doi:10.3969/j.issn.1001-0505.2015.06.004

# 基于隐性标识符的零权限 Android 智能终端识别

王研昊[1] 马媛媛[2] 杨 明[1] 罗军舟[1]

([1] 东南大学计算机科学与工程学院, 南京 211189)
([2] 国网智能电网研究院, 南京 211106)

**摘要**: 针对现有 Android 智能终端识别直接利用 IMEI, Android_ID 等显性标识符存在依赖敏感权限和易失效的问题,提出了一种基于隐性标识符组合的零权限设备识别方法. 通过调用系统 API 并执行 Linux Shell 命令,从物理层、应用层以及用户层 3 个层次上获取设备型号、屏幕信息、内核编译信息、User Agent、系统语言、字体大小、字体列表、用户安装程序列表等 8 个隐性标识符,并将其组合形成设备指纹. 然后,提出了指纹精确匹配算法和变化指纹关联匹配算法,用于仅 1 个隐性标识符发生变化且变化前后相似程度大于 0.85 时的设备关联识别. 通过采集真实用户数据进行实验验证,结果表明所提算法的设备识别准确率达到 94.52%.

**关键词**: Android 系统;智能终端识别;隐性标识符;指纹匹配算法
**中图分类号**: TP311 **文献标志码**: A **文章编号**: 1001-0505(2015)06-1046-05

# Zero permission Android device identification based on implicit identifiers

Wang Yanhao[1] Ma Yuanyuan[2] Yang Ming[1] Luo Junzhou[1]

([1] School of Computer Science and Engineering, Southeast University, Nanjing 211189, China)
([2] State Grid Smart Grid Research Institute, Nanjing 211106, China)

**Abstract**: In the current technologies of Android device identification, explicit identifiers are usually adopted, such as IMEI (international mobile equipment identity) and Android_ID, which depend on sensitive permissions and are liable to fail. To solve these problems, a zero permission device identification method based on the implicit identifier group is proposed. From the physic layer, the application layer and the user layer, eight implicit identifiers including device type, screen information, kernel compile information, user agent, system language, font size, font list and user installed package list are obtained by calling system API (application programming interface) and executing Linux Shell commands. These eight implicit identifiers are combined to form the device fingerprint. Then, the corresponding fingerprint exact matching algorithm and the changing fingerprint associated matching algorithm are proposed, which can be applied to device related identification when only one implicit identifier changes and the similarity ratio is above 0.85 before and after the change. Finally, the experimental verification is carried out based on the real user data, and the results show that the device identification accuracy rate of the proposed algorithm can reach 94.52%.

**Key words**: Android system; device identification; implicit identifiers; fingerprint matching algorithm

目前智能终端呈现普及化趋势,其中 Android 系统占据了主导地位[1]. 在其发展过程中,智能终端识别技术(即在终端设备中提取设备指纹从而将不同的终端进行区分)被广泛应用于移动广告

**收稿日期**: 2015-03-31. **作者简介**: 王研昊(1990—),男,硕士生;杨明(联系人),男,博士,副教授,yangming2002@seu.edu.cn.
**基金项目**: 国家自然科学基金资助项目(61272054, 61402104, 61572130, 61320106007)、国家高技术研究发展计划(863 计划)资助项目(2013AA013503)、国家电网公司科技资助项目(EPRIXXKJ[2014]2244)、东南大学江苏省网络与信息安全重点实验室资助项目(BM2003-201)、东南大学计算机网络和信息集成教育部重点实验室资助项目(93K-9).
**引用本文**: 王研昊,马媛媛,杨明,等. 基于隐性标识符的零权限 Android 智能终端识别[J]. 东南大学学报:自然科学版,2015,45(6):1046-1050. [doi:10.3969/j.issn.1001-0505.2015.06.004]

精确投放、安全认证和访问控制等领域. 然而,现有的 Android 智能终端识别技术均基于显性标识符[2],例如操作系统信息(Android_ID)、设备相关信息(IMEI,Build. Serial)和 Sim 卡相关信息(IMSI、手机号码)等. 这些显性标识符存在 2 方面问题:① Android_ID,Build. Serial 并不完全可靠,可以被硬件制造商设置[3],也较容易被人为篡改、伪造;② 其他显性标识符(如 IMEI、IMSI、手机号码等)都需要获取 READ_PHONE_STATE 这一敏感权限,会导致权限滥用和用户隐私泄露等问题[4]. 为此,研究人员提出了包含物理层、协议层、应用层、用户层的一系列使用隐性标识符(特征)的设备识别方法,这些特征可以通过以下 2 种方式获取:① 在终端内部通过运行程序获取;② 在终端外部通过对其产生的流量、信号进行分析获取.

基于物理层的特征主要来源于设备硬件上的差异[5]. 基于此,Dey 等[6]在研究 Android 智能终端的三轴加速度器时发现,即使同一个型号的加速度器也会因为制造等原因产生细微的差别,但其所需相同振动刺激的条件较为理想,且采集程序需要常驻后台,采集周期较长;Zhou 等[7-8]通过智能终端麦克风记录声音来抽取特征,但在记录声音前需要申请 RECORD_AUDIO 权限. 基于协议层的特征主要来源于协议栈参数或协议驱动算法[9]. 基于应用层的特征主要来源于所运行应用类型及其行为[10];基于此,Stöber 等[11]在研究 Android 智能终端领域时,通过比对后台运行程序所产生的流量模式发现智能终端安装的应用程序列表. 基于用户层的特征主要来源于用户操作[12-13]. 上述 4 个层次的特征不是相互隔离的. Eckersley[14]针对浏览器指纹进行研究时,选取的特征涵盖了物理层、应用层以及用户层.

本文将在零权限的前提下针对 Android 智能终端进行识别. 由于研究的应用场景基于 Android 应用程序,故需要在 Android 智能终端内部获取隐性标识符. 基于物理层、应用层以及用户层,选择合适的隐性标识符形成设备指纹,并提出了相应的指纹识别算法,从而完成终端识别.

## 1　零权限指纹生成

### 1.1　指纹生成方法

不同于显性标识符,单个隐性标识符一般不能用来进行唯一识别,即多个智能终端存在一定的概率共享同一属性值. 虽然单个隐性标识符不具备唯一识别能力,但是将多个隐性标识符组合起来,就可以显著提高标识能力.

本文提出了一种基于隐性标识符组合的 Android 智能终端指纹生成方法,这些隐性标识符可以在零权限前提下通过调用系统 API 以及执行 Linux Shell 命令获取. 所有隐性标识符与其获取结果形成〈key,value〉键值对形式的字典,从而构成对应智能终端的设备指纹. 参考 Google 提供的 Android 官方文档并借鉴浏览器终端识别的相关工作[13-14],选用表 1 所示的隐性标识符组合.

**表 1　零权限获取的隐性标识符组合**

| 层次 | 序号 | 标识名称 | 获取途径 | 数据类型 |
|---|---|---|---|---|
| 物理层 | 1 | 设备型号 | 获取 Android. os. Build. MODEL 对应的值 | 字符串类型 |
| | 2 | 屏幕信息(宽度、高度、分辨率) | 通过 Resources 对象的 getDisplayMetrics( )方法获取 | 字符串类型,形式为宽度×高度×分辨率 |
| 应用层 | 3 | 系统内核编译信息 | 执行 Linux Shell 命令:cat/proc/version | 字符串类型,包含系统内核版本、编译者以及编译时间等信息 |
| | 4 | HTTP 数据包的 User Agent | 新建一个 WebView 对象,执行方法 getUserAgentString( ) | 字符串类型 |
| 用户层 | 5 | 系统语言 | 通过 Locale 对象的 getDisplayLanguage( )方法获取 | 字符串类型 |
| | 6 | 字体大小 | 通过 Resources 对象获取到系统 Configuration,进一步获取到 fontScale | 浮点数类型,系统默认字体大小为 1.0 |
| | 7 | 系统字体列表 | 执行 Linux Shell 命令:ls-la/system/fonts | 列表类型,其中每一项包含字体文件名(.ttf 文件)及其文件大小 |
| | 8 | 用户安装的应用程序列表 | 通过 PackageManager 获取用户安装程序列表 | 列表类型,其中每一项包含应用包名(如 com. sina. weibo)及其对应的 UID 号(10 000 ~ 19 999 之间的整数) |

如果不同用户使用的 Android 智能终端来自不同厂商,可以利用隐性标识符 1,2 进行区分. 如果不同用户使用同一款智能终端,不同批次出货的设备所带操作系统信息可能会存在差异,尤其是当用户进行刷机操作时,通过隐性标识符 3,4 便可进行区分. 如果用户恰巧使用了同一批次的同一款智

能终端,且没有进行刷机操作,但由于用户的使用习惯并非完全一样,不同用户对于系统语言、字体显示的需求以及使用的应用程序都会存在差异,故可通过隐性标识符 5 ~ 8 进行区分. 尤其是对于用户应用程序列表,除了应用包名外,还包含对应 UID 号,即使用户选择安装的应用列表完全相同,安装顺序的不同也会导致对应的 UID 号不同,发现其中差异便可实现不同用户间的区分. 除应用程序列表外,其他 7 个隐性标识符单独使用时区分能力较弱,组合使用时便能获得良好的区分效果.

### 1.2 理论分析

令 $F(y)$ 为 Android 设备 $y$ 所对应的设备指纹,$P(f_n)$ 为 $F(y)$ 分布所对应的离散概率密度函数,$f_n$ 为可能出现的设备指纹,且 $n \in [0,1,\cdots,N]$.

首先,引入信息论中惊异值 $I$ 的概念,其定义如下:

$$I(F(y)=f_n) = -\log(P(f_n)) \tag{1}$$

设备指纹由多个隐性标识符组合而成,可以通过计算每个隐性标识符的惊异值来判断其区分效果. 设某个隐性标识符为 $s$,其惊异值和信息熵分别为

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

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

针对某一特定的设备指纹,计算其中隐性标识符的惊异值,可得到各隐性标识符在该特定组合中的贡献信息;而针对总体样本,则计算隐性标识符的信息熵(即惊异值的期望),得到该隐性标识符总体上所具有的信息量.

## 2 指纹匹配算法

Android 智能终端设备指纹由多个隐性标识符组合而成,可以通过 2 个字典的精确匹配进行识别,但该方法没有考虑设备指纹可能发生的变化.

为了解决由于用户操作导致设备指纹发生变化影响匹配结果的问题,需要在精确匹配无法找到结果的情况下,引入关联匹配算法. 令已有指纹库为 $S$,新到来的设备指纹为 $x$,则根据关联匹配算法,在指纹库中找到指纹 $q$,满足 $x$ 与 $q$ 对应同一个设备,且由于用户操作等原因, $x$ 和 $q$ 不完全相同. 考虑到设备指纹变化的幅度不会太大,本文借鉴 Eckersley[14] 针对浏览器指纹的相关工作,提出了一种仅允许 1 个隐性标识符发生变化且变化前后相似程度大于 0.85 时的 Android 设备指纹关联匹配算法.

**算法 1** Android 设备指纹关联匹配算法

输入:已有设备指纹库 $S$,待关联的设备指纹 $x$.

输出:关联上的设备指纹,或 NULL(关联匹配失败).

```
C = dict(), R = [ ]
for all j in {设备型号, 屏幕信息, 系统内核, User Agent, 系统语言, 字体大小, 字体列表, 应用列表} do
    C[j] = [ ]
end
for all e in S do      //扫描整个指纹库 S
    if x 与 e 只有一个弱标识 j 不相等
        C[j].append(e)      //形成只有 1 个隐性标识符变化的待选设备指纹集合 C
end
for all j in C do      //在 C 中进行筛选找出最有可能变成 x 的设备指纹,用列表 R 存储
    if j in {设备型号, 屏幕信息, 系统内核, User Agent, 系统语言, 字体大小}
        for all e in C[j] do
            R.append(e)
        end
    else if j in {字体列表, 应用列表}
        M_f = NULL, M_p = 0
        for all e in C[j] do
            p = ListSimilarity(x[j], e[j])      //ListSimilarity 用杰卡德距离计算
            if p > 0.85 and p > M_p      //选择相似程度最大且不低于 0.85 的加入 R
                M_p = p, M_f = e
        end
        if M_f ! = NULL
            R.append(M_f)
end
if len(R) = = 1
    return R[0]
else
    return NULL
```

需要指出的是,算法中阈值 0.85 的选取是通过收集测试用户一段时间内的指纹数据并统计前后 2 条指纹中列表类型数据的变化情况分析得到的,同时该取值与文献[14]一致.

## 3 实验与结果分析

### 3.1 指纹数据的采集与分析

本实验设计了 Android 数据采集模块 DeviceFingerprint,收集设备指纹数据,同时该模块申请了 READ_PHONE_STATE 权限以获取设备的 IMEI 值. IMEI 值是可靠的显性标识符,会随着设备指纹一起上传,由此便可为数据集进行标记,以备后续评估分析设备指纹识别算法的效果. 为了采集真实用户数据,该模块被集成到东南大学先声网

客户端内，在全校范围内进行实验. 2014 年 9 月 22 日开始，截止到 2014 年 11 月 26 日，共收集到 22 798 条上传的设备指纹数据，其中共出现了 1 268 个不同的 IMEI 值，即对应了 1 268 个不同的 Android 智能终端.

由 1.2 节中的理论分析可知，根据式(1)计算数据集中不同设备指纹的惊异值，数据集中没有完全相同的 2 个设备指纹，故每台设备指纹惊异值均为 10.308 bit. 然后，根据式(2)和(3)分别计算设备指纹中各个隐性标识符的惊异值和信息熵. 信息熵越大，表示该隐性标识符所包含的信息量越大. 隐性标识符的信息熵和发生变化的次数 $U$ 见表 2. 由表可知，应用程序列表的信息熵最大，变化次数也最多，其他应用层和用户层隐性标识符也会发生变化.

**表 2　隐性标识符的信息熵**

| 参数 | 信息熵/bit | $U$ |
|---|---|---|
| 用户应用列表 | 10.306 | 5 403 |
| 内核信息 | 8.789 | 432 |
| User Agent | 8.039 | 132 |
| 设备型号 | 7.435 | 0 |
| 字体列表 | 6.890 | 150 |
| 屏幕信息 | 2.551 | 0 |
| 字体大小 | 1.172 | 42 |
| 语言 | 0.263 | 64 |

设备指纹与其发生变化隐性标识符数量的对应关系见图 1. 由图可知，不超过 1 个隐性标识符发生变化的指纹数占所有指纹数的 97.7%，说明算法 1 中只允许 1 个隐性标识符发生变化的限制是合理的.

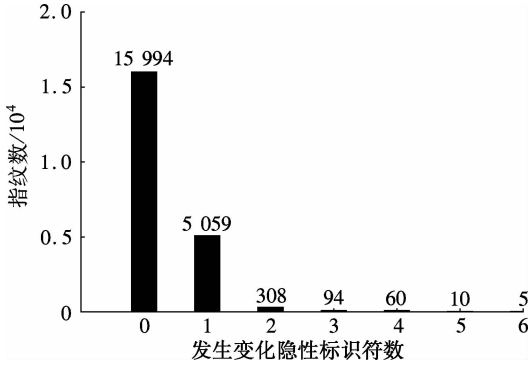


**图 1　数据集中设备指纹与发生变化隐性标识符数量间的关系**

### 3.2　匹配算法结果分析

为了验证匹配算法的执行效果，需要在数据集上进行测试. 按照每条指纹到来的时间顺序进行测试，对于新来的一条设备指纹，如果在指纹库中查找到匹配上的设备指纹，可依据 IMEI 值验证结果属于真阳性还是假阳性；如果在指纹库中没有找到，也可依据 IMEI 值验证结果属于真阴性还是假阴性；该条指纹匹配完成后，根据其 IMEI 值更新指纹库. 将准确率 $R_{\text{Accuracy}}$、误报率 $R_{\text{FPR}}$、漏报率 $R_{\text{FNR}}$ 作为匹配算法执行效果的评价指标，其计算公式分别为

$$R_{\text{Accuracy}}=\frac{N_{\text{TP}}+N_{\text{TN}}}{N_{\text{TP}}+N_{\text{TN}}+N_{\text{FP}}+N_{\text{FN}}} \tag{4}$$

$$R_{\text{FPR}}=\frac{N_{\text{FP}}}{N_{\text{TP}}+N_{\text{TN}}+N_{\text{FP}}+N_{\text{FN}}} \tag{5}$$

$$R_{\text{FNR}}=\frac{N_{\text{FN}}}{N_{\text{TP}}+N_{\text{TN}}+N_{\text{FP}}+N_{\text{FN}}} \tag{6}$$

式中，$N_{\text{TP}}$，$N_{\text{FP}}$，$N_{\text{TN}}$，$N_{\text{FN}}$ 分别表示结果属于真阳性、假阳性、真阴性、假阴性的数量.

精确匹配算法和关联匹配算法的测试结果见表 3. 由表可知，采用精确匹配算法时，在 22 798 条指纹中有 15 977 条指纹被正确匹配，1 268 条指纹被正确识别出不在指纹库中，5 553 条指纹被错误认为不在指纹库中；识别准确率为 75.6%，误报率为 0，漏报率为 24.4%. 这说明精确匹配总体效果不甚理想，误报率为 0 是期望的结果，但漏报率较高，究其原因在于没有考虑设备指纹发生变化的情况. 采用关联匹配算法时，识别准确率为 94.52%，误报率为 0.03%，漏报率为 5.45%. 相比精确匹配算法，关联匹配算法的识别准确率明显提高，同时漏报率也显著下降. 需要指出的是，结果中存在漏报是不可避免的，这是因为关联匹配算法仅允许 1 个隐性标识符发生变化，如果用户操作使得超过 1 个隐性标识符发生变化，算法便无法将同一个设备的 2 条指纹进行关联.

**表 3　2 种算法的测试结果**

| 测试结果 | 精确匹配算法 | 关联匹配算法 |
|---|---|---|
| $N_{\text{TP}}$ | 15 977 | 20 282 |
| $N_{\text{FP}}$ | 0 | 6 |
| $N_{\text{TN}}$ | 1 268 | 1 266 |
| $N_{\text{FN}}$ | 5 553 | 1 244 |

## 4　结语

本文提出了一种在零权限条件下基于隐性标识符的 Android 设备识别技术，将多个隐性标识符组合形成设备指纹，并对指纹识别算法进行了研究. 实验结果表明，隐性标识符组合具有较强的标识能力，所提的指纹匹配算法具有较好的识别效果. 在后续工作中，将尽可能挖掘潜在的隐性标识符，利用特征选择算法确定最优特征子集，并尝试其他机器学习识别算法.

## 参考文献(References)


[1] IDC. Smartphone OS market share[EB/OL]. (2015-01-15)[2015-01-31]. http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[2] Han S, Jung J, Wetherall D. An empirical study of third-party tracking by mobile applications in the wild [R]. San Jose, CA, USA: NSDI, 2012.

[3] Google Inc. Identifying app installations [EB/OL]. (2011-03-01) [2015-01-31]. http://android-developers.blogspot.jp/2011/03/identifying-app-installations.html.

[4] Grace M C, Zhou W, Jiang X, et al. Unsafe exposure analysis of mobile in-app advertisements[C]//*Proceedings of the Fifth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. Tucson, Arizona, USA, 2012: 101-112.

[5] Kohno T, Broido A, Claffy K C. Remote physical device fingerprinting[J]. *IEEE Transactions on Dependable and Secure Computing*, 2005, **2**(2): 93-108.

[6] Dey S, Roy N, Xu W, et al. AccelPrint: imperfections of accelerometers make smartphones trackable [C]// *Proceedings of the Network and Distributed System Security Symposium*. San Diego, CA, USA, 2014: 1-16.

[7] Zhou Z, Diao W, Liu X, et al. Acoustic fingerprinting revisited: generate stable device ID stealthily with inaudible sound[C]//*Proceedings of the* 2014 *ACM SIGSAC Conference on Computer & Communications Security*. New York, USA, 2014: 429-440.

[8] Das A, Borisov N, Caesar M. Do you hear what I hear? Fingerprinting smart devices through embedded acoustic components [C]//*Proceedings of the* 2014 *ACM SIGSAC Conference on Computer & Communications Security*. New York, USA, 2014: 441-452.

[9] Franklin J, McCoy D, Tabriz P, et al. Passive data link layer 802.11 wireless device driver fingerprinting[C]// *Proceedings of the* 15*th Conference on USENIX Security Symposium*. Vancouver, Canada, 2006: 167-178.

[10] Erman J, Arlitt M, Mahanti A. Traffic classification using clustering algorithms [C]//*Proceedings of the* 2006 *SIGCOMM Workshop on Mining Network Data*. Pisa, Italy, 2006: 281-286.

[11] Stöber T, Frank M, Schmitt J, et al. Who do you sync you are smartphone fingerprinting via application behaviour[C]//*Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*. Budapest, Hungary, 2013: 7-12.

[12] Pang J, Greenstein B, Gummadi R, et al. 802.11 user fingerprinting [C]//*Proceedings of the* 13*th Annual ACM International Conference on Mobile Computing and Networking*. Montreal, Canada, 2007:99-110.

[13] Acar G, Juarez M, Nikiforakis N, et al. FPDetective: dusting the web for fingerprinters[C]//*Proceedings of the* 2013 *ACM SIGSAC Conference on Computer & Communications Security*. Berlin, Germany, 2013: 1129-1140.

[14] Eckersley P. How unique is your web browser? [C]//*Proceedings of the* 10*th International Conference on Privacy Enhancing Technologies*. Berlin, Germany, 2010: 1-18.