情報処理学会論文誌 Vol.59 No.9 1678–1688 (Sep. 2018)

JDWPによる動的解析を利用したAndroidアプリケーションの端末識別情報利用実態調査

福田 泰平[1,a)] 明田 修平[1,†1] 瀧本 栄二[1] 齋藤 彰一[2] 毛利 公一[1]

受付日 2017年11月28日, 採録日 2018年6月8日

概要：Android アプリケーション（App）では，利用者や端末を一意に識別することを目的にグローバル ID が利用されている．グローバル ID の中には，利用者が変更できないために個人の特定につながる危険性を有するものがあり，Android ID，IMEI，IMSI，ICCID，電話番号，MAC アドレスなどが該当する．これらの非推奨なグローバル ID に対し，利用者が変更可能な安全なグローバル ID として，Ad ID や Instance ID が登場した．しかし，安全なグローバル ID を利用するかどうかは App の開発者に委ねられており，その利用実態は定かではない．また，App には，App 開発者以外の事業者などが作成した外部モジュールが組み込まれていることが多く，外部モジュールによる非推奨なグローバル ID の利用も懸念される．以上の背景から，グローバル ID の送信の実態を明らかにするために，マーケットに存在する 1,761 検体の App を対象に，JDWP に基づく手法による動的解析を用いてモジュールからの API 呼び出しとそのパッケージ名を観測した．調査の結果，約 26%の App が非推奨なグローバル ID を送信していることが分かった．また，約 15%の App では外部モジュールがそれらを送信していることが明らかとなった．

キーワード：Android，API トレース，グローバル ID，プライバシ保護

A Study of Usage of Device Identifiers on Android Applications Using Dynamic Analysis with JDWP

Taihei Fukuda[1,a)] Shuhei Aketa[1,†1] Eiji Takimoto[1] Shoichi Saito[2] Koichi Mouri[1]

Received: November 28, 2017, Accepted: June 8, 2018

Abstract: In Android applications (apps), global IDs are used for identifying a device or a user. Some global IDs, such as Android ID, IMEI, IMSI, ICCID, Phone Number and MAC Address, have a risk of identifying an individual because they cannot be changed by a user. As a measure against these unrecommended IDs, Advertising ID and Instance ID appeared as substitutes that can be changed by a user. However, whether the substitute IDs are used or not is fully depend on developers of apps, and the circumstances are unclear. On the other hand, many apps use external modules developed by other companies, etc., not an app's developer. So there is a concern of the use of the unrecommended IDs by the modules. In this paper, we conducted a dynamic analysis on 1,761 apps on a app market in order to clarify the circumstance of transferring global IDs outside. To achieve the purpose, we observed API calls from modules, and their package name, by the method based on JDWP. As a result, we revealed that approximately 26% of apps sent the unrecommended IDs outside. Moreover, we revealed that approximately 15% of apps sent them outside via external modules.

Keywords: Android, API tracing, global ID, privacy protection

[1] 立命館大学
Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[2] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555, Japan
[†1] 現在，トヨタ自動車株式会社
Presently with TOYOTA MOTOR CORPORATION
[a)] tfukuda@asl.cs.ritsumei.ac.jp

1. はじめに

近年，スマートフォン向けの OS として Android が普及している．International Data Corporation 社の調査 [1] では，2017 年第 1 四半期において約 85%のスマートフォンに Android が搭載されていると報告されている．また，

Androidでは，世界中の開発者によって作成されたAndroidアプリケーション（以下，Appと記す）をインストールすることで様々なサービスを受けることができる．AppBrain社の調査[2]では，2017年7月時点で約307万個のAppがAndroidの公式マーケットであるGooglePlay[3]に存在することが報告されている．

Appには，利用者や端末を一意に識別することを目的に，複数の事業者で継続して共有可能な識別子であるグローバルID[4]を取得しているものが存在する[5]．Appによって取得されるグローバルIDには，システムに固有なAndroid ID，端末に固有なIMEI，SIMカードに固有なIMSI，ICCID，電話番号，NICに固有なMACアドレスなどがある．グローバルIDの中には利用者によって変更できない識別子があり，Appがその識別子を取得すると，複数の事業者での名寄せなど他のプライバシ情報の紐づけに利用されて個人の特定につながる可能性がある．米連邦取引委員会は，文献[6]で利用者が変更可能なグローバルIDを使用した非特定化への考慮が重要であると指摘している．このようなグローバルIDのプライバシ上の危険性に対して，Googleは，2014年8月より利用者が変更可能なAdvertising ID[7]（以下，Ad IDと記す）を導入した．さらに，2015年5月には，Appの再インストールで変更可能なグローバルIDとしてInstance ID[8]を導入した．また，App開発者に対してUUID，Instance ID，Ad ID以外のグローバルIDを利用しないようにガイドラインを設けている[9]．また，2015年10月にリリースされたAndroid6.0ではMACアドレスの定数化[10]，2017年8月にリリースされたAndroid8.0ではAndroid IDの仕様変更[11]といった対策が行われた．このような対策の中，IMEI，IMSI，ICCID，電話番号はいまだAppによって取得可能であり，Android IDとMACアドレスもAndroidバージョンの古い端末では取得されてしまう可能性がある．さらに，利用者にプライバシポリシなどを明示し，ユーザがグローバルIDの取得を許諾した場合であっても，プライバシ上の危険性を考慮すると，非推奨なグローバルIDを用いることは好ましくない．

一方で，グローバルIDの送信に関与するのはApp開発者のみであるとは限らない．多くのAppには，広告掲載，App利用状況の把握，開発工程の削減などを目的にApp開発者以外が作成した外部モジュールが組み込まれており，これらの外部モジュールがグローバルIDの送信に関与していることが報告されている[5]．また，Graceら[12]によると，GooglePlayに存在する10万検体のAppに含まれていた広告モジュールを対象にした調査で，50個中33個の広告モジュールがIMEI，IMSI，ICCID，電話番号などを収集していることが示されている．このような，グローバルIDを送信する外部モジュールが組み込まれることで，多くのAppで非推奨なグローバルIDの送信が行われることにつながる．

以上の背景から，本論文では，Appによる非推奨なグローバルIDの外部送信の実態調査を実施した．この調査は，2017年7月から8月にかけて収集したGooglePlayの新着無料App 1,761検体を対象に，利用者のプライバシに配慮しない非推奨なグローバルIDの利用があるかを明らかにするものである．グローバルIDを送信するAppは，他のプライバシ情報との紐づけに利用されていることが推測されるため，この調査ではプライバシに配慮しない非推奨なグローバルIDの取得を試みるAppをプライバシの問題を有するものとして位置づけ，これらグローバルIDに着目して調査した．また，この調査では，外部モジュールと非推奨なグローバルIDの送信にどのような関連があるかも調査した．この調査では，App実行時の正確な挙動を基に実態を明らかにするため，1,761検体のAppに対して動的解析を行った．動的解析にあたり，Appによって実際に呼び出されたAPIを観測する機構（以下，APIトレース機構と記す）を構築した．具体的には，Javaアプリケーションのデバッグに利用されるJava Debug Wire Protocol（JDWP）[13]を用いて，グローバルID取得用APIや外部送信用APIにブレイクポイントを設定し，実行時のローカル変数や引数から通信内容を観測することでグローバルIDの送信が実際に行われているかを確認した．さらに，非推奨なグローバルIDの送信がどのモジュールによって行われているかを区別するため，JDWPを用いたAPIトレース機構にAPIを呼び出したモジュールを観測する機能を実装した．具体的には，APIトレース機構でAPI呼び出し時のスタックフレーム情報を取得することでAPIの呼び出し過程を取得する．取得したスタックフレーム情報からパッケージ名を取り出すことで，API呼び出しに関与したモジュールをパッケージ単位で識別した．

上記で述べたAPIトレース機構を用いた実態調査により，以下の実態が明らかとなった．

- 調査対象の約26%に及ぶ451個のAppが非推奨なグローバルIDを外部へ送信していた．
- 調査対象の約15%に及ぶ262個のAppでは外部モジュールが非推奨なグローバルIDを外部へ送信していた．

以下，本論文では，2章でグローバルIDの送信を観測するために作成したAPIトレース機構について述べ，3章でAPIトレース機構を利用した公式マーケットに存在するAppを対象をとした実態調査の内容と結果について述べる．4章で調査結果の考察について述べ，5章で関連研究について述べる．

2. APIトレース機構

Appによるグローバルの送信を確認するため，スタックフレーム情報を利用したモジュールを区別可能なAPIト

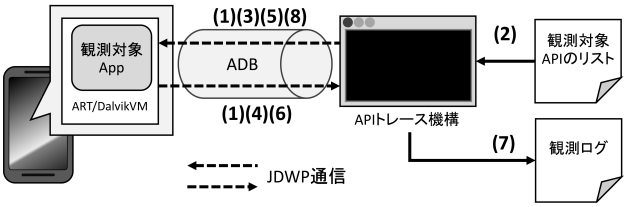

図 1 JDWP を利用した API トレース機構
Fig. 1 Mechanism of API tracing with JDWP.

レース機構を構築した．本章では，Java のデバッグインタフェースである JDWP を利用した API トレース機構の概要と，API トレース結果を利用してグローバル ID の送信の有無を明らかにする解析手順について述べる．また，スタックフレーム情報を基にした，API 呼び出し元モジュールを区別する手法について述べる．

2.1 JDWP を利用した API トレース機構

JDWP を利用して API の呼び出しを観測する機構を構築した．JDWP は，Android の Java VM である Android Runtime（以下，ART と記す）とデバッグ情報をやりとりすることができるプロトコルであり，ソケット通信を介してデバッグ対象の App を観測することができる．

API 呼び出しの観測は，API にブレイクポイントを設定することで行う．観測時に取得する API ごとの詳細情報（以下，API 情報と記す）は，ブレイクポイント到達時にローカル変数，引数，返り値，インスタンスフィールドを取得する（2.2 節で後述）．また，API の呼び出し元モジュールは，ブレイクポイントに到達したスレッドが管理しているスタックフレーム情報から呼び出し元モジュールのパッケージ名を取り出すことで特定する．以下，API トレース機構の具体的な動作について述べる（**図 1** 参照）．

(1) 観測対象の App を実行している ART と，API トレース機構を接続する．接続は，ADB [14] の JDWP 通信をフォワーディングする機能を利用して行う．
(2) 観測対象 API とブレークポイント到達時に取得するべき情報を記述した設定ファイルを読み出す．
(3) (2) の情報を基に観測対象の API にブレイクポイントを設定する．
(4) App のスレッドが (3) で設定したブレイクポイントに到達すると，ART はそのスレッドを停止させ，ブレイクポイントの位置（クラス名や行数など）を通知する．
(5) API トレース機構は，API 情報を取得するために，API 情報が格納されたローカル変数，返り値，インスタンスフィールド，および当該スレッドのスレッドオブジェクト内のスタックフレーム情報を要求する．
(6) (5) で要求した情報を受信する．
(7) API 呼び出し観測ログとして出力する．
(8) (4) で停止したスレッドを再開させる．以後，(4) から (8) を繰り返す．

2.2 観測対象 API

App によるグローバル ID の外部送信を観測するため，**表 1** の API を観測対象とした．

A. 外部通信に関する API App による外部との通信を観測するためには，通信内容と通信先の情報が必要である．そこで，ソケット通信や暗号化通信を行う API を観測対象とする．また，HTTP リクエストの送信に利用される API や，Android の WebView を使用してウェブページを表示するための API も観測対象とする．さらに，Android では，暗黙的インテントによって URL を指定することでブラウザ App を起動することができる．そのため，Activity 起動に使用される API も観測対象とすることで，ブラウザ App に送られた URL 情報を観測する．通信内容と通信先は，API 呼び出しの観測時に，API の引数やローカル変数から取得する．

B. グローバル ID 取得に関する API App による通信内容にグローバル ID が含まれていることを確認する必要がある．そこで，グローバル ID の取得に使用される API を観測対象とし，実際に取得されたグローバル ID を観測する．App によって取得されたグローバル ID は，API の返り値から取得する．

C. スタックフレーム情報の紐づけに必要な API スタックフレーム情報はスレッド単位で存在するため，スレッドを超えて呼び出し元モジュールを特定するための情報が必要となる．そこで，スレッドの生成とスレッド間通信を行う API を観測対象とすることで，スタックフレーム情報を紐づける情報を取得する．

2.3 観測ログ

API トレース機構が出力する観測ログの例を**図 2** に示す．図 2 は，スレッドを生成する API である Thread オブジェクトの start メソッドを観測したときのログである．観測ログには，API のパッケージ名を含むクラス名（reference）とメソッド名（method）や API を呼び出したスレッドの TID（thread.id）などを記録する．さらに，API 呼び出し時のスタックフレーム情報（api_info.stack）と API 呼び出し時のローカル変数やインスタンス変数の値を記録する．API 呼び出し時のスタックフレーム情報からパッケージ名を確認することで，どのモジュールが API の呼び出しに関与したかが分かる．

2.4 スタックフレーム情報の紐づけ

スタックフレーム情報はスレッド単位で管理されているため，複数のスレッドを介して行われた処理は，複数のス

表 1 観測対象 API
Table 1 Target APIs.

分類	API 名	取得する情報（API 情報）
A	libcore.io.Posix.sendto	通信内容
	com.android.org.conscrypt.OpenSSLSocketImpl$SSLOutputStream.write	通信内容
	com.android.okhttp.internal.huc.HttpURLConnectionImpl.connect	HTTP リクエストの内容
	org.apache.http.impl.client.DefaultRequestDirector. execute	HTTP リクエストの内容
	android.webkit.WebView.loadUrl	URL
	android.webkit.WebView.postUrl	URL，ポストされた情報
	android.webkit.WebView.loadDataWithBaseURL	URL
	android.app.Activity.startActivityForResult	ブラウザ App に送った URL
	android.app.Activity.startActivityAsUse	ブラウザ App に送った URL
B	android.provider.Settings$Secure.getString	Android ID
	android.telephony.TelephonyManager.getDeviceId	IMEI
	android.telephony.TelephonyManager.getSubscriberId	IMSI
	android.telephony.TelephonyManager.getSimSerialNumber	ICCID
	android.telephony.TelephonyManager.getLine1Number	電話番号
	android.net.wifi.WifiInfo.getMacAddress	MAC アドレス
	com.google.android.gms.ads.identifier.AdvertisingIdClient$Info.getId	Ad ID
	java.util.UUID.toString	UUID
	com.google.android.gms.iid.InstanceID.getId	Instance ID
C	java.lang.Thread.start	生成されたスレッドの ID
	android.os.MessageQueue.enqueueMessage	Message のオブジェクト ID


```
{
  ”time”: ”17:06:36.216”,
  ”reference”: ”java.lang.Thread”,
  ”method”: ”start”,
  ”signature”: ”()V”,
  ”pid”: 0,
  ”tag”: ”CALLED”,
  ”thread”: {
    ”id”: ”17900”,
    ”name”: ”pool−1−thread−1”
  },
  ”api_infos”: {
    ”stack”: [
      ”java.lang.Thread.start”,
      ”java.util.concurrent.ThreadPoolExecutor.addWorker”,
      ”java.util.concurrent.ThreadPoolExecutor.execute”,
      ”bje.run”,
      ”java.util.concurrent.ThreadPoolExecutor.runWorker”,
      ”java.util.concurrent.ThreadPoolExecutor$Worker.run”,
      ”java.lang.Thread.run”
    ],
    ”this.id”: 17901
  }
},
```


図 2 API トレースログの例
Fig. 2 Example of API tracing log.

タックフレーム情報を紐づける必要がある．本節では，スレッドの生成と Message オブジェクトによる処理委譲への対処方法について述べる．

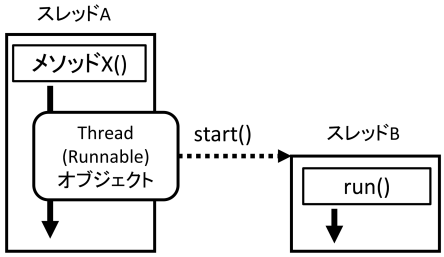

図 3 スレッド生成の流れ
Fig. 3 Flow of thread generation.

2.4.1 スレッド生成への対処

Java アプリケーションによるスレッド生成の様子を図 3 に示す．スレッド A は，スレッド B を生成するために Thread オブジェクトの start メソッドを呼び出す．スレッド B では Thread オブジェクトの run メソッドが実行される．両者のスタックフレーム情報を紐づけるためには，スレッド A の start メソッドとスレッド B の run メソッドを紐づける必要がある．

スレッド生成では，TID を利用することで紐づける．その手順を以下に示す．

(1) スレッド A によるスレッドの生成時は，インスタンスメソッドである start メソッドの呼び出し時に this オブジェクト（生成される Thread オブジェクト）から TID を取得する．

(2) スレッド B で実行された API は観測時には，実行されたスレッドの TID を取得する．

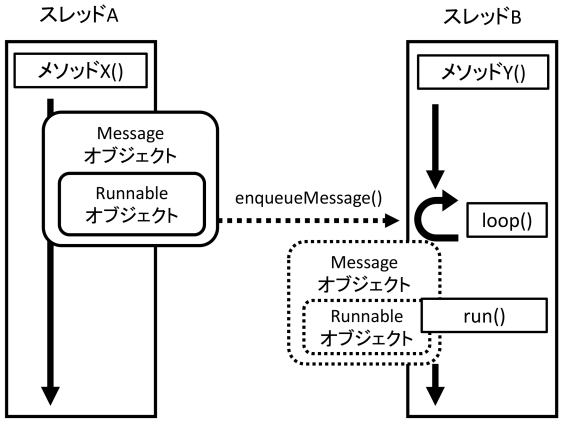

図 4 Message オブジェクトを利用した処理委譲の流れ
Fig. 4 Flow of processing delegation between threads with Message object.

スタックフレーム情報の最下位のメソッドが run メソッドである場合は，スタックフレーム情報が途切れていると判断できる．そこで，(1) と (2) で取得した TID が同一のスタックフレーム情報を紐づけることで対処する．

2.4.2 Message オブジェクトによる処理委譲への対処

Android では，Message オブジェクトを利用して，すでに存在するスレッドに任意の処理を委譲することができる（**図 4** 参照）．スレッド A が動作中のスレッド B に任意の処理を委譲する場合，まずスレッド A で Message オブジェクトを生成する．生成された Message オブジェクトには，処理させたい内容が実装された Runnable オブジェクトを保持させることができる．スレッド B は，Message オブジェクトを受信するために Looper クラスの loop メソッドで待機する．スレッド A は，Handler オブジェクトの enqueueMessage メソッドを利用して Message オブジェクトをスレッド B に送信する．スレッド B は，受信した Message オブジェクト中の Runnable オブジェクトで実装されている run メソッドを呼び出す．

両者のスタックフレーム情報を紐づけるために，やりとりされた Message オブジェクトのオブジェクト ID を利用する．その手順を以下に示す．

(1) スレッド A では，Message オブジェクトを enqueueMessage メソッドを利用してスレッド B に送信する．Message オブジェクトのオブジェクト ID を enqueueMessage メソッドの引数情報から取得する．
(2) スレッド B では，観測対象の API が呼び出された際に，loop メソッド呼び出し時のスタックフレーム情報を取得する．その後，スタックフレーム情報から loop メソッド呼び出し時のコンテキスト（ローカル変数や引数）を取得する．さらに，受信した Message オブジェクトが格納された loop メソッドのローカル変数から Message オブジェクトのオブジェクト ID を取得する．

スタックフレーム情報に loop メソッドが含まれている場合は，スタック情報が途切れていると判断できる．そこで，(1) と (2) で取得したオブジェクト ID が同一のスタックフレーム情報を紐づけることで対処する．

2.5 観測ログの解析

API トレース機構から出力された観測ログを解析し，グローバル ID の送信の有無を検出するとともに，送信に関与したモジュールをパッケージ名を基に抽出する．

グローバル ID の送信の確認 API トレース機構で観測した通信内容（表 1 の A）にグローバル ID（表 1 の B）と一致する文字列があるか比較することで判定する．なお，文字列比較を行う際には，グローバル ID に対して MD5，SHA1，SHA256，SHA512 でハッシュ化した値も比較対象とする．

関与したモジュールの抽出 API 呼び出し時に観測されたスタックフレーム情報について，必要であれば 2.4 節で述べた紐づけを行ったうえで，それらに含まれるパッケージ名を取り出すことで行う．

3. グローバル ID の外部送信の実態調査

本章では，2 章で述べた解析環境を利用して行った App によるグローバル ID の外部送信の実態調査とその結果について述べる．

3.1 調査目的と調査内容

本調査の目的は，App によるグローバル ID の外部送信の実態を明らかにすることである．また，App 内のモジュールを区別することで，グローバル ID の送信と組み込まれた外部モジュールとの関連を明らかにする．対象とするグローバル ID は，ハードウェアまたは Android システムに固有に割り当てられた端末の利用者によって変更することが困難な以下のグローバル ID を対象とした．

Android ID Android システムに割り当てられる識別子．Android の初期起動時にランダムに生成され，ファクトリーリセットを行うまで変更できない．

IMEI 携帯電話メーカによって端末に一意に割り当てられる識別子．携帯電話事業者が端末の接続拒否や機能停止を行う際に端末の識別に利用される．Android での取得には，READ_PHONE_STATE パーミッションが必要である．

IMSI SIM カードに格納された識別子．携帯端末の利用者を一意に識別できる．携帯電話事業者が利用者を識別するために利用される．Android での取得には，READ_PHONE_STATE パーミッションが必要である．

ICCID SIM カードに割り当てられた固有の識別子．IMSI と同様に携帯端末の利用者を一意に識別できる．

Android での取得には，READ_PHONE_STATE パーミッションが必要である．

電話番号 SIM カードに格納され，端末を一意に識別するために使用できる．Android での取得には，READ_PHONE_STATE パーミッションが必要である．

MAC アドレス NIC に一意に割り当てられる物理アドレス．Android のバージョン 6.0 以降では，Android フレームワークを使用して得られる MAC アドレスは定数化されており，App からは取得できない．

また，Google が外部送信での利用を推奨している以下のグローバル ID も調査対象とした．

UUID 任意のソフトウェア上で生成できるグローバル ID．時刻や乱数などを使用した様々な生成方法が存在する．App では，UUID を生成・保持することで，App がアンインストールされるまで App インスタンスの識別に利用することができる．

Instance ID App で利用可能な App インスタンスを識別できる識別子．セキュリティトークンや認証機能などに利用するための API も用意されている．

Ad ID GooglePlay 開発者サービスが提供する広告用のグローバル ID．ユーザによるリセットが可能である．

本調査では，これらのグローバル ID の利用傾向を明らかにするため，グローバル ID を App の内部で取得した検体数と外部へ送信した検体数を API トレース機構の観測ログから算出した．また，外部へ送信されたグローバル ID と外部モジュールとの関連を明らかにするため，外部モジュールごとに各グローバル ID 送信への関与が確認された検体数を算出した．

3.2 データセットと調査環境

本調査では，広く利用されている GooglePlay の App を対象とし，グローバル ID の利用実態調査を行った．本調査では，AdID や InstanceID，非推奨なグローバル ID の仕様変更後の最新の実態について明らかにするため，新着無料の App を対象とした．期間は 2017 年 7 月 7 日～8 月 5 日の間で毎週 1 回収集した．また，App が特定の種類に偏ることがないよう，全 32 のカテゴリから 20 個ずつ選択し，重複する App を除いた合計 1,761 検体を実態調査の対象とした．実行端末には，Android 6.0.1 を搭載した Nexus5 を使用した．

3.3 調査手順

API トレース機構を用いて以下の手順で行った．

(1) App を観測用端末にインストールし，API トレース機構で挙動を観測した．本調査では，グローバル ID および関連する情報の取得を目的とするモジュールは，それらが取得可能となると早い段階で送信を試みると想定し，App において必ず通る実行パスである起動からタイトル画面の出現までを観測範囲とした．また，実行時間や操作回数などの違いによって観測結果の違いが出ないように，以下の方針で App を実行した．
- 各検体をタイトル画面の到達まで実行し，その間の API 呼び出しを観測した．
- 実行中に画面に現れるランタイムパーミッションの要求はすべて許可した．
- タイトル画面の前の App 固有のログイン処理などは，可能であればスキップするようタップ操作した．

(2) 観測ログを 2.5 節で述べた方法で解析し，各グローバル ID ごとに送信が行われた検体数を算出した．

(3) (2) で確認したグローバル ID の送信に対して，スタックフレーム情報のパッケージ名を基に外部モジュールの関与がある検体数を算出した．なお，外部モジュールとは，10 検体以上の App に共通して組み込まれていたモジュールから，パッケージ名をもとにモジュールの提供元が確認できたものを外部モジュールとして定義した．本調査で用いたデータセットでは，42 の外部モジュールが得られた．

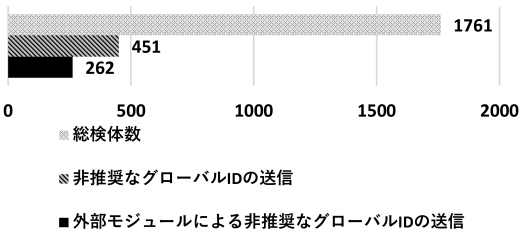

図 5 非推奨なグローバル ID の送信検体数
Fig. 5 Usage of global IDs without privacy consideration.

3.4 調査結果

3.4.1 グローバル ID の送信状況

利用が非推奨なグローバル ID である Android ID, IMEI, IMSI，ICCID，電話番号，MAC アドレスの送信が確認された App の検体数を**図 5** に示す．検体の約 26%（451 個）の App で非推奨なグローバル ID の送信が行われているという結果となった．また，検体の約 15%（262 個）の App で，外部モジュールによってそれらが送信されていることが分かった．このことから，非推奨なグローバル ID 送信の半数以上（約 58%）が，外部モジュールによるものであるということが明らかになった．

調査対象の 1,761 検体において，各グローバル ID ごとの取得した検体数と，それを外部へ送信した検体数を**図 6** に示す．多く取得されたのは，UUID と Android ID の約 68%であった．これらは，取得にパーミッションの要求が不要なため，端末の識別によく用いられる．最も多く外部へ送信されたのは Android ID であり 24%と高い割合で

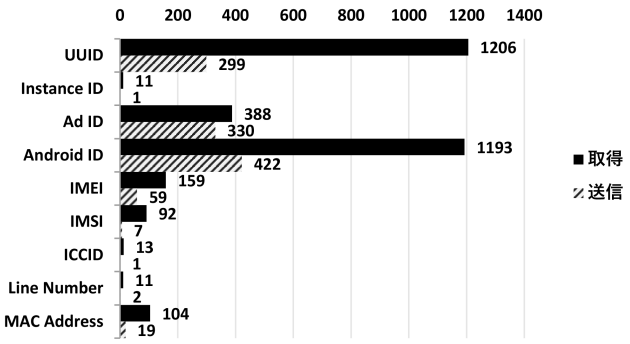

図 6 グローバル ID の取得と送信が確認された検体数
Fig. 6 Numbers of apps that fetch and send global IDs.

外部モジュール	組み込まれたApp数	モジュールによる送信を確認したApp数							
		推奨			非推奨				
		UUID	Instance ID	Ad ID	Android ID	IMEI	IMSI	ICCID	Line Number	MAC Address
Ad 01	65			1	62	1				
Ad 02	52	23		31	2					
Ad 03	46	29		12						
Ad 04	46	1		24						
Ad 05	33			3						
Ad 06	31	3		3						
Ad 07	29									
Ad 08	21			10						
Ad 09	20	14		2	13	1				
Ad 10	16			4						
Ad 11	16									
Ad 12	15			14						
Ad 13	13			3	2	2				2
Ad 14	10									
Ad 15	10									
Engine 01	72	13		19	53	4				
Engine 02	62				3					
Engine 03	48	3		4	1					
Engine 04	42			2		1				
Engine 05	28									
Monitor 01	141	40								
Monitor 02	131	36		36	97					
Monitor 03	99				4	2				
Monitor 04	73	21		19	8	28				
Monitor 05	32			12	12	3				
Monitor 06	29			13	13					
Monitor 07	10	6		1	2	4				
Support 01	275	63	1	79	6	1		1		
Support 02	105									
Support 03	74									
Support 04	64			18						
Support 05	50	6				1				1
Support 06	41									
Support 07	39				1					
Support 08	36	32								
Support 09	33									
Support 10	19									
Support 11	18									
Support 12	15	1		1	1					14
Support 13	14									
Support 14	12									
Support 15	11			5						

図 7 外部モジュールの利用検体数と各グローバル ID の送信検体数
Fig. 7 Numbers of apps that send global IDs for each external modules.

あった．Android ID は，Google のガイドラインでは外部への送信を控えるべきとされている．Android ID の代替として Google が提供した広告向けの Ad ID の送信検体数は 2 番目に多い 19%であった．今回の調査では，Ad ID よりも Android ID の方が送信検体数が多く依然 Android ID が多用されている結果となった．

3.4.2 送信モジュールの状況

外部モジュールごとのグローバル ID 送信について App 数を算出した結果を図 7 に示す．図 7 では，これらの外部モジュールを広告を表示する機能を持つ広告モジュール（Ad），App 開発用のフレームワークを提供する開発エンジン（Engine），App の利用傾向や不具合を監視する機能を持つ監視モジュール（Monitor），画像処理，ログイン処理，データベース，通知機能などの開発を支援する開発支援モジュール（Support）の 4 つに分類して示す．図 7 より，今回得られた 42 の外部モジュールのうち，18 の外部モジュールで非推奨なグローバル ID の送信が行われていた．一方で，監視モジュールのみに着目すると，7 個中 6 個で Android ID と IMEI が送信されており，非推奨なグローバル ID を利用する傾向にあることが分かった．広告モジュールは，15 個中 9 個で UUID や Ad ID が利用されており，安全なグローバル ID を利用している傾向が見られるが，Ad 01，Ad 09，Ad 13 のような一部の広告モジュールでは依然非推奨なグローバル ID を利用していた．開発支援モジュールは，モジュールの持つ機能によって利用するグローバル ID にばらつきがあり，15 個中 8 個の外部モジュールではグローバル ID が送信されていなかった．一方で，Support 05，Support 07，Support 12 など Android ID，IMEI，MAC アドレスの送信に関与した開発支援モジュールも確認できた．

4. 考察

3 章で述べた調査では，1,761 検体のうち約 26%の App で非推奨なグローバル ID が送信されていることが明らかとなった．また，約 15%の App では，外部モジュールによってそれが送信されていることが分かった．すなわち，非推奨なグローバル ID 送信の約 58%は外部モジュールが関与したものであり，App によって送信されるグローバル ID が，組み込まれた外部モジュールに大きく影響されていることが明らかとなった．

また，図 6 では，利用が推奨されていない Android ID を利用する App が多いことが明らかとなった．Android ID の取得や送信が多かった背景として，Android ID がパーミッションを要求することなく取得可能なため，利用するための敷居が低いことが要因としてあげられる．さらに，Android 端末の識別子として選択肢が少ないことがあげられる．iOS では，広告での使用を目的とした IDFA（Identification For Advertisers）とその他での使用を目的とした IDFV（Identifier For Vender）の 2 つのグローバル ID を提供している．IDFA はユーザの任意のタイミングでリセットが可能であり，IDFV は同一提供元の App をすべ

て削除すると変更されるグローバル ID である．IDFA は Ad ID に相当するグローバル ID であるが，IDFV のようなグローバル ID は Android で利用できない．Android においても，IDFV のような提供元単位のグローバル ID を設けることによって端末識別性の高いハードウェア識別子の利用を抑える効果が期待される．また，Android ID は，2017 年 8 月にリリースされた Android 8.0 において App 単位で値が変化する識別子に仕様が変更される [11] ため，今後の Android ID の利用傾向が注目される．

図 7 では，多くの外部モジュールで UUID と Ad ID が利用されているが，42 のうち 18 の外部モジュールで非推奨なグローバル ID を利用しているという結果となった．図 7 では同じ外部モジュールでも送信されるグローバル ID に違いが見られた．これは，App や組み込まれた外部モジュールのバージョンなどによって処理が異なるためであると推定される．また，本論文の実態調査では，App のタイトル画面が表示されるまでの期間が観測対象となっている．そのため，外部モジュールは組み込まれているがグローバル ID の送信まで実行されなかった場合が存在していると考えられる．このことから，図 7 で示したグローバル ID の送信状況は最低限観測された数であり，その数以上の App がグローバル ID 送信のリスクをかかえている可能性が考えられる．たとえば，IMEI の送信が確認された開発支援モジュール Support 05 を組み込んだ 50 検体の App は，今回観測された 1 検体と同様に IMEI が送信されるリスクをかかえている．

さらに，図 7 より，Ad 01，Engine 01，Monitor 02，Monitor 04，Support 12 のように，1 つのモジュールが多数の App に組み込まれ，多くの App でプライバシに配慮しない非推奨なグローバル ID の利用につながることが明らかとなった．そこで，図 7 において，Android ID，IMEI，MAC アドレスの送信で特徴的な Ad 01，Engine 01，Monitor 02，Monitor 04，Support 12 が組み込まれた App の開発者について追加調査を行った．その結果，Engine 01 と Monitor 02 については，幅広い App に組み込まれており，App 開発者による傾向は確認されなかった．一方で，Ad 01，Monitor 04，Support 12 については，App の提供元について**表 2** のような特徴的な傾向が得られた．表 2 は，Ad 01，Monitor 03，Support 12 による Android ID，IMEI，MAC アドレスの送信が確認された App の提供元と App の種類をまとめたものである．表 2 より，Ad 01 は提供元 04，11，15 で集中的に利用されていることが分かった．Monitor 03 も提供元 04，11 によって多用されている．特に，Support 12 を組み込んだ App はすべて提供元 16 によって提供されていた．また，表 2 の提供元の多くは，Android のホーム画面の見た目をカスタマイズするための着せ替え App を提供していた．さらに，Ad 01 を組み込んだ App を 31 個提供している提供元 15 も，ホーム画面上に気象情報を表示するウィジェット App を提供していた．このように，少数の開発者が，非推奨なグローバル ID を送信する外部モジュールを組み込んだ多数の App をリリースしたことが表 2 から明らかとなった．さらに，Ad 01 と Monitor 04 については，両者を含んでいる App が 23 検体確認され，着せ替え App を中心に Ad 01 と Monitor 04 に関連が見られた．このような着せ替え App やウィジェット App は，つねに端末上で動作し続けるため，Android ID，IMEI，MAC アドレスによる個人特定のリスクが非常に高く危険である．

表 2 App 提供元と Ad 01，Monitor 04，Support 12 が組み込まれた App 数

Table 2 Numbers of apps contains Ad 01, Monitor 04 and Support 12 for each app's releasers.

提供元	Ad01	Monitor04	Support12	App の種類
01	1	1		着せ替え
02	2	2		着せ替え
03	3	3		着せ替え
04	8	8		着せ替え
05	2	1		着せ替え
06		1		着せ替え
07	1	1		着せ替え
08		1		着せ替え
09	1	1		着せ替え
10	2			着せ替え
11	8	4		着せ替え
12		1		端末最適化
13		2		端末最適化
14	3	2		着せ替え
15	31			Widget，画面ロック
16			14	着せ替え

以上から，App に組み込まれた外部モジュールを考慮した様々な対策の必要性が確認できた．App 作成者は，図 7 のような非推奨なグローバル ID を送信する可能性がある外部モジュールについて認識し，利用を控えることで，プライバシに配慮しないグローバル ID の利用を抑制することができる．App のマーケットでは，非推奨なグローバル ID を送信する外部モジュールを利用している App を公開しないようにすることで，実際にグローバル ID が外部へ送信されるのを未然に防ぐことができる．外部モジュール提供者に対しては，UUID，Instance ID，Ad ID への移行を促すことで改善が期待できる．

5. 関連研究

App による情報漏洩を検出する研究に TaintDroid [15] に代表されるテイント解析技術を利用した研究 [16], [17], [18] がある．これらは，テイント解析技術を利用し，端末内の情報にタグを付与し，タグを追跡することによって外部へ情

報が漏洩したことを検出するものである．TaintDroid は，総務省が発行しているスマートフォンプライバシアウトルック [19] で第三者が情報漏洩を検出する動的解析技術として取り上げられている．ただし，これらは，送信に関与したモジュールを正確に特定できない．本論文では，API 呼び出し時のスタックフレーム情報を利用することで，グローバル ID の送信に関与したモジュールをパッケージ単位で特定した．

外部モジュールに関しては，外部モジュールを検出する研究 [20] や駆除をする研究 [21] などの外部モジュールに起因したプライバシ上のリスクを軽減する取り組みが行われている．また，外部モジュール上のコードに含まれるセキュリティリスクを対象にしたマーケット調査 [22], [23], [24], [25] が行われている．Taylor ら [22] は，パーミッションや外部モジュールの脆弱性が時間経過にともないどのように変化するか調査し，Watanabe ら [23] は，脆弱性が生じる要因を明らかにするため静的解析により外部モジュールに脆弱性が含まれているか調査しており，無料 App 200 万検体中約 70%，有料 App 3 万検体中 50%で外部モジュールによる脆弱性が確認された．Tang ら [24] は，Android パーミッションが必要な API がどのパッケージ中のコードに含まれているか静的解析することによって，マーケット上の App 10,710 検体のうち 2,762 検体で広告モジュールによる Android パーミッションの要求があることを明らかにした．ただし，これらの調査 [22], [23], [24] は，静的解析の技法を用いているものが多い．App は，外部との通信，リフレクションを用いたメソッド呼び出し，動的クラスロードなどが行われるため，静的解析では到達不能コードや通信内容などの App 実行時の環境に依存する動作まで解析することは困難である．本論文の調査では，プライバシに配慮しない非推奨なグローバル ID の送信について，外部モジュールの実挙動を基にした実態を明らかにした．動的解析を適用した調査も行われてきた．磯原ら [25] は，実行時に外部モジュールから送信された利用者情報の有無についてまとめている．しかし磯原らの調査は，2013 年時点のもので，1 章で述べた代替 ID の導入などのプライバシ保護対策が反映されていない．本論文の調査では，2017 年時点の新着無料 App を対象とし，プライバシ保護対策が進んでいく中でのグローバル ID の利用状況の変遷，実態を明らかにしている．また，本論文の調査では外部モジュールごとにグローバル ID の送信数について具体的に示しており，モジュールの利用状況が非推奨なグローバル ID の送信機会の増加につながっていることを明らかにした．

6. おわりに

本論文では，App によるグローバル ID の送信の実態を明らかにするため，1,761 検体の App に対して API トレースによる動的解析を行った．その結果，以下の事実が明らかとなった．

- 調査対象である 1,761 検体のうち，約 26%の App が非推奨なグローバル ID を送信しており，約 15%の App では外部モジュールが非推奨なグローバル ID を送信していた．
- 非推奨なグローバル ID 送信の約 58%は，外部モジュールが関与した送信であった．
- グローバル ID の中で最も外部へ送信された回数が多いものは，Android ID であった．
- IMEI，IMSI，MAC アドレスなどのハードウェア識別子を利用する外部モジュールが存在していた．
- App の利用率や不具合を監視する機能を持つ監視モジュールは，Android ID と IMEI を外部へ送信する傾向があった．
- 着せ替え App やウィジェット App は，Android ID，IMEI，MAC アドレスを送信していた．

以上のことから，2017 年 8 月時点においても，App によって非推奨なグローバル ID が送信され，個人が特定されるリスクがあることが確認できた．また，特定の App 提供元によって Android ID，IMEI，MAC アドレスを送信する可能性のある外部モジュールが多用されていることが明らかになった．App や外部モジュールによる個人が特定されるリスクは，利用者からは判断できないため，App 開発者によるこれらのリスクに対する正確な理解やモジュールの提供元による改善が求められる．

参考文献

[1] IDC: Smartphone OS Market Share 2017 Q1, IDC (online), available from ⟨https://www.idc.com/promo/smartphone-market-share/os⟩ (accessed 2017-10-16).

[2] AppBrain: Ad networks – Android libraries statistics, AppBrain (online), available from ⟨http://www.appbrain.com/stats/libraries/ad⟩ (accessed 2017-10-16).

[3] Google: Google Play, Google (online), available from ⟨https://play.google.com/store?hl=ja⟩ (accessed 2017-10-16).

[4] 高木浩光：緊急起稿 パーソナルデータ保護法制の行方 その 1，高木浩光@自宅の日記（オンライン），入手先 ⟨http://takagi-hiromitsu.jp/diary/20140422.html⟩（参照 2017-11-21）．

[5] 総務省：スマートフォンプライバシーイニシアティブ利用者情報の適正な取扱いとリテラシー向上による新時代イノベーション，利用者視点を踏まえた ICT サービスに係る諸問題に関する研究会 (2012).

[6] FTC: Internet of things Privacy & Security in a Connected World, FTC Staff Report (2015).

[7] Google：Android 広告 ID の使用–広告–収益化と広告–Developer Policy Center，Google Play デベロッパーポリシーセンター（オンライン），入手先 ⟨https://play.google.com/intl/ja/about/monetization-ads/ads/ad-id⟩（参照 2017-10-16）．

[8] Google: What is Instance ID? - Instance ID – Google Developers, Google Developers (online), available from ⟨https://developers.google.com/instance-id/?hl=ja⟩ (accessed 2017-10-16).

[9] Google: Best Practices for Unique Identifiers – Android Developers, Android Developers (online), available from ⟨https://developer.android.com/training/articles/user-data-ids.html⟩ (accessed 2017-10-16).
[10] Google：Android 6.0 の変更点 – Android Developers，Android Developers（オンライン），入手先 ⟨https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html⟩（参照 2017-10-16）.
[11] Google：Android O での動作変更点 – Android Developers，Android Developers（オンライン），入手先 ⟨https://developer.android.com/preview/behavior-changes.html⟩（参照 2017-10-16）.
[12] Grace, M.C., Zhou, W., Jiang, X. and Sadeghi, A.-R.: Unsafe Exposure Analysis of Mobile In-app Advertisements, *Proc. WISEC '12*, pp.101–112, ACM (2012).
[13] Oracle: Java Debug Wire Protocol, Oracle (online), available from ⟨https://docs.oracle.com/javase/jp/8/docs/technotes/guides/jpda/jdwp-spec.html⟩ (accessed 2017-11-19).
[14] Google: Android Debug Bridge, Android Developers (online), available from ⟨https://developer.android.com/studio/command-line/adb.html?hl⟩ (accessed 2016-10-03).
[15] Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L.P., Jung, J., McDaniel, P. and Sheth, A.N.: TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones, *ACM Trans. Computer Systems* (*TOCS*), Vol.32, No.2, pp.1–29 (2014).
[16] Schuette, J., Kuechler, A. and TItze, D.: Practical Application-Level Dynamic Taint Analysis of Android Apps, *Proc. 2017 IEEE Trustcom/BigDataSE/ICESS*, pp.17–24 (2017).
[17] Yoon, M.-K., Salajegheh, N., Chen, Y. and Christodorescu, M.: PIFT: Predictive Information-Flow Tracking, *SIGPLAN Not.*, Vol.51, No.4, pp.713–725 (2016).
[18] Shankar, V.G., Somani, G., Gaur, M.S., Laxmi, V. and Conti, M.: AndroTaint: An efficient android malware detection framework using dynamic taint analysis, *Proc. 2017 ISEA Asia Security and Privacy* (*ISEASP*), pp.1–13 (2017).
[19] 総務省：スマートフォン上のアプリケーションにおける利用者情報の取扱いに係る技術的検証等の諸問題に係る実証調査研究，スマートフォンプライバシーアウトルック II，スマートフォンアプリケーション プライバシーポリシー普及・検証推進タスクフォース (2015).
[20] Soh, C., Tan, H.B.K., Arnatovich, Y.L., Narayanan, A. and Wang, L.: LibSift: Automated Detection of Third-Party Libraries in Android Applications, *Proc. 2016 23rd Asia-Pacific Software Engineering Conference* (*APSEC*), pp.41–48 (2016).
[21] Pan, J.Y. and Ma, S.H.: Advertisement removal of Android applications by reverse engineering, *Proc. 2017 International Conference on Computing, Networking and Communications* (*ICNC*), pp.695–700 (2017).
[22] Taylor, V.F. and Martinovic, I.: To Update or Not to Update: Insights From a Two-Year Study of Android App Evolution, *Proc. ASIA CCS '17*, pp.45–57, ACM (2017).
[23] Watanabe, T., Akiyama, M., Kanei, F., Shioji, E., Takata, Y., Sun, B., Ishi, Y., Shibahara, T., Yagi, T. and Mori, T.: Understanding the Origins of Mobile App Vulnerabilities: A Large-scale Measurement Study of Free and Paid Apps, *Proc. MSR '17*, pp.14–24, IEEE (2017).
[24] Tang, J., Li, R., Han, H., Zhang, H. and Gu, X.: Detecting Permission Over-claim of Android Applications with Static and Semantic Analysis Approach, *Proc. 2017 IEEE Trustcom/BigDataSE/ICESS*, pp.706–713 (2017).
[25] 磯原隆将，川端秀明，竹森敬祐，窪田 歩：Android アプリの静的解析を用いた利用 API と外部モジュール検知によるプライバシーポリシー作成支援機構，研究報告セキュリティ心理学とトラスト（SPT），Vol.2013-SPT-6, No.63, pp.1–8 (2013).

福田 泰平 （学生会員）

1994 年生．2017 年立命館大学情報理工学部情報システム学科卒業，同年同大学大学院情報理工学研究科博士前期課程情報理工学専攻に入学，現在に至る．システムソフトウェア，コンピュータセキュリティ等に興味を持つ．

明田 修平 （正会員）

2013 年立命館大学情報理工学部情報システム学科卒業，2015 年同大学大学院情報理工学研究科博士前期課程情報理工学専攻修了，2018 年同大学院情報理工学研究科博士後期課程情報理工学専攻修了．同年トヨタ自動車株式会社入社，現在に至る．オペレーティングシステム等の研究開発に従事．博士（工学）．

瀧本 栄二 （正会員）

1999 年立命館大学理工学部情報学科卒業，2001 年同大学大学院理工学研究科博士前期課程修了，2005 年同研究科博士後期課程単位取得退学，同年（株）ATR 適応コミュニケーション研究所専任研究員，2010 年立命館大学情報理工学部情報システム学科助手，2017 年立命館大学情報理工学部情報理工学科助教，現在に至る．主にシステムソフトウェア，無線ネットワークに関する研究に従事．博士（工学）．電子情報通信学会会員．

齋藤 彰一 （正会員）

1993 年立命館大学理工学部情報工学科卒業．1995 年同大学大学院博士前期課程修了．1998 年同大学院博士後期課程単位習得中退．同年和歌山大学システム工学部情報通信システム学科助手．2003 年同講師，2005 年同助教授．2006 年名古屋工業大学大学院助教授，2007 年同准教授，2016 年同教授，現在に至る．オペレーティングシステム，インターネット，セキュリティ等の研究に従事．博士（工学），ACM，IEEE-CS 各会員．

毛利 公一 （正会員）

1994 年立命館大学理工学部情報工学科卒業，1996 年同大学大学院理工学研究科修士課程情報システム学専攻修了，1999 年同研究科博士課程後期課程総合理工学専攻修了．同年東京農工大学工学部情報コミュニケーション工学科助手，2002 年立命館大学理工学部情報学科講師，2004 年同大学情報理工学部情報システム学科講師，2008 年同准教授，2014 年同教授となり，現在に至る．博士（工学）．オペレーティングシステム，仮想化技術，コンピュータセキュリティ等の研究に従事．電子情報通信学会，ACM，IEEE-CS，USENIX 各会員．