *SBG*
SBS	The Sensitive Behavior Set of Android-based software
S_t	The set of secure sensitive methods
CRF	The critical feature representation
F	The 0-1 vector of features
P	The permission list of software applications
MFM	The multi-feature model for mobile devices
v_d	$v_d \in V_d$ is one of the secure sensitive method
v_n	$v_n \in V_n$ is not secure sensitive method
μ,α,β,γ	The marking function

V_d and V_n are subsets of the *SBG*'s node set. Any $v_d \in V_d$ is one of the secure sensitive methods. Each $v_n \in V_n$ is not secure sensitive but directly or indirectly calling secure sensitive methods. $E \subseteq V_n \times V_d$ is a set of edges of a software sensitive graph, which has a call relation between different methods. Each edge $e = (v_n, v_d) \in E$ indicates that the non-secure sensitive method v_n in the software S directly or indirectly calls a secure sensitive method V_d. e also represents a V_n in component C_s and triggers a method V_d in component C_t through ICC. $\mu : V_d \rightarrow \langle ID, EntryType, Para\rangle$ is a marking function to mark contents contained in vertices of a graph, including the methods' *ID*, entry point types *EntryType* and parameters *Para*.

Although the *SBG* of the software in Android-based mobile devices can well describe the behavior features, the software usually has too many functions to make thousands of edges of its function call graph. It has low extraction rate when directly extracting its common features. Thus, we make further processing on the basis of the *SBG* to extract the Sensitive Behavior Set (*SBS*) of Android-based software.

Definition 2: A *SBS* for an Android-based software can be described as a set of (2).

$$SBS = \{S_1, S_2, \cdots, S_m\} \tag{2}$$

Where: $S_t = \{v | (v_t, v) \in E \cap (v_t \in V_n, v \in V_d)\}$ is a set of secure sensitive methods, which indicates that in a *SBG*, the *ith* non-sensitive secure method directly or indirectly call the set of all secure sensitive methods. Here, the length of *SBS* set is $m = |V_n|$.

To a certain extent, the programs' malice is associated with attributes in malware installation packages and permission information of software applications. For example, when a malware tries to get a root permission, it usually has root exploiting database files in the installation package, and then cheat the installation software to set jar files, dex files or apk files in its packages. In order to avoid the detection, some of malware usually modify suffix names of malicious loading files, such as .mp4 and .png, and place them in resource folders (e.g. assets, res) to act as normal files. We can find suspicious resource files through comparing file types with their contents and their suffixes. Permissions of software applications reflect possible behaviors of software. Therefore, it can effectively express software features by analyzing permissions of malware applications.

Definition 3: A Critical Feature Representation (*CFR*) can be described as a triple as shown in (3).

$$CFR = (SBS, F, P) \tag{3}$$

Where, F is a 0-1 vector of features from the software installation package. Software features corresponding to the vector include: whether there is a lib/so file, whether it has a root exploiting database file, whether it has a subroutine (jar, dex, or apk), whether there exists an abnormal file (that is, the file suffix does not match the file type). P is the permission list of software applications.

B. MULTI-FEATURE MODEL OF MALWARE FAMILY

Here, we extract features of malware families and establish a Multi-Feature Model (*MFM*) for mobile devices. Intuitively, if malware samples are from the same family, critical features can be represented as a common structure. However, malware usually has a large number of variants. Although their behaviors have certain similarities, the details are quite different. If selecting common structures of all samples, we only can extract few features which would lead to a higher false-positive rate in the test [3]. We build multiple marking functions and take the probability of certain behaviors in the sample as weights in the malware family. The definition of the *MFM* of a malware family is given below.

Definition 4: A*MFM of a m*alware family for Android-based mobile devices can be described as a six-tuple as shown (4).

$$MFM = \left(SBS^c, \alpha, F^c, \beta, P^c, \gamma\right) \tag{4}$$

Where:

1) $SBS^c = \left\{S_1^c, S_2^c, \cdots, S_m^c\right\}$ is a *SBS* of the malware family C, which can be obtained by statistically analyzing the *SBS* of samples in the same malware family .

2) Marking function $\alpha : S_i^c \rightarrow \lfloor 0, 1 \rfloor$ represents the probability of the sensitive method set in the malware family.

3) F^c is the feature of software installation package of malware family C.

4) Marking function $\beta : f \in \mathrm{F}^c \rightarrow \lfloor 0, 1 \rfloor$ represents the probability that each feature in the F^c is in the malware family sample.

5) P^c is the permission list of frequent applications in the malware family C. It can be obtained by analyzing the permission list P in the same malware family.

6) Marking function $\gamma : p \in P^c \rightarrow \lceil 0, 1 \rceil$ represents the probability that each permission of P^c is in the malware family sample.

C. EXTRACTION OF SECURE SENSITIVE FEATURES IN ANDROID-BASED SYSTEM

Here, we discuss critical features of malware which can be extracted from the *SBS*, installation package features and permission features.

1) SENSITIVE BEHAVIOR SETS

We divide the construction of the *SBS* into four steps: The first step is to use FlowDroid [20] tool to analyze Android-based software samples and construct software call graphs. In the second step, we use IC3 software [28] to locate ICCs for getting a complete call graph of the software function. In the third step, we determine the available secure sensitive method, seek programs that directly or indirectly call the method, and constitute the *SBS* of the software. And in the fourth step, we build the P from secure sensitive methods which are called by all non-secure sensitive methods in the graph.

The three types of secure sensitive methods concerned here are based on function names or related marks. For the permission protecting method, please refer to the permission-mapping table provided by PSout [25]. For the information flow source method or sink method,v please refer to SuSi [26] research results. Other suspicious methods are introduced as follows:

(a) Dynamic loading function – it can load a new APK or jar package during the software execution, so that the software can dynamically obtain new functions. The malware dynamically loads malicious codes, which makes it difficult to obtain its execution logic for software's static-based analysis. Methods concerned here include: DexClassLoader() and PathClassLoader().

(b) Java language – it can dynamically construct and call objects, which improves the flexibility of malware logic

TABLE 2. Common root exploit file.

Name	The attacked program	Used malicious families
Asroot	Linux kernel(…ß)	Asroot
Exploid	init (<=2.2)	DroidDream zHash DroidKungFu
GingerBreak	vold (<=2.3.3)	GingerMaster
KillingInThe NameOf	ashmem (<=2.2.1)	--
RATC	adbd (<=2.2.1)	DroidDream BaseBridge DroidKungFu
Zimperlich	zygote (<=2.2.1)	DroidDeluxe DroidCoupon
zergRush	libsysutils (<=2.3.6)	--

execution. The corresponding function of the reflection mechanism is java.lang.reflect.Method.invoke() [18].

(c) Encryption and decryption functions – they can protect software data and improve the security of the software. However, attackers encrypt malicious loading files in the malware, which makes it difficult to analyze the security. Here, we mainly focus on the API function in javax.crypto package [21].

(d) Native Development Kit (NDK)– it uses Java Native Interface (JNI) to call native codes. However, the use of native codes cannot be limited by software application permissions, and native nodes can use vulnerabilities of the system to perform illegal operations, such as trying to get root permissions. When stating relevant functions, there are native words in codes to locate native functions.

For each located function, the function ID will be first recorded, and then the information of secure sensitive is extracted by using analysis techniques of data flow and control flow [29], including entry point that triggers a secure sensitive method, parameters of a method, and a marking function that constructs a *SBG*.

2) FEATURES OF INSTALLATION PACKAGES

Except that malware have malicious features in their execution logic, installation package files and application permissions also reflect the malice to some degree. In order to extract the information to construct the installation package feature vector F, our work is accomplished as follows: (a) As the fact that the file type of the APK installation package in Android-based software is zip compression package, we use decompression command to decompress the package file when extracting installation package features. (b) We traverse all files in the folder, use Apache Tika tool kit [28] to analyze the file type based on data content, and check whether the file type matches its file suffix. (c) We determine if there is a .so file, and determine whether the database file is a root file (as shown in Table 2 [31]) according to the MD5 value. (d) We determine whether there are subroutines in the file, including .jar files, .dex files and .apk files. (e) We construct the feature vector F of the software installation package according to the above analysis.

3) PERMISSION FEATURES

In order to improve the security of Android systems, Google provides three major security mechanisms: the permission mechanism, the signature mechanism, and the sandbox mechanism [32]. The permission mechanism is used to restrict APIs, resources, and components who have a limited application accesses. In order to enable a program component to access sensitive resources, the program must apply for the corresponding permissions. All permissions that need to be applied for are declared in a AndroidManifest.xml (a Manifest file). When installing a software, the system will list all permissions for the software application. And only with authorization, these permissions would allow it to be installed and use corresponding functions during its operation.

Permissions of software applications reflect possible behaviors. Therefore, it can effectively express software features by analyzing permissions of malware applications. We can obtain permissions to apply for the software by analyzing the AndroidManifest.xml file. The file is encrypted in the installation package, and we can use APKParser [30] to decrypt it. Here, we use APKParser to process the file and extract the permission information to form the permission list P of the software.

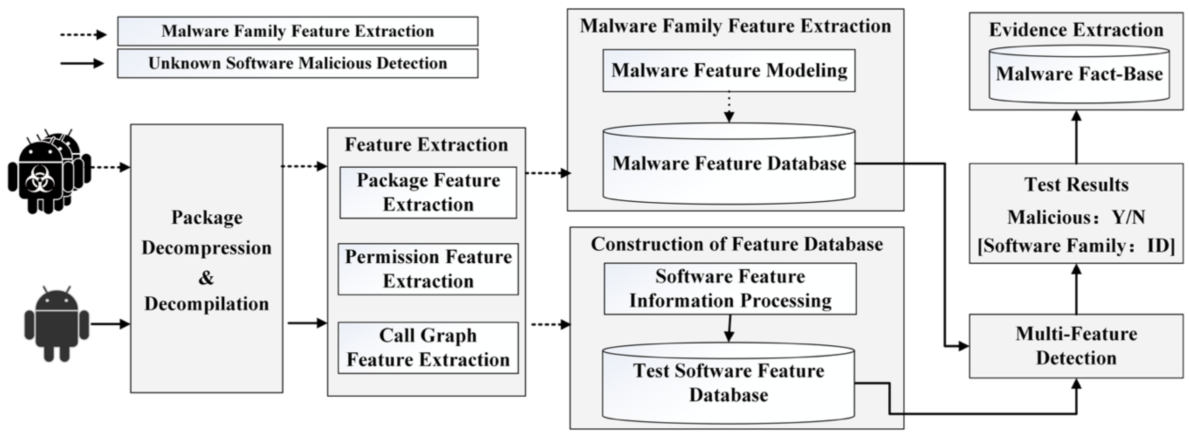
By using the way above, it is possible to analyze sample programs of the malware family, extract the critical features of each sample, and then construct malware family features in each sample.

D. CONSTRUCTION OF THE MULTI-FEATURE MODEL OF ANDROID-BASED MALWARE FAMILY

After extracting critical features of each sample, we construct the *MFM* of a malware family. We use the method given in sub-section 3-A to extract features of training samples from a malware family. The set constructed by the *SBS* of all samples is $S = \{SBS_1, SBS_2, \cdots, SBS_n\}, \quad S \in SBS_i, i = 1, \cdots n.$

For each set, we calculate its probability of occurrence in all samples' *SBSs*. If it is greater than 50%, the method S is added to the set SBS^c, and the mapping is built between S and its probabilities in the marking function α. The installation package feature in the *MFM* of a malware family consists of a marking function and a feature vector whose length is m. The extracted feature vector set of malware samples is $F = \{F_1, F_2, \cdots, F_m\}$. Here, the installation package feature is presented as $F^c = \left\{f_1^c, f_2^c, f_3^c, f_4^c\right\}$ and the marking function γ. For the *kth* feature, we calculate its probability that appears in F. If its value is greater than 50%, f_k is assigned as a value of 1, and the corresponding probability is assigned to $\beta\left(f_k^c\right)$.

The permission list applied by the *ith* sample is $P_i = \{p_1, p_2, \cdots, p_l\}$, then the set of the feature list for all samples



Malware Family Feature Extraction
Unknown Software Malicious Detection
Package Decompression & Decompilation
Feature Extraction
Package Feature Extraction
Permission Feature Extraction
Call Graph Feature Extraction
Malware Family Feature Extraction
Malware Feature Modeling
Malware Feature Database
Construction of Feature Database
Software Feature Information Processing
Test Software Feature Database
Evidence Extraction
Malware Fact-Base
Test Results
Malicious: Y/N
[Software Family: ID]
Multi-Feature Detection

FIGURE 3. Structure of multi-feature detection of Android malware

of the family is $P = \{p_1, p_2, \cdots, p_l\}$. For the feature whose probability is greater than 50%, we add it into the permission feature p^c of the *MFM*, and assign corresponding probability to $\gamma\,(P^c)$. By using the above way, we can extract multiple features in a malware family to build the feature database of the malware family for Android-based mobile devices.

IV. MULTI-FEATURE DETECTION OF ANDROID MALWARE

The structure of the multi-feature malware detection on Android-based mobile devices is shown in Fig.3. For an Android-based software, first establish the *CFR* from Definition 3. Second, compare the*CFR* of the malware family database, and compare its similarity with each family to detect the software. If the software is malware, its belonging family should be given. Third, extract evidence of malicious codes and add the related information to the malware fact-base.

A. MULTI-FEATURE DETECTION

For improving the detecting ability of malware variants, we propose a fuzzy comparison method by marking functions (from Definition 4) to detect the multi-feature. When calculating the similarity during extracting the *CFR* of the test software, we adopt three different parts of the software's *CFR*.

1) $SBS^c = \left\{S_1^c, S_2^c, \cdots, S_m^c\right\}$ is the to-be-matched *SBS* of multiple features of the malware family. The corresponding marking function is α, and the similarity calculation is shown in (5).

$$S_{sbs} = \omega_{sbs} \sum_{i=1}^{m} \alpha\left(S_{\mathrm{i}}^c\right)\zeta\left(S_i^c\right), S_i^c \in SBS^c$$
$$\zeta\left(S_i^c\right) = \begin{cases} 0, & if\ semiContain(SBS, S_i^c) = false \\ 1, & if\ semiContain(SBS, S_i^c) = true \end{cases} \tag{5}$$

Here, $semiContain(SBS, S_i^c) = true$ means there is a set S in *SBS*, and it has similar elements with the set S_i^c. Here, we assume that the proportion of the similar elements in the two sets is greater than 80%. And the meaning of $semiContain(SBS, S_i^c) = false$ is opposite. In order to prevent that malware families with more features replace the one having fewer features, we add a correction factor $-\omega_{sbs}$. It is equal to the number of all sets S_i^c, which makes $semiContain(SBS, S_i^c) = true$ in *SBS*, divided by the length of the set SBS^c.

2) Feature vector of the test software is $F = \{f_1, \cdots, f_m\}$. For the to-be-matched feature vector $F^c = \{f_1^c, \cdots, f_m^c\}$ in the *MFM* of a malware family, the corresponding tokenize function is β, and the similarity calculation is shown in (6).

$$S_f = \omega_f \sum_{i=1}^{m} f_i f_i^c \beta(f_i) \tag{6}$$

We calculate the similarity according to the probability of each feature. If the value of the feature vector in the *MFM* of the malicious family is 0, the similarity is 0. The calculation of the correction factor ω_f is that the number of all features satisfied with $f_i f_i^c = 1$ in F, divided by the number of features whose value is 1 in F^c.

3) The permission list of the test software is P. $P^c = \{p_1^c, p_2^c, \cdots, p_n^c\}$ is the to be matched permission list in the *MFM*, the corresponding marking function is γ, and the similarity calculation method is shown in (7).

$$S_p = \omega_p \sum_{i=1}^{n} \gamma\left(p_i^c\right) \psi\left(p_i^c\right), p_i^c \in P^c$$
$$\varphi(p_i^c) = \begin{cases} 0, & if\ p_{\mathrm{i}}^c \notin P \\ 1, & if\ p_{\mathrm{i}}^c \in P \end{cases} \tag{7}$$

Here, ω_p is the correction factor. It equals to the quantity number, which belongs to p^c in the permission set P, divided by the length of the set p^c. Thus, we can calculate the similarity between features of a software sample and a certain malware family as shown in (8).

$$S_{score} = S_{sbs} + S_f + S_p \tag{8}$$

We select the maximum value of similarity calculation results, and determine whether it exceeds the threshold. If it

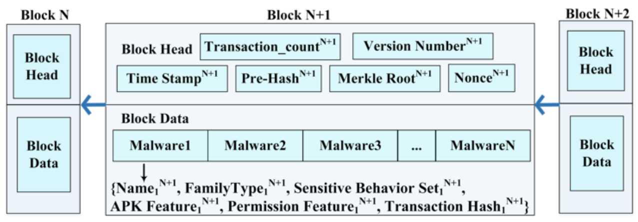


FIGURE 4. The partial structure of the blockchain.

exceeds the threshold, it is considered as a malware, and the corresponding malware family ID is outputted, otherwise, the test software is regarded as an original software.

B. THE FACT-BASE OF MALICIOUS CODES BASED ON BLOCKCHAIN

In the case of detecting malicious codes, if a problem is found in the result of the detection, the related information about the detection will be collected to constitute a block data and submit to the fact-base, which can be used as the evidence of malicious codes and to update the feature base of the malware family. The information can also be analyzed to provide support for the new detecting rules. In our CB-MDEE, each data block includes some testing fact information, such as sensitive behavior feature, installation package feature and permission feature of software. There is some basic information, including timestamp, hash value of the previous block, and a random number for verifying hash values. The data block structure is shown in Fig. 4, where (1) Pre-Hash is the hash value of the previous block with a size of 32 bytes. (2) Version number is used to track software / protocol updates with a size of 4 bytes. (3) Timestamp records the time that the block produces with a size of 15 bytes. (4) Transaction_count is the number of test results in the current block, with a size of 3 bytes. (5) Merkle root is a hash value calculated by all malicious codes detected in the block with a size of 32 byte, which is used to check whether a test result exists in this blocks. (6) Nonce is a random number before the block, which is recognized as a formal block. If a block is identified as a formal block, it is filled with 0000, and the size is 15 bytes. Here, the hash values are unique to ensure the integrity of the blockchain and to prevent frauds effectively. They change synchronously because of the changeable blocks. When the newly generated data blocks are added to the chain, the information in the blockchain can no longer be changed for conforming to the evidentiary requirements.

In each data block, detection result includes: 1) Name: the name of the malware. 2) FamilyType: indicates which malicious family the malware belongs to. 4) APK Feature: the feature of the installation package of the malware. 5) Permission Feature of the malware. 6) Transaction hash value of the detection result that uniquely identifies the record.

Each node calculates the hash value of the block when the block is received from the network, and it is a digital fingerprint obtained by the quadratic hash calculation of the block. Whether the block is uploaded on the network or is stored on a permanent storage device of a node as part of the blockchain, the hash value of the block is unique and it clearly identifies a block.

TABLE 3. Partial features of gappusin family.

Feature	Characteristic Component	Probability
Installation Package Feature	Feature Vectors	（1.0, 0.17, 0.0,0.028）
Permission Features	android.permission.ACCESS_NETWORK_STATE	1.0
	android.permission.INTERNET	1.0
	android.permission.WRITE_EXTERNAL_STAORAGE	1.0
	android.permission.READ_PHONE_STATE	0.94
	…	…
Call Graph Feature	323,20,41,8255	1.0
	293,91	0.96
	…	…

V. EXPERIMENTAL RESULTS

A. DATA SETS AND EXPERIMENTAL ENVIRONMENT SETTINGS

We perform our experiments in the CPU– Intel Core i7-3770, whose main memory is 16 GB. The operating system is ubuntu 15.10. The malware data set comes from the Drebin data set [34]. The benign software data is extracted by modifying the crawler programing – Google Play [2]. We select 4486 malware samples and 2140 benign software samples from 24 malware families. For example, a.dex file in a real-world Android malware, whose MD5 value is 3de513a148400b457dd8d8fa9238804db3ec031a0b526d4a0 4b77e5112aa2dcf [22]. For the benign software, we manually perform tests to determine it is not a malware by using VirusTotal (https://www.virustotal.com/#/home/upload) and Dr. Web Anti-virus (https://download.drweb.com/?lng=en).

In the malware samples, 75% samples of the 24 malware families were selected as the feature extraction to construct a signature database. For example, the partial feature information of the malware family Gappusin [34] is obtained after the analysis as shown in Table 3.

Due to the large number of files and the list of files in the package, here we give partial features of permission and call graph features in Table 4, where the methods involved in the call graph feature are replaced by numbers.

B. EXPERIMENTS OF MALWARE DETECTION

In order to verify our CB-MDEE model, we first classify and detect the remaining 25% samples by using the constructed feature database of the malware family. The results are shown in Table 5. Due to the limited space, we only present the results of 10 families. The first column of the table

TABLE 4. API explanation of partial malware family gappusin.

API ID	Function Name
20	java.lang.System: long currentTimeMillis() org.apache.http.util.EntityUtils:
41	java.lang.String toString(org.apache.http.HttpEntity)
91	android.content.Intent: void <init>(android.content.Intent)
293	android.app.Activity: void startActivity(android.content.Intent)
323	android.app.Activity: android.view.View findViewById(int)
8255	android.util.Log: int d(java.lang.String,java.lang.String)

TABLE 5. Detection results of malware in drebin dataset.

Software Family	Number of Samples for Feature Extraction	Number of tested samples	FN	FP	Accuracy
DroidKungFu	464	158	10	1	93.7%
FakeInstaller	678	239	13	10	94.6%
Opfake	429	183	14	1	92.3%
BaseBrige	285	98	9	0	90.8%
Plankton	243	85	0	1	100.0%
Iconosys	136	46	1	0	97.8%
FakeDoc	112	40	0	0	100.0%
Kmin	123	44	0	1	100.0%
Gappusin	60	18	0	0	100.0%
DroidDream	76	25	1	2	96.0%
Others	735	209	14	4	93.6%
Total	3341	1145	62	20	94.6%

is names of malware families, the second column is numbers of samples for feature extraction for, and the third column is numbers of tested samples. The fourth and fifth columns are numbers of false-negatives (FN) and false-positives (FP) in test results respectively. FN indicates that a software p belongs to the malware family M, but it is judged as a benign software or classified as the sample number of other families. FP indicates that a software p does not belong to the malware family M, but it is incorrectly classified as the sample number of the M family. The last column is the overall accuracy of the current malware family classification, which is calculated by dividing the numbers of correctly sorted samples in the total number of the samples.

From Table 6, we can find that CB-MDEE can be well used for the detection of Android malware families, and the accuracy is 94.6%. In particular, it can reach a 100% detecting rate for the malware family Gappusin, and can greatly solve problems that Drebin cannot detect Gappusin. For the detection of the software in Drebin data set, the time cost distribution is shown in Fig. 5.

As we can see from Fig.5, in Drebin dataset, 1145 samples are detected, and more than 1081 (94%) of the samples are used for the feature extraction and detection.

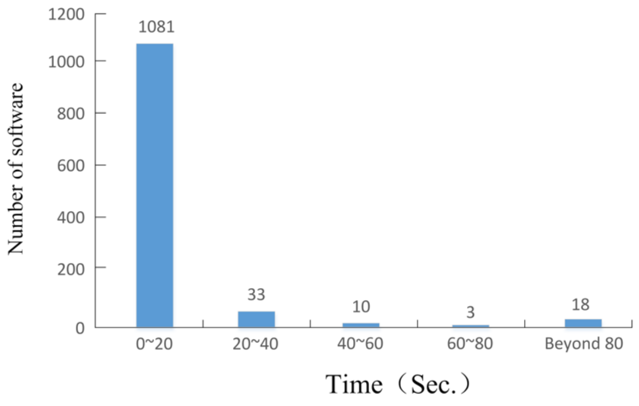


FIGURE 5. Time vs. detected number software with Drebin dataset.

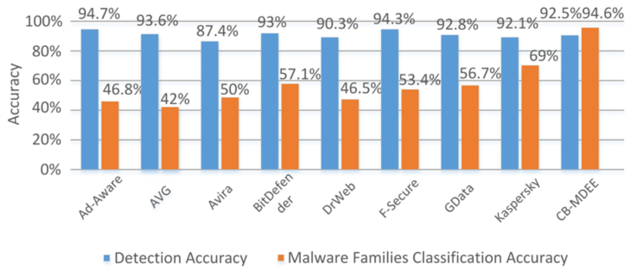


FIGURE 6. Comparison of different detecting engines.

The total required time is less than 20 seconds. The average time cost for each sample test is 7.9 seconds. Although the detecting time is higher than that of Drebin, the detecting accuracy (94.6%) is slightly higher than that of Drebin (93.9%).

C. EXPERIMENTS OF UNKNOWN SOFTWARE DETECTION

In order to verify detecting capabilities of our CB-MDEE, 2140 benign software samples were mixed with the remaining 25% of the 1145 malware samples as the input data. At the same time, in order to further verify describing capabilities of the multi-feature model proposed here, we use the subset of the multi-feature model as the classification feature. The test results are shown in Table 6. Here, we use two indicators: the accuracy and the recall rate to evaluate the detecting results. In table 6, we set the True Positive (TP) as the number of samples that are correctly detected as malware in test results, then the calculating methods of the accuracy and recall rate in table 7 are: accuracy = TP/(TP + FP), recall rate = TP/(TP + FN).

As we can see from Table 6, compared with other features, the extraction and comparison of the call graph feature need more time. However, the feature can be effectively used to classify malware. If the call graph feature is used to detect malware alone, it can only achieve 79.5% accuracy and 78.6% recall rate. Meanwhile, the average time cost for the detection of testing software is 131 seconds, which is significantly higher than detecting the software in Drebin data

TABLE 6. Detecting results by different features on unknown software.

Features	Average Time (Sec.)	FN	FP	Accuracy	Recall rate
Permission	32.3	445	351	66.6%	61.1%
Installation Package	69.5	357	304	72.2%	68.8%
Call graph	130654.5	245	232	79.5%	78.6%
Permission/ Installation Package	97.2	297	287	74.7%	74.1%
Permission & Call Graph	130698.4	135	141	87.7%	88.2%
Total	131041.0	62	86	92.5%	94.6%

set. The number of the software in Drebin data set is smaller. Of all 5560 samples, only 277 software is larger than 5 MB. The majority of the software in the benign software data set is larger than 5 MB, and the majority of the software (1647) is between 10 MB and 20 MB. The extracted software call graph features is directly related to the .dex file size.

We can improve the detecting accuracy by detecting different feature sets. For example, we select the permission features and call graph features as the basis for the detection, and the detecting effect is significantly higher than the individual detection of the permission or the call graph. If the multiple features proposed here are used for the malware detection, it can achieve 92.5% detecting accuracy and 94.6% recall rate.

We submit the hash value (SHA256) of the test dataset samples used in this paper to VirusTotal, and use the online detecting engine for the online detection. VirusTotal provides a total of 58 online detecting engines. In this experiment, 13 kinds of detecting engines with poor detecting results are removed, including CMC, K7AntiVirus, Malwarebytes, Panda, SUPERAntiSpyware, TheHacker, TotalDefense, Trustlook, VIPRE, ViRobot, WhiteArmor, nProtect and Yandex. The reason for their poor detection may be that they did not provide services during the experiment, or they did not include the submitted samples. Furthermore, we select 8 detecting engines that have good detecting effects and compare them with the detecting results. The comparison of analyzing results is shown in Fig.6.

From Fig.6, we can find that the 8 detecting engines in the comparison can achieve higher detecting accuracies, but it has a lower accuracy of malware families. In order to remove the statistical errors caused by the naming methods of the same malware family, we deal with the results of the VirusTotal. Taking the samples of the Goldream family as an example, this paper examines the detecting results with words like Goldream, Golddrea, Glodream, GoldDream, Golddream and GDream as correct classifications. The results of the comparison show that our method can achieve better accuracies in different malware families.

VI. CONCLUSION

In this paper, we construct a framework CB-MMDE to detect and classify malware on Android-based mobile devices through Blockchain technology. We analyze multiple features of malware families, propose a malware feature model–*MFM* for mobile devices based on Android system, and design a malware detection and classification algorithm. The experimental results of the Drebin data set and the benign software data set show that CB-MMDE can effectively detect and classify known malware and perform malice determination and malware family classification on unknown software with a higher accuracy and lower time cost.

REFERENCES

[1] R. Uncheck, "Mobile malware evolution 2016," Kaspersky Lab., Moscow, Russia, Tech. Rep. 28. Feb. 2017.

[2] N. J. Percoco and S. Schulte, "Adventures in BouncerLand—Failures of automated malware detection within mobile application markets," Trustwave Holdings, Inc., Chicago, IL, USA, Tech. Rep. 1, 2012.

[3] X. Du, M. Guizani, Y. Xiao, and H. H. Chen, "Secure and efficient time synchronization in heterogeneous sensor networks," *IEEE Trans. Veh. Technol.*, vol. 57, no. 4, pp. 2387–2394, Jul. 2008.

[4] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day Android malware detection," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services*. Low Wood Bay, U.K., 2012, pp. 281–293.

[5] Y. Xiao, X. Du, J. Zhang, and S. Guizani, "Internet protocol television (IPTV): The killer application for the next generation Internet," *IEEE Commun. Mag.*, vol. 45, no. 11, pp. 126–134, Nov. 2007.

[6] L. Wu, X. Du, and X. Fu, "Security threats to mobile multimedia applications: Camera-based attacks on mobile phones," *IEEE Commun. Mag.*, vol. 52, no. 3, pp. 80–87, Mar. 2014.

[7] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, Mar. 2017.

[8] J. J. Sikorski, J. Haughton, and M. Kraft, "Blockchain technology in the chemical industry: Machine-to-machine electricity market," *Appl. Energy*, vol. 195, pp. 234–246, Jun. 2017.

[9] Z. Zheng, S. Xie, H. Dai, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data, Big Data Congr.*, Honolulu, HI, USA, Jun. 2017, pp. 557–564.

[10] I.-C. Lin and T.-C. Liao, "A survey of blockchain security issues and challenges," *Int. J. Netw. Secur.*, vol. 19, no. 5, pp. 653–659, Sep. 2017.

[11] X. Du and H. H. Chen, "Security in wireless sensor networks," *IEEE Wireless Commun. Mag.*, vol. 15, no. 4, pp. 60–66, Aug. 2008.

[12] X. Du, M. Guizani, Y. Xiao, and H.-H. Chen, "Transactions papers a routing-driven elliptic curve cryptography based key management scheme for heterogeneous sensor networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 3, pp. 1223–1229, Mar. 2009.

[13] X. Du, D. Wu, W. Liu, and Y. Fang, "Multiclass routing and medium access control for heterogeneous mobile ad hoc networks," *IEEE Trans. Veh. Technol.*, vol. 55, no. 1, pp. 270–277, Jan. 2006.

[14] H. Cai and B. G. Ryder, "Understanding Android application programming and security: A dynamic study," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Shanghai, China, Sep. 2017, pp. 364–375.

[15] F. Fischer *et al.*, "Stack overflow considered harmful? The impact of copy&paste on android application security," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Jose, CA, USA, May 2017, pp. 121–136.

[16] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, Nov. 2015.

[17] K. O. Elish *et al.*, "Profiling user-trigger dependence for Android malware detection," *Comput. Secur.*, vol. 49, pp. 255–273, Mar. 2015.

[18] X. Xiao, Z. Wang, Q. Li, Q. Li, and Y. Jiang, "ANNs on co-occurrence matrices for mobile malware detection," *KSII Trans. Int. Inf. Syst.*, vol. 9, no. 7, pp. 2736–2754, Jul. 2015.

[19] S. Arzt and E. Bodden, "StubDroid: Automatic inference of precise data-flow summaries for the Android framework," in *Proc. IEEE/ACM 38th Int. Conf. Softw. Eng. (ICSE)*, Austin, TX, USA, May 2016, pp. 725–735.

[20] D. Octeau *et al.*, "Effective inter-component communication mapping in android with Epicc: An essential step towards holistic security analysis," in *Proc. 22nd USENIX Conf. Secur. (SEC)*, Washington, DC, USA, 2013, pp. 543–558.

[21] J. Burket, L. Flynn, W. Klieber, J. Lim, W. Shen, and W. Snavely, "Making DidFail succeed: Enhancing the CERT static taint analyzer for Android app sets," Softw. Eng. Inst., Washington, DC, USA, Tech. Rep. CMU/SEI-2015-TR-001, Mar. 2015.

[22] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications," in *Proc. Eur. Symp. Res. Comput. Secur.*, Wrocław, Poland, 2014, pp. 163–182.

[23] X. Du, Y. Xiao, M. Guizani, and H.-H. Chen, "An effective key management scheme for heterogeneous sensor networks," *Ad Hoc Netw.*, vol. 5, no. 1, pp. 24–34, Jan. 2007.

[24] 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. Int. Conf. Softw. Eng.*, Florence, Italy, May 2015, pp. 303–313.

[25] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "PScout: Analyzing the Android permission specification," in *Proc. ACM Conf. Comput. Commun. Secur.*, Raleigh, NC, USA, 2012, pp. 217–228.

[26] S. Rasthofer, S. Arzt, and E. Bodden, "A machine-learning approach for classifying and categorizing Android sources and sinks," presented at the NDSS Symp., San Diego, CA, USA, Feb. 2014, pp. 1–15.

[27] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: Scalable and accurate zero-day Android malware detection," in *Proc. Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, Low Wood Bay, U.K., 2012, pp. 281–293.

[28] (2016). *Apache Tika—A Content Analysis Toolkit*. [Online]. Available: https://tika.apache.org/

[29] X. Du, Y. Xiao, H.-H. Chen, and Q. Wu, "Secure cell relay routing protocol for sensor networks," *Wireless Commun. Mobile Comput.*, vol. 6, no. 3, pp. 375–391, May 2006.

[30] (2011). *Xml-Apk-Parser Project*. [Online]. Available: https://code.google.com/archive/p/xml-apk-parser/

[31] T.-H. Ho, D. Dean, X. Gu, and W. Enck, "PREC: Practical root exploit containment for Android devices," in *Proc. 4th ACM Conf. Data Appl. Secur. Privacy*, San Antonio, TX, USA, Mar. 2014, pp. 187–198.

[32] W. Shin, S. Kiyomoto, K. Fukushima, and T. Tanaka, "Towards formal analysis of the permission-based security model for Android," in *Proc. 5th Int. Conf. IEEE Wireless Mobile Commun. (ICWMC)*, Cannes, France, Aug. 2009, pp. 87–92.

[33] Q. Xia, E. B. Sifah, K. O. Asamoah, J. Gao, X. Du, and M. Guizani, "MeDShare: Trust-less medical data sharing among cloud service providers via blockchain," *IEEE Access*, vol. 5, pp. 14757–14767, 2017.

[34] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, San Diego, CA, USA, Feb. 2014, pp. 1–15.

JINGJING GU received the B.E. degree in computer science and the Ph.D. degree in computer science and technology from the Nanjing University of Aeronautics and Astronautics (NUAA), China, in 2005 and 2011, respectively. She is currently an Associate Professor with the Institute of Artificial Intelligence and Pattern Computing, NUAA.

Her current research interests include flying ad hoc networking, mobile security, wireless sensor network, and data mining.

BINGLIN SUN received the B.E. degree in computer science from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2016, where he is currently pursuing the master's degree with the College of Computer Science and Technology. His area of interest includes information security and reverse engineering.

XIAOJIANG DU (SM'09) received the B.S. and M.S. degrees in electrical engineering from Tsinghua University, Beijing, China, in 1996 and 1998, respectively, and the M.S. and Ph.D. degrees in electrical engineering from the University of Maryland at College Park in 2002 and 2003, respectively.

He is currently a Professor with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA. His research interests are security, wireless networks, and systems. He has over 250 journal and conference papers in these areas, and a book published by Springer. He received over U.S. 5 million in research grants. He is a Life Member of the ACM.

JUN WANG received the B.E. degree in computer science and the M.E. degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2014 and 2016, respectively.

His research includes android security and machine learning.

YI ZHUANG received the degree from the Department of Computer Science, Nanjing University of Aeronautics and Astronautics, in 1981. She is currently a Professor and the Ph.D. supervisor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics.

Her research interests include network distributed computing, information security, and dependable computing.

ZIWANG WANG received the M.S. degrees in computer science and technology from Guizhou Normal University, Guiyang, China, in 2016. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China.

His research includes mobile security and system security.