Vol.2011-CSEC-53 No.3
Vol.2011-IOT-13 No.3
2011/5/12

Android パーミッションを悪用する Script の脅威と静的解析

川端秀明[†]　磯原隆将[†]　竹森敬祐[†]　窪田歩[†]

Android OS の特徴として，利便性の高いアプリケーション（以下，アプリ）を実現するパーミッションという機構があり，アプリケーションのインストール時にユーザが承認することで，端末の情報や機能へのアクセス権を制御している．また, アプリの可用性の向上のために web 機能をアプリに内包する webkit を搭載している．これを用いることで，Android アプリと HTML,CSS,JavaScript など Web アプリとを柔軟に連携できる．しかし，webkit を利用したアプリが，外部サーバから JavaScript を受け取り実行した場合，アプリに与えられたパーミッションの範囲で実行される脅威がある．要するに，アプリ単体では不正な動作をしないが，後から送り込まれた悪意の JavaScript によって端末を操作されてしまう．そこで本研究では，アプリの静的解析により得られるコードの特徴から，後から送り込まれる JavaScript の機能を把握し，潜在的な脅威を推定する手法を提案する．これはアプリの実行コードの逆コンパイルによるコード解析であり，外部サーバの JavaScript から呼び出されるメソッドを特定することで, 情報漏洩や端末の不正操作を推定する．

Threat of Script abuse Android Permissions and Static Analysis

Hideaki Kawabata[†] Takamasa Isohara[†]
Keisuke Takemori[†] and Ayumu Kubota[†]

The access permission framework is designed in the Android OS in order to develop useful applications. When the user confirms the access permissions, the application can access confirmed information and/or functions. In addition, the webkit in the Android OS provides a web rendering engine to the application. The Android application using the webkit can execute web applications, e.g., HTML, CSS, JavaScript. When the application using webkit receives and executes the JavaScript, the access permissions are delegated to the JavaScript that can access functions and/or information in the Android phone. Then, threats of the JavaScript should be evaluated. In this paper, we proposed code analysis technique that extracts potential threats from the web rendering application. The information leakage and/or misuse functions are detected, when malicious methods for the JavaScript are described in the Android application.

1. はじめに

スマートフォンの中でも，端末で管理される情報や機能を利用するための API が豊富に用意されている Android OS が注目を集めている[1]．スマートフォンは，誰もが自由にアプリを開発して公開でき，それをユーザが自由にインストールできるプラットフォームであり，様々な機能を柔軟かつ容易に拡張することができる．

スマートフォンと PC を比較した場合，スマートフォンは，利用者との結びつきが PC よりも緊密な機器であり，氏名，電話番号，メールアドレス，住所録，メールの送受信履歴，インターネットの閲覧履歴等の個人情報が集積している．また，端末の高機能化と，常に携帯するその利用手段から，従来の PC では収集されなかった位置情報等の情報も容易に取得できる．このような中，アプリの中には，OS の脆弱性を突いて管理者権限を奪うものや, 端末に格納された個人情報を収集して外部に送信する，悪性アプリが存在する．特に，正規のアプリを装いつつ，不正な振る舞いを行う機能が組み込まれたトロイの木馬が現れ，感染が拡がっている[2].

Android の特徴として，安全性と利便性のトレードオフをユーザに委ねるパーミッション機構[3]や，アプリの可用性の向上のために Web 機能をアプリに内包する Webkit[4]の搭載が挙げられる．

パーミッション機構とは，端末内の情報や機能へのアクセス権をアプリに承認を与えるものである．具体的には，アプリ開発者がマニフェストファイルにそのアプリで使用する機能をパーミッションとして定義しておき，アプリがインストールされる時に，Android OS からユーザへ使用許可を求める．しかしながら，スマートフォンに慣れていないユーザにとって，どのパーミッションがどのような危険性を持っているのか理解することは難しい．したがって，アプリが容易に入手可能な環境において個人情報を悪用される脅威から，スマートフォンの利用者を保護する必要がある．

次に HTML のレンダリングエンジンである webkit とは，アプリ開発者が webkit を利用することでブラウザ機能を簡単にアプリに登載できる機能である. Webkit を利用したアプリと，サーバ側で作成した HTML,CSS,JavaScript などの Web アプリを連携させた開発も盛んになってきている[5]．Web アプリと柔軟な連携を実現するために，アプリは指定した外部サーバに置かれた JavaScript を実行することができる．更に，この JavaScript はアプリに定義されたメソッドを呼び出し，メソッドに記述された API を実行することができる．したがって，外部サーバに公開されている JavaScript からアプリの宣言しているパーミッションの範囲で端末の個人情報を取得する権限，端末の操作権限を有する API を実行させることができる．この仕様を活用することで幅の広いアプリの開発が可能であり，Android の魅力の一つであるが，悪用された場合は

[†] 株式会社 KDDI 研究所 ネットワークセキュリティグループ
KDDI R&D Laboratories Inc. Network Security Laboratory

ⓒ2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.3
Vol.2011-IOT-13 No.3
2011/5/12

大きな脅威となる．例えば，アプリに個人情報を取得するメソッドを定義し，外部サーバの JavaScript を後から送り込み該当のメソッドを実行することで，JavaScript によって個人情報を漏洩する Android アプリを実現できる．

そこで本稿では, JavaScript を利用したアプリに潜在する脅威を推定する手法を提案する．これはアプリの実行コードを逆コンパイルして得られるコードから，外部サーバの JavaScript から呼び出されるメソッドを特定し，そのメソッド内に情報漏洩に関するコード，端末の不正操作に関するコードを検出することで，アプリに潜在する脅威を推定する手法である．そして，Android 端末にインストールされるアプリの逆コンパイルに要する時間を測定し，数秒で処理を完了できることから，Android 端末向けマルウェア対策アプリとして実装できる目処を示す．

2. アプリの実装手法とその課題

2.1 アプリと JavaScript の連携

Android OS では webkit の WebView クラスを利用することで，アプリと外部サーバに公開されている JavaScript を連携させることができる．外部サーバの JavaScript が Android アプリのメソッドを呼び出すことができるが，Android アプリにどのようなメソッドが JavaScript から実行できるのか定義しなくてはならない．つまり，ある webkit を利用したアプリ開発者以外の人が，何らかの手段でアプリに JavaScript を送り込み不正行為をするという攻撃は成立しない．以下，2.1.1 で Android 端末向けアプリの実装方法について，2.1.2 で Web 側の JavaScript の実装方法について説明する．

2.1.1 Android 端末向けアプリ側の実装

Android アプリで外部サーバの JavaScript と連携させるプログラムの例を図 1, 2 に示す．1 行目は，WebView のインスタンスを生成している．2 行目は，JavaScript 利用の有無を設定している．3,4 行目は，外部サーバの JavaScript から呼び出すことができるオブジェクトの登録をしている．外部サーバの JavaScript から実行させたいメソッドを記述したクラスを作成し(図 2)，そのインスタンスを 4 行目で登録している．5 行目は，実行させたい外部サーバの URL の設定となる．指定した外部サーバの JavaScript を実行させたい場合は，ここで連携させたい JavaScript の URL を記述する．5 行目が実行された時，アプリは外部サーバの JavaScript をダウンロードし端末内で JavaScript を実行する．

```
1:  WebView wv = new WebView(this);
2:  wv.getSettings().setJavaScriptEnabled(true);
3:  JavascriptObject jo = new JavascriptObject(this);
4:  wv.addJavascriptInterface(jo, "android");
5:  wv.loadUrl("http://www.akui-javascript.com");
```

図 1　JavaScript を利用するアプリのプログラム例

```
6:  public class JavascriptObject {
7:      public string method_name {
8:          //外部サーバから実行させたい処理を記述
9:      }
10: }
```

図 2　外部サーバの JavaScript から実行できるメソッドの作成

2.1.2 Web 側 JavaScript の実装

外部サーバに公開する JavaScript から，2.1.1 で実装したアプリ内のメソッドを呼び出すには，図 1 の 4 行目の第 2 引数と登録したオブジェクトのメソッド名を記述することで，JavaScript からアプリ内のメソッドを実行することができる．図 1,2 の例では，android.method_name と記述することで JavaScript から図 2 のメソッドを実行することができる．メソッドを実行して得た値は，JavaScript の機能を使用することができる．したがって，JavaScript でユーザから不可視である HTML の form を作成し，アプリ内メソッドを実行して得た値を自動で post 送信するプログラムを作成することで，情報漏洩アプリを作成することができる．

2.2 従来研究

これまで，Android 向けアプリを評価する方式として，Logcat と呼ばれるアプリケーションを実行した際に出力されるログ情報に着目した動的解析手法が提案されてきた[6][7]．これは，正規表現を用いたシグネチャを作成し，Logcat ログに適用することで不正検知を行う手法であり，情報漏洩型と攻撃型マルウェアを検知できることを示している．

また，アプリを逆コンパイルし得られたコード情報を元にアプリの挙動を解析する静的解析手法がある．文献[8]の記事では，個人情報を取得するコード，取得した情報を外部サーバに送信するコードを発見し，アプリに潜在する脅威を判定している．

ⓒ2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.3
Vol.2011-IOT-13 No.3
2011/5/12

2.3 課題

外部サーバの JavaScript を利用した悪性アプリを検知する場合，動的解析・静的解析の手法において，それぞれ課題がある．

動的解析における課題は，悪性アプリの挙動が外部サーバから取得した JavaScript のコード内容によって変化するため，攻撃者が外部サーバの JavaScript を変更するだけでアプリの挙動を変化させることができる点である．すなわち，ある時点でアプリの動的解析を行うために取得した外部サーバの JavaScript に悪性コードが含まれない場合，悪性の挙動を見逃してしまう．

静的解析における課題は，外部サーバの JavaScript に，個人情報を取得するメソッドを実行するコード，個人情報を外部サーバに送信するコードが記述されているため，アプリのコードに個人情報漏えいに関する全コードが存在しない点である．すなわち，検証対象のアプリ内で個人情報を取得するコードと外部サーバに送信するコードの双方が含まれていない限り，悪性アプリを検知することができない．

3. 提案方式

前述の課題を解決するため，本稿では，静的解析による脅威の推定手法を提案する．これはアプリの実行コードを逆コンパイルして得られるコード情報を解析する手法である．外部サーバから呼び出されるメソッドを特定し，そのメソッド内に個人情報漏洩に関するコード，端末を不正操作するコード，などを検知することでアプリに潜在する脅威を推定する手法である．図 3 に実行コードの逆コンパイルによって外部サーバの JavaScript を利用した悪性アプリの脅威の推定手法の手順を示す．以下，順を追って説明する．

手順1： 逆コンパイルの実行

sample.apk は，脅威推定の対象となるアプリである．アプリは，実行コードである class.dex ファイル，アプリが利用するテキストファイルや画像ファイル等の各種リソースファイルなど一式を圧縮して保持する．

はじめに，この圧縮形式のファイルから，実行コードを抽出する．続いて，得られた実行コードを逆コンパイルし，その結果であるソースコードを得る．ソースコードは，クラスごとのファイルに分かれている．逆コンパイルを行う代表的なツールとしては，Dedexer[9], android-apktool[10] がある．以下，本稿では Dedexer を使用して逆コンパイルを行うものとする．（図 3 中，.ddx と書かれているファイルが各クラスである．）

手順2： WebView 利用の判定

アプリが WebView を利用するかの判定を行う．逆コンパイルして得られたソースコード全てに対して Webview のインスタンスを生成するコード

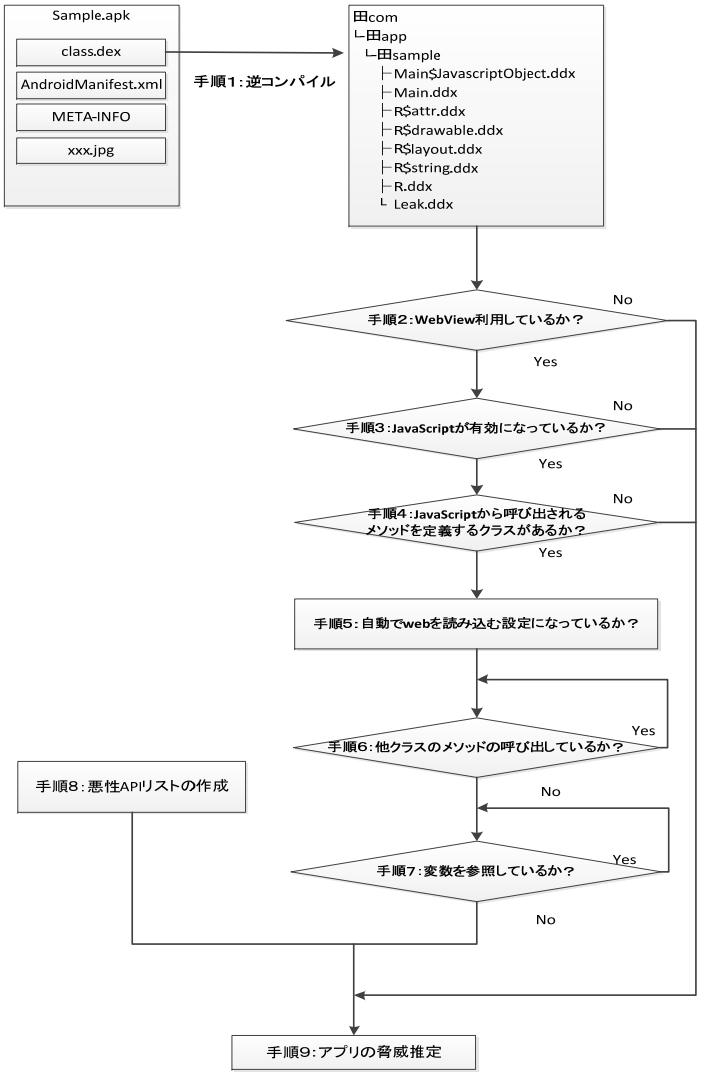

図 3 JavaScript を利用した Android アプリの脅威の推定手法

ⓒ2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.3
Vol.2011-IOT-13 No.3
2011/5/12

(android/webkit/WebView)が存在するか判定する．コードが存在する場合は手順 3 へ進み，コードが存在しなかった場合はアプリに潜在する脅威はないと判断し解析を終了する．図 4 に，WebView のインスタンスを生成するコードの例を示す.

```
1:    new-instance   v1,android/webkit/WebView
2:    invoke-direct   {v1,v4},android/webkit/WebView/<init>      ;
      <init>(Landroid/content/Context;)V
```

図 4 WebView インスタンスの生成のコードの例

手順3： JavaScript 利用の判定

アプリが JavaScript を実行できるのか判定を行う．逆コンパイルして得られたソースコード全てに対して JavaScript の利用可否を決定するメソッド名(setJavaScriptEnabled)が存在するか判定する．メソッド名が存在する場合は，メソッド名に入る第 2 引数を特定する．その後，第 2 引数を生成している箇所に遡り，代入される値を確認する．値が 0 であった場合は，JavaScript の利用が無効であり，1 であった場合は JavaScript の利用が有効である．有効になっていた場合は手順 4 へ進み，無効だった場合や利用可否を決定するメソッド名が存在しない場合はアプリに潜在する脅威はないと判断し解析を終了する．図 5 に JavaScript を利用するメソッド名の例を示す.

```
1:    const/4          v3,1     // v3=0 JavaScript 無効 v3=1 JavaScript 有効
2:    invoke-virtual   {v2,v3},android/webkit/WebSettings/setJavaScriptEnabled
      ; setJavaScriptEnabled(Z)V
```

図 5　JavaScript を利用するコード

手順4： JavaScript から呼び出だされるメソッドを定義するクラスの判定

外部サーバの JavaScript が，アプリに記述されたメソッドを呼び出し，実行可能か判定を行う．そのため，JavaScript から呼び出されるメソッドを定義するクラスが存在するか判定を行う．逆コンパイルして得られたソースコード全てに対して JavaScript から呼び出されるオブジェクトを設定するメソッド名（addJavscriptInterface）が存在するか確認する.メソッド名が存在した場合,メソッド名に入る第 2 引数を特定する．その後，第 2 引数を生成している箇所に遡り，クラス名を特定し，手順 5 に進む．JavaScript から呼び出されるクラスが無かった場合,アプリに潜在する脅威はないと判断し解析を終了する．図 6 に JavaScript から呼び出されるクラスを設定するコードの例を示す.

```
1:    new-instance   v0,com/app/sample/Main$JavascriptObject
2:        invoke-direct          {v0,v4,v4},com/app/sample/Main$JsObj/<init>
      ; <init>(Lcom/app/sample/Main;Landroid/content/Context;)V
3:    const-string     v2,"android"
4:    invoke-virtual   {v1,v0,v2},android/webkit/WebView/addJavascriptInterface
      ; addJavascriptInterface(Ljava/lang/Object;Ljava/lang/String;)V
```

図 6　JavaScript から呼び出されるクラスを設定するコード

手順5： web の自動読み込み設定の判定

アプリが自動で JavaScript を読み込み実行するのか判定を行う．もし，自動で読み込む設定にされていた場合，アプリを起動すると同時に JavaScript が実行される可能性があるため脅威に晒される確率が増す．逆コンパイルして得られたソースコード全てに対して外部サーバを読み込むメソッド名(loadUrl)が存在するか確認する．メソッド名が存在した場合，どこの web に接続するのか確認するため，メソッド名の第 2 引数を特定する.その後,第 2 引数を生成し代入される URL を確認し,手順 6 へと進む．図 7 に，外部サーバを読み込むコードの例を示す.

```
1:    const-string     v2,"http://www.akusei-javascript.com"
2:    invoke-virtual   {v1,v2},android/webkit/WebView/loadUrl    ;
      loadUrl(Ljava/lang/String;)V
```

図 7 外部サーバを読み込むコード

手順6： 他クラスのメソッド呼び出しの判定

手順 4 で外部サーバの JavaScript から実行可能なメソッドを定義するクラスを特定したが，特定したメソッド内で更に他のクラスのメソッドを呼び出すことが考えられる．そのため，外部サーバの JavaScript から実行可能なメソッドに加えて，そのメソッドから更に呼び出される他のクラスのメソッドも特定しなくてはならない．これらを判定するため，他のクラスを呼び出しているコード（new-instance）とメソッドを実行するコード(invoke-virtual)を確認する．図 8 では，Leak クラスのインスタンスを生成し，start-leak という String 型のメソッドが実行されている．同様に，Leak クラスの start-leak メソッド内に別のクラスに定義されたメソッドを実行していないか確認を行う．すべての判定が終了したならば，手順 7 へ進む.

ⓒ2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.3
Vol.2011-IOT-13 No.3
2011/5/12

```
1:    new-instance   v0,com/app/sample/Leak
2:    invoke-virtual {v0},com/app/sample/Leak/start-leak      ;
      start-leak()Ljava/lang/String
```

図 8　他のクラスを呼び出しているメソッドを実行するコード

手順7：　変数の参照の判定

手順 4,6 で特定したクラスのメソッド内で，変数の参照の有無を確認する．手順 4,6 で特定したメソッド内で変数を参照するコード(iget-object + Ljava/lang/)を確認し，呼び出されるクラスと変数名を特定する．図 9 の例では，Main というクラスに定義された variable という String 型の変数が参照されている．もし変数の参照が確認されたならば，特定したクラスで変数に代入しているコード(iput-object)を確認し，変数に値が入力されるメソッドを特定する．図 10 では，変数 variable に第 1 引数である v1 の値が入力され，v1 には getLine1Number を実行した値が入力されている．このように，変数の参照が確認されたならば，その変数に値が入力されるメソッドを特定する．すべての判定が終了したら，手順 8 へ進む.

```
1:    iget-object      v2,v1,com/app/sample/Main.variable Ljava/lang/String;
```

図 9 変数を参照するコード

```
1:    invoke-virtual  {v0},android/telephony/TelephonyManager/getLine1Number
      ; getLine1Number()Ljava/lang/String;
2:    move-result-object          v1
3:    iput-object     v1,v2,com/app/sample/Main.variable Ljava/lang/String;
```

図 10 変数に値を入力するコード

手順8：　悪性 API リストの作成

手順 6~8 で判定した外部サーバから実行することができるメソッド内に脅威となるコードがあるか判定するため, 予め悪性 API リストを作成する. 悪性 API リストとは，その API が実行された時の脅威の種別と危険度が記述された表である．例えば，端末の電話番号を取得する getLine1Number などが挙げられる. 表 1 に悪性 API リストの例を示す．ここで危険度は，Low，Medium，High の 3 段階で表し，危険性が高くなるほど，アプリ実行ユーザへのリスクが高まることを示している．

表 1 悪性 API リスト

悪性 API	脅威の種別	危険度
getDeviceID	端末識別番号の取得	Medium
getLine1Number	端末電話番号の取得	High
getSimSerialNumber	SIM の固有番号の取得	Medium
getDeviceSoftwareVersion	Android OS のバージョン情報の取得	Low
getAccounts()	Google アカウントの情報取得	High

手順9：　潜在脅威の推定

手順 4,で特定した外部サーバの JavaScript から実行されるメソッド内，手順 6 で特定した外部サーバの JavaScript から実行されるメソッドから更に呼ばれるメソッド内，手順 7 で特定した変数に値を代入するメソッド内と手順 7 で作成した悪性 API リストを照合し，アプリに潜在する脅威を推定する．抽出された悪性 API リストの一覧と手順 5 で判定した web 読み込みによる自動実行の有無を提示し，ユーザに対してアプリに潜在する脅威を伝える．図 11 に，抽出したクラスのメソッド内で実行されている悪性 API コードの例を示す.

```
1:    invoke-virtual  {v1},android/telephony/TelephonyManager/getDeviceId
      ; getDeviceId()Ljava/lang/String;
2:    invoke-virtual  {v1},android/telephony/TelephonyManager/getLine1Number
      ; getLine1Number()Ljava/lang/String;
3:    invoke-virtual  {v1},android/telephony/TelephonyManager/getSubscriberId
      ; getSubscriberId()Ljava/lang/String;
```

図 11 特定したメソッド内で悪性 API が実行されているコード

ⓒ2011 Information Processing Society of Japan

Vol.2011-CSEC-53 No.3
Vol.2011-IOT-13 No.3
2011/5/12

4. 評価

Android 端末では，CPU やメモリといったリソースが限られているため，3 章で提案したアプリの潜在脅威を推定する手法を Android 端末アプリとして実装できるか評価を行う．具体的には，Android 端末にインストールしたアプリを逆コンパイルするアプリを試作し，逆コンパイルにかかる時間を測定した．以下に，測定環境，結果，考察を示す．

4.1 逆コンパイル時間の測定

インストールしたアプリを逆コンパイルするアプリを試作した．逆コンパイルには，パブリックドメインである Dedexer を用いた．逆コンパイルするアプリは，AndroidSDK API Level 2.1[11] に含まれているオープンソースのサンプルアプリの中で実行コード class.dex のファイルサイズが一番小さい MultiRes と一番大きい ApiDemos を用いて測定を行った．表 3 に測定環境，表 4 に測定結果を示す．

表 3　逆コンパイル時間の測定環境

測定端末	IS03
Android OS	2.1-update1
CPU	1GHz
逆コンパイルツール	Dedexer

表 4 逆コンパイル時間の測定結果

アプリ名	class.dex ファイルサイズ [KByte]	処理時間 [ms]					平均処理時間[ms]
		1 回目	2 回目	3 回目	4 回目	5 回目	
MultiRes	5	650	660	690	1140	1020	832
ApiDemos	528	2640	3040	2680	2800	2680	2768

4.2 考察

逆コンパイル時間を測定した結果，コード量の多い ApiDemos アプリで平均して 3 秒以内に逆コンパイルが終了することが判明した．3 章で提案した推定手法を実装するためには，その後，脅威の推定に関する処理とそれらの結果をデーターベースに保存する処理が必要となる．逆コンパイルに関して，3 秒以下で実現できるという結果から，その後の処理も最適化していくことで，Android 端末のマルウェア対策アプリとして適応できうるものと推測する．

5. おわりに

本稿では，Android 端末向けのアプリにおいて，JavaScript を利用するアプリに潜在する脅威を推定する手法を提案した． これは，アプリを逆コンパイルして得られるコードに対する静的解析であり，外部サーバの JavaScript から呼び出されるメソッドを特定し，そのメソッド内に情報漏洩に関するコード，端末を不正操作するコードを検知することで，アプリに潜在する脅威を推定する．

Android 端末向けマルウェア対策アプリとして実装させるため，逆コンパイルに関する評価を行った．評価の結果，逆コンパイル時間に関して，コード量の多いアプリでも 3 秒以内で逆コンパイルが終了することを示し，提案方式を Android 端末向けマルウェア対策アプリとして実装できる目処を得た．今後は，逆コンパイル以降の処理もアプリとして実装し，悪性アプリの脅威を推定する．

尚，JavaScript を利用したアプリが悪性ということはなく，実装方法次第では様々な魅力あるアプリを開発できる可能性を秘めている．アプリ開発者は，ユーザが安心して利用出来るアプリを提供できるよう，不要なパーミッションは宣言せずに，取得した重要な情報はアプリの動作以外で決して使用しないアプリ開発を心がけていただきたい．

参考文献

[1] Android, http://www.android.com/jp/
[2] Lookout Blog, “Security Alert: DroidDream Malware Found in Official Android Market” http://blog.mylookout.com/2011/03/security-alert-malware-found-in-official-android-market-droiddream/
[3] Access permissions, http://developer.android.com/reference/android/Manifest.permission.html
[4] Android Webkit, http://developer.android.com/reference/android/webkit/package-summary.html
[5] Jonathan Stark: Android アプリケーション開発ガイド HTML+CSS+JavaScript による開発手法, オライリー・ジャパン(2011).
[6] 磯原隆将，竹森敬祐，窪田歩，高野智秋, “Android 向けアプリケーションの挙動に注目したマルウェア検知”, IEICE, SCIS2011, 3B3-2, 2011 年 1 月.
[7] 竹森敬祐，磯原隆将，窪田歩，高野智秋, “Android 携帯電話上での情報漏洩検知”, IEICE, SICS2011, 3B3-32011 年 1 月.
[8] Donato Ferrante, “Dissecting Android Malware”, http://www.inreverse.net/?p=1272
[9] Dedexer, http://dedexer.sourceforge.net/
[10] android-apktool, http://code.google.com/p/android-apktool/
[11] Android SDK, http://developer.android.com/sdk/index.html

ⓒ2011 Information Processing Society of Japan