Vol.2015-DPS-162 No.46
Vol.2015-CSEC-68 No.46
2015/3/6

Android アプリの危険性を見分ける支援ツールの提案

松崎 悠太[†1]　　千石 靖[†1]

近年，日本におけるスマートフォンの普及率が高くなってきている．その中で Android の普及率が 63%と高くスマートフォンを持っている人たちの過半数を占めている．それに伴い，Android 向けアプリの数も多くなっており，100 万種類以上存在している．しかし，それらのアプリの中にはユーザに害を与えてしまうマルウェア（モバイルマルウェア）も存在しており，個人情報が外部に漏えいされる問題やアプリを実行すると他のアプリもインストールされる問題などがある．モバイルマルウェアの被害件数は 2014 年の 3 か月間で新しく 75 万件以上発見され，過去 2 年間では約 400 万件にも上っている．本研究ではアプリをインストールする前にアプリの機能，動作やパーミッションを分析し，危険性を判断するサーバの構築を考え，初めて Android を使う人たちやアプリをインストールするのが心配な人たちを対象とした Android アプリの危険性を見分ける支援ツールを提案する．

Proposal of a support tool distinguishing the risk of Android application

YUTA MATSUZAKI[†1]　YASUSHI SENGOKU[†1]

In late years, the diffusion rate of the smartphones in Japan rises. Diffusion rate of Android is 63% and occupies the majority of people having a smartphone. The number of Android applications has also increased; it exists more than 1 million. However, the malware (mobile malware) harming a user exists in the applications. There are the problems that personal information is leaked to the outside. And there are the problems that other applications are installed when run the application. The numbers of mobile malware damages are discovered more than 750,000 newly in three months of 2014 and rises to approximately 4 million in the past 2 years. In this research, analyze of the application action and "Android permission" before installing application and think about the construction of a server judging the risk. We propose the support tool, which can distinguish the risk of the Android application for people who are anxious about people using Android for the first time and installing application.

1. はじめに

近年，スマートフォンの普及がさらに進んでいる．総務省が情報通信端末の世帯保有率を調査したところ，62.6%になり前年度に比べ 13.1 ポイント上がった[1]. 情報通信端末の世帯保有率を図 1 に示す．このグラフから，今後さらにスマートフォンが普及する可能性が見て取れる．

そのスマートフォンの OS の中で Android の普及率が高くなってきている．MM 総研の調査では，Android 普及率が 63%と高く，スマートフォンを持っている人たちの過半数を占めている[2].

それに伴い, Android 向けアプリの数も多くなっており，100 万種類以上存在している．しかし，それらのアプリの中にはユーザに害を与えてしまうマルウェア(モバイルマルウェア)も存在しており，個人情報が外部に漏えいされる問題やアプリを実行すると他のアプリもインストールされる問題などがある．図 2，図 3 は McAfee が調査したモバイルマルウェアの被害件数のグラフである．図 2 を見るとモバイルマルウェアの被害件数は2014年の3か月間で新しく 75 万件以上発見され，図 3 を見ると，過去 2 年間で被害件数は約 400 万件にも上っており，依然として増加傾向である[3]．そのため，どのようなアプリがマルウェアなのか見極めることがモバイルマルウェアの被害を減らせる方法だと考える．

よって，本研究ではアプリをインストールする前にアプリの機能，動作やパーミッションを解析し，危険性を判断するサーバの構築を考え，初めて Android を使う人たちやアプリをインストールするのが心配な人たちを対象とした Android アプリの危険性を見分ける支援ツールを提案する．

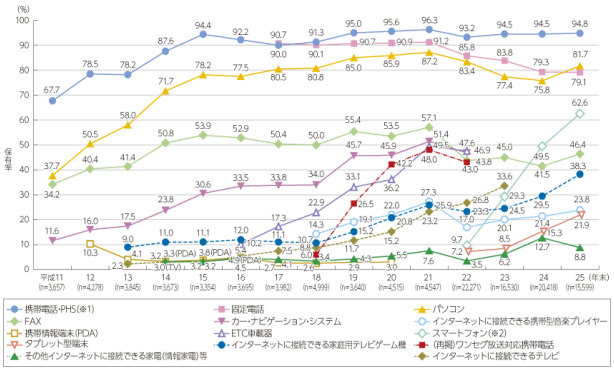

図 1　情報通信端末の世帯保有率

†1 金沢工業大学大学院
Kanazawa Institute of Technology Graduate School

ⓒ2015 Information Processing Society of Japan

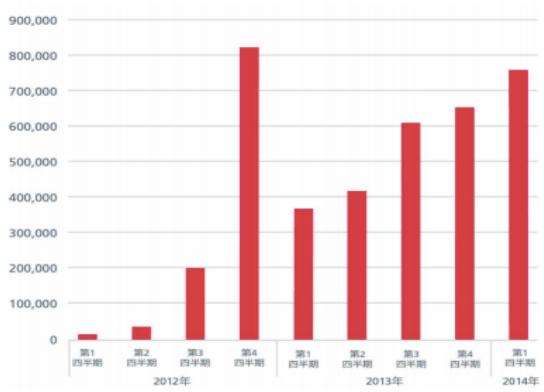

図 2　モバイルマルウェア被害件数

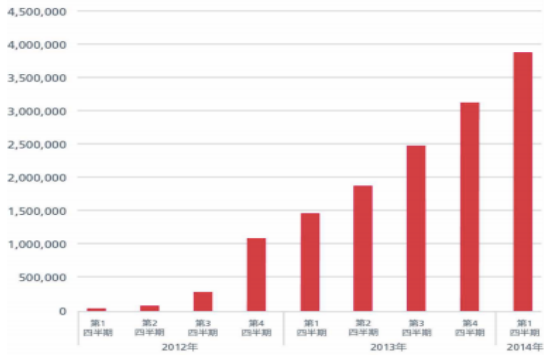

図 3 モバイルマルウェア累計被害件数

2. 被害事例と Android の問題点

ここでは，危険なアプリから起こった 2 つの被害事例の紹介と Android の問題点について述べる．

1 つ目の事例は，2014 年 3 月に McAfee が“Google Play から他のアプリのダウンロード，インストールおよび起動を行う危険な国内ユーザ向けアプリを Google Play 上で確認した”[4]と報告したものである．本来は，Google Play からアプリをインストールする際にアプリが使用する権限を確認し，ユーザが許可するとアプリがインストールされる．しかし，上記のアプリをインストールすると，本来のアプリをインストールする仕組みを無視して自動的に他のアプリをインストールしてしまう．この機能は，ユーザがインストールをする許可をしたことによって取得した Google アカウントの認可トークンを用いて，非公式な方法で通信を実現している．

2 つ目の事例は，2014 年 3 月に McAfee が“Android 端末ユーザの Google アカウント ID（多くの場合，Gmail メールアドレス），Facebook アカウント ID（ログイン用メールアドレス）および Twitter アカウント ID を密かに収集する不審なアプリを Google Play 上で確認した”[5]と報告したものである．この事例で問題になっているアプリを起動すると情報端末に登録されているそれぞれの ID が外部のサーバに送信されてしまう．これも，アプリのインストール時に権限の許可をしたことにより取得した権限を使ってユーザの情報を外部に送っているのである．

これらの事例では，アプリをインストールする際に行われる権限の要求を確認しないで許可してしまい何も考えずにアプリをインストールしてしまう人たちが多いことがわかった．そのため，モバイルマルウェアの被害が減らないのだと推測できる．図 4 に権限の要求をユーザに通知する画面を示す．

図 4 ユーザへ権限の要求・通知

図 4 を見てみると，権限はどのようなものなのかという簡単な記載しかされていないためユーザは権限を許可した場合の危険性について理解ができないので，多くのユーザはアプリの危険性を見極めることができない．よって，支援ツールを使うことで，ユーザがアプリに潜んでいる危険性を見極めることができれば，モバイルマルウェアなどによる被害が減少すると考えられる．

3. システムの提案

本研究では，アプリの機能，動作やパーミッションを解析し，その解析結果から危険性を割り出し，危険性の種類ごとに区別するツールを提案する．

提案するツールは，自作の Android アプリとサーバ間の連携でシステムを構築する．自作アプリでは，危険性を知りたいアプリの解析をサーバにリクエストする．次にサーバ側では，リクエストされたアプリを取得，解析し，その結果をデータベースに記録する．解析結果は、Web サービスを使って閲覧できるようにするが，その際に，同じ危険性を持っているアプリも紹介する．図 5 に提案するツールの処理の流れを示す．

ⓒ2015 Information Processing Society of Japan

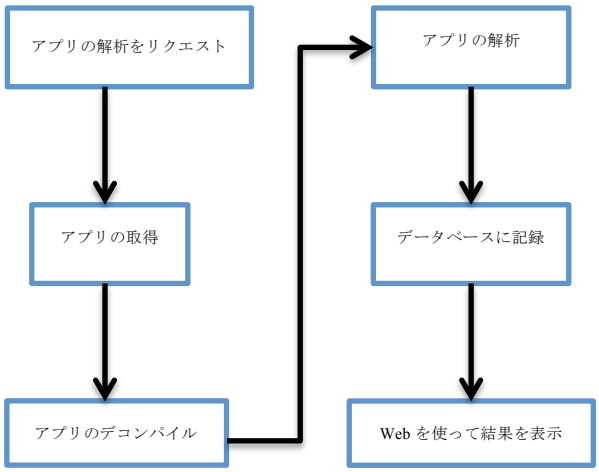

図 5 提案するツールの処理の流れ

3.1 Android アプリ

解析のリクエストに Android アプリを使う理由は，どのような場所にいてもリクエストすることができるからである．スマートフォンの機能の 1 つであるモバイルデータ通信機能が利用できるため，Wi-Fi 環境がないところでも解析リクエストができる．現在のモバイルデータ通信規格の主流は LTE になっているので通信速度も速い．

アプリとサーバとの通信方式として，HTTPS 通信を使用し，危険性を確認したいアプリの名前や種類，アプリを取得した URL などをサーバに送信する．アプリとサーバの通信方式には HTTP 通信と HTTPS 通信の 2 つの方法がある．この 2 つの方法には，セキュアな通信方式かどうかという違いがある．

表 1 通信方式の比較

		HTTP	HTTPS
特徴	URL	http://で始まる	https://で始まる
	通信内容の暗号化	なし	あり
	通信内容の改ざん検知	不可	可
	接続先サーバへの接続	不可	可
被害リスク	攻撃者による通信内容の読み取り	高	低
	攻撃者による通信内容の書き換え	高	低
	アプリの偽サーバへの接続	高	低

表 1 は日本スマートフォンセキュリティ協会（JSSEC）が作成した「Android アプリのセキュア設計・セキュアコーディングガイド」の HTTP 通信と HTTPS 通信の特徴と被害リスクの比較表である[6]．この表の被害リスクに注目してみると，HTTP 通信は攻撃者により通信内容を読み書きされるリスクや偽サーバに接続されるリスクが高い．自作アプリが送る通信内容にはセンシティブなものはないが，通信内容を読み書きされるといった改竄のリスクを避けるために HTTPS 通信を採用する．

3.2 対象アプリの取得方法

サーバ側では自作アプリから送られてきた通信内容から解析対象のアプリを検索し，対象アプリの apk ファイルを取得する．apk ファイルは，Android アプリを格納するためのファイル形式である．apk ファイルを手に入れてデコンパイル（本稿では，apk ファイルを解析可能な状態に展開することを指す.）し，アプリの機能を解析する．

Android アプリを手に入れる方法は 2 つある．1 つ目はスマートフォン上でアプリを取得する方法で，2 つ目は別デバイス（解析サーバ側）でアプリの apk ファイルを取得する方法である．

本システムでは 2 つ目の方法を採用する．理由として，サーバ側で取得すると解析作業が効率よくできるからである．スマートフォンでアプリをダウンロードしてしまうと，次の解析作業に移るためにスマートフォンで取得したアプリをサーバに移さなくてはいけなくなるため，作業効率が悪くなってしまう．また，スマートフォン上で Google Play を使ってアプリをダウンロードするとインストールまで自動的に進んでしまうため，通常の方法で apk ファイルを取得することができない．2 つ目の方法を使えば，アプリをインストールすることなく apk ファイルをストレージ上に保存でき，そのまま解析作業に移ることができるため，プロセスを減らすことができる．

3.3 対象アプリのデコンパイル

第 3.2 節で述べた方法で取得した apk ファイルをデコンパイルし，アプリの解析ができる形にする．デコンパイルは apktool を用いて行う．apktool は apk ファイルをアーカイブ前の状態に展開することができるツールである．apktool の大きな特徴として以下の 2 つが挙げられる[7].

- リソース（resources.arsc, XML, 9.png 等のファイルを含む）をほぼ元の形式にデコードし，再構築する．
- バイトコードを逆アセンブルし，smali 形式のファイル群を生成する．

この apktool を用いずに apk ファイルを UNZIP して展開することも可能だが，図 6 のように XML ファイルがバイナリデータになってしまい，図 7 のようなコード表記の形に戻らないためアプリの解析が難しくなってしまう．対象アプリの機能の解析には AndroidManifest.xml という XML ファイルも使用するため，apktool を用いてデコンパイルしたほうがいい．

ⓒ2015 Information Processing Society of Japan

```
AndroidManifest.xml
1  0300 0800 1007 0000 0100 1c00 8803 0000
2  1b00 0000 0000 0000 0000 0000 8800 0000
3  0000 0000 0000 0000 1a00 0000 3400 0000
4  5200 0000 7600 0000 9000 0000 9c00 0000
5  aa00 0000 b800 0000 d000 0000 dc00 0000
6  ee00 0000 4601 0000 4a01 0000 5c01 0000
7  7001 0000 9401 0000 9e01 0000 b201 0000
8  cc01 0000 e001 0000 fe01 0000 1c02 0000
9  2c02 0000 6402 0000 7802 0000 bc02 0000
10 0b00 7600 6500 7200 7300 6900 6f00 6e00
11 4300 6f00 6400 6500 0000 0b00 7600 6500
12 7200 7300 6900 6f00 6e00 4e00 6100 6d00
13 6500 0000 0d00 6d00 6900 6e00 5300 6400
14 6b00 5600 6500 7200 7300 6900 6f00 6e00
15 0000 1000 7400 6100 7200 6700 6500 7400
```

図 6 apktool を使用せずに展開した結果

```
AndroidManifest.xml
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest android:versionCode="1" android:versionName="1.0" package="com.
   exam.testapp"
3    xmlns:android="http://schemas.android.com/apk/res/android">
4      <application android:theme="@style/AppTheme" android:label="@string/
       app_name" android:icon="@drawable/ic_launcher" android:debuggable="
       true" android:allowBackup="true">
5          <activity android:label="@string/app_name" android:name=".
           MainActivity">
6              <intent-filter>
7                  <action android:name="android.intent.action.MAIN" />
8                  <category android:name="android.intent.category.LAUNCHER"
                   />
9              </intent-filter>
10         </activity>
11         <activity android:label="@string/app_name" android:name="com.exam.
           testapp.CheckerActivity" />
12     </application>
13 </manifest>
```

図 7 apktool を使用したデコンパイル結果

3.4 対象アプリの解析

対象アプリの解析方法としては，静的解析法を用いる．我々は，静的解析による Android パーミッションの可視化方法を提案した坂下らの研究[8]の Java コードの解析方法を参考に，smali ファイルを解析する．

apktool を使うと Java で作成されたアプリのプログラムが smali 形式のファイルとして復元される．smali から Java のソースコードに変換するツールは存在しているが，それを使用せずに解析することによって処理を簡略化することができる．

図 8 は逆アセンブルの実験用に自作したカレンダーアプリのコードである．図 9 はそれが逆アセンブルされて生成された smali ファイルのコードである．

図 8，図 9 を比較してみると，インターフェースは何を使っているのか，基となったソースファイルは何かなどが読み取れる．他にも，図 9 の 21 行目の「# virtual methods」というコードの記述内では MainActivity クラス内の各メソッドの中にある処理内容が記述されているといったことを読み取ることができる．

このことから，smali ファイルのコードは Java で作成したアプリのコードに類似性があるため，smali ファイルを解析することは可能である．

```
MainActivity.java
1  package com.exam.testapp;
2
3  import android.app.Activity;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.widget.CalendarView;
7  import android.widget.CalendarView.OnDateChangeListener;
8  //import android.widget.Toast;
9
10
11 public class MainActivity extends Activity implements OnDateChangeListener{
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17
18         CalendarView calendarView = (CalendarView)findViewById(R.id.calendarView);
19
20         calendarView.setOnDateChangeListener(this);
21     }
22
23
24
25     @Override
26     public void onSelectedDayChange(CalendarView view, int year, int month,
27             int dayOfmonth) {
28         // TODO 自動生成されたメソッド・スタブ
29         //Toast.makeText(this, String.format("%04d/%02d/%02d", year, month + 1, day
30             //  Toast.LENGTH_SHORT).show();
31         Intent intent = new Intent(MainActivity.this,CheckerActivity.class);
32         startActivity(intent);
33     }
34 }
```

図 8 Android アプリのソースコード

```
MainActivity.smali
1  .class public Lcom/exam/testapp/MainActivity;
2  .super Landroid/app/Activity;
3  .source "MainActivity.java"
4
5  # interfaces
6  .implements Landroid/widget/CalendarView$OnDateChangeListener;
7
8
9  # direct methods
10 .method public constructor <init>()V
11     .locals 0
12
13     .prologue
14     .line 11
15     invoke-direct {p0}, Landroid/app/Activity;-><init>()V
16
17     return-void
18 .end method
19
20
21 # virtual methods
22 .method protected onCreate(Landroid/os/Bundle;)V
23     .locals 2
24     .parameter "savedInstanceState"
25
26     .prologue
27     .line 15
28     invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/
       Bundle;)V
29
30     .line 16
31     const/high16 v1, 0x7f03
32
33     invoke-virtual {p0, v1}, Lcom/exam/testapp/MainActivity;->
       setContentView(I)V
```

図 9 逆アセンブルで生成された smali コード

3.5 データベース

アプリを解析した後，データベースを使用して記録する．解析結果を蓄積することで，同じアプリの解析を無くすことができる．また，アプリの解析結果から危険性の種類ごとに区別するため，データベースに記録することでそれらを簡略化することができる．

データベースは，SQL を使ってリレーショナルデータベースを構築する．作成するテーブルは，以下の 3 種類とする．

- アプリの解析結果をまとめたテーブル
- アプリを危険性の種類ごとにまとめたテーブル
- コンテンツカテゴリごとにまとめたテーブル

例として，「アプリ解析結果をまとめたテーブル」にはど

ⓒ2015 Information Processing Society of Japan

のようなフィールドを入れてデータベースを構築するのかを以下に示す.

- ID : 本システムで使用する番号を示す.
- アプリ名 : 解析した対象のアプリの名前を示す.
- アプリを取得したサイトの URL : アプリを取得した場所を示す.
- コンテンツカテゴリの種類 : ゲーム, ビジネスなどといったコンテンツカテゴリを示す.
- 解析日 : アプリを解析した日を示す.
- 使用しているパーミッション : 使用しているパーミッションの名前を示す.
- 危険性の有無 : 解析結果からアプリの危険性の有無を示す.
- 危険性の種類 : 解析結果からアプリの危険性の種類を示す.
- 危険性の内容 : 解析結果からどのような危険性が潜んでいるのかという説明をする.

上記のようなテーブル，フィールドを用いてデータベースを構築し，解析結果を記録し，Web サービスを使って解析結果を表示できるようにする.

3.6 Web サービス

解析結果は Web サービスを通じて誰でも手軽に閲覧できるようにする.

Web サービスでは解析結果を記録したデータベースの情報をユーザに伝える．また，第 3.1 節で述べた自作アプリでもリクエストしたアプリの解析結果を閲覧できるようにするために, インテント機能を使って Web サービスに遷移して解析結果を閲覧できるようにする.

図 10 は，解析結果を閲覧するための Web ページの外観である.

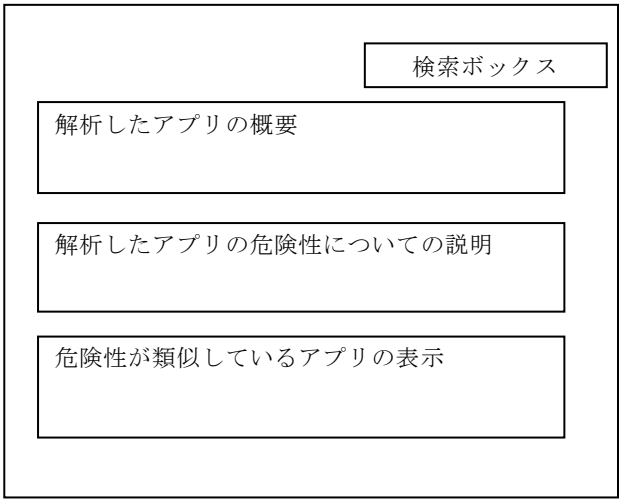

図 10 解析結果の表示方法

検索ボックスでは，アプリの名前を入力し解析結果を知りたいアプリを探すことができる．解析の結果として，解析したアプリの概要の部分にはそれがどのようなアプリなのかを説明する．次に解析したアプリの危険性についての説明を表示し，最後に潜んでいる危険性が類似しているアプリを表示して注意を促す.

スマートフォンやタブレットでも閲覧できるようにするため, レスポンシブな Web デザイン開発ができる Bootstrap を使う. Web ページとデータベースの連携には PHP を使用して行う.

3.7 システムの自動化

アプリの取得，デコンパイル，解析といった行程を手動で行うと，1 つのアプリに対してかかる時間が長くなり，多くの手間もかかってしまう．しかし，この工程を自動で行うことによって解析時間を大幅に短縮させることができ，手間も減らすことができる．よって，本システムでは，第 3.2, 3.3, 3.4 節で述べたシステムにスクリプト言語を使用して自動化する.

スクリプト言語とは，OS やアプリの動作や機能などをプログラムの形で記述できるプログラミング言語のことである.

スクリプト言語には，いくつか種類があり，その中でも有名なスクリプト言語には以下のようなものがある.

- Perl
- PHP
- Python
- JavaScript
- Ruby

表 2 は各スクリプト言語の特徴をまとめたものである．本システムでは，アプリ解析を重視するため, 正規表現を得意としたスクリプト言語である Perl を用いることにした.

表 2 各スクリプト言語の特徴

スクリプト言語	特徴
Perl	正規表現を扱えるので，文字列を扱う機能がすぐれている.
PHP	データベースとの連携が容易にできる.
Python	全ての環境で動作する．事実上すべてのタスクをこなせる.
JavaScript	クライアント・サーバ間の連携を容易にできる.
Ruby	柔軟な文法と強力な拡張性によって，自分にあったスタイルでプログラムを記述できる.

4. 考察

本研究では，アプリの機能，動作やパーミッションを解析し，その解析結果から危険性を割り出し，危険性の種類

ごとに区別するツールを提案する．提案したツールを使うことで，ユーザは対象アプリの危険性をインストールする前に知ることができる．また，類似した危険性が潜んでいるアプリを紹介することによって，どのようなアプリの種類に危険性が潜んでいるのか傾向が見えてくるようになり，ユーザがアプリの危険性を見極めることができるようになる．見極めることができることによって，モバイルマルウェアなどによる被害を減少できるのではないかと考えられる．

提案するツールに類似している製品では，Android 端末にインストールされているアプリケーションにはどのようなパーミッションを使用していて，危険性が潜んでいるのか確認できる tSpyChecker[9]やインストールする前にアプリの危険性を確認できる secroid[10]などがある．しかし，モバイルマルウェアの被害が増えているため，ほとんどのユーザはアプリの解析サービスといったものが存在していることを知らない，または使いづらいという理由から活用していないため，モバイルマルウェアの被害が減らないのではないのかと考える．我々は，ツールの使いやすさ，どのようにしてツールを多くに人に認知してもらえるかという点も重視していきたい．

5. 今後の取り組み

今後は, smali 形式のファイルからアプリの動作や機能を解析する方法を確立し，アプリに潜んでいる危険性の判断方法を明らかにする．また，スクリプト言語を使ったシステムの自動化を確立し，提案システムを開発する．第 4 節の考察で述べたように，ツールをどのようにして使いやすくするのか，どのような方法で多くの人に認知してもらうのかを考え，提案するツールに反映させていく．

そのあと，Android 端末を所持しているユーザに使用させ，評価をしてもらう．また，ツールを使うことで Android アプリに潜んでいる危険性があることを理解し，見極められるようになったかといったツールの有用性の検証も行う．

参考文献

1） 総務省，平成 26 年版情報通信白書，http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h26/html/nc253110.html
2） MM 総研，スマートフォン市場規模の推移と予測，http://www.m2ri.jp/newsreleases/main.php?id=010120131009500
3） 奥富 幸大(2014)，マカフィー株式会社，最新モバイル脅威レポート(日本スマートフォンセキュリティ協会 （JSSEC）技術部会主催「スマートフォンセキュリティアップデート（2014 夏）」発表資料)，https://www.jssec.org/dl/20140801_McAfee_Yukihiro_OKUTOMI.pdf
4） マカフィー株式会社，McAfee Blog，http://blogs.mcafee.jp/mcafeeblog/2014/03/1395.html
5） マカフィー株式会社，McAfee Blog，http://blogs.mcafee.jp/mcafeeblog/2014/03/1397.html
6） 一般社団法人日本スマートフォンセキュリティ協会，Android アプリのセキュア設計・セキュアコーディングガイド(2014).
7） Android -apktool-，https://code.google.com/p/android-apktool/
8） 坂下 卓弥，小形 真平，海谷 治彦，海尻 賢二: 静的解析による Android パーミッションの利用目的の可視化方法，研究報告コンピュータセキュリティ(CSEC)，2013-CSEC-62
9） タオソフトウェア株式会社, Android ソフトウェア - tSpyChecker, http://www.taosoftware.co.jp/android/spychecker/
10） ネットエージェント株式会社, Android アプリの潜在リスクチェックは secroid(セキュロイド), http://secroid.jp/

ⓒ2015 Information Processing Society of Japan