Vol.2017-DPS-170 No.14
Vol.2017-CSEC-76 No.14
2017/3/2

静的テイント解析と動的解析の組み合わせによるAndroidアプリの利用者情報漏えい検出手法の提案

瓶子 昇吾[1,a)]　小形 真平[2,b)]　岡野 浩三[2,c)]

概要：アプリマーケットへのマルウェアの登録を防ぐことは重要であるが，登録候補となる膨大なアプリを検査するマーケット業者の労力は高い．従来の支援として利用者情報漏えいの観点からマルウェアの判定支援を行う FlowDroid といった静的テイント解析ツールがあるが，漏えいしないものを漏えいすると誤って判定する偽陽性の問題がある．そこで，本研究では，偽陽性を低減して漏えいをより正確に検出できるように，静的テイント解析で得られる漏えいの疑いを動的解析で裏付ける手法を提案する．その核は，漏えいが疑われる部分的な制御フローを拡大して動的解析に必要な UI 操作に紐づける方法である．提案手法の有効性は，静的解析ツールテスト用アプリ群 DroidBench に対する FlowDroid の解析結果に基づく比較により評価される．

キーワード：Android，スマートフォン，静的解析，動的解析，利用者情報

1. はじめに

近年普及しているスマートフォンやタブレットといった端末の OS として Android は高いシェアを占めている．Android の特徴として低い敷居でアプリ開発・公開が可能な点があり，多くの Android アプリが Android アプリマーケットである GooglePlay で公開されている．

しかし Android が普及する一方で Android 端末の利用者情報 [1] を狙ったマルウェアも増加しており，TrendMicro の統計では 2014 年から 2015 年にかけてマルウェアの検体数が倍以上に増加している [2]．

ユーザをマルウェアによる被害から守るためには，マーケットにおけるアプリ登録審査段階でマルウェアを排除することが重要となる．その登録審査では，マルウェアと判定されたアプリの開発者に対する措置は重いものとなるため，誤判定は防がれなければならない．

利用者情報漏えいを検出するための従来技術の一つとして，静的テイント解析ツール FlowDroid[3] がある．FlowDroid では，利用者情報である Source が取得されてから，その利用者情報をアプリ外部に送信しうる操作である Sink までのフローを検出する．FlowDroid の特徴の一つとして，アプリの正確なライフサイクルモデルを考慮して解析することにより，従来の手法よりも利用者情報漏えいの見落としが少ない解析手法を実現していることが挙げられる．しかし，実際には利用者情報漏えいが生じないにも係わらず，漏えいが生じると誤って判定してしまう偽陽性の問題がある．

そこで，本報告では，マーケット業者が膨大なアプリからマルウェアと疑わしいものをより的確に絞り込めるように，利用者情報漏えいの判定における偽陽性の問題改善を主目的として，静的テイント解析ツール FlowDroid[3] と Android アプリ動的解析手法 [4] を組み合わせた利用者情報漏えい検出手法を提案する．

提案手法の有効性を評価するために，静的解析ツールテスト用アプリ群 DroidBench[5] 中の一部のアプリに対して，FlowDroid と提案手法をそれぞれ適用して，その解析結果を比較した．結果として FlowDroid の解析結果で生じていた偽陽性判定を提案手法により陰性と裏付けることができたため，提案手法が有効である見込みを得た．一方で，多様なアプリに対して静的解析の結果を裏付けるうえで，提案手法に課題も見受けられたため，その解決策を含めて議論する．

本稿の貢献は以下の通りである．

- FlowDroid の解析結果として表れる利用者情報漏えいが生じる可能性があるフローに基づいて，動的解析の探索・実行ができるよう両手法の橋渡しとなる手法を

[1] 信州大学大学院理工学系研究科
[2] 信州大学学術研究院
[a)] 15tm529k@shinshu-u.ac.jp
[b)] ogata@cs.shinshu-u.ac.jp
[c)] okano@cs.shinshu-u.ac.jp

新規に提案した．
- 提案手法により FlowDroid に比べて偽陽性の問題を改善できる見込みを示した．

2. 諸概念

2.1 利用者情報

総務省がスマートフォン端末における利用者情報の取扱いに関する対応として，スマートフォン プライバシー イニシアティブ [1] を公表しており，そこでは利用者の識別に係る情報，第三者の情報，通信サービス上の行動履歴や利用者の状態に関する情報を利用者情報と定義している．

本研究では，FlowDroid による静的テイント解析における Source となる利用者情報を対象に漏えいの検出を行う．この Source には，電話帳内の情報，端末所有者の個人情報，端末の固有 ID，各種履歴や位置情報などといった利用者情報が含まれる．

2.2 Android マルウェア

利用者の同意なしに行われる個人情報の外部送信やプライバシーの侵害，それらによって利用者が被害を受けるリスクはマルウェアによって顕在化する [1]．当該マルウェアは，たとえば利用者からの利用者情報の窃取やそれを用いた金銭の奪取など，端末から利用者情報にアクセスし，外部への送信やそれを引き起こす可能性があるプロセスへ渡す操作を行う．

本研究では，アプリが利用者情報をアプリ外部に漏えいするかどうかを検出する手法の構築を趣旨とする．そのため，対象となる Android マルウェアは利用者情報漏えいを行うものに限定される．

2.3 Android アプリ解析手法

マルウェア的な振る舞いの検出に用いられるアプリの解析手法には静的解析と動的解析がある．静的解析はアプリのマニフェストやコード，レイアウトファイルなどの構成要素からアプリに含まれるマルウェア的な振る舞いを検出する．動的解析はアプリ実行中の動作を監視し，観測された動作からマルウェア的な振る舞いを検出する．静的解析は動的解析に比べてアプリ全体を漏れなく解析しやすいが，動的束縛などによる誤判定のリスクがある．動的解析はアプリを実行した際に観測された動作に基づくため，観測した範囲では誤判定となることはないが，アプリ全体を漏れなく解析することが困難である．これらの特徴から静的解析と動的解析は互いに補完しあう性質を持つ．

本研究では，漏れが少なく偽陽性の低い利用者情報漏えい検出手法の実現をねらい，既存の静的テイント解析ツール FlowDroid[3] を用いて得られた結果を Android アプリケーション動的解析支援手法 [4] を用いて裏付けるために静的解析手法と動的解析手法を組み合わせる．

```
$r4 = virualinvoke $r3. < android.TelephonyManager:
                          java.String getDeviceId() >
$r0. < de.ecspride.Button2: java.lang.String imei > = $r4
$r4 = $r0. < de.ecspride.Button2: java.lang.String imei >
$r5 = virtualinvoke $r5. < java.lang.StringBuilder:
     java.lang.StringBuilder append(java.lang.String) > ($r4)
$r4 = virtualinvoke $r5. < java.lang.StringBuilder:
                    java.lang.String toString() > ()
staticinvoke < android.util.Log:
       int i(java.lang.String,java.lang.String) > ("TAG", $r4)
```

図 1 FlowDroid が出力する利用者情報漏えいフローの例

2.3.1 静的テイント解析ツール FlowDroid

FlowDroid[3] は Android アプリの静的テイント解析ツールである．FlowDroid における静的テイント解析では，まず解析対象のアプリの apk ファイルと Android のライブラリである android.jar を入力に受けてアプリのライフサイクルを構築する．そして，利用者情報を取得する Source をテイントの起点として，漏えいを行う可能性がある Sink にテイントされたデータが渡されるまでを追跡し，利用者情報漏えいの可能性としてのフローを出力する．

表 1 は，端末の電話番号を取得して漏えいするアプリを FlowDroid で解析したフローの一例である．これは，TelephoneManager クラスの getDeviceId() により変数 imei に与えられた端末番号が android.util.Log に渡されて漏えいするまでのフローを示している．

2.3.2 Android アプリケーション動的解析支援手法

既存の動的解析支援手法 [4] は Android アプリの起動時にアプリの画面構造と画面遷移のマッピングを行い，構造の中に含まれる UI 要素の View の探索と操作を繰り返すことでアプリの動作を誘発する．これによりアプリ内の UI への網羅的な操作とそれに伴う動作の解析を行う．

本手法では各画面をノード，画面間の遷移が生じるきっかけをユーザイベント，View への操作をエッジとして画面状態遷移グラフを構築する．そして，各画面の持つ UI 要素である View を探索してグラフを巡回し，未探索の View に対する操作を行うことで解析対象のアプリへ効率的な操作を行う．

本研究では動的解析ツールとして既存研究のシステムを実装し，更に FlowDroid から得られる情報漏えいを誘発する UI 要素の探索と操作を行えるように機能を追加する．

3. 提案手法

3.1 概要

提案手法では Android アプリの静的テイント解析 FlowDroid[3] の解析結果を基に利用者情報漏えいを発生させる View を抽出し，これを利用して既存手法 [4] をベースとして作成した動的解析ツールで情報漏えいを引き起こすユー

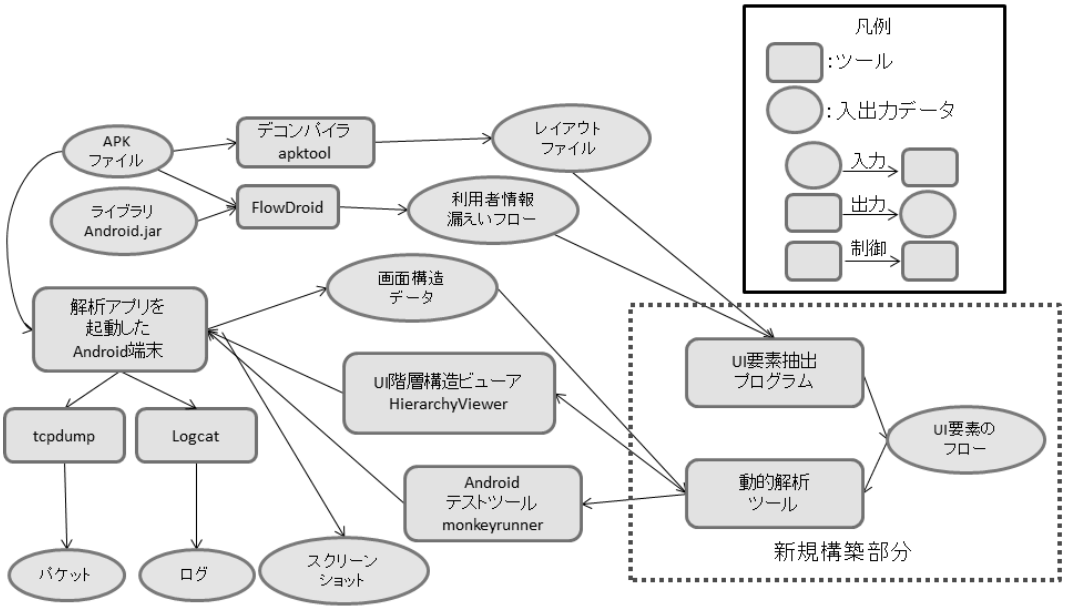

図 2 システム概要図

ザ入力を再現する．そして，このように動的解析を行う間のアプリの通信や画面から人手で静的解析の結果を裏付ける．提案手法で作成したシステムの全体図を**図 2** に示す．点線で囲んだ部分が提案手法で実装した機能である．

実装方法として，まず FlowDroid に apk ファイルと Android のライブラリである android.jar を与えて，利用者情報漏えいフローを検出する．加えて，apktool[10] により同 apk ファイルをデコンパイルして，アプリのレイアウトファイルを得る．そして，本研究で実現する UI 要素抽出プログラムにこれらの利用者情報漏えいフローとレイアウトファイルを与え，利用者情報漏えいフローに沿って UI を自動操作するための UI 要素の操作フローを抽出する．この UI 要素は Android では View と呼ばれるものであるが，この抽出方法は 3.2 節で詳述する．

その後，本研究で実現する動的解析ツールに UI 要素のフローを与え，Android 端末（エミュレータ）上で起動された解析対象のアプリの UI を自動操作する動的解析を行う．本動的解析では，実行中のアプリの UI 階層構造を取得するために Android 公式ツールの HierarchyViewer[8] を用い，取得された画面内の View を利用して Android 公式ツールの monkeyrunner[9] により自動操作を行う．本動的解析における動的解析ツールについては 3.3 節で詳述し，抽出した View を用いた動的解析の実行は 3.4 節で詳述する．

このとき，Android 端末（エミュレータ）では，tcpdump[11] によるパケットキャプチャと Logcat[12] による端末内部のシステムコールログ記録および monkeyrunner 主導による操作画面のスクリーンショット記録が行われており，これらが利用者情報漏えい検出のための参考情報として利用される．これらの情報を用いた利用者情報漏えい検出については，3.5 節で詳述する．

3.2 利用者情報漏えいフローからの View 抽出

図 3 に View 抽出における手順の概要図を示す．FlowDroid は解析対象の apk ファイルに含まれるバイトコードとアプリの構成ファイルに対して静的テイント解析を行い，漏えいの可能性がある利用者情報のフローを検出して出力を行う．フローは利用者情報やそれにアクセスするメソッドといった Source を起点にし，他インスタンスへの受け渡しやインターネット，Short Message Service(以下 SMS)，電話といった外部へ利用者情報を漏えいさせる可能性がある Sink を終点としている．また，フロー内の onClickListener のような View の操作時に動作するコールバック関数や View を配置するメソッドなどは，$r0，$r1，といった識別子で管理される．そこでは，フロー内部の定数に基づく識別子の値 (例：0x7f070000) を取得する．

UI 上に表示されるボタンなどの要素と内部プログラムとの紐付けは，アプリのレイアウトファイル (図 3 中の R_id ファイル) に記載される ID 番号を介して行われており，ここで取得した変数などの値はその ID 番号を指す可能性がある．そこで，レイアウトファイルの ID 番号と取得した値とを比較して一致したものを利用者情報漏えいを誘発する UI 要素とし，これをもつ UI（Android の View）を特定する．そして，Source から Sink までのフロー中で特定できる View を順にリスト化する．

3.3 動的解析ツール

Android アプリやセキュリティについて知識がある人間がつぎの観点からマルウェアを判定しやすい解析結果の出力をねらう．アプリに対する正確な操作と監視や，マーケットで用いる場合の膨大なアプリを処理する必要のために提案手法では人の手ではなく既存研究の手法を用いた

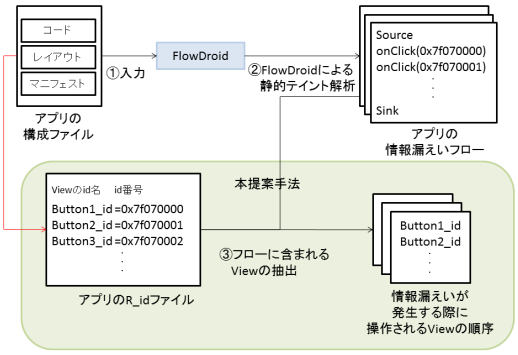

図 3 FlowDroid による解析結果からの利用者情報漏えいに関わる View 抽出

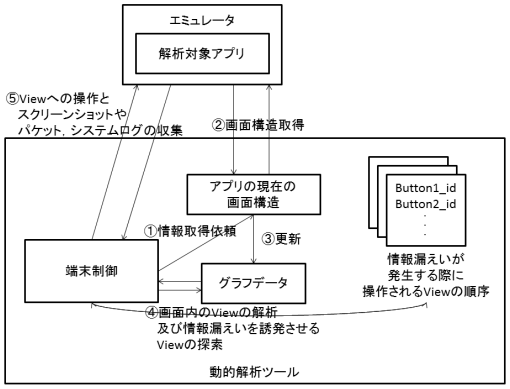

図 4 動的解析ツール概要図

ツールによる自動化を行う．その観点としては，1）プライバシーやセキュリティに係わる利用者情報をアプリ外部へ送信しているかどうか，2）ユーザに対して利用者情報の利用に関する通知が行われているかどうか，3）アプリの目的から利用者情報を扱う機能が不正なものではないかどうかを挙げる．そのため，本節で提案する動的解析ツールは，最終的にはアプリ操作中の通信パケットやシステムコール，スクリーンショットといったログを記録するよう構成される．これらのログの用法については 3.5 節にて詳述する．

動的解析ツールの概要図を**図 4** に示す．図 4 の①，②，③の操作から解析対象の画面の View 構造を取得する．④で取得した構造から未操作の UI 要素の View と FlowDroid から得た利用者情報漏えいを誘発する View フロー内の View を探索し，⑤で View フロー内の要素を順次操作することによって行う．アプリの実行には事前にダミーデータとして利用者情報を入力したエミュレータを用い，Android のツールキットに含まれているデバッグツール HierarchyViewer と monkeyrunner を用いてアプリ実行時の画面構造取得と UI の操作を行い，その操作中のパケット通信・エミュレータのシステムコールログ・操作毎のスクリーンショットの記録を行い，解析を行う．

3.4 抽出した View を用いた動的解析

本節では，3.2 節で FlowDroid の解析結果に基づいて得た View のリストと，3.3 節の動的解析ツールとを組み合わせた解析手順を説明する．アプリの UI から取得した View の ID 名と FlowDroid の解析結果から得られたフローの先頭 View の ID が一致した場合，一致したフローに含まれる View への探索と操作をフローの終端まで繰り返し，利用者情報漏えいの誘発を試みる．

フロー終了まで操作を行って画面遷移が発生した場合，フローの Source で取得された利用者情報が画面間で受け渡される可能性がある．そこで，遷移先の画面に受け渡された利用者情報を用いて遷移先の画面での利用者情報漏えいが行われる可能性を考慮して，遷移した画面での View から開始されるフローが存在する場合はそのフローの View に対する操作を行う．

フロー実行途中に次に操作されるべき UI 要素が発見できなかった場合，あるいは探索によってフローの起点部の UI が発見できなかった場合はそのフローは完遂不可能であったとして中断される．完遂不可能であったフローは記録され，操作途中で中断されたものは中断されるまでのログを残す．

3.5 利用者情報漏えい検出

本節では，利用者情報漏えい検出の視点から，提案手法によって得られたログの用法を説明する．大別して，通信パケットとシステムコールは，利用者情報をアプリ外部へ送信しているかどうか見極めるための参考情報となる．一方で，スクリーンショットは，ユーザへ適切な利用者情報の利用を通知しているかどうかや，アプリの目的から利用者情報を扱う機能が不正かどうかを見極めるための参考情報となる．

スクリーンショットはアプリに対する操作毎に記録する．通信パケットは動的解析実行中に ADB[13] で tcpdump を用いて取得し，パケットキャプチャソフトを用いて通信内容に含まれる利用者情報を探索することを想定する．システムコールは LogCat[12] を用いて取得し，Source となる利用者情報取得操作や Sink となる操作とその引数から利用者情報が取得されて，そしてそれが実際に漏えいしているかの確認に用いる．

そして，それぞれの記録が行われた時間から，どの操作を行った後に利用者情報の取得が行われ Sink に渡されたかを特定し，SMS などの機能の利用であれば動的解析に用いた端末の記録と比較して検証を行う．更に，その際にアプリから利用者情報に関するどのような通知があったかを確認して，アプリがマルウェアであるかどうかの判定に繋げる．**図 5**，**図 6**，**図 7** はそれぞれスクリーンショット，パケット，システムコールの例である．図 5 で記録された画像とその記録時間から動的解析の操作を行った

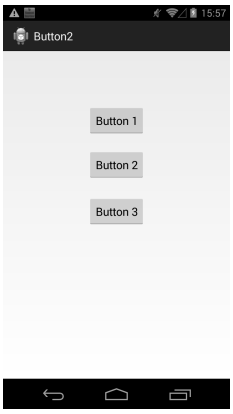

図 5 スクリーンショット

図 6 パケットキャプチャ

```
02-08 11:39:37.638 I/TAG     ( 2452): Button3: 000000000000000
02-08 11:39:43.548 W/AudioTrack( 1479): AUDIO_OUTPUT_FLAG_FAST denied by client
02-08 11:39:43.559 D/MmsService( 1705): getAutoPersisting
02-08 11:39:43.804 I/TAG     ( 2452): sendIMEI: 000000000000000
```

図 7 システムコール

時間とアプリからの通知を確認する．図 6 は実行中に取得したパケットを WireShark[14] で開いた画像である．パケットの送信時間や通信のストリームに含まれる内容から，利用者情報送信の有無と送信が行われた時間を確認する．図 6 の例では送信されたパケットのストリームに端末内のメールアドレスのダミーデータ“leak” が含まれていたことからメールアドレスの利用者情報の漏えいを示している．図 7 は実行中のアプリが出力したメインのログバッファの一部である．ログに含まれているテキストやタグから，利用者情報や Source や Sink となる操作の確認を行う．例では端末内の IMEI(ログ内のテキストに含まれている「000000000000000」) がログに出力されていることと，MmsService がタグのログからメッセージサービス機能が呼び出されていることがわかる．

4. 評価

提案手法の有効性評価のため，下記の Research Questions (RQs) に答える．本来であれば，解析結果を業務者に示して有効性を評価することが，本研究の目的に対する直接的な評価となるが，それ以前に解析性能が低ければ意味がない．そこで，本報告では，FlowDroid の解析結果と提案手法の解析結果を比較することによる評価とその結果を示す．なお，本評価では提案手法を支援する解析実行のための試作ツールを用いた．

RQ1 提案手法は FlowDroid による解析結果の偽陽性を低減できるか

RQ2 提案手法は利用者情報漏えいの検出精度を高めることができるか

RQ1，RQ2 に答えるためにそれぞれ実験 1，実験 2 を実施した．これらの実験の詳細は次節から述べる．

4.1 適用するアプリケーション

静的解析ツールのベンチマークアプリ群である DroidBench[5] を試作ツールに適用して実験を行った．実験 1 では RQ1 の回答のために，FlowDroid が偽陽性を検出した 4 アプリに提案手法を適用した．実験 1 の結果については 4.2 節で説明する．また，実験 2 では RQ2 の回答のために，FlowDroid が利用者情報漏えいに関して偽陽性を含めて陽性と判定した 25 アプリに提案手法を適用した．実験結果については 4.3 節で説明する．

4.2 実験 1：提案手法による偽陽性の低減に関する評価

FlowDroid が偽陽性を検出した 4 アプリ (ArrayAccess1.apk，ArrayAccess2.apk，ListAccess1.apk，Button2.apk) を提案手法に適用する．

結果として，全アプリの実行中のパケット送信は無かった．そして，ArrayAccess1.apk，ArrayAccess2.apk，ListAccess1.apk は UI 要素が存在しなかったが，アプリ起動時のシステムコールのログから，Source となる端末番号，利用者情報の取得は確認できた．そして，Sink として出力される情報には利用者情報が含まれていなかったことがそれぞれ確認された．

Button2.apk については，まず概要を述べた後．詳細を述べる．Button2.apk は UI 要素を持つアプリであったため，FlowDroid は UI 要素を含んだ複数の利用者情報漏えいフローを生成した．試作ツールが FlowDroid の出力したフロー毎に UI 要素の View 抽出と操作フローの生成を行い，その操作フローに基づき動的解析を実行した．その結果，FlowDroid が陽性と判定したフローは動的解析でも陽性と判定でき裏付けることができた．また，FlowDroid が偽陽性と判定したフローは動的解析では陰性と判定でき偽陽性を低減することに成功した．次節から UI の存在する Button2.apk に関する詳細を述べる．

4.2.1 Button2.apk の構成

Button2.apk は図 5 に示されるように 3 つのボタンを持つ一つの画面によって構成されるアプリである．3 つのボタンの操作時に行われるアプリの内部動作を**表 1** に示す．Button3 が変数 imei に端末番号を取得させる Source となる動作を引き起こすボタンとなる．また，imei 内の情報を引数として利用者情報を漏えいさせる可能性がある Sink を引き起こすボタンは Button1，Button3 が該当する．

4.2.2 Button2.apk に対する FlowDroid の解析結果

Button2.apk の利用者情報の漏えいフローとして FlowDroid は 3 つの経路を持つフローを出力した．各フローが

表 1　各ボタン操作時の動作

種類	説明
Button1	変数 imei 内の情報を SMS のテキストで送信し，ログで出力した後 imei に null を与える
Button2	imei に null を与え，imei の値をログに出力する
Button3	端末番号を取得し変数 imei に与え，ログに imei の値をログに出力する

表 2　FlowDroid の出力する利用者情報漏えいフロー

漏えい有りと検出されたフロー	Sink	実際の動作
Button3	ログ出力	漏えい有り
Button3 → Button1	SMS 送信	漏えい有り
Button3 → Button1	ログ出力	漏えい有り
Button3 → Button2 → Button1	SMS 送信	漏えい無し

※ここでは，各フローの最後の ButtonN を Sink を実行するきっかけと見ており，その Sink からの漏えいの有無を示している．そのため，Button3-¿Button2-¿Button1 では，Button3 のみで漏えい自体は生じるが，これは注目する Sink が引き起こしたものではないために漏えい無しとなっている．

実際に行われる場合に操作される各ボタンとその順序，また実際に漏えいが発生するかそうでないかを人の目で確認した結果を**表 2** に示す．これらのフローの中の Button3 → Button2 → Button1 では，Button3 で変数 imei に取得された端末番号が Button1 の操作により呼び出される Sink によって漏えいすると検出しているフローである．しかし，実際は Button2 の操作時に変数 imei が null の値で上書きされるため利用者情報の漏えいは発生しない．このように，FlowDroid では偽陽性判定が生じている．

4.2.3　FlowDroid の結果から生成される UI 要素の View フローと実行結果

FlowDroid の結果を元に提案手法を用いて操作フローと動的解析の自動化を試みた．まず，FlowDroid が出力した利用者情報の漏えいフローの中から Button1,2,3 の ID を引数とするメソッドを探索し，操作フローを生成した．具体的には，Button1 と Button2 を引数とするメソッドは onClick() メソッドであり，Button3 を引数とするメソッドは clickOnButton3() メソッドであるため，各メソッドがフローに現れた際に対応するボタンが操作されたとみなして操作フローを生成した．そして，操作フローに基づき動的解析を行った．

その結果を**表 3** に示す．先頭に ⋆ がついているフローは FlowDroid が出力したフローと一致しているフローである．onClick() メソッドが Button1 と Button2 どちらも引数に取る可能性があるため，FlowDroid の出力するフローよりも多くの操作フローが生成されているが，FlowDroid で検出した利用者情報漏えいのフローに対応するものは漏れなく生成でき，動的解析によって裏付けを行うことができた．実験 1 においては，FlowDroid の解析結果を動的解析で検証し偽陽性を低減できた．

表 3　FlowDroid の結果から生成される UI 要素の View フローと実行結果

提案手法の出力する利用者情報漏えいフロー	UI フローの操作完遂後の利用者情報漏えいの有無
⋆Button3	ログ出力による漏えい有り
⋆Button3 → Button1	SMS 送信による漏えい有り
⋆Button3 → Button1	ログ出力による漏えい有り
⋆Button3 → Button1 → Button2	漏えい無し
Button3 → Button2	フロー完遂後の漏えいは無し
Button3 → Button1 → Button1	漏えいなし（クラッシュ）
Button3 → Button2 → Button1	漏えいなし（クラッシュ）
Button3 → Button2 → Button2	フロー完遂後の漏えいは無し

表 4　実験 2 による精度評価の結果

	FlowDroid	提案手法
True Positive (TP)	26	12
False Positive (FP)	4	0
True Negative (TN)	0	4
False Negative (FN)	0	14
Precision (p) (TP/(TP+FP))	87%	100%
Recall (r) (TP/(TP+FN))	100%	46%
F-measure (2pr/(p+r))	0.93	0.63

※ TP：利用者情報漏えいをしており陽性として検出された数
FP：利用者情報漏えいをしていないが陽性として検出された数
TN：利用者情報漏えいをしておらず陰性として検出された数
FP：利用者情報漏えいをしているが陰性として検出された数

4.3　実験 2：提案手法による解析精度の評価

FlowDroid が，利用者情報漏えいに関して偽陽性を含めて陽性と判定した 25 アプリを用いて提案手法で解析を行った．25 アプリ中で利用者情報を漏えいしているアプリは 21 個あり，検出されるべきフローは 26 通り存在した．FlowDroid はこれらのアプリから 30 通りのフローを検出，過検出は 4 通りであった．提案手法を用いてこれらのアプリを解析した結果，検出したフローの数は 12 通り，検出漏れは 14 通りで過検出は 0 であった．この結果に基づいて適合率 (Precision)，再現率 (Recall) および F-measure を算出して比較した．結果は**図 4** のとおりである．F-measure の結果から提案手法を用いた解析では偽陰性判定が多く生じてしまい，利用者情報漏えいの解析精度の向上は見られなかった．

漏えいを検出できなかった原因として，主に 3 つが挙げられる．第一に，試作ツールではタッチ入力とスクロールのみを実装していたため，入力したテキストを SMS で送信するような種類の漏えいの検出に要するアプリには対応できていなかった．第二に，マルウェアや，端末の位置情報が更新された際に位置情報を取得して漏えいするようなアプリの操作でなく状態変化に対して漏えいが生じるアプ

リ対応できていなかった．この別の例としては，アプリがバックグラウンドで実行するように切り替わった後に情報を保存して停止する際に漏えいが生じるアプリや，端末のメモリが不足した際に漏えいが生じるアプリが挙げられる．第三に，HTTP 通信による漏えいが検出されないような一部漏えいした情報が取得できていなかったことが挙げられる．

4.4 考察

実験 1 により，FlowDroid で検出した利用者情報漏えいを動的解析で再現し検証することができたため，RQ1 の回答として，実験 1 では提案手法は FlowDroid による解析結果の偽陽性を低減できたといえる．しかし，その過程で生成された UI フローは FlowDroid の利用者情報漏えいフローよりも多く，動的解析実行時間を長引かせる原因となった．フロー増加の原因である各 UI 要素を引数にとり，onClick() メソッドのような各 UI 要素がクリックされたことを受けて動作するメソッドは，基本的に UI 要素毎に異なる動作をさせるために異なるメソッドなどを呼び出す．そのため UI 要素の抽出に UI 要素を引数に取るメソッドだけでなく，そのメソッドが後に呼び出すメソッドも紐付けして UI フローの生成を行うことで，UI フローが必要以上に増加することを抑えられると考えられる．

実験 2 により，試作ツールの機能の不足と解析を行うためのログ取得が不充分であることが判明した．これらを主な理由に，RQ2 の回答として，実験 2 では提案手法は利用者情報漏えいの検出精度を高めることができなかった．しかし，これらの問題の解決により精度を高めることは可能と考えられる．問題の解決策としては，まずタッチ入力とスクロールのみに対応している入力方法が限定的である問題に対しては，テキスト入力などの機能を実装することで改善を図る．つぎに，アプリの状態変化による漏えいが検出できない問題に対しては，たとえば，動的解析を行う範囲をアプリ起動後からの View に対する操作だけに限定せずに，アプリや端末の状態変化を発生させるようエミュレータの設定操作を自動化する機能などを拡充することで改善を図る．また，別の方法として，たとえば FlowDroid の結果から View のみを抽出するのではなく，フロー内に含まれる onSaveInstanceState や LocationListener といった端末やアプリの状態変化に対応するデータやメソッドを抽出して，これを基に対応する情報や状態を解析環境の端末・アプリ上で再現して操作を行うことでも改善を図る．最後に，HTTP 通信による漏えいが検出できない問題に対しては，プロキシやパケットキャプチャの解析環境の見直しを行うことで解決を図る．

また，提案手法で動的解析ツールが用いている HierarchyViewer と monkeyrunner による操作は，解析対象のアプリの UI 要素の位置情報の計算や操作プログラムの生成，実行を一々行っていたため時間がかかっており，長時間操作やアプリのエラー落ちの際にエミュレータとの接続が切れてツールの再起動が必要だった．そこでアプリテスト用ライブラリ Android Testing Support Library のライブラリを用いることで処理時間，更にツール作成のために必要な時間も削減できると考える．Android Testing Support Library は端末のアプリに表示されている UI 要素をまとめて取得したり，UI 要素の検索，詳細なパラメータの取得，UI 要素への操作，画面のスクリーンショットを容易に可能にする．また，実験の際にアプリがクラッシュする事例があり，その際は人力でテストの修正を行っていた．しかし Android Testing Support Library 内の UI Automator の機能 UI Watcher を用いることでエラーダイアログを感知することができる．これを用いればアプリがクラッシュした際もエラーダイアログを閉じ，アプリを再起動させて動的解析を続行する操作の自動化が可能になり，解析の自動化に貢献すると考えられる．

5. 関連研究

静的解析と動的解析の組み合わせの研究に関しては，動的解析がアプリの構造を静的解析を用いて解析し，テストケースの生成を作成するのに用いているものがある [15][16]．動的解析のみでは端末のアプリ実行時の情報しか得られないため，アプリの UI 要素に対する円滑な操作を行うために静的解析でアプリのレイアウトファイルを解析し，操作すべき UI 要素を得て動的解析に用いるというものである．しかし，これらの解析手法に用いられている静的解析はあくまで動的解析で円滑に UI 操作を行うためのものであり，静的解析で得た利用者情報漏えいの解析結果を，動的解析で裏付けるものではない．

6. おわりに

6.1 まとめ

本報告では，静的テイント解析ツール FlowDroid と既存の動的解析手法とを組み合わせた利用者情報漏えい検出手法を提案した．FlowDroid がアプリから検出した利用者情報漏えいのフローから，その利用者情報漏えいが発生する際に操作されるアプリの UI 要素を抽出して，動的解析ツールに入力し操作させることでアプリの利用者情報漏えいを誘発し，静的解析の結果を動的解析で裏付ける．

また，提案手法を元に試作したツールを用いて行った実験により，FlowDroid の検出した情報漏えいフローからの UI 抽出および動的解析による検証に成功した．これにより FlowDroid の偽陽性を低減できる見込みを得た．

6.2 今後の課題

今後の課題としては，動的解析で操作する UI 要素のフロー生成過程の見直し・View 抽出の最適化，動的解析ツー

ルの機能拡充，利用する API の変更・更新，利用者情報漏えい検出のための情報整理の自動化，より広範な種類のアプリに対する解析性能評価が挙げられる．

View フロー生成における余分なフローの生成の原因は View を引数に取るメソッドなどのみから操作されるであろう View を選択していることにある．その View とその View を操作した後に実行されるメソッドやコールバック関数とを互いに紐付けることによって操作されるべき View の候補が絞り込まれ，余分な View フローの生成が抑制できると考えられる．そのためにアプリの静的コード解析による各メソッド間の関係と引数の絞込に関する手法の検討を行う．また，View 以外にもアプリや端末の状態変化によって呼び出されるリスナークラスのメソッドや，アプリ起動時などに読み込まれるバンドルといった要素から，漏えいを誘発する可能性があるアプリや端末の操作の抽出を行う．

提案手法で用いた動的解析の試作ツールではタッチ操作やスクロール操作を実装していたが，それだけでは誘発することができない利用者情報の漏えいがあった．漏えいを誘発するための操作を十分に行うことができるよう，テキスト入力やホームボタンなどの操作の実装や，解析に用いたエミュレータの端末情報に対する操作のための動的解析ツールの機能を拡充をする．

動的解析に用いる API の更新については，Android アプリテスト用ライブラリ Android Testing Support Library[17] を用いることを考える．今回用いた試作ツールでは複数の API の機能を組み合わせたり，API から取得する情報の整理や計算などをツール側で行っていた．このアプリテスト用ライブラリを用いることでツール側で行うべき操作が簡略化され，計算などに用いていた時間や記録すべき情報の軽量化が図れると考えられる．

利用者情報漏えい検出のための情報整理の自動化について，提案手法では動的解析で得られたパケットやシステムコールログ，スクリーンショットなどは逐一人の目によって確認される．Source で取得された利用者情報と Sink で送信された情報の比較や，操作中のどこで Source や Sink に係る操作が行われたのかを時系列ごとの情報の統合・整理などの自動化方法を検討し，利用者情報漏えい検出の効率化を目指す．

より広範な種類のアプリに対する解析性能評価に関しては，今回の提案手法の評価は FlowDroid の偽陽性の問題を解消することを中心的に扱ったが，実際のマルウェア検体に対する適用や，より多くのアプリに対する提案手法の適用を行う．

参考文献

[1] 総務省: スマートフォン プライバシー イニシアティブ, 入手先 ⟨http://www.soumu.go.jp/main_content/000171225.pdf⟩（参照 2017-02-03）.
[2] トレンドマイクロセキュリティブログ: 1000 万個を突破した Android 不正アプリの「これから」, 入手先 ⟨http://blog.trendmicro.co.jp/archives/12960⟩（参照 2017-02-03）.
[3] Arzt, S., et al.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps., ACM Sigplan Notices, 49(6), pp.259-269, (2014).
[4] 塩治 榮太朗, 秋山 満昭, 岩村 誠, 針生 剛男: GUI 構造に基づいた Android アプリケーション動的解析支援の検討, Computer Security Symposium 2012, pp.36-43, (2012)
[5] Secure Software Engineering: DroidBench - Benchmarks, 入手先 ⟨https://blogs.uni-paderborn.de/sse/tools/droidbench/⟩（参照 2017-02-03）.
[6] Android Developers: システム パーミッション, 入手先 ⟨https://developer.android.com/guide/topics/security/permissions.html?hl=ja⟩（参照 2017-02-03）.
[7] Material design guidelines: Patterns - Permissions, 入手先 ⟨https://www.ipsj.or.jp/faq/chosakuken-faq.html⟩(参照 2017-02-04).
[8] Android Developers: Hierarchy Viewer, 入手先 ⟨https://developer.android.com/studio/profile/hierarchy-viewer.html⟩（参照 2017-02-10）.
[9] Android Developers: monkeyrunner, 入手先 ⟨https://developer.android.com/studio/test/monkeyrunner/index.html⟩（参照 2017-02-10）.
[10] Apktool: Apktool - A tool for reverse engineering Android apk files, 入手先 ⟨https://ibotpeaches.github.io/Apktool/⟩（参照 2017-02-10）.
[11] TCPDUMP: TCPDUMP, 入手先 ⟨http://www.tcpdump.org/tcpdump_man.html⟩（参照 2017-02-10）.
[12] Android Developers: logcat コマンドライン ツール, 入手先 ⟨https://developer.android.com/studio/command-line/ logcat.html?hl=ja⟩（参照 2017-02-08）.
[13] Android Developers: Android Debug Bridge, 入手先 ⟨https://developer.android.com/studio/command-line/adb.html?hl=ja⟩（参照 2017-02-08）.
[14] WireShark: WireShark, 入手先 ⟨https://www.wireshark.org/⟩（参照 2017-02-08）.
[15] Carter, P., Mulliner, C., Lindorfer, M., Robertson, W., and Kirda, E.: CuriousDroid: Automated User Interface Interaction for Android Application Analysis Sandboxes., Proceedings of the 20th International Conference on Financial Cryptography and Data Security (FC), (2016).
[16] Tam, K., Khan, S. J., Fattori, A., and Cavallaro, L.: CopperDroid: Automatic Reconstruction of Android Malware Behaviors., NDSS '2015, (2015).
[17] Android Developers: Testing Support Library, 入手先 ⟨https://developer.android.com/topic/libraries/testing-support-library/index.html?hl=ja⟩（参照 2017-02-08）

—