IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 18, 2016, accepted October 31, 2016, date of publication November 8, 2016,
date of current version November 28, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2626395

Efficient Fingerprinting-Based Android Device Identification With Zero-Permission Identifiers

WENJIA WU, JIANAN WU, YANHAO WANG, ZHEN LING, AND MING YANG
School of Computer Science and Engineering, Southeast University, Nanjing 211189, China

Corresponding author: W. Wu (wjwu@seu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61402104, Grant 61572130, Grant 61502100, Grant 61532013, Grant 61632008, and Grant 61320106007, in part by the Jiangsu Provincial Natural Science Foundation of China under Grant BK20140648 and Grant BK20150637, in part by the Jiangsu Provincial Key Technology Research and Development Program under Grant BE2014603, in part by the Jiangsu Provincial Key Laboratory of Network and Information Security under Grant BM2003201, and in part by the Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grant 93K-9.

ABSTRACT Mobile device identification techniques can be applied to secure authentication, and will be of particular importance for the security of mobile networks, such as avoiding spoofing attacks. For Android devices, explicit identifiers, e.g., Android ID, are used to uniquely identify a device. However, permissions are required to gain such identifiers, and this could cause the permission abuse and the leakage of user privacy. To address these issues, we use the combination of implicit identifiers that cannot identify a device individually. We first investigate 38 implicit identifiers that are acquired without requesting any permission. Then, a feature selection algorithm is used to choose effective identifiers as the device fingerprint, and three algorithms are designed to identify the devices. Finally, we conduct experimental evaluations on 50 830 fingerprints from 2239 different Android devices. The empirical results demonstrate the effectiveness and efficiency of our algorithms.

INDEX TERMS Device fingerprinting, android, implicit identifier, zero-permission.

I. INTRODUCTION

In recent years, the smartphone market is in a time of rapid growth, and the worldwide smartphone shipments are expected to reach 1.46 billion units in 2016, according to the latest forecast from the International Data Corporation (IDC) Worldwide Quarterly Mobile Phone Tracker [1].

Features acquired from a smartphone can be used to identify a user of the smartphone. Different from traditional computers, a smartphone is a personal belonging that is playing an increasingly important role in people's everyday lives, such as communication, shopping and traveling. In other words, a smartphone can usually be bound to a user. When the user accesses a system through mobile networks, the system can authenticate the user's identity with identifying his/her smartphone so that an attacker trying to impersonate the device will be detected. Therefore, device identification is of great importance for secure user authentication in mobile networks and has attracted attentions of many researchers. Since Android is currently dominant in mobile operating systems (OS), we focus on Android device identification in this paper.

Traditional methods utilize an explicit identifier that can uniquely identify a device, such as International mobile equipment identity (IMEI), International mobile subscriber identity (IMSI), or Android ID.[1] However, some explicit identifiers are unreliable, and can be easily tampered or even forged, such as Android ID. Moreover, it is necessary to request sensitive permissions before acquiring them. For example, the READ_PHONE_STATE permission is required when acquiring the information about IMEI, IMSI and phone number. This will cause the abuse of sensitive permissions and the leakage of user privacy [2], [3].

To address these issues, we adopt implicit identifiers for Android device identification in this paper. Unlike explicit identifiers, the implicit identifiers investigated in this paper cannot be used to identify devices individually, as devices may share the same value for an identifier. However, a combination of these identifiers shows great device-related characteristics, so that we can use the set of them as a

[1] https://developer.android.com/reference/android/provider/Settings.Secure.html

fingerprint for device identification. More specifically, we explore several suitable implicit identifiers that can be obtained without requesting any permission, generate device fingerprints through the combination of the implicit identifiers, and propose three algorithms to identify Android devices, respectively. The main contributions of this paper are as follows:

- We present 38 implicit identifiers that cover the features of physical layer, application layer, and user layer in Android system, and take these identifiers as features for fingerprinting. What's more, these features can be acquired without requesting any permission.
- We utilize a feature selection algorithm to remove irrelevant and redundant features. On this basis, we propose a fingerprint matching algorithm (FMA), a fingerprint association algorithm (FAA) and a naïve bayes classifier based algorithm (NBCA) to achieve Android device identification, respectively.
- We conduct experimental evaluations on 50830 fingerprint data records from 2239 different Android devices. The results show that the NBCA algorithm outperforms the two other algorithms, and its precision is over 99%, while its recall is over 98%.

The rest of the paper is structured as follows. Section II presents feature collection and selection. We then propose three algorithms to implement Android device identification in Section III. Next, we conduct experimental evaluations in Section IV. Section V discusses some related work. Finally, Section VI concludes the paper.

II. FEATURE COLLECTION AND SELECTION

In this section, we introduce and analyze the features extracted from Android system, and then select features for fingerprint formulation.

A. FEATURE COLLECTION

In Android system, the information about hardware, software and user settings can be seen through the settings menu. According to the information, we try to find some suitable features which can be obtained without requesting any permission. By investigating the official documents of Google [4], we study three types of features, mainly involving hardware-related implicit identifiers in physical layer, OS-related implicit identifiers in application layer, and user-setting-related implicit identifiers in user layer. From the three types of features, we eventually choose 38 different features that can be obtained through system API or Linux Shell command. Most of them are user-setting-related implicit identifiers, such as timezone, ringtone, and input methods. The others are hardware-related implicit identifiers and OS-related implicit identifiers, e.g., screen information and kernel information. The fingerprint of an Android device can be constituted by these features and the corresponding values, formatted as a list of (identifier, value) pairs. Table 1, 2 and 3 present all the features that are chosen and the corresponding approaches to acquire their values.

Among these features, the list of user package, denoting the information about applications installed on the device, is the most noteworthy. That is because hundreds of thousands of applications are provided in Android market and the sets of applications installed on user devices differ from each other in thousands of ways. Moreover, each application has an UID that is assigned in the stage of application installation and remains unchanged until application uninstallation. Since the UID owned by a removed application will be reclaimed by system for future assignment and a new coming application will adopt the smallest one of available UIDs. Despite the two devices have the same set of installed applications, their lists of user package will still be different due to the different orders of application installation. Hence, the list of user package has the high ability of device identification. For the list of system package, although system applications in the ROM are installed in the same order, users may remove or install some system applications. This will make the lists of devices different from each other. Thus, the list of system package is also a good feature for device identification. However, some

TABLE 1. List of hardware-related implicit identifiers.

Name of identifier	Acquisition approach	Data type
Device model	Get the value of Android.os.Build.MODEL	String
Equipment manufacturer	Get the value of Android.os.Build.MANUFACTURER	String
Screen information (width, height, resolution)	Call getDisplayMetrics method of Resources object	String, formatted as "width × height × resolution"
Capacity of internal storage	Call the method of Environment object	Integer, the unit is MB
Capacity of external storage	Call the method of Environment object	Integer, the unit is MB

TABLE 2. List of os-related implicit identifiers.

Name of identifier	Acquisition approach	Data type
Kernel information	Execute shell command "cat /proc/version"	String
Android version	Get the value of Build.VERSION.RELEASE	String, e.g., "4.4.2"
User-agent string in HTTP	Create a WebView object, and call getUserAgentString method	String
Structure of system storage	Execute shell command "df"	List
Structure of system root directory	Execute shell command "ls -la"	List

TABLE 3. List of user-setting-related implicit identifiers.

Name of identifier	Acquisition approach	Data type
Timezone	Call getTimeZone method of Calendar object	String, e.g, "China Standard Time"
Format of hour	Call is24HourFormat method of DateFormat object	Enumeration, 12 or 24
Format of date	Call getDateFormatOrder method of DateFormat object	Enumeration, "yyyy-MM-dd" or others
Automatic time synchronization	Get the value of AUTO_TIME in Settings object	Enumeration, True or False
Automatic selection of timezone	Get the value of AUTO_TIME_ZONE in Settings object	Enumeration, True or False
Time of screen locking	Get the value of SCREEN_OFF_TIMEOUT in Settings object	Integer, the unit is second
Notification of Wi-Fi available	Get the value of WIFI_NETWORKS_AVAILABLE_NOTIFICATION_ON in Settings object	Enumeration, True or False
Policy of Wi-Fi sleeping	Get the value of WIFI_SLEEP_POLICY in Settings object	Enumeration
Way of getting location information	Get the values of GPS_PROVIDER and NETWORK_PROVIDER in LocationManager object	Enumeration
Lock pattern enabled	Get the value of LOCK_PATTERN_ENABLED in Settings object	Enumeration, True or False
Lock pattern visible	Get the value of LOCK_PATTERN_VISIBLE in Settings object	Enumeration, True or False
Vibrating for phone unlocking	Get the value of LOCK_PATTERN_TACTILE_FEEDBACK_ENABLED in Settings object	Enumeration, True or False
Input methods	Call getEnabledInputMethodList method of InputMethodManager object	String
System language	Call getDisplayLanguage method of Locale object	String
Root or not	Check Android.os.Build.Tags and /system/app/Superuser.apk, and execute shell command "which su"	Enumeration, True or False
Font size	Get system configuration from Resources object, and then obtain fontScale	Float
List of typeface	Execute shell command "ls -la /system/fonts"	List
List of user package	Get list of user package through PackageManager	List
List of system package	Get list of system package through PackageManager	List
List of ringtone	Call setType method of RingtoneManager object	List
Ringtone	Call getRingtone method of RingtoneManager object	String
Alarm alert	Call getRingtone method of RingtoneManager object	String
Notification	Call getRingtone method of RingtoneManager object	String
Sound effects enabled	Get the value of SOUND_EFFECTS_ENABLED in Settings object	Enumeration, True or False
Showing password in text editors	Get the value of TEXT_SHOW_PASSWORD in Settings object	Enumeration, True or False
Screen brightness mode	Get the value of SCREEN_BRIGHTNESS_MODE in Settings object	Enumeration, True or False
Automatic rotation of screen	Get the value of ACCELEROMETER_ROTATION in Settings object	Enumeration, True or False
Current wallpaper	Get the wallpaper information from WallpaperManager object	String, MD5 value

security softwares, such as 360 Mobile Safe that has been installed on 68.5% of smartphones in China,[2] can control the acquisitions of lists of user package and system package, and the two features are unreliable to be acquired. Therefore, it is necessary to combine with other features to achieve device identification.

B. FEATURE ANALYSIS

In order to evaluation the goodness of these features, we introduce the concept of surprisal and entropy in information theory [5].

If x represents an Android device, $F(x)$ denotes the fingerprint of x, and $P(f_n)$ is the discrete probability density function corresponding to the probability distribution of $F(x)$, where f_n ($n \in [0, 1, ..., N]$) is a possible fingerprint. The surprisal I is defined as follows:

$$I(F(x) = f_n) = -\log_2(P(f_n)) \tag{1}$$

The device fingerprint is formed by several features, and we can calculate the surprisal of each feature to evaluate its distinguishing ability. Supposing that s is a feature, we can calculate its surprisal and entropy by Formula 2 and 3.

$$I(f_{n,s}) = -\log_2(P(f_{n,s})) \tag{2}$$

$$H(F_s) = -\sum_{n=1}^{N} P(f_{n,s}) \times \log_2(P(f_{n,s})) \tag{3}$$

For a certain device fingerprint, we can compute the amount of information that contributes to the fingerprint by the surprisal of each feature. For all device fingerprints, we can calculate the entropy of the feature, which represents the amount of information owned by the feature.

C. FEATURE SELECTION

As described above, we find a set of 38 features that can be used to form device fingerprints directly. However, some features are either redundant or irrelevant, and can be removed without degrading the performance of device identification. In addition, cardinality reduction of features will make the process of device identification efficient. Therefore, it is necessary to choose a subset of relevant features for fingerprint generation. In general, a feature is good if it is relevant to the class but it is not redundant to any of the other relevant features. Hence, the key is to find a proper evaluation indicator of the correlation between features and select features

[2]http://www.iimedia.cn/42606.html

based on the evaluation indicator. In this paper, we utilize a fast correlation-based filter (FCBF) algorithm proposed in [6] for feature selection. After feature selection, we can form compacted device fingerprints by the selected features.

III. FINGERPRINTING-BASED DEVICE IDENTIFICATION

In this section, we propose three algorithms to achieve device identification, respectively. Given a data set consisting of fingerprints of known Android devices, these algorithms can be used to identify unknown Android device.

A. FINGERPRINT MATCHING ALGORITHM

We first propose a simple FMA algorithm to exactly match the unknown fingerprint with fingerprints in the data set, as shown in Algorithm 1. The set of fingerprints of known Android devices is denoted by F, and the fingerprint of an unknown device is denoted by x. If there exists a fingerprint $f \in F$ that can be exactly matched with x, i.e., each (identifier, value) pair is the same for the two fingerprint, the unknown device will be identified as the device whose fingerprint is f. Otherwise, the unknown device cannot be identified, and the algorithm returns 'null'.

Algorithm 1 Fingerprint Matching Algorithm (FMA)

Input: fingerprints in the data set F, unknown fingerprint x
Output: matched fingerprint y

```
1  y ← null
2  for f ∈ F do
3      if x and f are exactly matched then
4          y ← f
5          return y
6  return y
```

B. FINGERPRINT ASSOCIATION ALGORITHM

The FMA algorithm is efficient, and its identification is precise. However, it cannot work well when the fingerprints of devices are changeable, since the changed fingerprint of a device and its previous fingerprint cannot be exactly matched. In order to address the issue, we propose an FAA algorithm that can be utilized when the precise matching between fingerprints is failed. According to the research work for desktop browser fingerprinting in [7], our algorithm is designed to associate an unknown fingerprint with its previous fingerprint if it does not change too much, specifically, only one feature is changed and the change range is relatively small.

Algorithm 2 shows the process of the FAA algorithm. We denote the set of features as S, where list-type features are involved in set S_{list} and other features are included in set $S_{no-list}$.

The algorithm mainly consists of two part. In the first part (lines 3-7), the set of fingerprints F is scanned, and the fingerprints that have only one feature different from fingerprint x will be added into set *candidates*. In the second part (lines 8-20), we choose the fingerprints that are most possible to change into fingerprint x from set *candidates*, and add them into set *foundF*. For each fingerprint in set *candidates*, if the changed feature is not a list-type feature, it will be added into set *foundF* directly. And if the changed feature belongs to the set of list-type features, the fingerprint with the maximum similar degree (no less than the threshold δ, usually $\delta = 0.85$) will be added into set *foundF*.

Algorithm 2 Fingerprint Association Algorithm

Input: fingerprints in data sets F, unknown fingerprint x (precise matching is failed)
Output: associated fingerprint y

```
1  y ← null
2  candidates ← dict()
3  for s ∈ S do
4      candidates[s] ← []
5  for f ∈ F do
6      if x and f are different on only one feature s then
7          candidates[s].append(f)
8  foundF ← []
9  for s ∈ S do
10     if s ∈ S_no−list then
11         for f ∈ candidates[s] do
12             foundF.append(f)
13     if s ∈ S_list then
14         maxF ← null, maxProb ← 0
15         for f ∈ candidates[s] do
16             prob ← ListSimilarity(x[s], f[s])
17             if prob >= δ and prob > maxProb then
18                 maxF ← f, maxProb ← prob
19         if maxF! = null then
20             foundF.append(maxF)
21 if len(foundF) == 1 then
22     y ← foundF[0]
23 return y
```

We adopt the concept of Jaccard distance, and define similar degree function *ListSimilarity* as follows:

$$\frac{|set(f[s]) \cap set(x[s])|}{|set(f[s]) \cup set(x[s])|} \tag{4}$$

where the operator *set* casts a list into a set so that the operations such as union and intersection can be done.

At the end of the algorithm, if there exists only one fingerprint in set *foundF*, the associated fingerprint will be returned. Otherwise, we return 'null', since we cannot decide the association when more than one fingerprint is in set *foundF*.

C. NAÏVE BAYES CLASSIFIER BASED ALGORITHM

The FAA algorithm can be applied to identify the devices that have only one feature changed, and the identification will fail when there is more than one feature changed. Considering that multiple features may be changed for a device, we utilize the classification method to implement device identification. We take the known devices in the set D as classes, i.e., $d_1, d_2, d_3, \ldots,$ and d_N. The fingerprints of the known devices are collected in data set F, and there may be multiple fingerprints corresponding to a device (class). Thus, we can formulate the device identification problem as a multiclass classification problem. For an unknown fingerprint x, if x is classified as $d_i \in D$, it means the unknown device can be identified as the known device d_i. To solve the classification problem, we propose an NBCA algorithm.

As irrelevant and redundant features have been removed through feature selection, we can assume that the features are independent and there is no correlation between features. The set of features are defined by S, i.e., $\{s_1, s_2, \ldots, s_M\}$. Thus, given an unknown fingerprint to be classified, represented by a vector $x = (x_1, x_2, \ldots, x_M)$ with some M features (independent variables), the probability of classifying it as class d_i, denoted by $P(d_i|x)$, can be calculated as follows:

$$\begin{aligned} P(d_i|x) &= P(d_i|(x_1, x_2, \ldots, x_M)) \\ &= \frac{P(d_i)}{P(x_1, x_2, \ldots, x_M)} \prod_{j=1}^{M} P(x_j|d_i) \end{aligned} \tag{5}$$

Since $P(d_i)$ and $P(x_1, x_2, \ldots, x_M)$ are taken as constants, Formula 5 can be expressed as:

$$P(d_i|(x_1, x_2, \ldots, x_M)) \propto \prod_{j=1}^{M} P(x_j|d_i) \tag{6}$$

Hence, the most possible class that assigned to x can be obtained by the following formula:

$$y = \arg\max_{d_i} \prod_{j=1}^{M} P(x_j|d_i) \tag{7}$$

In the following, we design methods of calculating $P(x_j|d_i)$ for each feature s_j. There are four categories of features: enumerate type, numerical type, string type, and list type.

For the enumerate-type features, we count the frequency of each value of feature s_j appearing in the fingerprints of d_i, and calculate the probability of the value x_j. We also perform Laplace smoothing, that adds one into the count of each value, such that no probability is ever set to be exactly zero.

For the numerical-type and string-type features, we consider the change/unchange event of the value of the feature s_j, and calculate the probabilities of the two events from the sequence of fingerprints of class d_i in data set. For the feature s_j of class d_i, the probability of the change event is denoted by $P(s_j \text{ is changed}|d_i)$, and the probability of the unchange event for the feature $P(s_j \text{ is unchanged}|d_i)$. In addition, the last fingerprint of class d_i is denoted by f_i, and the value of its j^{th} feature is $f_{i,j}$. Thus, $P(x_j|d_i)$ is defined as follows:

$$P(x_j|d_i) = \begin{cases} P(s_j \text{ is unchanged}|d_i) & \text{if } x_j = f_{i,j} \\ P(s_j \text{ is changed}|d_i) & \text{otherwise} \end{cases} \tag{8}$$

For the list-type features, we evaluate the difference between x_j and $f_{i,j}$ through Jaccard distance, and $P(x_j|d_i)$ is defined as follows:

$$P(x_j|d_i) = J(x_j, f_{i,j}) = \frac{max(|set(x_j) \cap set(f_{i,j})|, 1)}{max(|set(x_j) \cup set(f_{i,j})|, 1)} \tag{9}$$

Algorithm 3 Naïve Bayes Classifier based Algorithm (NBCA)

Input: set of the known devices D, fingerprints of know devices F, unknown fingerprint x
Output: identified device *findD*

```
maxProb ← 0, findD ← null
for d ∈ D do
    prob ← ∏_{j=1}^{M} P(x_j|d)
    if prob > maxProb and prob > λ then
        maxProb ← prob
        foundD ← d
return foundD
```

The NBCA algorithm is described in Algorithm 3. As seen from this algorithm, the threshold λ is an important parameter for classification. If λ is set to be high, some fingerprints of known devices cannot be correctly identified. If λ is set to be low, some fingerprints of new devices may be identified as a known device. Thus, in order to improve the performance of classification, we propose a threshold configuration algorithm to obtain the optimal value of threshold λ.

To evaluate the performance of classification algorithm, we define the standard classification metrics, i.e., precision, recall and F_β score. Denote TP as the number of the true positives, i.e., the number of fingerprints are correctly identified as a known device. Let FP be the number of false positives, i.e., the number wrongly identified as a known device. Let TN is denoted as the number of true negatives, i.e., the number of fingerprints are correctly identified as new devices. Likewise, FN is denoted as the number of false negatives, i.e., the number wrongly identified as new devices. Precision, recall and F_β score are defined by

$$precision = \frac{TP}{TP + FP} \tag{10}$$

$$recall = \frac{TP}{TP + FN} \tag{11}$$

$$F_\beta = (1 + \beta^2) \frac{precision \times recall}{\beta^2 \times precision + recall} \tag{12}$$

Since we place more emphasis on FP, i.e., weighing precision higher than recall, the parameter β is set to be 0.5, and the measure is denoted by $F_{0.5}$ [8].

Algorithm 4 Threshold Configuration Algorithm

Input: fingerprints of the known devices (as training data)
Output: the optimal threshold λ

```
λ ← 0
utilize training data to run Algorithm 3, and obtain TP, FP, TN, FN and probMap
maxF ← the current F_0.5
sort keys in probMap in increasing order
for key ∈ the set of keys do
    if key == 0 then
        continue
    Update TP, FP, TN, FN according to Formula 13
    Update F_0.5 denoted by newF
    if newF > maxF then
        maxF ← newF
        λ ← key
return λ
```

Algorithm 4 represents the threshold configuration algorithm that mainly involves two stages. In this first stage, the threshold λ is set to be 0, and the training data are utilized to run the classification algorithm. In this process, the TP, FP, TN and FN are counted, and we define a data structure $probMap$ that stores some (key, five-element tuple) pairs. For each fingerprint in the training data, its class d_i is obtain after classification, and the corresponding probability $p = \prod_{j=1}^{M} P(x_j|d_i)$ is calculated. Then, we verify the classification result, and update the value of TP, FP, TN, FN and $probMap[p]$. $probMap[p]$ is a five-element tuple:

- $probMap[p][0]$: the number of true positives where the corresponding probability is p;
- $probMap[p][1]$: the number of true negatives where the corresponding probability is p;
- $probMap[p][2]$: the number of false positives (the target fingerprint is in the data set) where the corresponding probability is p;
- $probMap[p][3]$: the number of false positives (the target fingerprint is not in the data set) where the corresponding probability is p;
- $probMap[p][4]$: the number of false negatives where the corresponding probability is p

In the second stage, $probMap$ is scanned by key in increasing order, and find the value of the key to maximize $F_{0.5}$. Initially, the TP, FP, TN, and FN are corresponding to the threshold λ, $\lambda = 0$, and the current $F_{0.5}$ can be calculated. Then, the n keys in $probMap$ are sorted, that is, $(key_0 =)0 < key_1 < key_2 < \ldots < key_n < (key_{n+1} =)1$. If λ is adjusted as $key_1 < \lambda < key_2$, the TP, FP, TN, and FN should be updated as follows:

$$\begin{aligned} TP &= TP - probMap[key_1][0] \\ FP &= FP - (probMap[key_1][2] + probMap[key_1][3]) \\ TN &= TN + probMap[key_1][3] \\ FN &= FN + (probMap[key_1][0] + probMap[key_1][2]) \end{aligned} \tag{13}$$

Thus, we iteratively adjust λ as $key_i < \lambda < key_{i+1}$, $i \in [1, n]$, and update the TP, FP, TN, and FN, so that the $F_{0.5}$ corresponding to the new threshold can be calculated. After the iterations, we can find the threshold that is corresponding to the maximum $F_{0.5}$.

IV. PERFORMANCE EVALUATION

A. DATA COLLECTION

We develop a data collection module in Android to collect and update the data of device fingerprints. In this module, we also apply the READ_PHONE_STATE permission to acquire the IMEI of the device. As a reliable explicit identifier, the IMEI is uploaded with the fingerprint, so each fingerprint is marked with an IMEI in the data set.

In order to collect the data of real users, we integrate the data collection module into Xiansheng APP that provides services of information aggregation and query for the students of Southeast University. [3] In the campus, the APP has a large number of users. Every time a user opens the APP, the data collection module will collect the implicit identifiers of the current device, generate a fingerprint in the background, and asynchronously upload the fingerprint to the server.

From September 22, 2014 to November 7, 2015, 50830 fingerprints are collected in the data set with 2239 different IMEIs. It means that all the fingerprints are generated by 2239 different Android devices. The set of fingerprints are denoted by $DFSet$.

B. DATA ANALYSIS

In statistics, all the devices in the data set are from 48 different manufacturers and the top 10 are shown in Figure 1. Meanwhile, we also find the numbers of the devices

[3] http://herald.seu.edu.cn

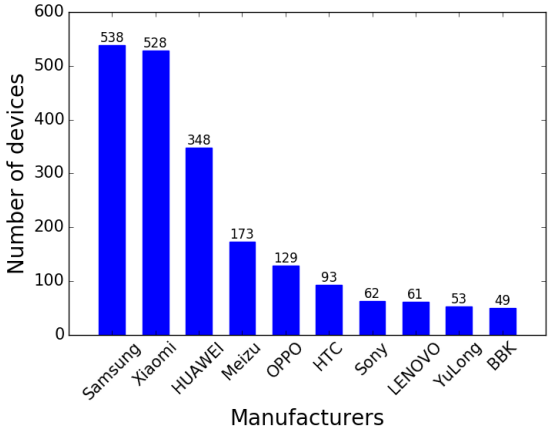


FIGURE 1. Top 10 equipment manufacturers in dataset.

TABLE 4. Manufacturers whose devices appear no more than 3 times.

Frequency of appearance	Device Manufacturers
3	DOOV, YUSUN
2	TPAL-A8P, TCT, koobee, IUNI, Hisense, CMDC
1	YUEAEE, WST, W3T61, Unkown, rockchip, QXH, Philips, Nokia, MT6589, MOFUT, MIT, Letv, L7REAXDM5, L1E, iPhone, InFocus, GT-I9300, GO, FEIXUN Baidu, DKL01, China nobile, BOWAY, Batmobile, android, AMOI

from some manufacturers are no more than 3, and these manufacturers are listed in Table 4.

As seen from the information about manufacturers, our data set has a broad coverage, involving mainstream Android devices and non-mainstream devices. These minor manufacturers are easy to be identified. Likewise, some manufacturers in Table 4 are not real manufacturers, such as Baidu, since the third-part ROM is modified. This make the devices easy to be identified.

C. FEATURE ANALYSIS

In the data set, there may be multiple fingerprints corresponding to each device, and only the last fingerprint is considered in feature analysis. We first count the different values of each feature, and calculate its entropy. Then, the features are sorted by their entropy in decreasing order. Table 5 shows the top 10 features and the last 10 features.

It can be seen from Table 5 that there are 2237 different lists of user package for the 2239 Android devices, and most of the devices can be uniquely identified by this feature. Thus, this feature has a high level of diversity.

TABLE 5. Number of values and entropy of features.

Feature	Number of values	Entropy
Top 10		
List of user package	2237	13.076
Current wallpaper	1989	10.576
List of system package	2112	10.407
Kernel information	1262	8.977
Structure of system storage	1094	8.793
List of ringtone	1259	8.558
Ringtone	1341	8.530
User-agent string in HTTP	735	7.868
Capacity of external storage	604	7.426
Device model	537	7.285
Last 10		
Sound effects enabled	3	0.738
Notification of Wi-Fi available	2	0.689
Automatic selection of timezone	3	0.669
Automatic time synchronization	3	0.627
Format of hour	2	0.588
Root or not	2	0.552
Automatic rotation of screen	2	0.486
Format of date	3	0.470
System language	5	0.215
Timezone	14	0.181

In the data set, there is a sequence of fingerprints corresponding to a device. Due to system update, configuration, and package installation, the fingerprint of a device is constantly changing. We count the number of changes from these fingerprint sequences for each feature, as shown in Table 6.

TABLE 6. Number of changes for each feature in the data set.

Feature	Number of changes	Feature	Number of changes
List of user package	14000	Time of screen locking	1716
List of system package	1588	Way of getting location infomation	1404
Current wallpaper	3230	Showing password in text editors	112
Kernel information	1395	Font size	138
Ringtone	1200	Policy of Wi-Fi Sleeping	90
List of ringtone	1485	Lock pattern visible	227
Structure of system storage	4414	Lock pattern enable	359
User-agent string in HTTP	332	screen brightness mode	2076
Capacity of external structure	213	Sound effects enabled	154
Device model	17	Automatic selection of timezone	203
Structure of system root directory	421	Automatic time synchronization	296
Capacity of internal storage	77	Notification of Wi-Fi available	260
List of typeface	569	Format of hour	173
Input methods	907	Root or not	71
Notification	560	Format of date	116
Alarm alert	604	Automatic rotation of screen	1123
Equipment manufacturer	4	Timezone	210
Android version	201	Vibrating for phone unlocking	14
Screen information(width, height, resolution)	191	System language	170

From this table, we can see that the list of user packages changes frequently over time, and so the stability of this feature is poor.

D. FEATURE SELECTION

Based on the *DFSet*, we utilize the FCBF algorithm to select features, and choose 9 features as $QIDS_1$. All the features are listed in Table 7.

TABLE 7. Feature list $QIDS_1$ after selction.

Type of features	Name of feature
OS-related	User-agent string in HTTP
User-setting-related	List of system package
	Current wallpaper
	Ringtone
	List of ringtone
	Input methods
	Time of screen locking
	Showing password in text editors
	Policy of Wi-Fi sleeping

Since some security softwares may prohibit to acquire the list of user package and system package, we discard the two features and then do feature selection. Thus, 10 features are chosen in Table 8 as $QIDS_2$.

TABLE 8. Feature list $QIDS_2$ after selction.

Type of features	Name of feature
hardware-related	Capacity of external storage
OS-related	Kernel information
User-setting-related	Current wallpaper
	Ringtone
	List of ringtone
	Input methods
	Time of screen locking
	Showing password in text editors
	Way of getting location information
	Policy of Wi-Fi sleeping

E. DEVICE IDENTIFICATION

We implement the proposed identification algorithms, and evaluate the performance of the algorithms based on the *DFSet*.

We first compare the performance of the FMA algorithm and FAA algorithm under different feature sets, i.e., $QIDS_1$ and $QIDS_2$. The results are shown in Figure 2. From the figure, we can see that the precision of the FAA algorithm is almost the same as that of the FMA algorithm, and the recall of the FAA algorithm is much improved, compared to that of the FMA algorithm. That is because that the FAA algorithm can recognize the device fingerprints with one feature changed, and avoid them to be wrongly identified as an unknown device.

Then, we compare the performance of the NBCA algorithm and FAA algorithm under different feature sets, i.e., $QIDS_1$ and $QIDS_2$. For the NBCA algorithm, we choose 40% of *DFSet* for training (denoted by $DFSet_{Train}$), and the rest is used for testing (denoted by $DFSet_{Test}$). For the FAA algorithm, it runs on $DFSet_{Test}$. To get the optimal threshold of the NBCA algorithm, we use $DFSet_{Train}$ to run the threshold configuration algorithm under $QIDS_1$ and $QIDS_2$ respectively, and obtain the optimal threshold $\lambda = 2.05E-4$ for $QIDS_1$ and $\lambda = 3.19E-5$ for $QIDS_2$. The FAA algorithm runs on $DFSet_{Test}$. The results are shown in Figure 3.

From this figure, we can see that the precision of the NBCA algorithm is almost the same as that of the FAA algorithm, and the recall of the NBCA algorithm is more than that of the FAA algorithm. That is because that the NBCA algorithm can recognize the device fingerprints with multiple features changed, while the FAA algorithm consider the fingerprints with only one feature changed.

Finally, we run the NBCA algorithm and FAA algorithm without feature selection, that is, all the 38 features are considered. The results are shown in Table 9. We can see from this table that the performance of the NBCA algorithm without feature selection is almost the same as that with feature selection. This illustrates the reasonability of feature selection, that is, the feature sets $QIDS_1$ and $QIDS_2$ is good subsets of features with no loss in precision and recall.

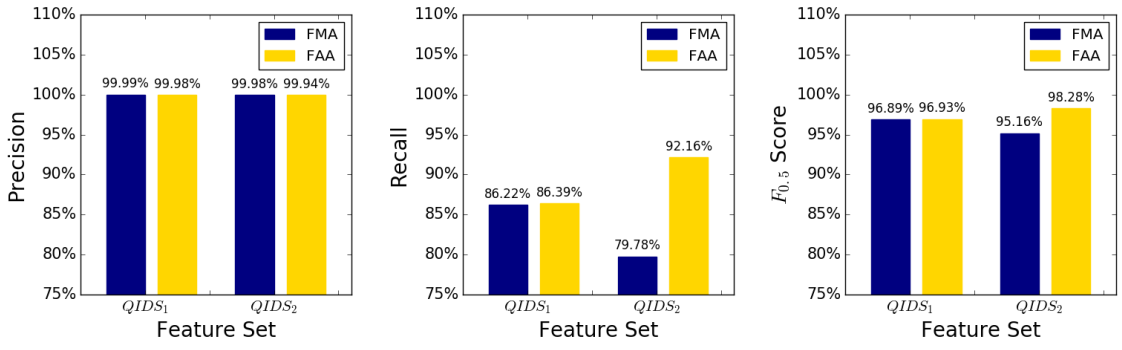


FIGURE 2. Performance of the FMA algorithm and FAA algorithm on *DFSet*.

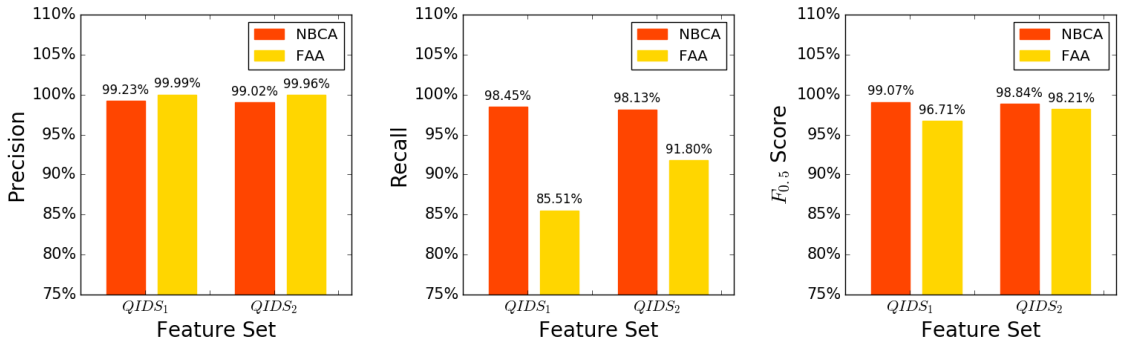


FIGURE 3. Performance of the NBCA algorithm and FAA algorithm on *DFSet_Test*.

TABLE 9. Performance of NBCA algorithm and FAA algorithm on *DFSet_Test* (Without feature selection).

Algorithm	Precision	Recall	$F_{0.5}$ score
Naïve bayes classifier based algorithm	99.68%	98.55%	99.46%
Fingerprint association algorithm	99.99%	78.96%	94.94%

In addition, the recall of the FAA algorithm without feature selection is decreased, compared to that with feature selection. That is because some additional features will disturb the process of fingerprint association, and negatively affect the success of device identification.

V. RELATED WORK

Various techniques for device identification have been proposed for user authentication, intrusion detection, and so on. Due to the limitations of explicit identifiers, device fingerprinting based on implicit identifiers is becoming one of the most common techniques. Several approaches have been proposed, and can be classified as physical-layer identification, protocol-layer identification, application-layer identification and user-layer identification.

Physical-layer device identification aims at identifying devices on characteristics of devices that are observable from the physical hardware and their communication at the physical layer [9]. For fingerprinting a physical device, Kohno et al. [10] exploit microscopic deviations in device hardware, that is, clock skews, which can be estimated remotely using TCP and ICMP timestamps. Gerdes et al. [11] analyze variations in the analog signal of the Ethernet devices, and propose a corresponding method to achieve device identification through using as few as 25 Ethernet frames. Brik et al. [12] design and implement a technique to identify the 802.11 wireless device through passive radio-frequency analysis, which leverages the hardware imperfections of network interface cards. Moreover, Vo-Huu et al. [13] extract physical characteristics of the Wi-Fi signal with the software-defined radio receiver. In addition, the smart mobile devices are equipped with several sensors, which can also be utilized for device fingerprinting, such as accelerometers [14] and speakers [15], [16].

For protocol-layer identification, its features are extracted from the parameters of protocol stacks or algorithms in network drivers. Franklin et al. [17] characterize the implementation-dependent probing algorithms for 802.11 networks, and propose an approach to passively identify the wireless drive employed by a device. Pang et al. [18] analyze sizes of broadcast packets from 802.11 traffic, and find implicit identifiers that are exposed by design flaws of the 802.11 protocol.

Application-layer identification utilizes features about the types of running applications and their behaviors, since the installed applications and their background communication generate a fingerprint for identification. Erman et al. [19] propose machine learning algorithms to classify network traffic and identify the types of applications. Dyer et al. [20] show that despite the use of traffic morphing, features of applications can still be found to implement device identification. Stöber et al. [21] analyze characteristic Wi-Fi/3G traffic from the background activities in Android system, and extract features available as side-channel information such as timing and data volume, so that the smartphones can be accurately identified.

User-layer identification is mainly according to features about user operations and settings. In 802.11 wireless networks, users add service set identifiers (SSIDs) into a preference network list, and transmit the SSIDs in a probe request. Since the SSID lists among different users are distinguishing, Pang et al. [18] take the SSID list as a feature for device fingerprinting. Kurtz et al. [22] propose an approach to fingerprinting mobile devices using personalized device configurations of apple's iOS platform, since mobile device configurations have now become so highly personalized that they can be utilized to generate a unique fingerprint for every user.

Moreover, the features from different layers are not independent, and can be combined for device fingerprinting. For example, Pang et al. [18] not only extract the features from the protocol layer, but also consider the features from the user layer. Yen et al. [23] construct device fingerprints using user-agent string, IP address, cookie ID and user login ID,

and identify devices with high precision and recall. Hupperich et al. [24] conduct a systematic study of browser, system, hardware, and behavioral features, and extract the corresponding features to design a fingerprinting system.

VI. CONCLUSION

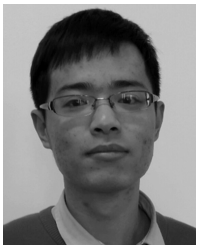
Device identification is an effective technology to enhance secure authentication. In this paper, we propose an approach to identify Android devices, utilizing implicit identifiers that can be acquired without requesting any permission. Firstly, we present 38 implicit identifiers (features) that cover the features of physical layer, application layer, and user layer, and introduce the corresponding ways to acquire these features. Then, we analyze the information entropy of these features, and do feature selection for device fingerprinting. Next, we propose an FMA algorithm, an FAA algorithm and an NBCA algorithm to achieve Android device identification, respectively. Finally, we conduct experimental evaluations on 50830 fingerprints from 2239 different Android devices. The results show that the performance of the NBCA algorithm is better than the two other algorithms, and its precision is over 99%, while its recall is over 98%.

In the future, we plan to apply our device identification approaches for user authentication, and analyze the security of our authentication mechanism.

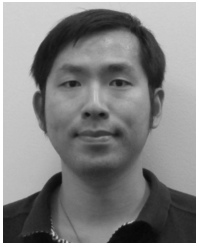
REFERENCES

[1] IDC. (Sep. 2016). *Flat Smartphone Growth Projected for 2016 as Mature Markets Veer Into Declines, According to IDC*. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS41699616
[2] S. Yu, "Big privacy: Challenges and opportunities of privacy study in the age of big data," *IEEE Access*, vol. 4, pp. 2751–2763, 2016.
[3] Z. Fang, W. Han, and Y. Li, "Permission based Android security: Issues and countermeasures," *Comput. Secur.*, vol. 43, pp. 205–218, Jun. 2014.
[4] Google. *Android API Guide*, accessed on Sep. 2016. [Online]. Available: http://developer.android.com/guide/index.html
[5] (Sep. 2016). *Self-Information*. [Online]. Available: https://en.wikipedia.org/wiki/Self-information
[6] L. Yu and H. Liu, "Feature selection for high-dimensional data: A fast correlation-based filter solution," in *Proc. 20th Int. Conf. Mach. Learn.*, Washington, DC, USA, 2003, pp. 856–863.
[7] P. Eckersley, "How unique is your Web browser?" in *Proc. 10th Int. Conf. Privacy Enhancing Technol.*, Berlin, Germany, 2010, pp. 1–18.
[8] (Sep. 2016). *F1 Score*. [Online]. Available: https://en.wikipedia.org/wiki/F1_score
[9] B. Danev, D. Zanetti, and S. Capkun, "On physical-layer identification of wireless devices," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 6:1–6:29, 2012.
[10] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Dependable Secure Comput.*, vol. 2, no. 2, pp. 93–108, Apr./Jun. 2005.
[11] R. M. Gerdes, T. E. Daniels, M. Mina, and S. F. Russell, "Device identification via analog signal fingerprinting: A matched filter approach," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2006, pp. 1–18.
[12] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proc. 14th ACM Int. Conf. Mobile Comput. Netw. (MobiCom)*, San Francisco, CA, USA, 2008, pp. 116–127.
[13] T. D. Vo-Huu, T. D. Vo-Huu, and G. Noubir, "Fingerprinting Wi-Fi devices using software defined radios," in *Proc. 9th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Darmstadt, Germany, 2016, pp. 3–14.
[14] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "AccelPrint: Imperfections of accelerometers make smartphones trackable," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2014, pp. 1–16.
[15] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting revisited: Generate stable device id stealthily with inaudible sound," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Scottsdale, AZ, USA, 2014, pp. 429–440.
[16] A. Das, N. Borisov, and M. Caesar, "Do you hear what I hear?: Fingerprinting smart devices through embedded acoustic components," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Scottsdale, AZ, USA, 2014, pp. 441–452.
[17] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. V. Randwyk, and D. Sicker, "Passive data link layer 802.11 wireless device driver fingerprinting," in *Proc. 15th Conf. USENIX Secur. Symp.*, Vancouver, BC, Canada, 2006, pp. 167–178.
[18] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall, "802.11 user fingerprinting," in *Proc. 13th Annu. ACM Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2007, pp. 99–110.
[19] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proc. SIGCOMM Workshop Mining Netw. Data*, Pisa, Italy, 2006, pp. 281–286.
[20] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-Boo, I still see you: Why efficient traffic analysis countermeasures fail," in *Proc. IEEE Symp. Secur. Privacy*, San Francisco, CA, USA, May 2012, pp. 332–346.
[21] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are?: Smartphone fingerprinting via application behaviour," in *Proc. 16th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Budapest, Hungary, 2013, pp. 7–12.
[22] A. Kurtz, H. Gascon, T. Becker, K. Rieck, and F. Freiling, "Fingerprinting mobile devices using personalized configurations," in *Proc. Privacy Enhancing Technol. Symp.*, Darmstadt, Germany, 2016, pp. 4–19.
[23] T. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi, "Host fingerprinting and tracking on the Web: Privacy and security implications," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2012, pp. 1–16.
[24] T. Hupperich, D. Maiorca, M. Kührer, T. Holz, and G. Giacinto, "On the robustness of mobile device fingerprinting: Can mobile users escape modern Web-tracking mechanisms?" in *Proc. 31st Annu. Comput. Secur. Appl. Conf.*, Los Angeles, CA, USA, 2015, pp. 191–200.

WENJIA WU received the B.S. and Ph.D. degrees in computer science from Southeast University in 2006 and 2013, respectively. He is currently a Lecturer with the School of Computer Science and Engineering, Southeast University. His research interest is wireless and mobile networks.



JIANAN WU is currently pursuing the master's degree with the School of Computer Science and Engineering, Southeast University. His research interests include mobile networks.



YANHAO WANG is currently pursuing the master's degree with the School of Computer Science and Engineering, Southeast University. His research interests include mobile networks.

ZHEN LING received the B.S. degree from the Nanjing Institute of Technology, China, in 2005, and the Ph.D. degree from Southeast University, Nanjing, China, in 2014, all in computer science. He joined the Department of Computer Science, City University of Hong Kong, from 2008 to 2009 as a Research Associate, and the Department of Computer Science, University of Victoria, from 2011 to 2013 as a Visiting Scholar. He is currently a Lecturer with the School of Computer Science and Engineering, Southeast University. His research interests include network security, privacy, and forensics.

MING YANG received the M.Sc. and Ph.D. degrees in computer science and engineering from Southeast University in 2002 and 2007, respectively. He is currently an Associate Professor with the School of Computer Science and Engineering, Southeast University. His research interests include network security and privacy, and wireless networks.