情報処理学会論文誌 Vol.60 No.12 2259–2268 (Dec. 2019)

Androidにおける端末識別情報送信検出のための動的解析システム

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

受付日 2019年3月8日, 採録日 2019年9月11日

概要：Android アプリケーション（App）には，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 を利用しているものがある．グローバル ID の中には，利用者による変更ができないという性質から個人特定につながる危険性が指摘されている．先行研究（福田ら，2018）では，App によるグローバル ID の利用を，動的解析により通信内容を確認することで，明らかにした．このような App の動的解析を行うためには，App の起動だけでなく解析中の GUI 操作が必要である．しかし，GUI 操作は解析者の経験則に依存していたり，グローバル ID の送信が行われたタイミングや送信の契機となった GUI 操作などの具体的な状況を知ることができないといった問題がある．以上の背景から，GUI 操作の自動化とグローバル ID 送信時の状況を取得可能とする動的解析システムを提案する．このシステムは，API トレースによる動的解析と GUI 操作と記録を同時に行うことで目的を達成している．評価では，マーケットに存在する実 App を対象に動的解析を実行し，画面遷移や GUI 操作の座標など，グローバル ID の送信の契機となった GUI 操作の傾向や送信タイミングを明らかにすることができた．

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

Dynamic Analysis System for Detection of Device Identifier Transmission in Android

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

Received: March 8, 2019, Accepted: September 11, 2019

Abstract: In the most of Android applications (apps), global IDs are generally used for identifying a specific user or mobile device. Some global IDs have a risk of identifying an individual because they cannot be changed by any user. In the previous study (Fukuda et al., 2018), we have revealed the usage of global IDs by apps through dynamic analysis and confirmed the communication contents. In order to perform dynamic analysis of an app, apart from launching the app, it is also necessary and important to analyze the GUI operation during the app execution. However, there exist some problems, e.g., the GUI operations usually depend on the heuristics of the analyzer. In addition, it is impossible to know the specific situations such as the transmission timing of the global IDs and the GUI operations that triggered the transmission. In this paper, we propose a dynamic analysis system that is able to automate the GUI operations and acquire the situation where the global IDs are transmitted. The proposed system achieves these above purposes by simultaneously performing the dynamic analysis, GUI operation, and GUI operation recording. In the evaluation, we analyzed real apps from GooglePlay with the proposed system. As a result, we clarify the tendency of GUI operations that trigger these transmissions of global IDs, including transmission timing, screen transition, and the coordinates.

Keywords: Android, dynamic analysis, 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，アドソル日進株式会社
Presently with Ad-Sol Nissin 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．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，2019 年 1 月時点で約 252 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]．このグローバル ID には，利用者による変更ができないという性質を持つもの（以下，非推奨なグローバル ID と記す）があり，複数の事業者による利用者情報の紐づけや永続的な個人の追跡につながるなどの危険性が指摘されている [1], [6]．このような，危険性から App 利用者を守るためには，実態把握によるマーケットの健全な運用が求められる．

我々は，このグローバル ID の持つ危険性についてその実態を明らかにするため調査を実施した [1]．その結果，Java のデバッグインタフェースである JDWP を利用した動的解析により，App による外部通信を観測できた．これにより，GooglePlay に存在する App 1,761 検体の約 26%による非推奨なグローバル ID の送信を明らかにした．この調査を行うためには，App 実行時の挙動を動的解析するために実際に App を実行する必要がある．また，多くの App は，タップやスクロールなどに起因したイベントドリブンな設計になっているため，解析中の GUI 操作が必要である．しかし，動的解析中の App への GUI 操作や実行時間は，解析者の経験則にのっとって行われていることが多い．我々の調査 [1] でも，App 起動後からタイトル画面の出現までの挙動に着目しており，その判断や GUI 操作は解析者にゆだねられ，動的解析の目的達成に十分であるか確認することができない．同様に，App の動的解析技術やそれらを利用した調査 [7], [8], [9], [10], [11] でも，実行時間や GUI 操作に関する検討が不十分であったり経験則に任されていることが多い．

本論文では，グローバル ID の送信を引き起こした GUI 操作やそのときの画面の状態などを取得可能とする手法を提案する．具体的には，App 実行時の動的解析を API トレースにより行い，画面の取得や GUI 操作は UI Automator [12] を用いて機械的に実行および記録することで実現する．このとき，両者から得られるログのタイムスタンプに着目することで，動的解析の対象となる App の挙動に起因した GUI 操作を抽出することを可能にする．評価では，マーケットに存在する App に対して提案手法を適用し，グローバル ID 送信までに要する画面遷移や GUI 操作に関する情報を得られるか確認する．

以下，本論文では，2 章で提案手法について述べ，3 章で GUI 操作自動化およびその記録を行う処理について具体的に述べる．4 章でマーケットに存在する App を対象に行った提案手法の評価について述べ，5 章で関連研究について述べる．

2. 提案手法

App による動的振舞いと GUI 操作を紐づけるためには，App の動的解析と機械的な GUI 操作の記録を行う必要がある．そこで，両者を同時に実現する App の解析手法を提案する．提案手法の全体像を**図 1** に示す．本手法は，JDWP による API トレースを行う API トレース部，GUI 操作の自動化および記録を行う GUI 操作自動化部，その 2 つの連携によって実現される．以下，本章ではその概要を述べる．

API トレース部

API トレース部では，App の動的振舞いを観測する API トレース機構 [1] が中心となり解析を行う．API トレース機構は，JDWP を利用して Android フレームワーク内にブレークポイントを設定することで，App による API の呼出しを観測する．また，観測対象 API の呼出しごとに API 名，実行されたスレッド名，ローカル変数および返り値を取得してログとして出力することで，App によって取得された利用者情報や外部との通信内容を観測可能とする．これにより，App による非推奨なグローバル ID の送信がいつ行われたか観測ログから確認することができる．

GUI 操作自動化部

GUI 操作自動化部では，API トレース部での動的解析を進める際に必要な解析対象 App への GUI 操作を自動化する．自動化を実現するため，Android SDK に付属している GUI テストツールである UI Automator を利用する．UI Automator により，操作対象の App がブラックボックスなものであっても，App 実行時に取得した画面の構成に

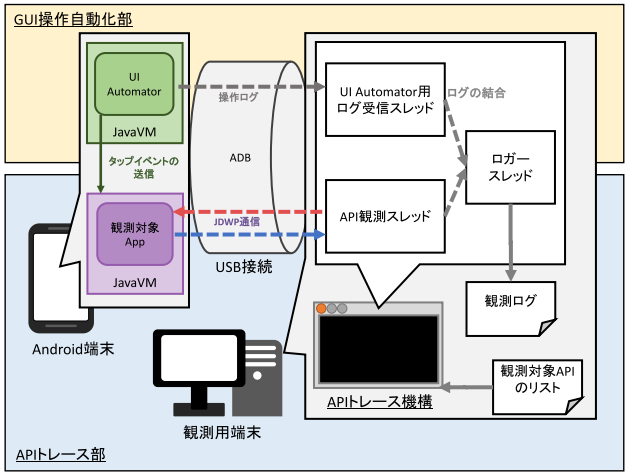

図 1 提案手法の概要
Fig. 1 Overview of proposed method.

基づいて，操作可能な箇所に GUI 操作を行う．

両機構の連携

API トレースログと GUI 操作を紐づけるため，自動化により実行した GUI 操作の記録を行う．さらに，操作中のスクリーンショットを撮ることで，動的解析時の GUI 操作や画面に掲示されていた内容を復元することを可能にする．記録された GUI 操作のログは，API トレース機構に送信され，ロガースレッドによって API トレースログとともに観測ログに書き込まれる．これにより，API トレースログで観測した App の動的振舞いと GUI 操作の関係を観測ログから解析することができる．

3. GUI 操作自動化および記録

3.1 Android における GUI の構成要素

App の画面は，Activity と呼ばれる単位で状態遷移する．Activity の外観は，GUI 部品を表現する View オブジェクトの集合によって構成される．App の画面構成を担っている各要素とその関係について**図 2** に示す．

View View は，Android の画面上にある GUI 部品の基底クラスである．すべての GUI 部品は View を継承している必要がある．View には，ラベル，ボタン，チェックボックス，ラジオボタンなど App の見た目に影響するものが多い．

ViewGroup ViewGroup は，View の子クラスであり複数の View オブジェクトを内部に保持する機能を持つ View である．ViewGroup によって，1 画面を構成する View の集合は，階層構造として表現することができる．ViewGroup は，レイアウトを整えるための機能を持つものが多く，代表的なものに複数の View オブジェクトを線形に整列させる LinearLayout がある．また，View の階層構造の起点にある DecorView も ViewGroup の一種である．

Window Window は，Android 端末の画面の矩形領域を管理する単位である．Window は，DecorView を 1 つ保持し，それに連なる View の階層構造を基に矩形領域内に GUI を描画する．1 つの Activity は，1 つの Window を生成し，その中で App の GUI が画面に表示される．

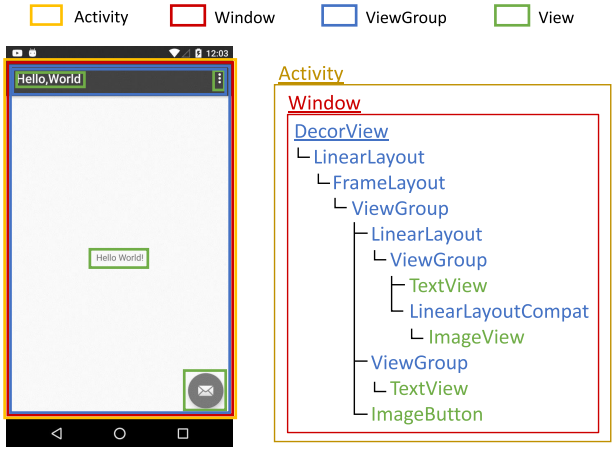

図 2 Android の画面と構成要素
Fig. 2 Component of Android's GUI.

3.2 UI Automator による GUI 操作の自動化

3.2.1 UI Automator の概要

UI Automator は，App の GUI をテストするフレームワークである．Java の単体テストに用いられる JUnit と同様にテストシナリオを記述し，Android 端末上にインストールされた JUnitRunner によってテストが実行される．通常の JUnit を使用したテストではテスト対象のクラスなどが定義されたソースコードが必要であるが，UI Automator では Android SDK で提供されている API を使用することで，ソースコードがなくても View オブジェクトの取得やタップ操作のエミュレーションが行える．

UI Automator 用に作成されたテストシナリオは，App と同様に APK ファイルにビルドされる．端末内にインストールされた後は，ADB を介してクラスを指定することでテストシナリオが実行される．

3.2.2 状態識別手法

GUI 操作の自動化を効果的に行うためには，画面の状態の把握や操作済み View の区別により重複した GUI 操作を削減する必要がある．一方で，UI Automator では，Activity 名を取得できないため，テスト対象の App の状態を識別する他の手法が必要である．さらに，UI Automator で取得された View がすでに GUI 操作済みであるかどうか判別する必要もある．そこで，提案手法では，観測対象 App の GUI 構成から得られる情報を基に現在の画面の状態と View を識別する．具体的には，(1) GUI 情報の取得，(2) 画面の識別，(3) View の識別，の 3 段階の処理により画面の区別，およびクリック済み View の区別を行う．

(1) GUI 情報の取得

UI Automator では，テストシナリオ内で GUI の解析を行うために dumpWindowHierarchy メソッドが用意されている．dumpWindowHierarchy メソッドは，Android システム中で現在画面に表示されている View オブジェクトすべてを XML 形式でダンプした結果を呼出し元に提供する．XML 形式のデータには，View を表す node と名付けられたエレメントが格納されている．View 階層構造は，node エレメントの入れ子関係によって表現されている．node エレメントの属性には**表 1** で示すものがある．

(2) 画面の識別

画面の識別には，画面全体を構成する GUI 情報を基に識別するのが適切である．そこで dumpWindowHierarchy メソッドの結果を利用して画面に固有

表 1 node エレメントの属性
Table 1 Attribute of node element.

属性	値
index	親から見た要素番号
text	表示されているテキスト
resource-id	プログラムからアクセスするための識別子
class	View の種類（クラス名）
content-desc	代替テキスト
checkable	チェック操作の可否
checked	チェックの有無
clickable	クリック操作の可否
long-clickable	長押しクリック操作の可否
enabled	有効か無効か
scrollable	スクロール操作の可否
password	パスワード用テキストエリアであるか
selected	選択されているか
visible-to-user	端末のユーザが視認できるか
bounds	描画エリア（絶対座標）

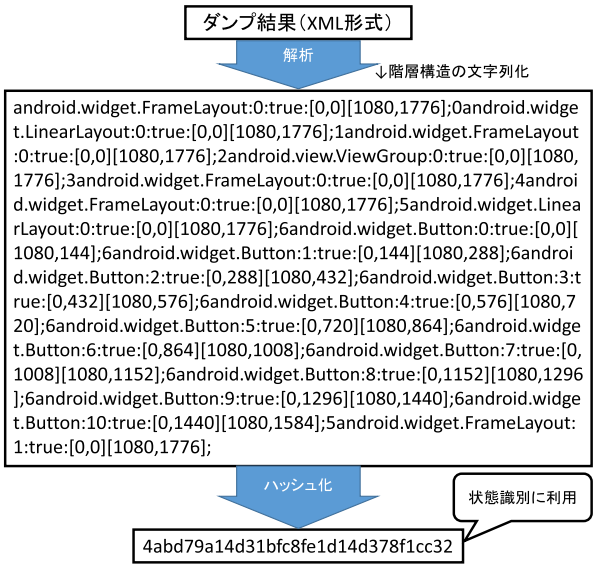

図 3 画面状態の識別子生成手順
Fig. 3 Method of generating GUI identifier.

な値を生成することで識別する．

画面の識別子を生成する手順を図 3 に示す．まず，dumpWindowHierarchy メソッドで取得した画面構成のダンプ結果をパースし，node エレメントの構成から View 階層構造を取得する．取得した View から属性や描画エリアなどの情報を取得し，View 階層構造に関する情報も含めて文字列化する．文字列化したデータを MD5 でハッシュ化し，その結果を状態識別子として使用する．

View 階層構造の文字列化に利用する属性が View 階層構造（階層の深さ，親から見て何番目の要素であるか）と View の種類（View のクラス名）の 2 つであると，同じ View 階層構造を持つ異なる見た目の Activity を区別することができない．そのため，表 1 の属性

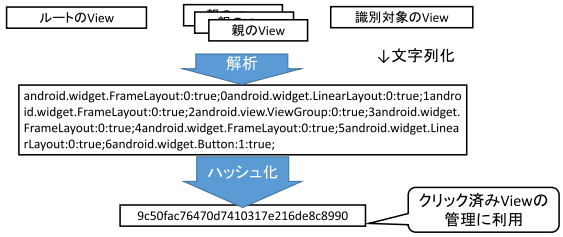

図 4 View の識別子生成手順
Fig. 4 Method of generating View identifier.

から文字列化する情報を追加し，より細かく画面の状態を識別できるようにする．提案手法による GUI 操作自動化では，テキストボックスへの入力やチェックボックスへの操作もエミュレーションの対象とする．また，スクロール操作による View の位置の変化も状態変化に含める．そのため，画面の状態識別子生成に利用する文字列には，node エレメントの階層構造と class，index，text，checked，enabled，bounds を含める．

（3）View の識別

操作済みの View を識別するためには，現在の画面に対して View を一意に識別する必要がある．そこで，複数の画面状態間での重複は許容し，画面状態識別子とともに利用することでクリック済みの View に固有な値を生成することで識別する．

View の識別子を生成する手順を図 4 に示す．画面の状態識別と同様に，クリック対象の View の識別も View の解析結果を文字列化し，MD5 でハッシュ化する．ハッシュ化する情報は，View 階層構造全体ではなく，識別子を生成したい View から親に向かってたどることで解析できる，View 階層構造上の位置を使用する．そこで，文字列化する情報には，ルート View から識別対象の View までの View 階層構造とクラス名を列挙したものを文字列化する情報として使用する．

3.2.3 GUI 操作の流れ

UI Automator を利用して API トレース中の GUI 操作を行うテストシナリオのフローチャートを図 5 に示す．GUI 操作開始直後，ループ処理に入る．この 1 ループが GUI 操作 1 回分に相当する．以下，ループ内の処理について説明する．

画面が更新されるまで待機

提案手法では，UI Automator と API トレース機構を併用する必要があるため，操作対象の App に生じる API トレースのオーバヘッドを考慮する必要がある．そのため，App による画面の更新のタイミングを正しく観測することが重要である．そこで，本段階では UI Automator で用意されている API を利用して 2 段階で待機処理を行う．待機処理のコードを図 6 に示す．図 6 の (1) にある waitForWindowUpdate メソッ

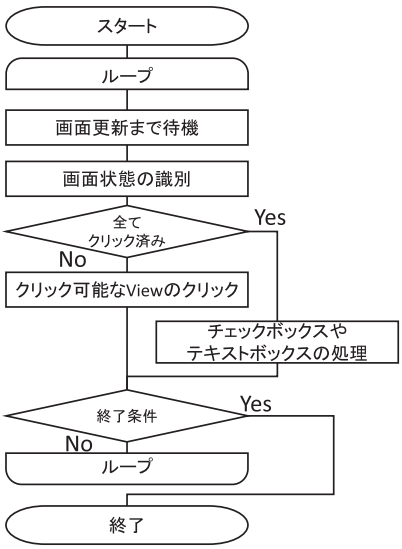

図 5 テストシナリオのフローチャート
Fig. 5 Flow chart of test scenario.

```
long UPDATE_TIMEOUT = 5000;
long SEARCH_TIMEOUT = 5000;
String TARGET_PACKAGE = "対象App のパッケージ名";

UiDevice mDevice = UiDevice.getInstance(…);
//(1)
mDevice.waitForWindowUpdate(null, UPDATE_TIMEOUT);
mDevice.waitForIdle();
//(2)
mDevice.wait(Unitl.findObjects(By.clickable(true).
    pkg(TARGET_PACKAGE)), SEARCH_TIMEOUT);
```

図 6 UI Automator のテストシナリオに記述する待機処理
Fig. 6 Code of waiting in test scenario of UI Automator.

ドは，現在表示されている画面に変化があるまで待機する処理である．その後，図 6 の (2) にて，クリック可能かつ観測対象の App に含まれる View が発見されるまで待機する．この (2) の待機処理によって，API トレース機構のオーバヘッドによって GUI が更新される前に GUI 構成の取得処理が行われてしまう現象を回避できる．

画面の状態識別

画面の状態識別は，3.2.2 項（2）の方法を用いて生成したハッシュ値を利用する．またこのとき，初めて出現したハッシュ値だった場合は，現在の画面のスクリーンショットを撮影し，ハッシュ値を名前として端末の内部ストレージに保存する．これにより App 実行中にユーザに提示されていた情報を解析時に復元できる．また，テストシナリオでは，1 回の GUI 操作ごとに画面状態の識別を行う．そのため，時間経過による画面遷移など，View の操作以外に起因した状態変化は対象にしない．

View オブジェクトの操作

画面の状態識別により特定された未操作の View オブジェクトに対してクリック操作やスクロール操作を実行する．このとき，クリック対象の条件には，以下の条件にすべて一致するものを対象とする．

- 有効である（enable=true）．
- パッケージが観測対象の App のパッケージ名と一致する．
- クリック可能である（clickable=true）．
- チェックボックスではない（checkable=false）．
- テキストボックスではない（clazz not contain “EditText”）．

チェックボックスのクリックやテキストボックスへの入力は，その操作自体で画面状態が変化してしまうため，クリック可能な View をすべてクリックした後に一括で行う．これにより画面状態数が爆発的に増加するのを防ぐ．

終了条件の判定

UI Automator による GUI 操作は以下にあげる 3 条件のうち 1 つ以上満たされたときに終了する．

- SIGINT を受信する．
- 指定したタイムアウト時間が経過する．
- 一定時間以上待機しても，クリック可能な View が出現しない．

3.2.4 ログ出力機能

UI Automator は Android 端末上で実行されるため，端末外にログを送信する機能を追加実装する必要がある．この機能を実現するために，GUI 操作の自動化開始直後にテストシナリオ内でログ送信用のサーバソケットを開き，接続したプロセスが GUI 操作に関するログを受信できるようにした．このソケットに対して，API トレース機構がクライアントとして接続することで，GUI 操作ログの送受信を行う．

1 回の GUI 操作によって送信されるログの内容を**図 7** に示す．ログには，GUI 操作に関する情報と操作を行ったときの画面の状態が記録されている．GUI 操作に関する情報は，クリック，スクロール，待機などの操作の種類や View オブジェクトのクラス名や操作対象の View オブジェクトの表示エリアを示すバウンディングボックスなどが記録される．タイムスタンプは，操作が行われたタイミングで取得したものを使用し，ログの送受信にかかる遅延は後から修正できるようにした．

3.3 API 呼出しの引き金となった GUI 操作の抽出

GUI 操作のログとスクリーンショットを API トレースログと合わせて利用することで，グローバル ID 送信時の以下のコンテキストを確認することができる．

- グローバル ID 送信の引き金となった GUI 操作の内容やその操作に至るまでの経緯．
- グローバル ID 送信の直前にユーザに示されていたテ

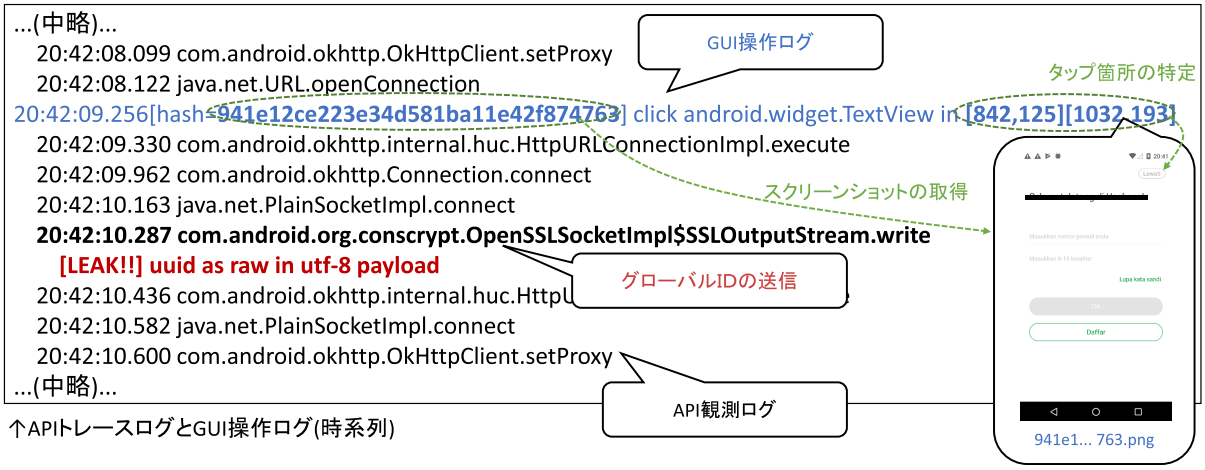

図 8 API 呼出しの引き金となった GUI 操作の抽出
Fig. 8 Extraction of GUI operation triggering API call.

```
{
  "action": "click",                  #操作の種類
  "bounds": "[196,648][883,792]",  #View の表示エリア
  "current_state": {                #操作時の画面の状態
    "checkables": "(0/2)",          #チェック操作の進捗
    "clickables": "(0/2)",          #クリック操作の進捗
    "depth": "1",                   #画面到達時の深さ
    "editables": "(noedit/0)",      #テキスト編集の進捗
    #画面のハッシュ値
    "hash": "996ec3f7d9276838495835473c904c7e",
    "scrollables": "(0/0)"          #スクロール操作の進捗
  },
  "number": "0",                    #通し番号
  "next_state":{                    #操作後の画面の状態
     (略: 項目は,current_state と同じ)
  },
  #操作したView のクラス名
  "reference": "android.widget.Button",
  "tag": "OPERATION",    #タグ (API ログ orGUI 操作ログ)
  "time": "16:03:55.712"            #タイムスタンプ
},
```

図 7 GUI 操作のログ
Fig. 7 Log of GUI operation.

キストや画像などの情報.

ログとスクリーンショットを利用してグローバル ID 送信時のコンテキストを確認する手順を**図 8** を用いて説明する．図 8 のログは，提案手法によって API トレースログと GUI 操作のログを結合したものである．図 8 のログでは，GUI 操作（青字），API 観測ログ（黒字），グローバル ID 送信を示すメッセージ（赤字）が時系列で並んでいる．グローバル ID 送信の検出は，先行研究 [1] で述べた方法で API トレースログを解析することで実現する．

図 8 では，20:42:10.287 の時点で呼び出された write メソッドによって，UUID が App によって送信されている．また，そのログに一番近い上位の GUI 操作ログを確認することで，図 8 で示すスクリーンショットの画面中にある座標 $(937, 160)$ をクリックしたことが，UUID の送信の発端となったことが確認できる．また，スクリーンショットを確認することで，図 8 の場合は画面右上にある「Lewati」と記述されたボタンのタップが引き金となっていたことが分かる．この解析のように，グローバル ID 送信以前に行われた GUI 操作ログに着目することで，GUI 操作箇所やそこに表示されていたコンテンツを後から確認することができる．

3.4 制約事項

本手法は，3.2.2 項で述べたとおり，GUI 構成の解析に View 階層を用いる．そのため，GUI 構成が View 階層に現れないものについては，GUI 構成の解析および GUI 操作が正常に行えない可能性がある．

また，本手法では，API 呼出しと GUI 操作の紐づけにタイムスタンプを用いる．そのため GUI 操作以外のイベントが起因した API 呼出しの紐づけを誤る可能性がある．本論文では，GUI 操作以外のイベントによる影響を抑えるため，端末の固定，端末のスリープ機能の解除，解析後 App のアンインストールにより，Android システムや他の App からのイベントが最小限になるようにした．

4. 評価

4.1 概要

提案手法を評価するため，マーケットに存在する App を対象に提案手法を適用し，グローバル ID 送信を動的解析する際に必要な GUI 操作の抽出が行えるか確認した．検体は，マーケットにおける一般的な母集団を対象とするため，Google Play に存在する新着無料 App を 2018 年 10 月 26 日から 2019 年 1 月 16 日にかけて毎週各カテゴリから収集した 6,242 検体の APK ファイルを対象にした．6,242 検体は，APK ファイルの重複を除いた結果であり，カテゴリごとに収集された数は**図 9** に示すとおりである．グローバル ID は，利用者による変更ができない非推奨なグロー

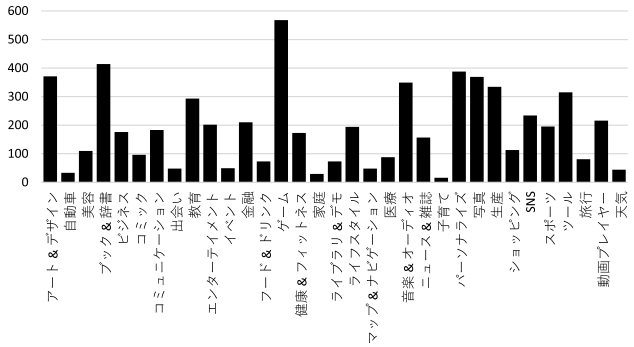

図 9 6,242 検体が属する App カテゴリの分布

Fig. 9 Distribution of app categories to which 6,242 apps belong.

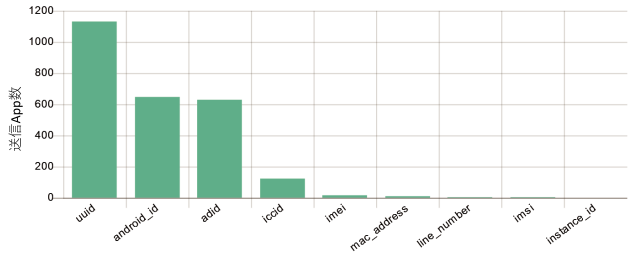

図 10 グローバル ID の送信を送信した App 数

Fig. 10 Count of apps that sent gloabl ID.

バル ID である Android ID，IMEI，IMSI，ICCID，MAC アドレス[*1]，電話番号と，利用者による変更が可能な推奨されたグローバル ID である UUID，Ad ID [13]，Instance ID [14] の合計 9 種類のグローバル ID を対象にした．検体の実行時間は，6,242 検体を実行できる現実的な時間として 5 分間に設定した．抽出した GUI 操作は集約し，グローバル ID 送信までに要した画面遷移の深さ，GUI 操作ログに記録された座標，操作座標にあったテキストなどの情報の 3 点について傾向をまとめた．実行端末には，Android 6.0.1 を搭載した Nexus5（ディスプレイ解像度は 1,080 × 1,920）を使用した．

4.2 グローバル ID の送信 App 数

API トレース機構による動的解析によって，明らかになったグローバル ID ごとの送信 App 数を**図 10** に示す．図 10 より，6,242 検体の App のうち，UUID は 1,131 検体で約 18.12%，Ad ID は 629 検体で約 10.08%の App でグローバル ID の送信が行われていた．また，非推奨なグローバル ID である Android ID は 647 検体で約 10.38%の App で送信されていた．

4.3 グローバル ID の送信に至るまでの画面遷移の深さ

グローバル ID 送信までの画面遷移の深さを**図 11** に示す．画面遷移の深さは，起動直後を 0，起動後最初に出現した画面を 1 として，遷移元の画面遷移の深さから 1 を足した値としてカウントした．図 11 より，GUI 遷移の深さが 0 のときの送信回数は 4,944 回，1 以上のときの送信は 4,737 回であり，約 51.07%のグローバル ID 送信は起動直後に行われていることが分かった．このことから，多くのグローバル ID 送信は，GUI 操作を行わなくても動的解析できることが分かる．

一方で，約 48.93%のグローバル ID 送信は，GUI 操作が引き金となっていることが分かった．また，グローバル ID 送信数は，画面状態遷移の深さの増加にともない減少しており，深さが 10 以上になると送信数が横這いであることが分かった．

[*1] Android6.0 以降，プライバシへの配慮のため MAC アドレスは定数化．

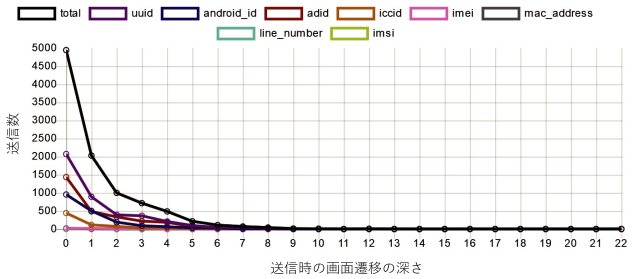

図 11 グローバル ID 送信時の画面遷移の深さ

Fig. 11 Depth of display transition when global ID were sent.

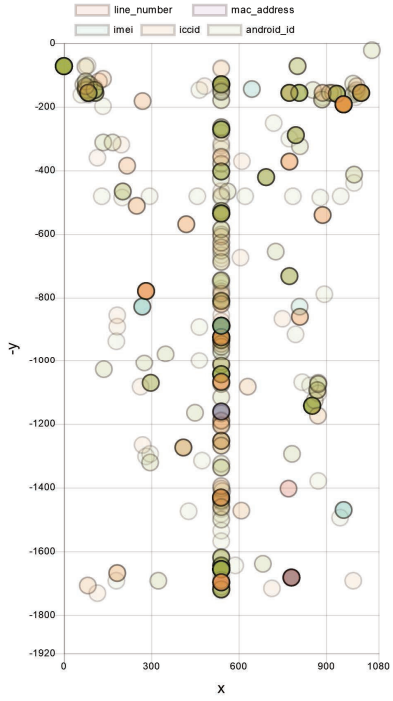

図 12 非推奨なグローバル ID の送信に起因した GUI 操作座標

Fig. 12 GUI operation coordinates caused of unrecommended global ID transmissions.

4.4 グローバル ID 送信に起因した GUI 操作座標

図 11 の非推奨なグローバル ID について，送信の引き金となった GUI 操作の座標を**図 12** に示す．図 12 では，画

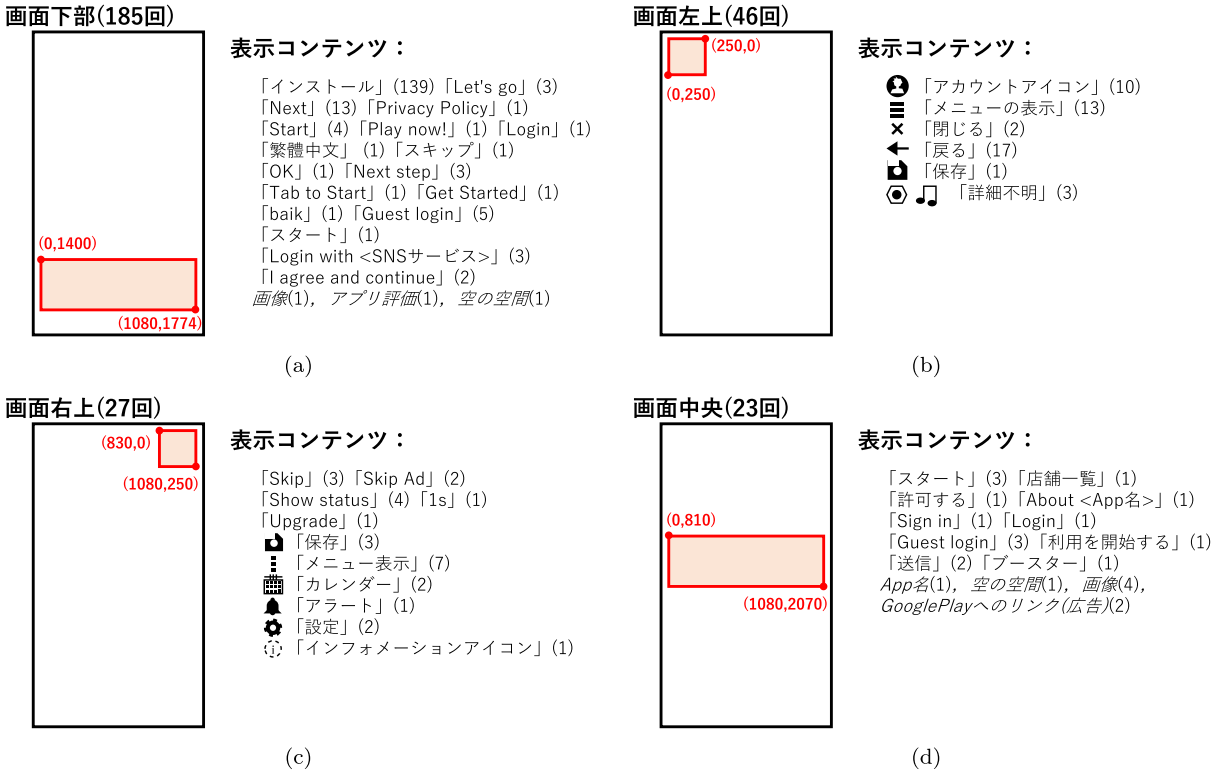

図 13 Android ID の送信に起因した GUI 操作が集中していた箇所と表示コンテンツ
Fig. 13 Contents in the place where GUI operation caused of Android ID transmission were concentrated.

面座標の原点が左上であることを考慮し，Y 軸を負数にして示している．各円は，半透明で描画しており，色の濃い部分は，円が重なっていることを示す．

図 12 より，画面中央である X = 540 に沿って上下に多く分布していることが分かった．特に画面中心や Y 座標が −1650 の位置に多く分布していることが分かった．また，Y 軸上部の左右に多く分布していることが分かった．図 12 で明らかになった GUI 操作箇所の分布を基にタップする場所に優先度を設けることで，効果的な動的解析による挙動の観測が可能になる．

4.5 GUI 操作箇所に存在するコンテンツ

図 12 で Android ID の送信の原因となった GUI 操作について，その領域に表示されていたテキストなどの表示コンテンツについて調査を行った．この調査では，グローバル ID 送信の直前に撮影されたスクリーンショットと GUI 操作座標を確認することで行った．

図 12 のうち，特定の箇所に集中していた 909 回中 281 回分（約 30%）の GUI 操作についてまとめた結果を**図 13** に示す．図 13 では，画面下部，左上，右上，中央の各々における表示コンテンツごとの操作数を括弧内に示す．

図 13 (a) および図 13 (d) で示した画面下部と画面中央にあった 208 回の GUI 操作のうち，「Install」「Start」「Let's Go」「Login」「Next」を含む View に対する GUI 操作は 183 回であり，88%を占めていた．また，図 13 (b) および図 13 (c) で示した画面左上と画面右上にあった 73 回の GUI 操作のうち，62 回はメニューの表示や機能を呼び出すアイコンであり，85%を占めていた．

4.6 考察

本評価では，提案手法を App 6,242 検体に対して適用し，グローバル ID が送信されるタイミングや要した GUI 操作などの抽出を行った．本評価により，多くのグローバル ID 送信は起動直後に行われていることや，画面遷移数に対して観測できるグローバル ID の送信数は減少傾向にあることが明らかとなった．このことから，5 分以上の実行により観測されるグローバル ID 送信数が増加する可能性があるが，その数に大きな変化は生じないと考えられる．また，図 12 と図 13 の結果から，グローバル ID の送信の引き金となる操作箇所や表示コンテンツの分布や傾向が明らかとなった．この結果より，提案手法によって，動的解析の目的である挙動の引き金となる画面遷移や GUI 操作といったこれまで明らかにされなかった情報が得られることが分かった．これまで App 解析者の経験則で行われてきた動的解析中の GUI 操作を，本評価で得られた指標を用いることで効果的に行うことが期待できる．

本評価では，グローバル ID の送信に利用される API の呼出しと GUI 操作との関連を明らかにした．この API に

加えて，ファイル入出力やプロセス間通信に利用されるAPIなどをGUI操作と照合することで，マルウェア解析や脆弱性検査といった他の用途の動的解析であっても提案手法により必要なGUI操作を抽出することが可能であると考えられる．また，他の動的解析技術を利用した調査 [7], [8], [9], [10], [11] も，提案手法と組み合わせて動的解析で得られる結果を照合することで，より効果的な動的解析の実現が期待できる．さらに，図9のカテゴリごとにGUI操作の傾向を明らかにし，フィードバックさせることで，カテゴリに合わせて最適なGUI操作を実現できる可能性がある．

一方で，3.3節で述べたGUI構成がView階層に現れない検体が影響し，本論文で述べた結果は解析された結果よりも多い可能性がある．また，GUI操作以外のイベントに起因したグローバルIDの送信が余分にカウントされている可能性もある．今後，API呼出しとGUI操作をより正確に紐づけられるようにすることで，厳密な結果を得られると考えられる．

5. 関連研究

Appの動的解析時のGUI操作の自動化を行っているものにMobileSandbox [15], Dynodroid [16], DroidBot [17]がある．MobileSandbox [15] やDynodroid [16] は，MonkeyRunner [18] を利用してマルウェアの動的解析やテスト中のGUI操作を自動化した．また，EHBDroid [19] では，GUI操作によって実行されるイベントハンドラを直接呼び出すことでGUI操作をシミュレーションした．DroidBot [17] では，Accessability Service [20] を利用したGUI操作自動化を行い，画面の状態遷移図を記録する手法を実現した．これらの研究によって，動的解析時のコードカバレージの向上やテスト中のエラー検出率が向上した．本論文では，GUI操作の自動化によりGUI操作と動的解析の結果を照合できる仕組みを実現することで，App自身の動的振舞いとGUI操作の関係について明らかにした．これにより，動的解析時に必要なGUI操作に関する傾向を解析可能にした．

6. おわりに

本論文では，Appによる動的振舞いとGUI操作との関連を抽出する仕組みを提案した．提案手法では，GUI操作自動化によって機械的に記録されたログをAPIトレースログと照合することで，API呼出しの引き金となったGUI操作を特定した．提案手法の評価では，グローバルID送信に要する画面遷移の深さや，グローバルIDの送信を引き起こすGUI操作箇所と表示コンテンツの傾向を明らかにできることを示した．今後の課題には，今回の評価で得られた結果をGUI操作自動化の処理にフィードバックさせることで，より効果的なグローバルID送信の解析を行う手法の検討がある．

参考文献

[1] 福田泰平, 明田修平, 瀧本栄二, 齋藤彰一, 毛利公一：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，情報処理学会論文誌，Vol.59, No.9, pp.1678–1688 (2018).

[2] AppBrain: Number of Android apps on Google Play, AppBrain (online), available from ⟨https://www.appbrain.com/stats/number-of-android-apps⟩ (accessed 2018-01-19).

[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] 竹森敬祐, 松井利樹, 磯原隆将, 川端秀明, 渡辺 龍, 窪田歩：スマートフォンアプリ向け独自IDの生成・管理，研究報告コンピュータセキュリティ（CSEC），Vol.2012-CSEC-59, No.8, pp.1–8 (2012).

[7] 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).

[8] 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).

[9] You, W., Liang, B., Shi, W., Wang, P. and Zhang, X.: TaintMan: An ART-Compatible Dynamic Taint Analysis Framework on Unmodified and Non-Rooted Android Devices, *IEEE Trans. Dependable and Secure Computing*, No.99, p.1 (2017).

[10] 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).

[11] Bhatia, T. and Kaushal, R.: Malware detection in android based on dynamic analysis, *2017 International Conference on Cyber Security And Protection of Digital Services* (*Cyber Security*), pp.1–6 (2017).

[12] Google: UI Automator — Android Developers, Android Developers (online), available from ⟨https://developer.android.com/training/testing/ui-automator⟩ (accessed 2018-05-07).

[13] 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）．

[14] 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).

[15] Spreitzenbarth, M., Freiling, F., Echtler, F., Schreck, T.

and Hoffmann, J.: Mobile-sandbox: Having a Deeper Look into Android Applications, *Proc. 28th Annual ACM Symposium on Applied Computing*, *SAC '13*, pp.1808–1815, ACM (2013).
[16] Machiry, A., Tahiliani, R. and Naik, M.: Dynodroid: An Input Generation System for Android Apps, *Proc. 2013 9th Joint Meeting on Foundations of Software Engineering*, pp.224–234 (2013).
[17] Li, Y., Yang, Z., Guo, Y. and Chen, X.: DroidBot: A Lightweight UI-guided Test Input Generator for Android, *Proc. 39th International Conference on Software Engineering Companion*, *ICSE-C '17*, pp.23–26, IEEE Press (2017).
[18] Google: monkeyrunner, Android Developers (online), available from ⟨https://developer.android.com/studio/test/monkeyrunner/⟩ (accessed 2019-01-23).
[19] Song, W., Qian, X. and Huang, J.: EHBDroid: Beyond GUI testing for Android applications, *Proc. 2017 32nd IEEE/ACM International Conference on Automated Software Engineering* (*ASE*), pp.27–37 (2017).
[20] Google: AccessibilityService — Android Developers, Android Developers (online), available from ⟨https://developer.android.com/reference/android/accessibilityservice/AccessibilityService.html⟩ (accessed 2019-01-23).

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，2019 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．同年アドソル日進株式会社入社，現在に至る．システムソフトウェアの開発に従事．

鄭 俊俊 （正会員）

2010 年福建師範大学（中国）工学部ソフトウェア学科卒業，2013 年広島大学大学院工学研究科情報工学専攻博士課程前期課程修了，2016 年同大学大学院工学研究科情報工学専攻博士課程後期課程修了，同年同大学大学院工学研究科外国人客員研究員，2018 年立命館大学情報理工学科助教，現在に至る．博士（工学）．ソフトウェア信頼性，性能評価，ソフトウェアフォールトトレラント技術，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，日本信頼性学会，日本オペレーションズ・リサーチ学会，IEEE 各会員．

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．