情報処理学会論文誌 Vol.56 No.2 720–732 (Feb. 2015)

推薦論文

Androidにおける実行中のクラスに基づくパーミッション制御手法

日置 将太[1,a)] 高瀬 拓歩[1] 齋藤 彰一[1] 毛利 公一[2] 松尾 啓志[1]

受付日 2014年3月10日, 採録日 2014年11月10日

概要：Android では端末の機能や情報へのアクセスをパーミッションという単位で制御するが，パーミッションはアプリ単位で与えられるため，広告ライブラリに代表される第三者の作成した外部ライブラリがホストアプリに与えられたパーミッションを使用できるという問題がある．そのため，第三者の作成したライブラリが，ホストアプリ用のパーミッションを利用して障害を引き起こす可能性がある．本稿では，アプリ開発者が実行中のクラスに基づいてパーミッションを制御するシステムを提案する．提案システムを導入することで，ユーザの判断を求めることなしにホストアプリと外部ライブラリのパーミッションを分離し，それぞれ最小のパーミッションで動作させることが可能となる．AndroidOS に対して提案システムを実装し，評価によって提案システムの導入によるパフォーマンスへの影響が小さく，アプリ開発者への負担も小さいことを確認した．

キーワード：Android，パーミッション制御，権限分離，第三者ライブラリ，広告ライブラリ

A Permission Control System Based on an Executing Class in Android

Shota Hioki[1,a)] Takuho Takase[1] Shoichi Saito[1] Koichi Mouri[2] Hiroshi Matsuo[1]

Received: March 10, 2014, Accepted: November 10, 2014

Abstract: An android permission system controls a privilege to access functions and information in terminals. However, the system gives the whole of the application permissions, so that it has a problem that third party libraries, as exemplified in an advertisement library, can use permissions which are only allowed the host applications. Therefore third party libraries can abuse them and cause failures. We propose a novel system that controls permissions without user decision based on an executing class. The proposed system can separate permissions for host applications and third party libraries, and allow them the least permissions. We implemented the proposed system and confirmed that a performance impact is small and developers have a light workload.

Keywords: Android, permission control, privilege separation, third party library, advertisement library

1. はじめに

Android [1] のアプリケーション（以下，アプリという）の開発において，アプリ開発者は Android に標準搭載された API を利用できるほか，第三者の作成したライブラリを自らのアプリ（以下，ホストアプリという）に組み込むことができる．第三者の作成したライブラリ（以下，外部ライブラリという）はさまざまな用途向けに多数存在する．その中でも AdMob [2] に代表される広告ライブラリは，ホストアプリ内で広告を表示することで無料のアプリであっ

[1] 名古屋工業大学
Nagoya Institute of Technology, Nagoya, Aichi 466–8555, Japan
[2] 立命館大学
Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[a)] android-sec@mail.ssn.nitech.ac.jp

本論文の内容は 2013 年 7 月の CSEC62 研究発表会にて報告され，同研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

ても収入を得られる仕組みを提供し，多くのアプリに組み込まれている．しかし，文献 [3] によると，ユーザに無断でユーザの情報を収集する広告ライブラリが存在することが報告されている．また，文献 [4] によると，広告ライブラリにセキュリティリスクが存在することが報告されている．このように，広告ライブラリには，ホストアプリ開発者がその安全性を完全に保障できないという問題がある．

Android ではアプリに対するセキュリティのためにパーミッション機構を搭載している．これは端末の情報へのアクセス権や機能の利用権限をパーミッションという単位で扱い，アプリの動作に必要となるパーミッションをユーザに示すことでユーザにインストールの可否を仰ぐ機構である．しかし，パーミッション機構には，パーミッションを与える粒度が粗いという問題がある．パーミッション機構ではアプリ単位でパーミッションを制御するため，外部ライブラリを組み込んだアプリの場合にはホストアプリと外部ライブラリに許可されるパーミッションは区別されない．そのため，外部ライブラリは自身が要求したパーミッションのほかにホストアプリが要求したパーミッションも利用可能となり，ホストアプリのパーミッションを利用して情報を漏洩できるという問題を発生させている．外部ライブラリの一種である広告ライブラリでも，これを悪用して情報の収集などを行っている事例が報告されている [5]．しかし，アプリ開発者は外部ライブラリの詳細を把握しているわけではないため，導入時の障害の有無を判別することは困難である．

広告ライブラリが正当に与えられたパーミッションのみを使用できるようにする研究として，ホストアプリと広告ライブラリを別のプロセスで動作させる研究がある．AdSplit [6] は，アプリを再コンパイルすることでホストアプリと広告ライブラリのパーミッションとプロセスを分割する．AFrame [7] では，マニフェストファイルとソースコードを変更することで，ホストアプリと広告ライブラリを別のプロセスで動作させる．これらの研究ではホストアプリ開発者がホストアプリを変更することで，ホストアプリと広告ライブラリのプロセスを分離し，強固なパーミッション分離を実現している．しかし，これらの研究には問題点が 2 点ある．1 点目に，広告ライブラリのプロセスだけがユーザに停止されることで，Android のビジネスモデルが脅かされる可能性がある．2 点目に，広告ライブラリの導入方法の一種であるレイアウトファイルに記述して広告ライブラリを動作させる方法に，AdSplit と AFrame は対応できない．

また，パーミッションの利用を動的に禁止することで，広告ライブラリのパーミッションの悪用を制御する研究 [8] がある．この研究では，パーミッション単位のほかに API 単位での権限制御機構と，アプリインストール後にユーザがパーミッションの許可および拒否を変更できる機構を提供している．さらに，スタックトレースを用いることで API 呼び出し元のクラス名を取得し，クラス名にホストアプリのパッケージ名が含まれているかどうかでホストアプリか外部ライブラリかを判断してユーザに通知する．ユーザは，この通知に基づいて外部ライブラリが利用しようとするパーミッションを拒否することで，外部ライブラリに対するセキュリティの向上が得られるとしている．この手法は，パーミッションの利用可否についてユーザに積極的に判断させる方式といえる．この手法の問題点は，1 点目としてユーザはアプリに関する十分な知識を持っていない場合が多く，アプリ実行時にパーミッションと API の利用について許可および拒否を判断することは困難であると考えられる．適切にパーミッションを判断できるユーザが少ないことは文献 [9] でも報告されており，パーミッション制御が正常に機能しない可能性がある．2 点目は，アプリは難読化されることがあり，この場合にはクラス名が無意味な文字列に変換されることから，ホストアプリと外部ライブラリの区別をつけることが困難である．3 点目として，ユーザによって個別にパーミッションや API が制限されることで，たとえば広告ライブラリの動作をすべて禁止するといったユーザが出現した場合，Android のビジネスモデルが機能しない可能性がある．

以上により，広告ライブラリによるパーミッションの悪用を防ぎつつ，広告ライブラリを正常に実行するために必要な要件（以下，広告ライブラリ実行要件という）を以下の 4 点に定める．

- Android の広告ビジネスモデルを維持する．
- 任意の広告ライブラリを利用できる．
- ユーザによる判断を必要最低限とする．
- ソースコードが難読化されたアプリの保護ができる．

本稿では，ホストアプリと広告ライブラリを別プロセスに分離せず，アプリ内でパーミッションのみを分離し，実行中のクラスに基づいてパーミッションを制御するシステムを提案する．提案システムでは，ホストアプリ開発者がクラスごとのパーミッションを指定する．これにより，アプリ全体に対するパーミッションを廃止し，パーミッションはクラスごとに与えられ，広告ライブラリとホストアプリのパーミッションの分離が可能である．なお，提案システムはカーネル制御のパーミッションには対応していないため，一部の広告ライブラリに対しては適用することができず，今後の課題である．

提案システムにより，広告ライブラリによるホストアプリ用のパーミッションの悪用を，ユーザが判断することなしに防止することができる．さらに，アプリ開発者がクラス単位のパーミッション設定をマニフェストファイルに記述するのみで利用することができ，任意の広告ライブラリを利用可能である．難読化に対応したツールを併用することによって，アプリの難読化によるソースコードの保護も

可能とする．また，広告ライブラリがユーザに停止されることも発生しないため，Android のビジネスモデルに対する影響はない．

なお，提案システムの主な対象は広告ライブラリであるが，外部ライブラリ全般に対して適用可能であるため，以降では対象を外部ライブラリ一般とする．

以下，2 章で Android のパーミッション機構とその動作について述べる．3 章と 4 章で提案とその実装について述べ，5 章で評価，6 章で考察を述べる．7 章で関連研究を述べ，最後に 8 章でまとめる．

2. Android のパーミッション機構

本章では提案システムの基盤となる Android のパーミッション機構について概要と動作ついて述べる．

2.1 概要

Android ではすべてのアプリが異なるプロセスとしてサンドボックス内で実行される．そのため，アプリは基本的に他のプロセスや外部の機能と情報へアクセスすることができない．

アプリがプロセス外部の機能と情報へアクセスするための権限を管理する機構として，Android はパーミッション機構を備えている．パーミッションが必要な機能と情報は，ネットワークアクセス，電話番号情報，端末識別子情報や電話帳情報などがある．パーミッションが必要な機能と情報をアプリが利用するためには，アプリ開発者があらかじめマニフェストファイルに必要なパーミッションを記述する必要がある．なお，外部ライブラリに必要なパーミッションに関してはマニュアルなどに記載されており，アプリ開発者はこの情報をもとにパーミッションを追記する．

マニフェストファイルに記述されたパーミッションは，アプリインストール時にユーザに提示される．ユーザは提示されたパーミッションをすべて承認してインストールするか，インストールしないかを選択する．アプリ実行時にはパーミッションが承認されているかどうか検証が行われ，アプリの動作を制御する．

パーミッション機構は，アプリケーション・フレームワーク制御とカーネル制御の 2 種類の制御機構を持っており，API によっていずれかで検証される．なお，1 つのパーミッションは複数の API と関連付けられている．アプリケーション・フレームワーク制御のパーミッションはそのパーミッション名で管理される．一方，一部のパーミッションが持つカーネル制御のパーミッションは Linux の GID で管理され，アプリケーション・フレームワーク制御パーミッションで特定のパーミッションが設定されている際にアプリのプロセスに与えられる．たとえば INTERNET パーミッションであれば “inet” の GID，Bluetooth を利用するための BLUETOOTH パーミッションであれば “net_bt” の GID が当該プロセスに与えられる．

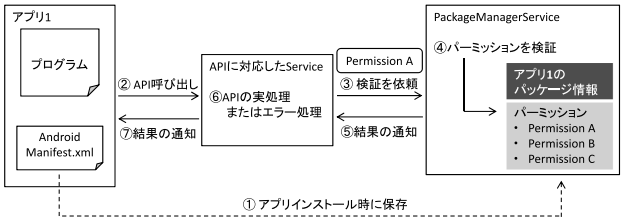

図 1 API 呼び出し時のパーミッション機構の動作
Fig. 1 Process follow of Android permission mechanism.

2.2 動作

アプリケーション・フレームワーク制御のパーミッション機構の動作を**図 1** に示す．まず，アプリが端末にインストールされるとき，マニフェストファイルに書かれた情報が PackageManagerService 上にパッケージごとに保存される（図 1 の①）．アプリが API を呼び出すと，プロセス間通信を用いて対応するサービスに処理が移る（図 1 の②）．当該 API の実行にパーミッションが必要な場合は，対応するサービスは PacakgeManagerService に対してパーミッションの検証を依頼する（図 1 の③）．このとき，検証対象となるパーミッションの名前を PackageManagerService に渡す．PackageManagerService では，API を実行したアプリのパッケージの情報を参照し，検証対象のパーミッションを含んでいるか否かを検証し（図 1 の④），結果を返す（図 1 の⑤）．検証結果を受け取った API に対応するサービスでは，検証が成功していれば正常に処理を行い，失敗していればセキュリティエラーとして処理を失敗させ（図 1 の⑥），結果を通知する（図 1 の⑦）．

カーネル制御のパーミッション機構では，パーミッションを必要とする関数が実行された際に，API を実行したアプリのプロセスが，パーミッションに対応する GID を設定されているかどうかをカーネルで判断する．

3. 提案

提案システムは，クラスごとにパーミッションを設定し，指定外のクラスからのパーミッション利用を許可しない．クラスごとのパーミッションの指定は，アプリ開発者がマニフェストファイルに記述することで行い，ユーザがパーミッションの利用に関して判断を行うことはない．これらにより，ホストアプリ用のパーミッションと外部ライブラリのパーミッションを分離し，ユーザの判断を仰ぐことなしに外部ライブラリによるホストアプリ用のパーミッションの悪用を防止する．さらに，マニフェストファイルに対する追加記述やソースコードの難読化に対しても，専用のツールによりアプリ開発者の負担増加を抑える．

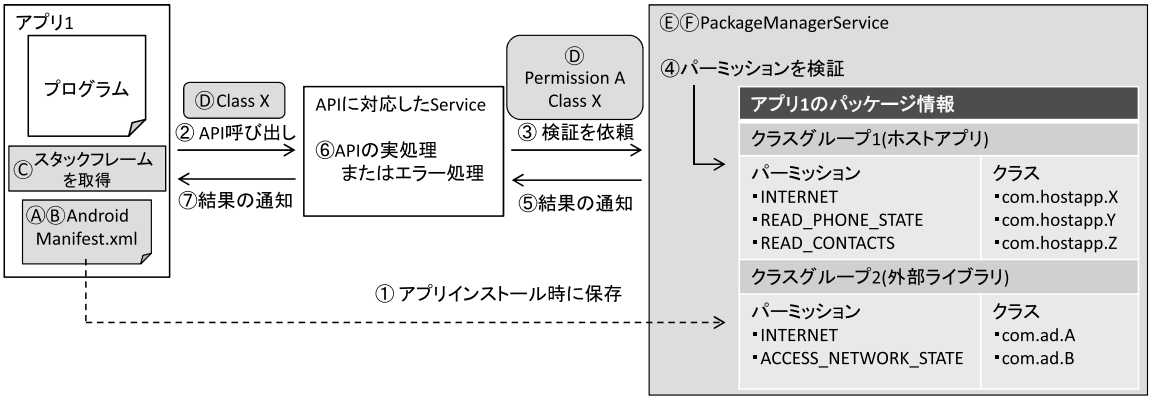

図 2 提案システムの概要
Fig. 2 Overview of the proposed system.

4. 提案システムの実装

本章では，提案システムの実装の詳細について述べる．まず，4.1 節で実装の概要について述べ，4.2 節でクラス単位でのパーミッションの設定をまとめて記述できるクラスグループについて述べる．4.3 節以降で実装の詳細について述べる．

4.1 概要

図 2 に，提案システムの概要を示す．灰色に塗られた部分が既存システム（図 1 参照）に追加した提案部分である．大きく分けて 5 点の実装を行った．以下それぞれについて述べる．

1 点目に，提案システムでは，クラスごとのパーミッション指定をクラスグループという単位を用いてマニフェストファイルに記述する．このため，クラスグループを定義できるように，マニフェストファイルの書式を変更した（図 2 のⒶ）．2 点目に，アプリ開発者のマニフェストファイルへのクラスグループの設定記述を補助するツールを実装した．このツールはクラスグループ設定記述の半自動化とソースコードの難読化対応を行う（図 2 のⒷ）．3 点目に，パーミッションの検証に必要な API を呼び出したクラス名を取得するために，各 API にスタックフレームを取得する処理の追加した（図 2 のⒸ）．スタックフレームから得られた API を呼び出したクラス名は，API 呼び出し時に API に対応したサービスに渡され，パーミッション名とあわせて PackageManagerService に渡される（図 2 のⒹ）．4 点目に，PackageManagerService のパーミッション検証処理をクラス単位で行えるように拡張した（図 2 のⒺ）．5 点目に，クラスグループの情報を表示するための API を新規作成した（図 2 のⒻ）．なお，処理の流れ①～⑦は既存システムと同じである．

4.2 クラスグループ

提案システムでは，パーミッションの設定が多くのクラスに必要になることから，複数のクラスに対する設定をまとめて記述するためにクラスグループという設定方法を導入する．クラスグループは，同一のパーミッションを利用する 1 つ以上のクラスを含み，アプリ開発者がいくつでも定義できる．

クラスグループの構成例を図 2 の PackageManagerService 内に示す．この例ではホストアプリのクラス群（com.hostapp.*）をクラスグループ 1 として 3 つのパーミッションを設定している．また，外部ライブラリのクラス群（com.ad.*）をクラスグループ 2 として 2 つのパーミッションを設定している．もし，2 つ以上の外部ライブラリを使用する場合には，さらに別のクラスグループを設定することで別のパーミッションを設定できる．アプリ開発者がクラスごとに詳細なパーミッションを設定する場合には，クラスグループをクラスごとに作成することで対応できる．

4.3 クラスグループによるパーミッション設定

クラスグループを用いてパーミッションを設定するために，マニフェストファイルの書式を一部変更した．従来のパーミッション記述例との比較を**図 3** に示す．Android 標準のパーミッションの設定は，パーミッションごとに uses-permission 要素を記述する．図 3 の従来のパーミッション記述例では，INTERNET と READ_PHONE_STATE のパーミッションを設定している．提案システムでは，クラスグループごとに class-group 要素を用意し，その中にパーミッション名を uses-class-permission 要素で，クラス名を join-class 要素で記述する．提案システムの記述例では 2 つのクラスによって構成されるクラスグループに，INTERNET と READ_PHONE_STATE のパーミッションを設定している．

PackageManagerService では，クラスグループによって分類されたクラス名とパーミッション名の組をキーとしたハッシュテーブル（以下，クラスパーミッションテーブルという）を作成する．ハッシュテーブルを用いることによ

```
<manifest ...>
...
 <!--従来のパーミッション記述例 -->
 <uses-permission
  android:name="android.permission.INTERNET"/>
 <uses-permission
   android:name="android.permission.READ_PHONE_STATE"/>

 <!--提案システムのパーミッション記述例 -->
 <class-group>
  <uses-class-permission
   android:name="android.permission.INTERNET"/>
  <uses-class-permission
   android:name="android.permission.READ_PHONE_STATE"/>
  <join-class android:name="com.ad.A" />
  <join-class android:name="com.ad.B" />
 </class-group>
...
</manifest>
```

図 3 マニフェストファイルの記述例
Fig. 3 Class-group element of AndroidManifest.xml.

```
<!--ツール適用前-->
<class-group android:name="ad.jar">
  ...
</class-group>

<!--ツール適用後-->
<class-group>
  ...
  <join-class android:name="com.ad.A" />
  <join-class android:name="com.ad.B" />
  <join-class android:name="com.ad.C" />
</class-group>
```

図 4 ツールによる class-group 要素の記述例
Fig. 4 Example of converting a class-group element by the support tool.

り，クラス名とパーミッション名による探索時間は，ハッシュテーブルの検索時間となり，クラス数とクラスグループ数とパーミッション数には依存しない．

4.4 パーミッション設定補助ツールと難読化対応

アプリ開発者のマニフェストファイルへの記述を補助するツールを作成した．記述補助ツールは 2 つの機能を持つ．1 つ目はクラスグループ記述の半自動化である．本ツールは，jar ファイル形式によって与えられたクラス群を 1 つのクラスグループとして，マニフェストファイルの class-group 要素を生成する．外部ライブラリは jar ファイル形式で配布されることが多いため，本ツールを用いることで，アプリ開発者は外部ライブラリのクラス構成を調べることなくクラスグループを設定できる．また，ホストアプリのクラス群についても，jar ファイルにまとめることで同様にクラスグループ設定を記述できる．記述例を**図 4** に示す．図 4 は 3 つのクラスで構成された ad.jar という外部ライブラリから，クラスの一覧を抽出して書き出している．このツールを使用することで，アプリ開発者の設定の負担はクラス数ではなく jar ファイル数に依存するだけとなり，クラス数の増加に対しても負担軽減を実現できる．

2 つ目は難読化への対応である．ProGuard [10] などの難読化ソフトは，ソースコードの変換後に，変換前後の

```
<!--難読化対応前-->
<class-group>
  ...
  <join-class android:name="com.hostapp.X" />
  <join-class android:name="com.hostapp.Y" />
  ...
</class-group>

<!--難読化対応後-->
<class-group>
  ...
  <join-class android:name="a.a.A" />
  <join-class android:name="a.a.B" />
  ...
</class-group>
```

図 5 ツールによる難読化対応例
Fig. 5 Example of the obfuscation support tool.

名称のマッピング情報を出力する．このマッピング情報を用いてマニフェストファイルの該当するクラス名を難読化後のクラス名に変換する．ツールを用いて難読化に対応した例を**図 5** に示す．例では，難読化前のクラス名（com.hostapp.*）を難読化された後のクラス名（a.a.*）に変換している．

4.5 API を呼び出したクラス名の取得

実行にパーミッションを必要とする各 API に対して，getStackTrace API によってスタックフレームを取得し，API を呼び出したクラスを特定する処理を追加した．この方式は文献 [8] と同様である．API は呼び出されると，API 呼び出し処理によって対応したサービスのメソッドをプロセス間通信を用いて実行する．追加したクラス特定処理は，API 呼び出し元クラスを取得するために getStackTrace API を呼び出して Java のスタックフレームを取得する．取得したスタックフレームから Android 標準 API のクラスを除いた最上位のクラスを API 呼び出し元クラスとして取り出して，各 API に対応するサービスのメソッド呼び出し時に当該 API 呼び出し元クラス名を渡す．API 呼び出し元クラス名を受け取った各サービスは，PackageManagerService へのパーミッション検証依頼時にパーミッション名とともに API 呼び出し元クラス名を渡す．

4.6 パーミッション検証

PacakgeManagerService では，各サービスから受け取ったパーミッション名と API 呼び出し元クラス名に基づいてパーミッションを検証する拡張を行った．実装を行ったパーミッションを**表 1** に示す．今回実装したパーミッションは全体の約 20%であるが，アプリで利用される主要なパーミッションであり，多くのアプリに対して適用可能であると考えられる．

パーミッション検証では，まずクラスパーミッションテーブルの有無を確認する．もし，テーブルが登録されていない場合は，提案システムに未対応のアプリと判断し，従来どおりのパーミッション検証を行う．一方，テーブル

表 1 適用したパーミッション一覧
Table 1 List of applied permissions.

パーミッション名	クラス	説明
INTERNET	WebView	ネットワークへのフルアクセス（カーネル制御のパーミッションは除く）
READ_PHONE_STATE	TelephonyManager	端末のステータスと ID の読み取り
ACCESS_NETWORK_STATE	ConnectivityManager	ネットワーク接続の表示
GET_ACCOUNTS	AccountManager	この端末上のアカウントの検索
KILL_BACKGROUND_PROCESSES	ActivityManager	他のアプリの終了
RESTART_PACKAGES	ActivityManager	他のアプリの終了
GET_TASKS	ActivityManager	実行中のアプリの取得
GET_PACKAGE_SIZE	PackageManager	アプリのストレージ容量の測定
CLEAR_APP_CACHE	PackageManager	アプリの全キャッシュデータの消去
ACCESS_COARSE_LOCATION	LocationManager	おおよその位置情報（ネットワーク基地局）
ACCESS_FINE_LOCATION	LocationManager	正確な位置情報（GPS とネットワーク基地局）
WAKE_LOCK	PowerManager	端末のスリープを無効にする
DISABLE_KEYGUARD	KeyguardManager	画面ロックの無効化
ACCESS_WIFI_STATE	WifiManager	Wi-Fi 接続の表示
CHANGE_WIFI_STATE	WifiManager	Wi-Fi からの接続と切断
MODIFY_AUDIO_SETTINGS	AudioManager	音声設定の変更
VIBRATE	Vibrator	バイブレーションの制御
BLUETOOTH	BluetoothAdapter	Bluetooth の設定へのアクセス（カーネル制御のパーミッションは除く）
BLUETOOTH_ADMIN	BluetoothAdapter	Bluetooth デバイスのペアの設定（カーネル制御のパーミッションは除く）
READ_SYNC_SETTINGS	ContentResolver	同期設定の読み取り
WRITE_SYNC_SETTINGS	ContentResolver	同期の ON/OFF の切替え
READ_CONTACTS	ContentResolver	連絡先の読み取り
WRITE_CONTACTS	ContentResolver	連絡先の変更
READ_SMS	ContentResolver	テキストメッセージ（SMS または MMS）の読み取り
WRITE_SMS	ContentResolver	テキストメッセージ（SMS または MMS）の編集
READ_CALL_LOG	ContentResolver	通話履歴の読みとり
WRITE_CALL_LOG	ContentResolver	通話履歴の書き込み
RECEIVE_BOOT_COMPLETED	BroadcastQueue	起動時に自動的に開始

が登録されていた場合は，API 呼び出し元クラス名と使用されたパーミッション名の組をキーとして，クラスパーミッションテーブルを検索する．検索が成功した場合は当該パーミッションは許可されていると判断し，検索に失敗した場合は不許可と判断する．

本実装は，多くのパーミッションに対して有効である．しかし，独自の対応が必要なパーミッションとして，端末起動時の BroadcastIntent を受信するための RECEIVE_BOOT_COMPLETED がある．このパーミッションに関しては，受信クラスが当該パーミッションを持っているかどうかを検証することで対応した．なお，このパーミッションの検証にはスタックトレースの確認は不要である．

4.7 クラスグループに基づくパーミッションの表示

Android では，API を用いてマニフェストファイルに記述されたパーミッションを取得することができる．そこで，提案システム用の拡張されたマニフェストファイルに対応したパーミッション取得 API を作成した．本 API は，PackageManagerService 上に実装をしており，PackageManagerService にあるパーミッションの情報を取得するものである．本 API は，インストーラへの組み込みやパーミッション検査ツールでの利用を想定している．本実装では，本 API を用いたパーミッション検査アプリを開発し，その表示例を**図 6** に示す．図 6 は，図 2 のクラスグループ設定を表示したものである．このアプリを利用することで，ユーザはアプリが組み込んでいる外部ライブラリと利用するパーミッションの情報を詳細に知ることが可能となる．ただし，クラスグループの情報を理解できるユーザは限られていると考えられる．難読化された場合はさらに理解が困難である．このため，外部ライブラリの開発者の詳細な情報を表示するなど，ユーザの理解を助けるための表示内容の検討やインストーラへの組み込みが必要であり，今後の課題である．

5. 評価

本章では，提案システムが正しく動作することの検証と，オーバヘッドの評価について述べ，最後に 1 章で述べた広

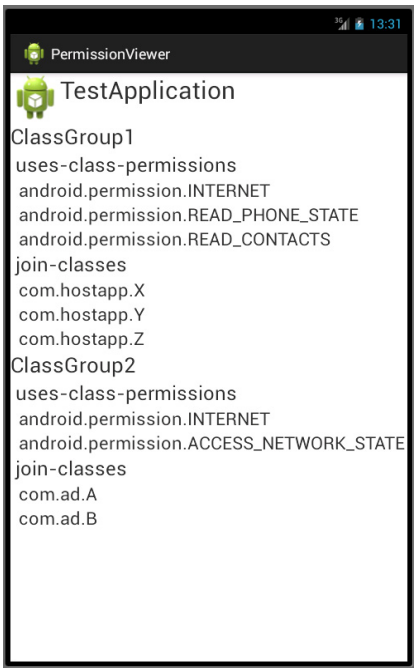

図 6 クラスグループの表示例
Fig. 6 Display of class group.

告ライブラリ実行要件について議論する．

5.1 提案システムによる制御

提案システムが正しくパーミッションを制御できていることを評価した．提案システムでは，検証が失敗すると従来のパーミッション機構と同様にセキュリティエラーを出力してアプリを停止終了させる．そこで，パーミッションを必要とするクラスをクラスグループに含まないアプリを動作検証用に作成した．AndroidSDK 付属の logcat [11] で取得したアプリ動作時のログを**図 7** に示す．図 7 は com.ad.A クラスに READ_PHONE_STATE のパーミッションがないため，その処理が失敗したことを示している．このセキュリティエラーが出力された際には，アプリは強制終了している．以上により，提案システムは実際のアプリにおいても正常に動作することを確認した．

5.2 アプリへの適用

提案システムが実際のアプリで利用可能であることを確かめるために，提案システムを搭載した Android OS 上で 25 のアプリを動作させた．各アプリが要求しているすべてのパーミッションをマニフェストファイルに記述した場合と，一部のパーミッションを記載した場合として広告ライブラリが属するクラスグループから一部のパーミッションを削り動作検証を行った．削ったパーミッションはカーネル制御のパーミッションを持つ INTERNET パーミッションと，カーネル制御のパーミッションを持たない ACCESS_NETWORK_STATE パーミッションである．なお，提案システムはカーネル制御のパーミッションに対応

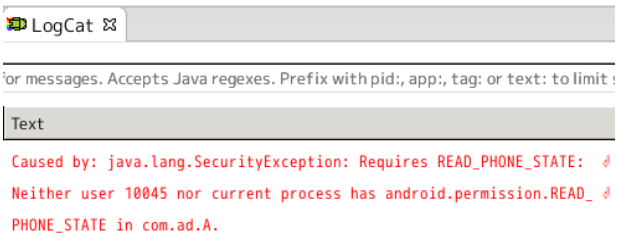

図 7 提案システムによるセキュリティエラーの出力例
Fig. 7 Example security error of the proposed system.

表 2 INTERNET パーミッションを削った場合の広告の動作
Table 2 Actions of advertisements after removing INTERNET permission.

確認された動作	広告種類数
何も表示されない	3
代替の画面が表示	1
Web page not abailable と表示	1
変化なし	5

していないため，カーネル制御のパーミッション（GID）は従来どおり与えている．

5.2.1 適用したアプリの概要

動作させたアプリの種類は，ゲームアプリ，タスク管理アプリ，電話帳アプリ，ライブ壁紙アプリ，Web 検索アプリである．アプリが持つ外部ライブラリの種類として，広告ライラリが 21 種類，クラッシュレポートを送信するライブラリ，ユーザの行動解析を行うライブラリ，ランキング機能を提供するライブラリ，3D エンジンやアニメーションに対応するためのライブラリ，twitter や facebook，SQLite を利用するためのライブラリを確認した．広告ライブラリ 21 種類のうち，11 種類の広告ライブラリは，ソースコードがリンクされていることは確認できたが，実行されていることは確認できなかったため，残りの 10 種類の広告ライブラリの動作についてまとめる．なお，すべてのパーミッションを記載した場合にアプリが正常に動作することを確認した．

5.2.2 パーミッションを削った場合の動作

INTERNET パーミッションのみを削って動作させた結果を**表 2** に示す．変化が見られた広告は 5 種類あり，何も表示されなくなった広告が 3 種類，代替の画面が表示された広告が 1 種類，Web page not abailable と表示された広告が 1 種類である．変化が見られなかった 5 種類の広告に関しては，提案システムが対応していないネイティブ API を用いて広告の取得を行っており，かつ，ネイティブ API の実行に際して必要なパーミッションが与えられているか否かを確認するコードを含んでいないために，そのまま動作したと考える．逆に，動作に変化が見られた広告は，ネイティブ API を用いていない，またはネイティブ API の実行に際してパーミッションの確認を行うコードを含んで

表 3 ACCESS_NETWORK_STATE パーミッションを削った場合の広告の動作
Table 3 Actions of advertisements after removing ACCESS_NETWORK_STATE permission.

確認された動作	広告種類数
何も表示されない	2
代替の画面が表示	1

いると考える．ネイティブ API とは，カーネル制御でパーミッションを検証する API であり，Java の標準ライブラリの API と，AndroidNDK で用いる API の両方を指す．確認のためにカーネル制御のパーミッションである “inet” の GID も削ったところ，変化が見られなかった広告を含むアプリがパーミッションエラーにより強制終了することを確認した．

次に，ACCESS_NETWORK_STATE パーミッションのみを削ってアプリを動作させた．ACCESS_NETWORK_STATE パーミッションは，動作を確認できた 10 種類の広告ライブラリのうち 3 種類の広告ライブラリが必要としている．動作結果を**表 3** に示す．ACCESS_NETWORK_STATE パーミッションを必要とする 3 種類すべての広告で変化が見られ，何も表示されなくなった広告が 2 種類，代替の画面が表示された広告が 1 種類である．以上の結果により，提案システムはネイティブ API を利用しない広告ライブラリにおいては正常に機能する．

5.2.3 広告ライブラリにおけるカーネル制御のパーミッション利用状況

広告のビジネスモデルを維持するためには INTERNET パーミッションを与えることは必須である．そこで，21 種類の広告ライブラリの INTERNET 以外のカーネル制御のパーミッションに関わるネイティブ API の利用状況を調べたところ，利用されていないことを確認した．この結果により，提案システムは多くの広告ライブラリに対して適用することが可能である．

5.3 提案システムのオーバヘッド

提案システムでは，パーミッションが必要な API の処理に追加実装を行ったため，その分のオーバヘッドが発生する．そこで API の呼び出しに要する時間を測定した．測定に用いた API は，Android の端末識別子を取得する API である getDeviceId であり，1,000 回実行した際の平均値を測定した．

評価環境を**表 4** に示す．提案システムの評価は，総クラス数と総パーミッション数を変化させてオーバヘッドを測定した．なお，それぞれのオーバヘッドを測定するときには，他方の数は 1 に固定している．従来手法のパーミッション機構はクラス数を考慮していないため，総パーミッション数の変化のみ測定した．なお，クラスグループは，登録時のパーミッション設定のみに用いており，アプリ動作時には使用されないため評価パラメータとして用いていない．

表 4 評価環境
Table 4 Evaluation environment.

測定端末	PandaBoard ES [12]
CPU	1.2 GHz ARM Cortex-A9 デュアルコア
memory	1 GB
OS	Android OS 4.2 に機能を追加

表 5 総クラス数ごとの処理時間（パーミッション数は 1）
Table 5 Processing time of getDeviceId (number of permissions is 1).

クラス数	1	10	20	50
提案システム (msec)	1.403	1.455	1.396	1.380

表 6 総パーミッション数ごとの処理時間（クラス数は 1）
Table 6 Processing time of getDeviceId (number of classes is 1).

パーミッション数	1	10	20	50
従来 (msec)	0.631	0.605	0.604	0.631
提案システム (msec)	1.403	1.404	1.396	1.380

総クラス数に応じた評価結果を**表 5**，総パーミッション数に応じた評価を**表 6** に示す．表 5 と表 6 の提案システムの結果より，提案システムの処理時間はクラス数とパーミッション数に依存することなく約 1.4 ミリ秒で一定であることが分かる．また，表 6 により，提案システムと従来手法の差をオーバヘッドとして求めると，約 0.8 ミリ秒である．このうち，API 呼び出し元クラスを取得するための getStackTrace API に要する時間は約 0.6 ミリ秒であり，オーバヘッドの大部分を占めることが分かる．このため，提案システムの負荷軽減には，API 呼び出し元クラス取得方法の改良が必要であると考えられる．

5.4 広告ライブラリ実行要件

1 章で広告ライブラリ実行要件として，広告ライブラリのビジネスモデルを維持できること，任意の広告ライブラリを利用できること，ユーザによる判断を必要最低限にすること，ソースコードが難読化されたアプリの保護ができることを定めた．

この要件に対して，提案システムはアプリ開発者が設定変更を行うため，広告ライブラリの実行に必要なパーミッションを削ることはなく，広告ライブラリのビジネスモデルを維持できる．次に，5.2 節で広告ライブラリのソースコードを書き換えることなく運用可能であることを示したが，一部のネイティブ API を含む広告ライブラリでは正常に制御できなかった．そして，4.7 節で示したように，提案

システムはアプリが組み込んでいる外部ライブラリと利用するパーミッションの情報提供が可能であり，ユーザによる判断を助けることができる．最後に，4.4 節で難読化に対応するツールを作成することで難読化されたアプリの保護が可能であることを示した．以上により提案システムは広告ライブラリ実行要件を満たすが，ネイティブ API を利用する広告ライブラリの利用については今後の課題である．

6. 考察

本章では，提案システムに関する考察として，アプリの用途によるオーバヘッドについて，ネイティブの API について，アプリ開発者の負担についてと，提案システムに対する攻撃とそれによる制限について述べる．

6.1 アプリの用途によるオーバヘッドの考察

API の呼び出し頻度によってはオーバヘッドが無視できなくなる可能性がある．そこでアプリを以下の 3 種類に分類し，それぞれが受ける影響について考察する．

- ゲームなどのリアルタイム性を必要とするアプリ
- 電話帳など端末の情報を扱うアプリ
- ネットワーク通信を行うアプリ

6.1.1 ゲームなどのリアルタイム性を必要とするアプリ

画面描画処理や音楽を鳴らす処理にはパーミッションが必要ないため，ゲームなどのリアルタイム性を必要とするアプリでは，パーミッションが必要な API が高頻度で呼び出されることはないと考えられる．そのため提案システムが高頻度で働くことはなく，影響は小さいといえる．

6.1.2 電話帳などの端末の情報を扱うアプリ

電話帳などのアプリでは，端末に記憶された情報を得るために API が呼び出される．しかし，API の呼び出しは，最初の情報取得時と情報の更新時のみであるため，いずれも高頻度ではないと考えられる．そのため影響は小さいといえる．

6.1.3 ネットワーク通信を行うアプリ

本分類に該当するアプリとして，たとえば Web ブラウザと，ネットワークの状態を定期的に調べる広告ライブラリと Web から情報をダウンロードするアプリがある．Web ブラウザは，WebView クラスを用いてインターネットにアクセスして Web ページを表示する．しかし，WebView クラスは Web ページを取得するときではなくコンストラクタ生成時にのみパーミッション検証を行うため，検証の頻度は低いといえる．次に，ネットワークの状態を定期的に調べる広告ライブラリは，5.2 節で評価した 21 種類の広告ライブラリ中の 2 種類が該当する．これらの広告ライブラリにおけるネットワーク状態確認のためのパーミッションの検証間隔を確認したところ，状態確認間隔が約 1 秒であった．この間隔は，提案システムのオーバヘッド（約 0.8 ミリ秒）に比べて非常に長いため，影響は小さいといえる．

表 7 カーネル制御のパーミッション一覧（Android 4.2）
Table 7 List of kernel controlled permissions (Android 4.2).

パーミッション名	GID
BLUETOOTH_ADMIN	net_bt_admin
BLUETOOTH	net_bt
BLUETOOTH_STACK	net_bt_stack
NET_TUNNELING	vpn
INTERNET	inet
CAMERA	camera
READ_LOGS	log
READ_EXTERNAL_STORAGE	sdcard_r
WRITE_EXTERNAL_STORAGE	sdcard_rw
WRITE_MEDIA_STORAGE	media_rw
ACCESS_MTP	mtp
NET_ADMIN	net_admin
ACCESS_CACHE_FILESYSTEM	cache
DIAGNOSTIC	input，diag
READ_NETWORK_USAGE_HISTORY	net_bw_status
MODIFY_NETWORK_ACCOUNTING	net_bw_acct

最後に，Web から情報を頻繁にダウンロードするアプリは，ソケットなどのネイティブ API を用いて通信を行う．提案システムはネイティブ API には対応していないためオーバヘッドの影響外となる．

6.2 ネイティブ API

Android ではネイティブ API を利用することができる．これらの API のパーミッションはカーネル制御のパーミッション機構によって制御されるため，提案システムはネイティブ API に対しては機能しない．そのため提案システムがカーネル制御のパーミッションに対応する必要がある．対応するためには，カーネル側で呼び出し元クラスを取得する方法を確立することと，本来プロセス単位で与えられる GID をクラス単位で割り当てることが必要である．さらに，ネイティブ API の中には DalvikVM を介さず実行される API もあるため，それらの API に対しては Java のスタックトレースは利用できず，ネイティブ API の呼び出し元を取得することは困難である．また，GID の割当て変更にも大きな変更が必要であると考えられる．提案システムが対応できないカーネル制御のパーミッションを**表 7** に示す．これらの実現とカーネル制御のパーミッションへの対応は今後の課題である．

6.3 アプリ開発者の負担

提案システムでは，アプリ開発者がマニフェストファイルに従来よりも多くの記述をする必要があるため，アプリ開発者へ負担が増すと考えられる．しかし，ホストアプリのクラスはアプリ開発者が自身で作成したものであるため，記述することは容易である．外部ライブラリのクラスに関しては，外部ライブラリが jar ファイルの形式で配布

されるため，ファイル構成を容易に知ることができ記述は可能である．また，4.4 節で述べたように jar ファイルからクラス名を自動で記述するツールを作成した．これによりアプリ開発者のマニフェストファイル記述のための負担を増やさない．

6.4 提案システムへの攻撃

Android 上に実装された提案システムに対して攻撃が行われる可能性がある．ここではリフレクションを用いた攻撃と標準の DalvikVM では対処が困難な攻撃，最後にメモリに対する攻撃について述べる．

6.4.1 リフレクションを用いた攻撃

Android は Java 言語をベースとしているため，リフレクションを利用することができる．リフレクションを用いると，メソッド名やフィールド名で対応するメソッドやフィールドにアクセスすることができる．このリフレクションを用いて 2 種類の攻撃が可能となる．

1 つ目は，情報を持つフィールドを参照する攻撃である．Java ではフィールドを private 宣言することでクラスの外からのアクセスを防ぐことができる．しかし，リフレクションを用いることで，private のフィールドを参照し情報を盗む攻撃が可能となる．

2 つ目は，リフレクションを用いてホストアプリのメソッドを実行する攻撃である．たとえば，パーミッションを必要とする API を実行してその結果を返すだけのメソッドをアプリ開発者が実装した場合，攻撃者は，リフレクションを用いて当該メソッドを呼び出すことでパーミッションを持つホストアプリのクラスに情報を取得させることが可能となる．

これらの攻撃に対しては，invoke メソッドと setAccessible メソッドの呼び出しをフックし，権限の異なるクラスに対するアクセスであるか検証する手法が有効である．ただし，クラスグループを確認するためのスタックトレースがオーバヘッドとなる可能性があり，その場合は軽量なスタックトレースを DalvikVM に実装する必要がある．

6.4.2 標準の DalvikVM では対処困難な攻撃

提案システムは DalvikVM に変更を加えずに実装されているために，対処困難な攻撃が 2 点存在する．

1 つ目は，DexClassLoader を用いた攻撃である．DexClassLoader はクラスを動的にロードするときに用いるクラスである．このクラスの loadClass メソッドは，すでにクラスがロードされているかどうか判定する機能を持つ．しかし，この判定機能は loadClass メソッドをオーバライドすることで迂回することができる．そのため，すでにロードされたクラスを任意のクラスで上書きすることが可能となる．この機能を悪用することで，ホストアプリのクラス名を持つ任意のクラスを作成することができる．この攻撃に対しては，loadClass メソッドの呼び出しをフックし，権限の異なるクラスのロードを禁止することで対応可能であると考えられるが，このフックも当該メソッドのオーバライドで迂回される可能性がある．この対応には，DalvikVM が loadClass メソッドを呼び出すときに，ロードするクラスを検証する必要がある．DalvikVM が検証することで，アプリによる迂回はできなくなる．

2 つ目は，提案システムを迂回する攻撃である．提案システムは API にスタックトレースを取得し渡す処理を追加している．そのため，リフレクションなどを用いて，この処理部分を迂回してサービスと通信することでスタックトレース情報を偽装することができ，提案システムの制御をすり抜ける攻撃が考えられる．この対応には，パーミッション検証を依頼するサービスが，アプリのスタックトレースを直接取得できるようにする必要がある．サービス側が直接取得することで，アプリ内での迂回や偽装を行う意味がなくなる．以上で述べたように，攻撃に対する対策を考慮した場合，アプリやサービスの実行基盤である DalvikVM 内において検証を行う機構や，異なるプロセスのスタックトレースを取得する機構が必要となる．しかし，これらの対策には DalvikVM の変更が必要となるため今後の課題である．

6.4.3 メモリに対する攻撃

Android では AndroidNDK を用いてネイティブコードを用いた開発が可能である．そのため，ホストアプリのメモリを参照して情報を取得したり，メモリを改ざんしてスタックを偽装したりする攻撃が考えられる．この攻撃に対しては，ネイティブコードの実行を禁止するか，ネイティブコードが悪意ある動作をしていないかを確認する必要があり，今後の課題となる．

7. 関連研究

7.1 パーミッションの分離

広告ライブラリのパーミッションをホストアプリから分離する研究としては，1 章で述べた AdSplit と AFrame のほかに，広告を AndroidOS 側の機能として実現することで広告ライブラリを排除する AdDroid [13] がある．この手法では広告ライブラリそのものが存在しないため AndroidOS 側で提供される広告機能のみが利用可能となり，広告ライブラリの自由度が低下する．あるいは多様な広告ライブラリに対応するために，頻繁な OS のアップデートが必要となる．提案システムでは広告ライブラリ側に変更がないため従来の広告ライブラリをそのまま利用可能である．

一方，これらの 3 つの手法はホストアプリと広告ライブラリを別プロセスで実行するため，提案システムにおけるメモリへの攻撃といった問題が発生しないというメリットがある．

7.2 パーミッションの制限

パーミッションの制限を強化する研究としては，1章で述べた手法 [8] や，アプリインストール時にユーザが承認するパーミッションを変更できる手法 [14]，パーミッションに時間や場所などのより詳細な条件を付与できるようにする手法 [15]，アプリを静的解析して過剰にパーミッションを宣言していないか判断する手法 [16] がある．また，パーミッションにより危険なアプリを判別する研究としては，パーミッションの組合せに着目する手法 [17] や，パーミッションの利用順序に着目する手法 [18] がある．これらの研究はいずれもアプリ全体に影響するため，ユーザにとっては外部ライブラリだけでなくアプリに対する安全性の向上が期待できる．しかし，いずれもユーザに判断を委ねている．パーミッションを適切に判断できるユーザが少ないことは文献 [9] で報告されており，現行のパーミッション機構よりもユーザの負担が大きくなることは望ましくない．

提案システムは，アプリ開発者の協力のもとでホストアプリと外部ライブラリのパーミッションを静的に分離して公開するための API を備えており，インストーラやアプリに組み込むことでユーザに詳細なパーミッション情報を表示することが可能である．文献 [19] では「アプリケーション提供者が情報収集モジュールを組み込む場合，アプリケーションを通じた情報収集の実態について明らかにするうえで，アプリケーション提供者は，自らが組み込んでいる情報収集モジュールの数，名称，提供者などの基本的な情報について，利用者に対して説明する.」とある．提案システムでは，外部ライブラリの数やパーミッションについて提供が可能なことから，ユーザによる外部ライブラリの理解を助け，負担を軽減することができる．

7.3 個人情報漏洩対策手法

パーミッション機構を対象とせず個人情報の漏洩を防ぐ手法も研究されている．TaintDroid [20] では機密情報に対してタグをつけることでアプリ動作時に動的解析を行うことができ，Aurasium [21] はアプリを再パッケージ化してセキュリティコードを埋め込み情報漏えいや権限昇格攻撃につながる動作を監視する．提案システムではパーミッションの不正利用を防ぐことが可能であるが情報についての追跡は行わないため，これらの手法と組み合わせることは有効である．

7.4 Android 以外での権限制御方式

Android 以外における権限の分離と制限に関する研究について述べる．FreeBSD に搭載されたファイルディスクリプタに対する動作の権限を制御する Capsicum [22]，保護ドメインごとにソースコードを分割してセキュリティポリシを分離する手法 [23]，実行コンテキストに応じてポリシを切り替える手法 [24] がある．提案システムでは，クラス単位で権限の分離を行っている点が異なっている．

8. まとめ

本稿では広告ライブラリを含む外部ライブラリが，ホストアプリのパーミッションを利用できるという問題を解決するために，実行中のクラスに基づいてパーミッションを制御するシステムを提案した．アプリ開発者がクラスグループを用いてパーミッションの設定を行うことで，ユーザにパーミッション利用の判断を委ねることなしに，ホストアプリと外部ライブラリをそれぞれ最小のパーミッションで動作させることが可能となる．提案システムの実装としては，アプリ開発者のマニフェストファイルへの記述を補助するツールを作成することでアプリ開発者の負担を軽減し，ソースコードの難読化にも対応している．評価として，25 種類のパーミッションと，カーネル制御のパーミッション部分を除く 3 種類のパーミッションにおいて制御が正しく行われていることを確認し，25 のアプリに含まれる 5 種類の広告で正しく動作することを確認した．そして，オーバヘッドの測定によって提案システムがパフォーマンスに与える影響が小さいことを示した．

今後の検討課題としては，カーネル制御のパーミッションへの対応と，提案システムへの攻撃に対応すること，外部ライブラリに関する情報の表示方法と表示内容の検討があげられる．

参考文献

[1] Android: 入手先 ⟨http://www.android.com⟩.
[2] AdMob: 入手先 ⟨http://www.google.com/ads/admob/⟩.
[3] 竹森敬祐：スマートフォンからの利用者情報の送信—情報収集の実態調査，総務省スマートフォンを経由した利用者情報の取り扱いに関する WG（第 1 回），資料 4 (2012).
[4] Grace, M.C., Zhou, W., Jiang, X. and Sadeghi, A.-R.: Unsafe Exposure Analysis of Mobile In-app Advertisements, *Proc. 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp.101–112 (2012).
[5] Security NEXT：不正ライブラリが組み込まれた Android アプリが Google Play 上に 32 件—偽広告 SDK を装った攻撃か，入手先 ⟨http://www.security-next.com/039598⟩（参照 2014-02-06）.
[6] Shekhar, S., Dietz, M. and Wallach, D.S.: AdSplit: Separating Smartphone Advertising from Applications, *Proc. 21st USENIX Conference on Security Symposium*, pp.28–28 (2012).
[7] Zhang, X., Ahlawat, A. and Du, W.: AFrame: Isolating Advertisements from Mobile Applications in Android, *Proc. 29th Annual Computer Security Applications Conference*, pp.9–18 (2013).
[8] 川端秀明，磯原隆将，窪田　歩，可児潤也，上松晴信，西垣正勝：Android における細粒度アクセス制御機構，情報処理学会論文誌，Vol.54, No.8, pp.2090–2102 (2013).
[9] Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E. and Wagner, D.: Android Permissions: User Attention, Comprehension, and Behavior, *Proc. 8th Symposium on Usable Privacy and Security*, pp.3:1–3:14 (2012).

[10] ProGuard: 入手先 ⟨http://proguard.sourceforge.net/⟩.
[11] logcat: 入手先 ⟨http://developer.android.com/tools/help/logcat.html⟩.
[12] PandaBoard ES: 入手先 ⟨http://pandaboard.org/content/pandaboard-es⟩.
[13] Pearce, P., Felt, A.P., Nunez, G. and Wagner, D.: AdDroid: Privilege Separation for Applications and Advertisers in Android, *Proc. 7th ACM Symposium on Information, Computer and Communications Security*, pp.71–72 (2012).
[14] Nauman, M., Khan, S. and Zhang, X.: Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints, *Proc. 5th ACM Symposium on Information, Computer and Communications Security*, pp.328–332 (2010).
[15] Ongtang, M., McLaughlin, S., Enck, W. and McDaniel, P.: Semantically Rich Application-centric Security in Android, *Sec. and Commun. Netw.*, Vol.5, No.6, pp.658–673 (2012).
[16] Felt, A.P., Chin, E., Hanna, S., Song, D. and Wagner, D.: Android Permissions Demystified, *Proc. 18th ACM Conference on Computer and Communications Security*, pp.627–638 (2011).
[17] Enck, W., Ongtang, M. and McDaniel, P.: On Lightweight Mobile Phone Application Certification, *Proc. 16th ACM Conference on Computer and Communications Security*, pp.235–245 (2009).
[18] Banuri, H., Alam, M., Khan, S., Manzoor, J., Ali, B., Khan, Y., Yaseen, M., Tahir, M.N., Ali, T., Alam, Q. and Zhang, X.: An Android Runtime Security Policy Enforcement Framework, *Personal Ubiquitous Comput.*, Vol.16, No.6, pp.631–641 (2012).
[19] 総務省：スマートフォンプライバシーイニシアティブ，総務省スマートフォンを経由した利用者情報の取り扱いに関する WG 最終取りまとめ (2012).
[20] Enck, W., Gilbert, P., 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, *Proc. 9th USENIX Conference on Operating Systems Design and Implementation*, pp.1–6 (2010).
[21] Xu, R., Saïdi, H. and Anderson, R.: Aurasium: Practical Policy Enforcement for Android Applications, *Proc. 21st USENIX Conference on Security Symposium*, pp.27–27 (2012).
[22] Watson, R.N.M., Anderson, J., Laurie, B. and Kennaway, K.: Capsicum: Practical Capabilities for UNIX, *Proc. 19th USENIX Conference on Security*, pp.3–3 (2010).
[23] Niu, B. and Tan, G.: Enforcing User-space Privilege Separation with Declarative Architectures, *Proc. 7th ACM Workshop on Scalable Trusted Computing*, pp.9–20 (2012).
[24] Shioya, T., Oyama, Y. and Iwasaki, H.: A Sandbox with a Dynamic Policy Based on Execution Contexts of Applications, *Proc. 12th Asian Computing Science Conference on Advances in Computer Science: Computer and Network Security*, pp.297–311 (2007).

推薦文

本論文では，Android アプリにおいて利用される外部ライブラリ（広告ライブラリなど）の権限をアプリ本体から分離することによって，外部ライブラリが本来のアプリのパーミッションを悪用して個人情報等を不正に詐取することを防止する方法を提案している．世界的に非常に競争の激しい研究分野の中で相応の成果を達成している点は評価に値する．今後の同研究分野の発展に有用であるため，推薦論文として推薦したい．

（コンピュータセキュリティ研究会主査　西垣正勝）

日置 将太 （学生会員）

2013 年名古屋工業大学工学部情報工学科卒業．同年同大学大学院工学研究科情報工学専攻博士前期課程入学，現在に至る．

高瀬 拓歩 （学生会員）

2014 年名古屋工業大学工学部情報工学科卒業．同年同大学大学院工学研究科情報工学専攻博士前期課程入学，現在に至る．

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，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．

松尾 啓志 （正会員）

1983 年名古屋工業大学工学部情報工学科卒業．1985 年同大学大学院修士課程修了．1989 年同大学院博士課程修了．同年名古屋工業大学電気情報工学科助手．講師，助教授を経て，2003 年同大学大学院教授．2006 年同大学情報基盤センターセンター長（併任），現在に至る．分散システム，分散協調処理に関する研究に従事．工学博士．電子情報処理学会，人工知能学会，IEEE 各会員．