平成 30 年度 東邦大学理学研究科情報科学専攻 修士論文

Androidアプリケーションのコンポーネント間通信による脅威分析の効率化と高度化

学籍番号　6517004

関岡　好史

金岡研究室

目 次

1　はじめに

AndroidOS が搭載された端末（以下 Android 端末と記す）に向けてさまざまなアプリケーション (以下アプリと記す) 開発がされており、AndroidOS とそれら AndroidOS 向けアプリに関するセキュリティは、Android 端末の増加に伴い避けることができない大きな問題となっている。PC 向けの OS である WindowsOS や MacOS、サーバ等でも広範に利用される Linux 等と比較して、AndroidOS に関してはまだ歴史が浅いことや他の用途と比較して多種多様なセンサーを搭載した端末を制御することから、セキュリティに関する検討や適切な対策が十分であるとはいいがたい。一方で、AndroidOS の普及は爆発的に増しており、スマートフォンなどの個人向け端末から家電等への進出などさらなる拡大が予想され、その脅威はますます高くなると言える。

Android アプリが持つ脅威についてはさまざまな視点から調査がされており、いまだ盛んなテーマとなっている。また、脅威に対する対策手法についてもさまざまなアプローチが取られており、爆発的な普及と学術的な研究が並行して行われている現状となっている。Android アプリに対するさまざまな調査の 1 つとして、複数のアプリの協調による情報集約の脅威がある。これまでの調査研究では、アプリ単体がどれほど情報漏えいの脅威を持つかについて強く焦点が当てられてきた。しかし、Android 端末には複数のアプリがインストールされることが可能であり、汎用性の高さから複数アプリがインストールされる形態での利用が通常となっている。

Android アプリにおける複数アプリ間の通信 (Inter-App Communication; IAC) は、Inter-Component Communication(以下 ICC と記す) により実現がされており、この ICC が原因で情報漏えいが起きるケースが存在している。近年の研究ではこの ICC からの情報漏えいに対する分析が盛んになっており、2015 年、Li らにより ICC の分析にテイント解析を採用した IccTA が発表され、その高度な分析能力に注目が集まっている。一方で、IAC の分析は、データセットが大量になる場合に組み合わせ数の爆発により現実的な時間ではデータセット全体の分析が難しく、また、IccTA を利用するに当たって分析結果から得られる情報から脅威であることを理解するには専門的な知識を要する。

そこで、本研究では IAC 分析の効率化、高度化を目的とした手法の提案、及び議論を行う。効率化においては、IAC の分析に関係しないアプリを分析対象から外すための事前処理をインテントに焦点を当てて行うことで組み合わせ数の削減を図り、高度化においては、分析結果から重要な項目のみの抽出、及び優先順位付けについて議論を行う。本論文の構成は以下の通りである。まず第 2 章で関連する前提の知識について紹介し、第 3 章では IccTA を中心とした関連研究について触れる。第 4 章では IccTA、及び IAC 分析に関する問題点を挙げ、第 5 章では効率化の手法を提案し、第 6 章にて効率化の評価を行う。第 7 章では評価の結果、及び高度化の面において考察を行い、第 8 章でまとめる。

2 前提知識

2.1 Android

2.1.1 Android 概要

Android は Google 社により開発されたプラットフォームであり、主にスマートフォンやタブレットなどの携帯端末をターゲットとしている。AndroidOS のカーネルは Linux をベースとしており、ミドルウェアやユーザインタフェースなどで様々な技術が組み合わされ 1 つのパッケージにされて提供されている。OS カーネルやミドルウェアは C または C++言語で記述されている。AndroidOS 上で動くアプリは、Google が開発した Dalvik 仮想マシン上で動作し、JavaSE のサブセットと独自の Android 拡張を合わせた言語で記述される

2.1.2 Android の ICC の仕組み

Android はコンポーネントと呼ばれる構成要素によってアプリが形成されている。そのコンポーネントはマニフェストにより記載され、アプリケーションパッケージ（APK）に同梱される。コンポーネントは以下の 4 つから成る。

- Activity（アクティビティ）
- Broadcast Receiver（ブロードキャストレシーバ）
- Content Provider（コンテントプロバイダ）
- Service（サービス）

Activity はユーザの目に見える部分を構成し、ユーザインタフェースを提供する。BroadcastReceiver は他のコンポーネントやシステムからのイベントメッセージを受信する機能を持つ。イベントメッセージには着信呼び出しやテキストメッセージなどがある。ContentProvider はアプリ間で構造化データを共有するための標準インタフェースとして機能する。そして Service はバックグラウンドでタスクを実行する。これらコンポーネント間での通信を ICC と呼ぶ。 ICC を実現するために Android では ICC メソッドと呼ばれる特定のメソッドを提供しており、ICC メソッドはインテント（Intent）と呼ばれるオブジェクトの引数として渡される。ICC はこのインテントによって実現されており、インテント内においてターゲットコンポーネントなどの指定を行い、AndroidOS に送ることで他コンポーネントとの通信が行われる。また、指定された情報によって AndroidOS の挙動が異なることから、インテントは明示的インテント（Explicit Intent）と暗黙的インテント（Implicit Intent）に大別される。明示的インテントはターゲットコンポーネントを指定し、送ることによってターゲットコンポーネントとの ICC が行われるが、暗黙的インテントではターゲットコンポーネントの指定を動作（Action）により指定する。 暗黙的インテントが呼ばれると、システムはその指定された動作を行うことが可能なインストール済みアプリを探し、ユーザにそのリストを提示、インテントのターゲットを指定させる。例えばインテントに URL が渡された場合、Android はその URL のコンテンツを閲覧可能なブラウザなどのアプリのリストを提供する。暗黙的インテントは、Category、Action、Data の 3 つの組み合わせで提供がされる。一方、暗黙的インテントを受け付けるアプリでは、マニフェストファイルにインテントフィルタ（Intent Filter）を用意し、受け取ることが可能な動作をそこで指定する。

2.1.3 ソースコード上での ICC の実現

本節ではソースコード上で ICC を実現する様子を示す。 まずは明示的インテントを紹介する。Listing 1 はインテントを送る側のアプリケーションにおけるソースコード記載である。Activity が onCreate メソッドで作成されるときに、Button オブジェクトを作成し、ボタンが押されると onClick メソッドが呼ばれるように設定した。onClick メソッドでは Intent クラスのインスタンスを生成し、ターゲットのパッケージ名（ここでは ”jp.ac.toho u.sci.is.klab.intent ”） とそこでのクラス名（ここでは ”jp.ac.toho u.sci.is.klab.intent.testActivity ”) を明示する。そして putExtra メソッドで文字列 ”Kanaoka Lab ” をインテントに格納する。最後にインテントは startActivity メソッドの引数として渡される。

Listing 1: 明示的インテントの例（送信側）

```
1  public class intentActivity extends Activity implements OnClickListener{
2
3          @Override
4          public void onCreate(Bundle savedInstanceState) {
5                  super.onCreate(savedInstanceState);
6                  setContentView(R.layout.main);
7
8                  Button btn = (Button)findViewById(R.id.Button01);
9                  btn.setOnClickListener(this);
10         }
11
12         public void onClick(View v) {
13                 Intent intent=new Intent();
14                 intent.setClassName("jp.ac.toho_u.sci.is.klab.intent","jp.ac.
                       toho_u.sci.is.klab.intent.testActivity");
15                  intent.putExtra("jp.ac.toho_u.sci.is.klab.intent.testString", "
                        Kanaoka_Lab.");
16
17                 startActivity(intent);
18         }
19 }
```

受信側では、Listing 2 にあるように、getIntent メソッドによりインテントを受信し、putExtra メソッドで格納された文字列を getStringExtra メソッドで取り出す。

Listing 2: 明示的インテントの例（受信側）

```
1  public class testActivity extends Activity {
2          @Override
3          public void onCreate(Bundle savedInstanceState) {
4                  super.onCreate(savedInstanceState);
5                  setContentView(R.layout.main2);
6
7                  Intent intent = getIntent();
8                  if(intent != null){
9                          String tmpStr = intent.getStringExtra("jp.ac.toho_u.sci.
                               is.klab.intent.␣testString");
10                 }
```

```
11          }
12  }
```

続いて暗黙的インテントを紹介する。Listing 3 はインテントを送る側のアプリケーションにおけるソースコード記載である。Intent クラスのインスタンス生成時に引数として動作（アクション）を指定（ここでは Intent.ACTION VIEW）し、その動作に渡す情報を指定（ここでは ”https://www.klab.is.sci.toho-u.ac.jp/ ”という URI）している。明示的インテントとは異なり、パッケージ名の指定をせず動作の指定をしていることがわかる。

Listing 3: 暗黙的インテントの例（送信側）

```
1 new Intent(Intent.ACTION VIEW,Uri.parse("https://www.klab.is.sci.toho-u.ac.jp/"))
      ;
```

Listing 4: 暗黙的インテントの例（受信側）

```
1  <activity
2          android:name="jp.ac.toho_u.sci.is.klab.intent.testActivity"
3          android:label="@string/app_name" >
4
5          <intent-filter>
6                  <action android:name="android.intent.action.MAIN"/>
7                  <category android:name="android.intent.category.LAUNCHER"/>
8          </intent-filter>
9
10         <intent-filter>
11                 <action android:name="android.intent.action.VIEW"/>
12                 <category android:name="android.intent.category.DEFAULT"/>
13                 <data android:scheme="https" />
14         </intent-filter>
15 </activity>
```

2.2 IAC

ICC はアプリ内の異なるコンポーネント間におけるコンポーネント間通信 (ICC) と異なるアプリ間におけるコンポーネント間通信 (Inter-App Communication; IAC)) の 2 種類に大別することができる。明示的インテントは指定されたターゲットコンポーネントとの ICC が行われることから ICC、IAC が実現されているが、暗黙的インテントは指定された動作が可能な他アプリとの ICC が行われるため IAC のみ実現されている。

2.3 テイント解析

テイント解析は、解析を行う技術者や自動マルウェア検出ツールに提供をするためにアプリケーションを分析し潜在的に悪意のあるデータフローを提供する。それにより、実際に情報漏えいがポリシー違反を構成するかどうかを判断することができる。テイント解析はあらかじめ定義されたソース（例えば、位置情報を返す API メソッド）から開始し、それが与えらえたシンク（達成点）に達するまでデータフローに従うことによって、アプリケーションを介して敏感なテイント情報を

追跡し、どこのデータが漏洩する可能性があるかについての正確な情報を提供する。その解析手法としては、動的と静的のいずれのものも存在する。

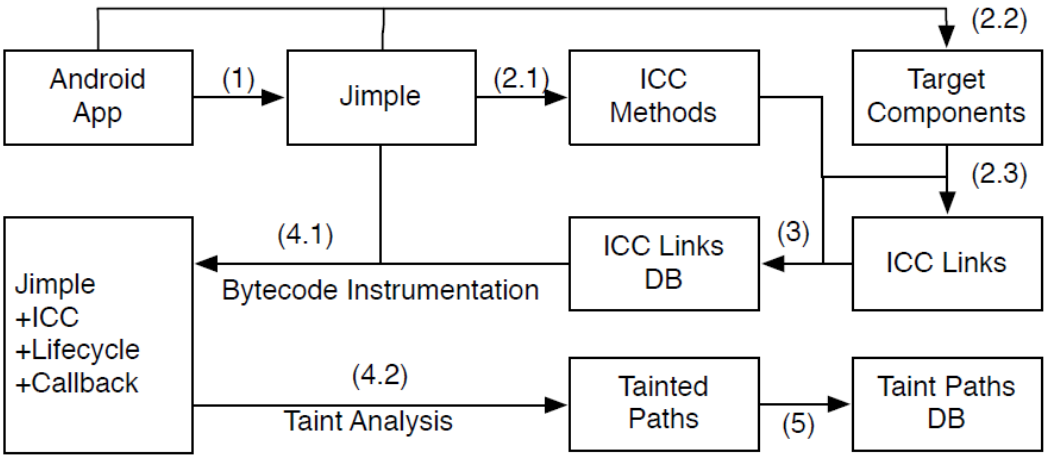

図 1: Li らの論文の Fig.2 ” Overview of IccTA. ” [1]

3 関連研究

3.1 IccTA

IccTA は Android アプリの ICC によるプライバシ情報漏えいのリスクを静的に解析するツールであり、2015 年に Li らにより提案された [1]。IccTA のソースコードはオープンソースとして公開されている [2]。IccTA はこれまでの ICC の研究にはなかったテイント解析を ICC の分析に組み入れ、それを実用レベルまで高度化して、2016 年には高速化の面で研究は進められている [3]。本研究では IccTA のフレームワークを用いて IAC 分析の効率化や高度化を行う。本節ではその IccTA の詳細を解説する。

3.1.1 IccTA のフロー

図 1 は、IccTA の概要である [1]。Android アプリが Java で実装されていても、アプリは従来の Java バイトコードの代わりに Dalvik バイトコードにコンパイルされる。IccTA は最初のステップで Dexpler[4] を使用して、この Dalvik バイトコードを Soot の内部表現である Jimple に変換する [5]。Soot は、Java ベースのアプリケーションを分析するための一般的なフレームワークである。ステップ (2) では、ICC リンクを抽出し、ステップ (3) で、それらを収集されたすべてのデータ（例えば、ICC 呼び出し時の引数やインテントフィルタの値など）とともに データベースに格納する。IccTA は ICC リンクに基づいてステップ (4.1) で Jimple での表現を変更してコンポーネントを直接接続し、コンポーネント間のデータフロー分析を可能にする。ステップ (4.2) では、Android アプリケーション用の高精度のコンポーネント間テイント解析ツールである FlowDroid[6] の修正バージョンを使用し、IccTA は Android アプリ全体の制御フローグラフを作成する。このフローグラフでは、Android コンポーネント間のインテントの値などのコンテキストを伝播させ、正確なデータフロー分析が可能になる。最後のステップ (5) では、IccTA は報告されたテイント解析で得られた漏えいの経路をデータベースに格納する。ステップ（3）と（5）の両方で、次の 4 つを含むすべてを格納する。

- URI やインテントなどの属性値

- インテントフィルタ値を持つターゲットのコンポーネント
- ビルドされた ICC リンク
- 分析結果で得られた ICC による情報漏えい

データベースに保存することで、アプリケーションを一度だけ分析し、それ以降はデータベースから結果を再利用することができる。次の 2 つのセクションでは、ステップ (2) と (4) にある IccTA の主な技術的な貢献について詳しく説明する。

3.1.2 ICC リンクの抽出

この節では、分析対象のアプリから ICC リンクを抽出する方法を紹介する。ICC リンクとは、アプリのソースコード内における ICC メソッドでの情報（明示的なインテントのクラス名、アクション、カテゴリ、mimetype など）とその情報に対応するターゲットコンポーネントを結びつけたものである。IccTA では図 1 のステップ (2.1),(2.2),(2.3) にてアプリから ICC リンクの抽出を行い、ステップ (3) でデータベースに格納する。まず、ステップ（2.1）においては Epicc[7] と呼ばれる Soot と Heros[8] に基づいて ICC メソッドとそのパラメータ値（アクション、カテゴリなど）を識別するためのツールを用いて、ICC メソッド、およびそれらのパラメータ（例えば、インテントのアクション）を得る。また、IccTA は Epicc のアイデアを実装した更に高度なツールである IC3[9] を使用することも可能である。ステップ（2.2）では、アプリケーションの構成ファイルである AndroidManifest を解析し、インテントフィルタの値を取得する。これにより、ステップ (2.1) において抽出した ICC メソッドに対応可能な全てのターゲットコンポーネントを特定する。このとき、ランタイムにおいてブロードキャストレシーバを登録される場合もあるため、バイトコードの分析も必要となる。次に、ステップ（2.3）ではターゲットコンポーネントと ICC メソッドをマッチングさせ、ステップ (3) でデータに格納する。

3.1.3 ICC へのテイント解析

この節では、ICC のテイント解析を実行する計測手法の詳細を説明する。まず、ICC メソッドに対して IccTA がどのように動作するかを示す。ステップ（4.1）では、図 1 に示すように、Jimple コードがコンポーネント同士の接続に従い IccTA によってコード変更が行われる。このコード変更は、IccTA が対応する全ての ICC メソッドに対して行われる。変換の主な考え方は、ICC メソッド呼び出しを、適切なインテントでターゲットコンポーネントのインスタンスと置き換えることにある。startActivity と startActivityForResult の 2 つの最もよく使用される ICC メソッドで、これらの変更を詳しく説明する。同様の方法でサービスとブロードキャストレシーバの ICC メソッドも扱われる。

StartActivity IccTA はまず、ソースと宛先のコンポーネントを接続するブリッジとして機能する IpcSC という名前のヘルパークラスを作成する。次に、startActivity メソッドが削除され、 生成されたヘルパーメソッド（redirect0）（A）を呼び出すステートメントに置き換えられる。そして、IccTA はインテントをパラメータとするコンストラクタメソッド、コンポーネントの関連するすべてのメソッド（ライフサイクルおよびコールバックメソッド）を呼び出す dummyMain メソッドを生成し、getIntent メソッドをオーバーライドする。そのときインテントは、Android システムによって呼び出し元コンポーネントから呼び出し先コンポーネントに転送される。カスタマイズ

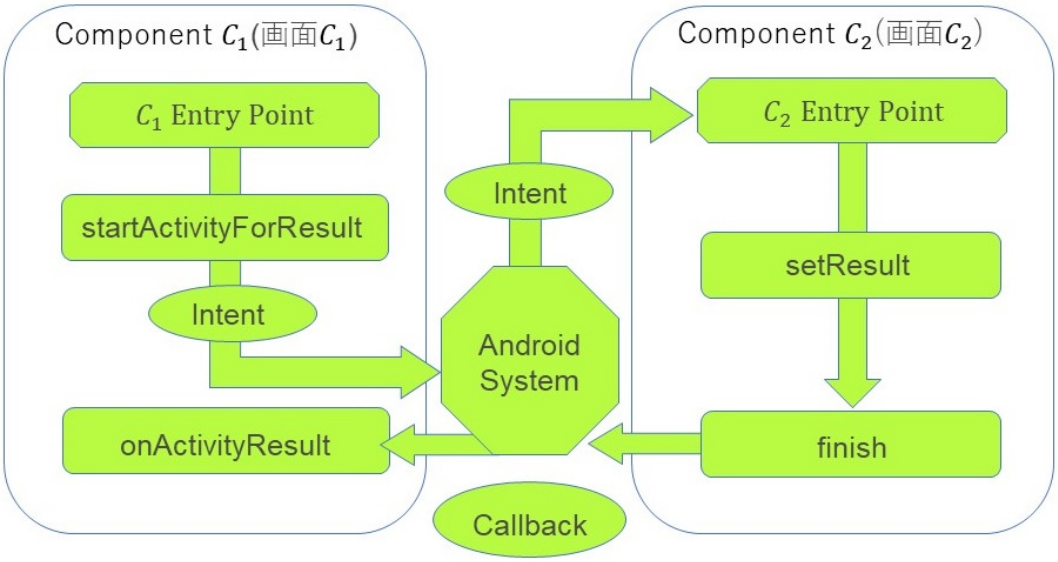

図 2: StartActivityResult 実行時の Android のフロー

されたコンストラクタメソッドである Activity2（Intent i）を使用してインテントをそのコンポーネントとして明示的に送ることにより、Android システムとしての動作をモデル化する。その後インテントをパラメータとして使用し、インテントを新しく生成されたフィールド intent for ipc に格納し、元の getIntent メソッドは Android システムに入ってくる Intent オブジェ クトを要求する。新しい getIntent メソッドは、新しいコンストラクタメソッドにパラメータとして与えられた Intent オブジェクトを返すことで、Android システムとしての動作をモデル化する。ヘルパメソッド redirect0 は、Activity2 型（ターゲットコンポーネント）のオブジェクトを構築し、ヘルパーメソッドのパラメータとして与えられたインテントを使用して新しいオブジェクトを初期化する。次に、Activity2 の dummyMain メソッドを呼び出す。ステップ（3）で格納された ICC リンクを使用することでターゲットコンポーネントの解決がされ、IccTA の内部におけるインテントベースの ICC の処理は明示的、暗黙的に関わらず同一である。

StartActivityForResult 図 2 のようにコンポーネント C1 は startActivityForResult メソッドを使用してコンポーネント C2 を開始することが可能である。このメソッドの場合、C2 の実行が終了すると、C1 は再び C2 から返された結果データで引き続いて実行される。startActivityForResult は、コンポーネント間の一方向通信（例えば、startActivity）のみをトリガする一般的な ICC メソッドと比較して、より複雑な動作が生じる。一般的な ICC メソッドとの一貫性を保つため、IccTA では onActivityResult メソッドを呼び出す C2 の仕上げメソッドを準備していない。その代わりに、Intent を格納する intent for ar フィールドを生成し、これを C1 に返す。返されるインテントは、setResult メソッドによって設定される。Intent の値を intent for ar へ格納することは、setResult メソッドをオーバーライドすることで実現される。ヘルパーメソッド IpcSC.redirect0 は、これらの 2 つのコンポーネントを直接リンクする 2 つの変更を行う。まず、宛先コンポーネントの dummyMain メソッドを呼び出し、次にソースコンポーネントの onActivityResult メソッドを呼び出す。

3.2 その他関連研究

単一のアプリによる情報漏えいの脅威に関する研究は、セカンドアプリと呼ばれる正規のアプリを解凍し悪性アプリを追加し、再圧縮して作成される Android マルウェアを検知する研究 [10] や、端末情報取得 API を実行する際に行われる遠隔手続き呼び出しによる情報取得に着目し、遠隔手続き呼び出しにおける手続き実行時に呼び出されたアプリを検出し、記録する手法を提案した研究 [11] がある。

一方、複数のアプリによる情報漏えいの脅威に関する研究では、端末にインストールされているアプリ同士が持つリンクを通した情報漏えいの脅威をグラフ化し、そのデータをベースに情報漏えいを防ぐ LinkDroid[12] や、アクセスを許可したアプリにプライバシー情報を渡すことで起こる情報漏えいを防ぐためにプライバシー情報をデバイス内で処理をし保護することを目的とした π Box を提案した研究 [13] などがある。しかしいずれもその現実的な脅威は限られたアプリを対象に調査されており大規模であるとはいえない。複数アプリ間の脅威についてはさまざまな面から注目を浴びているものの大規模に精査されてはおらず、それらの必要性が伺える。Android アプリの脅威を対象とした研究はほとんどが単一 Android アプリを対象としており、明示的に複数の Android アプリを対象として扱っている研究は我々の知る限りでは 7 つのみ [20] であり、これらの研究において公開を行っているツールは、Dexpler[4] を用いる BlueSeal[14] とテイント解析を用いる DidFail[15]、DroidSafe[16] である。

しかしながら、Li らの研究において IccTA の実験時に用いられたアプリの数が 15000 個であるのに対し、BlueSeal は 4039 個、DidFail は 0 個、DroidSafe は 24 個と実験に用いられたアプリ数が少なく、IccTA が最も信頼できる IAC ツールだといえる。

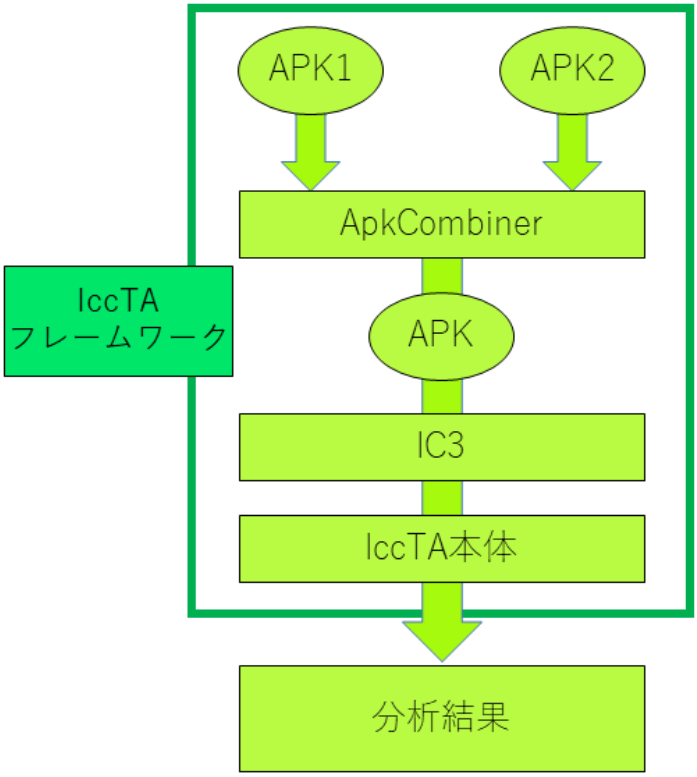

図 3: IccTA フレームワーク

4　IccTA によるアプリデータセット解析における問題点

IccTA の解析を大規模に行う場合、2 つの問題に直面する。1 つは、1 回の解析にかかる時間が長いという点である。そしてもう 1 つは、解析の対象がアプリのペアであり、広い解析対象のアプリの組み合わせを網羅的に調査するには、組み合わせ数が膨大なものとなる点である。

4.1　IccTA の解析に要する時間

3 章 1 節において IccTA のフローを解説したが、実際に IccTA を実装し、解析を行う場合、図 3 のように ApkCombiner、IC3、IccTA の 3 つのツールを通して解析が行われる。これまで仕組みとして IccTA の解説をしたが、図 3 のようにツールとしても IccTA の名称が用いられているため、本論文では今後はツールを IccTA 本体と呼び、仕組みを IccTA フレームワークと呼ぶことにする。各ツールの処理としては、ApkCombiner によって 2 つの APK を 1 つに合併し、IC3 により ICC リンクを抽出し、そして IccTA 本体によってテイント解析を行い、結果が得られるといった流れになる。

予備実験として Linux を用いて (CPU: Intel Core i7 (3.60GHz), RAM 16GB) を用い、各ツールの実行時間を調査した。以下にそれぞれの結果を示す。

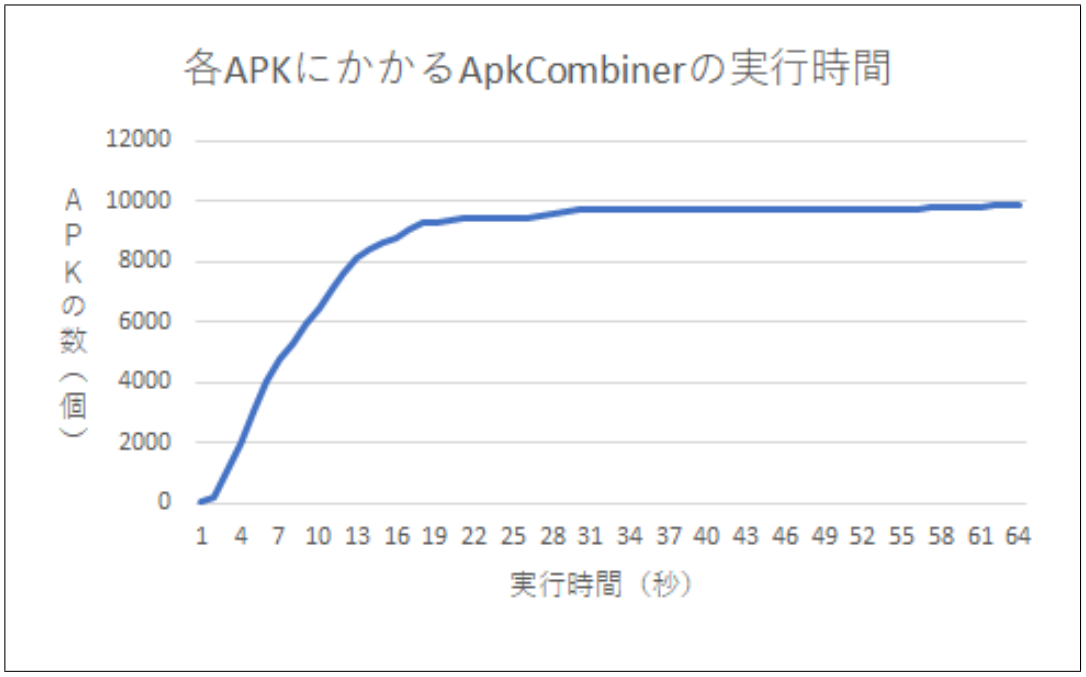

図 4: ApkCombiner の実行時間

4.1.1 ApkCombiner の実行時間

IccTA は 2 つの APK 間での ICC 解析を行うものであるが、IccTA 本体への入力は APK ファイル単体である。そのため、2 つの APK をコード変換し、1 つの APK に合併（Combine）する必要があり、ApkCombiner が用いられる。

図 4 は AndroZoo よりランダムで選出した 100 個の apk に対して、ApkCombiner を実行した際の実行時間のグラフである。全ての組み合わせは 63 秒以内に処理を終えており、全体の平均実行時間は 9 秒である。

4.1.2 IC3 の実行時間

図 1 において (1),(2.1),(2.2),(2.3),(3) の処理を行うツールは、IccTA 本体には含まれていない。そのため IccTA 本体を実行する前に、IC3 を用い、2 つの APK 間における ICC リンクの抽出、データベースに格納する。IC3 の実行時間はアプリの組み合わせによって大きく異なり、30 分以上経過しても終わらずに同じような処理を続けていたため、30 分の制限時間を設けて IC3 を実行することにした。

図 5 は前節の ApkCombiner により合併されたアプリ 3622 組の組み合わせに対して IC3 を実行した内、30 分以内に処理を終えた 3003 組のアプリと実行時間のグラフである。

(I) の 90%を終えるのに要した時間はおおよそ 420 秒、(II) の 95%には 650 秒と 10 分以内に処理を終えており、(III) の 99%には 1330 秒と、20 分程の時間を要していることがわかる。また、本処理を終えた 3003 組の IC3 平均実行時間は 2 分 13 秒であり、このことから多くのアプリは時間を要せずに分析を終えるが、いくつかのアプリで実行時間が多くかかっていることがわかった。

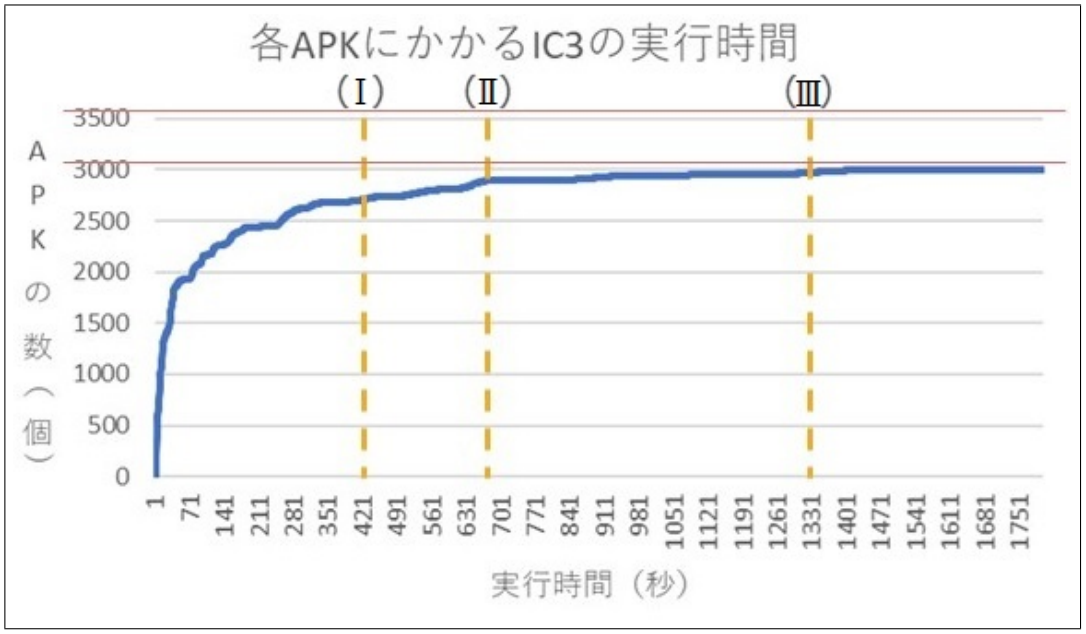

図 5: IC3 の実行時間

4.1.3 IccTA 本体の実行時間

図 1 において (1),(4.1),(4.2),(5) の処理を行うツールである。ApkCombiner により合併された APK を IccTA 本体に入力するが、IC3 にて抽出された ICC リンクを基にテイント解析を行うため、IC3 の実行後に行う。IccTA 本体の実行時間についても IC3 のように実行時間は一定ではなく、組み合わせにより実行時間は 1 時間から 10 時間以上と大きく異なる。本研究では 10 時間を制限とし、10 時間以内に解析が完了したものを図 6 に示す。

図 6 は前節の処理を終えた 60 組のアプリの組み合わせに対して IccTA を実行した内、10 時間以内に処理を終えた 57 組のアプリと実行時間のグラフである。(I) は 90%、(Ⅱ) は 95%、(Ⅲ) は 99% の組み合わせが実行を終えている時間を示している。

以上のことから IC3、IccTA 本体の解析に用いる実行時間は一定ではなく、効率良く解析を行う場合、制限時間を定める必要があるといえる。

4.1.4 全体の実行時間

各ツールでの実行時間を合わせた実行時間では、10 時間以上を要する場合が存在した。

4.2 データセット大規模化による組み合わせ爆発

Li らの論文では触れられていないが、IccTA の解析フローから解析するにあたって 2 つの APK を合併させなければならない。その際、APK のデータセット全体での IAC による情報漏えいを IccTA フレームワークを用いて分析するにはすべての組み合わせに対して分析を実施しなければならない。また、IccTA による分析結果は ApkCombiner に入力される APK の順序によって変わることから、データセットに含まれるアプリ数（APK 数）を n とすると、データセット全体の分析

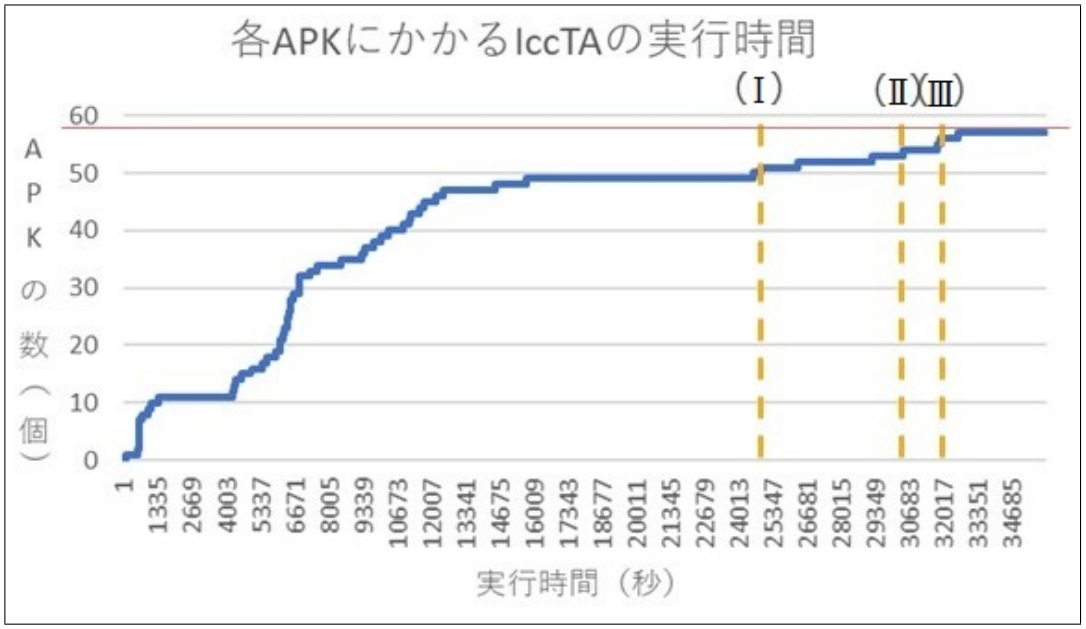

図 6: IccTA 本体の実行時間

に必要な IccTA フレームワークによる分析回数 N は

$$N = n * (n - 1) \tag{1}$$

となり、分析回数が膨大になることがわかる。そのため、4 章 1 節で挙げたようにすべてのアプリを分析することは現実的ではない。

近年、Android アプリ分析のためのデータセットは巨大化しており、The Statistics Portal の調べによると、2018 年 6 月の時点で GooglePlay に存在するアプリ数は 3,300,000 となっている [17]。PlayDrone が収集して公開しているデータセットは 1,402,894 のアプリ数であり [18]、AndroZoo が収集して公開しているデータセットは 7,019,769 のアプリ数となっている [19]。

表 1: intent に含めることのできる情報

フィールド	挿入される値の例
Component	MyActivity.class
Action	ACTION_VIEW
Data	IMAGE/JPEG
Category	CATEGORY_DEFAULT
Extra	EXTRA_EMAIL
Flags	FLAG_GRANT_READ_URI_PERMISSION

5 IccTA によるアプリデータセット解析の効率化

5.1 組み合わせ爆発を抑える事前処理の適用

本研究では、IAC による脅威分析の効率化に際して、組み合わせ数の爆発を抑える事前処理を提案する。事前処理は、図 3 の IccTA フレームワーク入力前にデータセット全体に実施し、入力する 2 つの APK の取捨選択を行う。インテントにはターゲットコンポーネントを指定する明示的インテントと ICC メソッドでの情報に対応したターゲットコンポーネントを指定する暗黙的インテントの 2 種類が存在するが、明示的インテントはアプリ内の ICC に用いられることが多く、暗黙的インテントは IAC にのみ用いられることから、本研究では暗黙的インテントに着目したスクリーニングを行う。また、IAC が行われる際、2 つのアプリは送信側アプリ、受信側アプリに分かれるが、ApkCombiner によるアプリの合併時においても送信側、受信側アプリが定められるため、送信側、受信側のそれぞれにおいてもアプリのスクリーニングを行う。

5.2 暗黙的インテントの送信側のスクリーニング

5.2.1 明示的インテントと暗黙的インテントの区別

表 1 はインテントに設定できる情報の一覧である。ICC は ICC メソッドにおいて Intent オブジェクトのインスタンス化、及び StartActivity メソッド、又は StartActivityForResult メソッドによって行われるが、その際、Intent インスタンスに含まれているデータによって明示的インテント、または暗黙的インテントでの ICC が行われる。本研究では暗黙的インテントの送信側のスクリーニングを行うため、対象アプリの Intent インスタンスに含まれているデータから暗黙的インテントであるか判断する必要がある。そのため、ICC メソッドを抽出し、ICC メソッドにより生成されている Intent インスタンスを調べることで暗黙的インテントを保有するアプリを送信側のアプリとしてスクリーニングを行う。インテントは明示的インテント、暗黙的インテントの 2 種類にのみ分類され、表 1 において、Component のフィールドを保持しているインテントが明示的インテント、その他のインテントを保持しているインテントが暗黙的インテントである。そのため、明示的インテントであるか調べることにより、暗黙的インテントを判別することができる。

5.2.2 暗黙的インテントの保有する情報

暗黙的インテントは送信側の保有する情報、受信側の指定する情報によって IAC が行われる対象が定まる。そのため暗黙的インテントであるかだけでなく、保有する情報に従ってアプリの分類

表 2: intentfilter で指定できる情報

情報名	インテントフィルタの記述例
Category	<category android:name= ”android.intent.category.DEFAULT”/>
Action	<action android:name= ”android.intent.action.SEND”/>
Data	<data android:mimeType= ”text/plain”/>

表 3: intentfilter の特殊な指定例

情報名	指定例
Category	CATEGORY_LAUNCHER
Category	CATEGORY_DEFAULT
Data	IMAGE/*

が必要である。次の節で説明するが、受信側で指定される情報は Category や Action、Data の 3 種類であるため、この 3 種類に従って送信側アプリの分類を行う。なお、Data においてはスキーム、MIME タイプなど複数の情報が存在するが、本研究では 2 つの Android アプリの IAC に着目していることから MIME タイプのみに着目し、分類するものとする。

5.3 暗黙的インテントの受信側のスクリーニング

暗黙的インテントにおける受信側のアプリはソースコードではなくマニフェストファイルにてインテントフィルタとして指定される。指定できる情報は表 2 に示す 3 種類である。そのため、マニフェストファイル内にインテントフィルタが指定されているアプリを受信側としてスクリーニングし、送信側と同様に Category、Action、MIME タイプによって分類する。

5.4 送信側、受信側の組み合わせ

1 節、2 節にてスクリーニング、分類されたアプリを組み合わせるが、この時表 3 のような指定がされた受信側のアプリには注意が必要である。例えば Category の場合、基本的には同様のカテゴリ名を保持する送信、受信側のアプリを組み合わせるが、カテゴリ名が「CATEGORY_LAUNCHER」である受信側のアプリの場合は IAC における受信側アプリの指定ではなく、端末からアプリを起動するための記述であるため、IAC の組み合わせ対象外である。また、カテゴリ名が「CATEGORY_DEFAULT」である受信側のアプリの場合は Category が設定されている送信側アプリには全て対応するため、Category が設定されている送信側アプリ全てと組み合わせる。同様に MIME タイプで分類されている送信、受信側アプリについても IMAGE の MIME タイプが設定されている送信側アプリは全て「IMAGE/*」が指定されている受信側アプリと組み合わせる。

表 4: 100 個の apk に対する送受信側のスクリーニング結果

情報名	送信側	受信側
Category	81 個	38 個
Action	79 個	59 個
MIME タイプ	76 個	4 個
合計	82 個	60 個

表 5: スクリーニング後の apk の組み合わせ数

情報名	組み合わせ数	削減率
Category	769 組	92.23%
Action	3029 組	69.40%
MIME タイプ	136 組	98.62%
合計	3138 組	68.30%

6 効率化手法の評価

6.1 スクリーニングによる組み合わせ数減少効果

本研究では AndroZoo よりランダムで選出した 100 個の apk に対してスクリーニングを行った。表 4 は各要素に従いスクリーニングした結果である。これにより選別された送信側、受信側アプリから各フィールドに従い作られた組み合わせを表 5 に示す。

本スクリーニングの結果、Category、及び MIME タイプの分析対象の削減率は 90%以上を示し、Action においてのみは 69.40%の削減率を示した。

6.2 スクリーニングの再検討及び実行

1 節のスクリーニングにより、全体で 68.30%と多くの不要な組み合わせの削減を達成することができた。一方で、マーケット上にあるアプリの膨大な組み合わせから見ると依然多くの分析対象が残っていることになる。そこで、さらなる削減を達成すべく、1 節の結果において最も削減率の低かった Action に焦点を当て、スクリーニングの再検討を行う。

6.2.1 脅威とならない組み合わせの排除

表 6 は各アクション名における Action の組み合わせ数の一部を示している。このことから、Action において最も組み合わせ数が多いアクション名は「ACTION_VIEW」であることが分かる。しかしながら、「ACTION_VIEW」と同時に保持され、用いられる情報は URI などの文字列であり、IAC による情報漏えいの視点で考えると脅威となるデータの通信が発生する可能性は低いと考えられる。そのため、本研究では「ACTION_VIEW」により組み合わされたアプリは分析対象外であるとした。同様の理由で表 6 においては「ACTION_SEND」以外のアクション名は対象外となり、スクリーニングの再検討が必要である。

表 6: アクション名別における Action の組み合わせ

アクション名	組み合わせ数
android.intent.action.VIEW	1554 組
android.net.conn.CONNECTIVITY_CHANGE	867 組
com.google.android.c2dm.intent.RECEIVE	777 組
android.intent.action.MAIN	441 組
android.intent.action.BOOT_COMPLETED	425 組
android.intent.action.USER_PRESENT	410 組
android.intent.action.PACKAGE_ADDED	308 組
com.google.android.c2dm.intent.REGISTRATION	299 組
com.android.vending.INSTALL_REFERRER	273 組
android.intent.action.SEND	240 組
android.intent.action.PACKAGE_REMOVED	80 組

表 7: カテゴリ名別における Action の組み合わせ

アクション名	組み合わせ数
android.intent.category.DEFAULT	630 組
android.intent.category.BROWSABLE	180 組
android.intent.category.HOME	27 組
android.intent.category.LEANBACK_LAUNCHER	5 組
com.wMyInsanityWorkoutRoutines	1 組

表 7 は各カテゴリ名における Category の組み合わせ数の一部を示している。本表においてはアクション名同様脅威となりうるデータの通信が行われるカテゴリ名は「android.intent.category.DEFAULT」のみとなる。

6.2.2 再スクリーニングによる組み合わせ数減少効果

前節にて Category、Action に脅威でない組み合わせがあることが分かった。しかし、Category、Action にはあらかじめ用意されているカテゴリ名、アクション名に加えてオリジナルのカテゴリ名、アクション名を追加することができる。そのため、全てのカテゴリ名、アクション名を調べ、脅威でないものを除外することは難しい。このことから本研究では、6 章 1 節にて行ったスクリーニング結果から特定の条件を加え再スクリーニングを行った。

表 8 は 100 組以上の組み合わせ数が存在する Action、及び 2 組以上の組み合わせ数が存在する Category の中で脅威でない組み合わせを除外する再スクリーニングの結果を示している。再スクリーニングの結果、Category の削減率は 93.59%、Action の削減率は 94.87%を示し、全体の削減率は 89.47%を示した。

表 8: スクリーニング後の apk の組み合わせ数

情報名	組み合わせ数	削減率
Category	634 組	93.59%
Action	507 組	94.87%
MIME タイプ	136 組	98.62%
合計	1042 組	89.47%

6.3 スクリーニングによる脅威分析率の上昇効果

事前処理によるスクリーニングは 1 度目では 68.30%、再スクリーニングでは 89.47%と高い削減率を達成した。しかし、依然分析対象の多さからより高い削減を実現することが望ましい。次章では、より高い削減の検討と、効率の良い解析について議論を行う。

また、本研究では特定のカテゴリ名、アクション名の除外を行ったが、さらに多くのカテゴリ名、アクション名について事前に調べ、脅威でないものを除外することでさらに削減率を上げることが可能であるだろう。Category、Action、MIME タイプの各情報においても、それぞれの脅威分析率を調べることでより十分な効率向上が図れると考えられる。

7　考察

本章では、得られた結果から更なる削減のための考察と、大規模分析における高度化についての考察を行う。

7.1　スクリーニングの効率向上

今回の提案手法では 89.47%の削減を実現し、一定のスクリーニングの効果があったことがわかる。一方で、組み合わせ数の爆発を考慮すると、より効率の良いスクリーニングが求められる。

しかし、Action、Category については、インテントに含められる情報としては一般的であることに加え、ICC 以外にも広く使われている情報であるため、Action、Category が記載されていることが必ずしも ICC、特にアプリ間での IAC が発生していることを示しているとは限らない。たとえば、OS や端末の操作行動などが渡されることも多い。これらをより詳細に調査し分類をすることで、より効果的なスクリーニングが実施可能であると考えられる。

また、1 回目のスクリーニングにおいて最も多くを占めた Action である ACTION_VIEW のように、MIME タイプと併用されるものでない限りは IAC 内において脅威とならない Action、Category は分析対象から除外すべきであり、これらの調査も検討すべきであろう。

ただし、MIME タイプにおいては IAC にて IMAGE データなどのファイルの通信が行われるため、スクリーニングの除外対象となる MIME タイプは存在せず、そしてその削減率は 98.64%となり、高い削減となる。いずれにせよ、より多くの分析を行い、その結果をさらにフィードバックすることが求められる。

7.2　スクリーニングから漏れたアプリ組み合わせの分析

スクリーニングの効果は数値として表せたものの、その結果重要な脅威が見逃されている可能性がある。本論文ではその脅威の見逃しについての評価が行えていなかったが、必要となる部分である。

当然、分析については全アプリの組み合わせを対象にすることは難しいため、より脅威が存在する可能性が高いアプリなどを対象にできるよう順位付けが必要である。例えば、本スクリーニングにより除外されたアプリの組み合わせにおいて、アクション名から IAC における脅威が低いと判断された組み合わせは分析対象としての優先度が高くあるべきだが、一方で、インテントを用いず IAC を行わない組み合わせは優先度が低いと考えられる。

7.3　データセット分析の高度化

組み合わせを削減することによる効率化が達成されても、まだ多くの解析対象が残る。これらのうちよりリスクが高いものから優先的に解析することで、有用な分析結果を素早く得ることも重要である。ここではいくつかの分析高度化の可能性について議論をする。

アプリの組み合わせを考えた場合、IAC 送信側アプリと IAC 受信側アプリの双方がリスクの低いアプリである場合は、その分析の優先順位を下げてもよいと考えられる。たとえば、あるアプリ A が重大な脅威を持つアプリであっても、アプリ A のダウンロード数が 100 程度であった場合、マーケット全体に及ぶリスクは低い。よって、送信側アプリと受信側アプリの双方がダウンロードが多いアプリの組み合わせから優先的に分析をすることで、リスクの高くなる可能性のある分析が

優先的に行うことができる。優先順位付けとしては単純に受信側アプリのダウンロード数と送信側アプリのダウンロード数の積を数値としてソートすることで実現は可能であろう。あるいはどちらかのアプリがジャンルにおいて支配的なアプリである場合は、そのアプリを軸に据えた網羅的な分析をすることも可能であろう。

アプリを分析するうちに得たノウハウを利用することで、そのノウハウを生かした優先順位付けも考えられる。本研究では Category、Action、MIME タイプに着目しスクリーニングを行ったが、これらの情報において特に脅威が発生しやすいアプリの特徴を調べることで、各情報ごとにおける脅威の発生率を算出することができる。そのため、アプリの特徴や保持する情報に従った優先順位付けや、各情報ごとにおけるアプリの脅威発生率、及び分析結果を基にそのアプリが脅威であるか否かといった分類を行う教師あり機械学習も十分に考えられる。

また、IccTA の分析時に出力する情報は分析結果、及び分析時の詳細な情報であるが、これらの情報から脅威であることを理解するには専門的な知識を要し、またアプリの組み合わせによっては 1 回の分析で 1GB を超える情報量を出力する。ユーザの求める結果はアプリの組み合わせが脅威か否かであるため、分析結果からさらに脅威か否かの情報のみ抽出、及びリスト化も考慮すべきであろう。

7.4 別分析の検討

IccTA の分析時間の長さから、別分析を用いることの検討が必要だと考えられる。IccTA の他に IAC 分析を行うツールは複数存在しているが、公開されているものは我々の知る限り BlueSeal[14] と IccTA 同様テイント解析を用いる DidFail[15]、DroidSafe[16] の 3 つである。また、IccTA の実行時間の長さについては、分析するたびに蓄積されるデータ量が原因であることが分かったことから、これらのデータを分析を行うたびに削除することで分析時間の削減を図る案も挙げられる。

8　まとめ

本研究では、Android のアプリ間で行われる IAC が原因で起こる情報漏えいを防ぐための分析手法として、IAC の脅威分析技術の効率化と高度化を目的とした手法の提案と実行、及び考察を行った。効率化では、大量のデータセットのケースで発生する組み合わせ数の爆発的増加を削減するためのスクリーニングを提案、及び実行を 2 度行い、89.47%と高い削減率を達成した。また、さらなる効率化の可能性として、スクリーニングの対象として行った Category、Action、MIME タイプについての詳しい調査、及び取捨選択を挙げ、スクリーニングの対象外とされたアプリ群に対しての効率的な分析方法の考案も行った。高度化の面においてはダウンロード数からリスクの高い組み合わせの優先順位、及び分析結果の集約化を考察した。本研究により効率化が達成されたものの、多くの解析対象及び解析に要する時間が大きいことからさらなる高度化の実現は必須である。今後はこれらを運用するより大きな仕組みや、絶えずアップデートするアプリ群に対する動的な対応など、実用化に向けたアプローチが重要になるであろう。

参考文献

[1] Li Li, Alexandre Bartel, Tegawend F. Bissyand, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. 2015. IccTA: detecting inter-component privacy leaks in Android apps. In Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE ’15), Vol. 1. IEEE Press, Piscataway, NJ, USA, 280-291.

[2] https://github.com/lilicoding/soot-infoflow-android-iccta

[3] Combining Static Analysis with Probabilistic Models to Enable Market-Scale Android Inter-component Analysis

[4] A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon. Dexpler: Converting android dalvik bytecode to jimple for static analysis with soot. In ACM Sigplan International Workshop on the State Of The Art in Java Program Analysis, 2012.

[5] P. Lam, E. Bodden, O. Lhot´ ak, and L. Hendren. The soot framework for java program analysis: a retrospective. In Cetus Users and Compiler Infastructure Workshop (CETUS 2011), 2011.

[6] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycleaware taint analysis for android apps. In Proceedings of the 35th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI 2014), 2014.

[7] D. Octeau, P. McDaniel, S. Jha, A. Bartel, E. Bodden, J. Klein, and Y. Le Traon. Effective inter-component communication mapping in android with epicc: An essential step towards holistic security analysis. In Proceedings of the 22nd USENIX Security Symposium, 2013.

[8] E. Bodden. Inter-procedural data-flow analysis with ifds/ide and soot. In Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program analysis, SOAP ’12, pages 38, 2012.

[9] D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel. Composite constant propagation: Application to android inter-component communication analysis. In Proceedings of the 37th International Conference on Software Engineering (ICSE), 2015.

[10] 磯原隆将, 川端秀明, 竹森敬祐, 窪田歩, 可児潤也, 上松晴信, 西垣正勝 ”セカンドアプリ内包型 Android マルウェアの検知” コンピュータセキュリティシンポジウム 2011

[11] 西本祐揮, 堀良彰, 櫻井幸一 ”動的解析を用いた Android における端末情報の取得検知手法” 情 報処理学会研究報告 IPSJ SIG Technical Report

[12] H. Feng, K. Fawaz, and K. G. Shin, ”LinkDroid: Reducing Unregulated Aggregation of App Usage Behaviors” In Proc. of the 24th USENIX Security Symposium (USENIX Security 2015)

[13] S. Lee, E. L. Wong, D. Goel, M. Dahlin, and V. Shmatikov ”π Box: A Platform for PrivacyPreserving Apps” In Proc. of the 22rd USENIX Security Symposium (USENIX Security 2013)

[14] F. Shen, N. Vishnubhotla, C. Todarka, M. Arora, B. Dhandapani, E.J. Lehner, S.Y. Ko, L. Ziarek, Information · P ws as a permission mechanism, in: Proceed- ings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ACM, 2014, pp. 515-526.

[15] W. Klieber, L. Flynn, A. Bhosale, L. Jia, L. Bauer, Android taint · P w analysis for app sets, in: Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis, ACM, 2014, pp. 1-6.

[16] M.I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, M. Rinard, Informa- tion- · P w analysis of android applications in droidsafe, in: Proc. of the Net- work and Distributed System Security Symposium (NDSS). The Internet Soci- ety, 2015.

[17] The Statistics Portal, ”Number of available applications in the Google Play Store from December 2009 to December 2016”, https://www.statista.com/statistics/266210/ number-of-available-applications-in-the-google-play-store/

[18] ”PlayDrone APK ’ s : Free Software : Download & Streaming : Internet Archive”, https: //archive.org/details/playdrone-apks

[19] ”Androzoo”, https://androzoo.uni.lu/

[20] Static analysis of android apps: A systematic literature review

[21] L. Li, A. Bartel, J. Klein, and Y. Le Traon. Detecting privacy leaks in android apps. International Symposium on Engineering Secure Software and Systems - Doctoral Symposium (ESSoS-DS2014), 2014.

[22] S. Han, J. Jung, D. Wetherall. A study of third-party tracking by mobile apps in the wild. UW-CSE-12-03-01, 2011

[23] R. Stevens, C. Gibler, J. Crussell, J. Erickson, and H. Chen, “Investigating user privacy in Android ad libraries, ”in Mobile Security Technologies (MoST) 2012, 2012