情報処理学会論文誌 Vol.54 No.8 2090–2102 (Aug. 2013)

推薦論文

Androidにおける細粒度アクセス制御機構

川端 秀明[1,a)] 磯原 隆将[1] 竹森 敬祐[1] 窪田 歩[1] 可児 潤也[2] 上松 晴信[2] 西垣 正勝[2]

受付日 2012年10月19日, 採録日 2013年5月18日

概要：Android を搭載した端末では，アプリケーションをインストールする際に，アプリケーションが利用する機能や情報をパーミッションという単位でユーザに通知して承認を得る必要がある．しかしながら，実行中のアプリケーションがパーミッションをどのように利用しているのか把握することができないという問題や，1 度ユーザがアプリケーションのパーミッションを承認してしまうと，実行中に取り消しを行うなどの制御が効かないという問題がある．そこで本論文では，実行中のアプリケーションに対して，利用する機能や情報へのアクセスをリアルタイムでユーザに通知して，制御を加えるフレームワークを提案する．これは，Android 端末の機能や情報を扱う API に対してフック処理を加えることによって，アプリケーションが動作する中でユーザ承認の確認機構を実現するものである．特に，スタックフレームを利用し，パーミッション利用の透明性を高め，細粒度のアクセス制御を可能としている点は，他に類を見ない提案方式の特長である．悪用された場合に危険となるパーミッションに付随する API にフック処理を実装し，提案フレームワークの評価を行ったところ，オーバヘッドは許容範囲内であること，および，アプリケーションが動作する中で機能や情報へのアクセスの様子をユーザが把握でき，アプリケーション実行中にユーザが適切な承認を与えることができることを確認した．

キーワード：Android，パーミッション，アクセス制御

Fine-grained Access Control Framework in Android

Hideaki Kawabata[1,a)] Takamasa Isohara[1] Keisuike Takemori[1] Ayumu Kubota[1]
Junya Kani[2] Harunobu Agematsu[2] Masakatsu Nishigaki[2]

Received: October 19, 2012, Accepted: May 18, 2013

Abstract: In order to reduce security risks caused by suspicious Android applications, users have to confirm permissions that define privileges to access APIs requested by an application at the installation phase. However, once the user authorize these permissions, it is hard to understand when and why these permissions are used for the application. Moreover, the user cannot check whether the application is properly using these permissions. In this paper, we propose a fine-grained access control framework that can control access to important information and/or functions requested by the application during its execution. The proposed framework allows users to confirm the access to the permission-protected APIs and to authorize the application to use them. In addition, our framework considers context of API calls by using stack-frame information for fine-grained control. In order to verify the effectiveness of the proposal, we modified the application framework of Android to implement runtime security policy enforcement mechanism by hooking several important API calls. Our performance evaluation result shows that the overhead of policy verification is negligible and does not affect the behavior of application's runtime.

Keywords: Android, permission, access control

[1] 株式会社 KDDI 研究所
KDDI R&D Laboratories Inc., Fujimino, Saitama 356–8502, Japan
[2] 静岡大学
Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[a)] kawabata@kddilabs.jp

本論文の内容は 2011 年 10 月のコンピュータセキュリティシンポジウム 2011（CSS2011）にて報告され，同プログラム委員長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．

1. はじめに

Android [1] では，アプリケーションから利用できる様々な機能や情報への API（Application Programming Interface）が公開されているため，利便性の高いアプリケーションを実現できる．特に，携帯電話として利用される端末では，電話番号，IMEI（端末固有識別子：International Mobile Equipment Identity），IMSI（SIM 識別子：International Mobile Subscriber Identity），アドレス帳，カレンダ，ブラウザ閲覧履歴など識別子やプライバシ情報を取得する API が用意されている．よって，これらの API が悪用された場合，なりすましやプライバシの侵害が問題になる．

Android のセキュリティ機構として，アプリケーションから利用する機能や情報を保護するパーミッションフレームワークがある．アプリケーションが利用する重要な機能や情報へのアクセスをパーミッションと呼ばれる単位で定義し，アプリケーションインストール時にユーザに対して承認を求める．

しかしながら，このパーミッションフレームワークには以下の課題がある．

- パーミッションの粒度

ユーザはアプリケーションをインストールするためには，アプリケーションが要求するすべてのパーミッションを許可する必要があり，パーミッション単位で承認することができない．また，1 つのパーミッションにリンクする機能や情報が複数あり，パーミッションの粒度が粗い．パーミッションに含まれる機能や情報の中でユーザが許可できるものと拒否したいものが混在している場合であっても，これらを切り分けて承認できない．

- パーミッション利用の透明性

パーミッションに付随する機能や情報を，「いつ」，「どのような機能や情報に」，「だれが」，アクセスするのか，ユーザは知ることができない．そのためアプリケーションの目的と利用実態を把握できず，機能や情報を不正に利用している場合の検知がきわめて難しい．また，アプリケーションに広告ライブラリなどの第三者ライブラリが組み込まれている場合，アプリケーションと第三者ライブラリのどちらがパーミッションを利用するのか分からない．ここで，「だれが」という用語を利用したが，実際に機能や情報を利用する動作の主体はプログラム実行コードであるため，「アプリケーション本体の実行コードか，内包する第三者ライブラリの実行コードか」という説明が正しい．しかし本論文の狙いには，広告事業者が利用者情報を収集し，利用者に効果的な広告を表示するなど，実際にはアプリケーション，第三者ライブラリ開発者という「人，事業者」が機能，情報を利用することも想定しているため，以後「だれが」という用語で説明する．

これまで，Android のパーミッションフレームワークに対する様々な研究 [2], [3], [4], [5] がなされているが，パーミッション粒度の課題や，パーミッション利用の透明性の確保に対する取り組みはなされていない．

そこで本論文では，Android のパーミッションで保護されている API にフックを仕掛け，実行中のアプリケーションに対してリアルタイムでユーザに通知して判断させる細粒度アクセス制御方式を提案する．これは，Android における機能や情報を扱う API にフック関数の導入と，フックを制御するアプリケーションを Android に実装することで実現する．特に，提案方式は，スタックフレームを活用することで，「だれが」API をコールしたのか判定することで，パーミッション利用の透明性を高め，細粒度のアクセス制御を可能としている点を特徴とする．ユーザ通知においては，ユーザビリティを考慮し，ユーザによる API の利用承諾を記憶するセキュリティポリシを設定することで API コールごとにユーザ通知が発生しない考慮をした．提案方式の実装を行い，実行中のアプリケーションの API コールを一時的に中断させ，ユーザによるアクセス制御を確実に挿入させる仕組みを実現する．提案方式の制御処理に要する負荷について評価を行い，オーバヘッドも十分小さいことを示す．

2. 背景

2.1 Android のアーキテクチャ

Android のアーキテクチャを図 1 に示す．Android は，Linux カーネル，ミドルウェア層，そしてアプリケーション層から構成されている．Linux カーネルは，プロセス管理，ファイルシステム，デバイスドライバなど OS としての基本機能を上位層に提供する．ミドルウェア層では，アプリケーションの実行環境としてレジスタベースの Dalvik 仮想マシン（DVM：Dalvik Virtual Machine），アプリケーション管理，情報や機能を利用するためのライブラリなどを提供する．Android では，アプリケーションの起動などアプリケーションの状態管理は ActivityManager，アプリケーションが持つパーミッションなどの権限管理は PackageManager が行っている．このような Android のコ

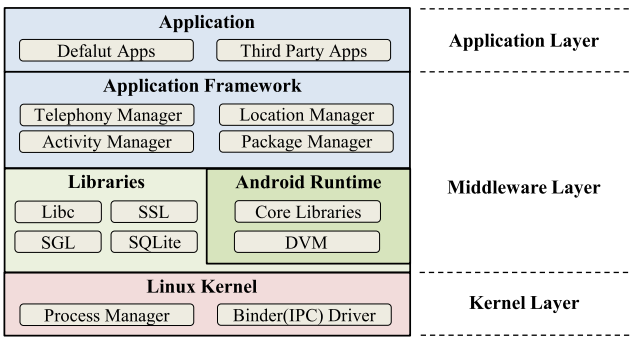


図 1 Android のアーキテクチャ
Fig. 1 Architecture of Android.

アとなるシステムサービスは system_server と呼ばれるプロセスで動作する．アプリケーション層では，プリインストールされている電話，インストーラなどの基幹となるアプリケーションや，第三者が開発したアプリケーションなどが Dalvik 仮想マシン上で動作する．Android アプリケーションの構成要素として，Activity，Service，Broadcast Receiver，Content Provider という 4 つのコンポーネントがある．Activity とは，ユーザに視覚的なインタフェース（画面）を提供するコンポーネントである．Service は，画面を持たずに，バックグラウンドで実行されるコンポーネントである．BroadcastReceiver は，システム全体にブロードキャストされる情報を受け取るコンポーネントである．Content Provider はデータを取得・保存できるコンポーネントである．Android では，プロセス間通信を実現する仕組みとして Binder があり，カーネル層でドライバが提供されている．Binder を利用すると，あるアプリケーションから異なるプロセスで動作するサービスに実装されたメソッドをコールするなど，プロセスを越えた連携が可能となる．アプリケーションどうしで連携するときや，アプリケーションから ApplicationFramework 層で用意された API をコールするときなど，Android で Binder は広く利用されている．

2.2 サンドボックス

Android アプリケーションはインストール時に，アプリケーションごとに異なる UID，GID が付与され，独立したプロセスとしてサンドボックス上で実行される．アプリケーションのファイルアクセスは，Linux のファイルシステムのパーミッションと同様の仕組みであるため，自身で作成したファイルと自身に Read，Write 権限が設定されたファイルしかアクセスできない．

2.3 パーミッションフレームワーク

Android には，アプリケーションからの重要な機能や情報へのアクセスをパーミッション（権限）と呼ばれる単位で定義し，ユーザ承認によってアクセス制御を行うパーミッションフレームワーク機構がある．パーミッションで制限される重要な機能として，ネットワーク接続，カメラの利用などがあり，重要な情報として，アドレス帳，端末固有識別子，位置情報などがある．アプリケーションがパーミッションで制限された機能や情報を利用するためには，AndroidManifest.xml と呼ばれるファイルに使用するパーミッションを宣言する必要がある．パーミッションを宣言すると，アプリケーションインストール時に，ユーザに対して，アプリケーションが利用する機能や情報が提示される．ユーザはアプリケーションが要求するすべてのパーミッションを承認してインストールするか，パーミッションを許容できないため，アプリケーションのインストールを諦めるかの選択をすることになる．パーミッションが許可されたアプリケーションは，Application Framework 層に設けられている API を用いて，OS 内の重要な機能や情報へアクセスすることが可能となる．アプリケーションがパーミッションで保護された API を実行すると，Android のパーミッションを判定する PackageManager と呼ばれる Android のコアシステムに，呼び出したアプリケーションの UID，パーミッション名が通知される．PackageManager は通知された情報を基に許可/拒否の判定を返答する．

2.4 パーミッションフレームワークの課題

Android のパーミッションに関する様々な研究がなされているが下記 2 つの課題がある．

課題 1：パーミッションの粒度

ユーザは，アプリケーションのインストール時に，アプリケーションが要求するすべてのパーミッションを許可する必要があり，パーミッション単位で承認することができない．また，1 つのパーミッションにリンクする機能や情報が複数あり，パーミッションの粒度が粗い．たとえば，READ_PHONE_STATE と呼ばれるパーミッションでは，電話番号，IMEI，IMSI，SIM シリアル番号という端末の利用者情報に関わるアクセス権を定義している．ユーザによっては，電話番号が漏えいするのは好ましくないが，端末識別子などの個人に直接コンタクトできない情報は許可してもかまわないというケースも考えられる．より細粒度のパーミッションの制御が可能であることが望ましい．

課題 2：パーミッション利用の透明性

パーミッションに付随する機能や情報が，アプリケーションの実行中に，「いつ」，「どのような機能や情報に」，「だれが」，アクセスするのか，ユーザは知ることができない．そのため，ユーザがアプリケーションの実行に必要だと判断してインストール時に承認したパーミッションが，実はアプリケーション内でユーザの意図しない動作に利用されていた場合に，それを検知することはきわめて難しい．この課題に関しては，Android では特に，パーミッション不正利用型マルウェアの問題，第三者ライブラリの問題が顕著である．

- パーミッション不正利用型マルウェア

パーミッション不正利用型マルウェアとは，アプリケーションに与えられたパーミッションを用いて，ユーザの意図とは異なる不正を働くマルウェアである．アプリケーションにパーミッションが必要だとユーザが判断して，アプリケーションをインストールするため，現行の Android パーミッションフレームワークをすり抜ける．たとえば，SMS のメールクライアントアプリの場合，SMS の送信を許可するパーミッションである SEND_SMS の使用が AndroidManifest.xml ファイルの中で宣言されていることは正しいと考えられる．しかしながら，このアプリケーションが，メールクライアントアプリとして正規の振舞い

をしている中で，バックグラウンドで勝手にSMSを送信していた場合，ユーザが気づくことはきわめて難しい．このように，正規の振舞いをしている裏で，ユーザの意図しない動作を行うアプリケーションに関しては，インストール時にユーザ自身がパーミッションを承認しているため，パーミッション機構による制御が働かない．アプリケーションがどのようにパーミッションを利用しているのかユーザが把握できる仕組みが必要となる．

- 第三者ライブラリ

Androidアプリケーションは，第三者が提供するライブラリを組み込み，アプリケーションの機能を拡張することができる．特に，広告機能を拡張するため，広告事業者 [6], [7] が提供する広告ライブラリを組み込む場合が多い．広告ライブラリの中には，ユーザに効果的な広告を表示するため，ユーザを一意に特定するID，位置情報，ユーザが選択した広告種別などを収集・分析するものもある．ユーザ情報の収集に関して，どのような情報を，何のために，どこに送信するか，ユーザに提示し同意をとる必要があるが，ユーザに無断で情報収集を行うケースもあり問題となっている．総務省では，ユーザが安心安全にアプリケーションを活用できるように，アプリケーション開発者，ライブラリ開発者に対して，スマートフォンを経由した利用者情報の取扱いに関する提言を行った [8]．また，Androidでは，アプリケーション単位に割り振られたUIDを元にパーミッションの制限をかけている．第三者ライブラリはアプリケーションに組み込まれるため，アプリケーションと第三者ライブラリを切り分けてパーミッション管理を行うことができない．そのため，パーミッションを「だれが」利用しているのか区別することができず，パーミッション利用の透明性がない．

3. 提案

本章では，2.4節で説明した課題を解決するため，パーミッションで保護されているAPI単位でフックを行い，スタックフレーム，ユーザ承認に基づく細粒度アクセス制御方式を提案する．まず，概要としてパーミッションフレームワークの課題に対する対応方針を説明する．その後，スタックフレームを利用した細粒度アクセス制御，ユーザビリティ向上のためのセキュリティポリシについて詳細に説明する．

3.1 概要

パーミッション粒度の課題，パーミッション透明性の課題を解決するため以下の対応を行った．

- パーミッション粒度の課題

パーミッション粒度の課題を解決するため，パーミッションに紐付くAPIに対してフック処理を追加し，パーミッションの粒度をAPI単位で制御できるようにする．

- パーミッション透明性の課題

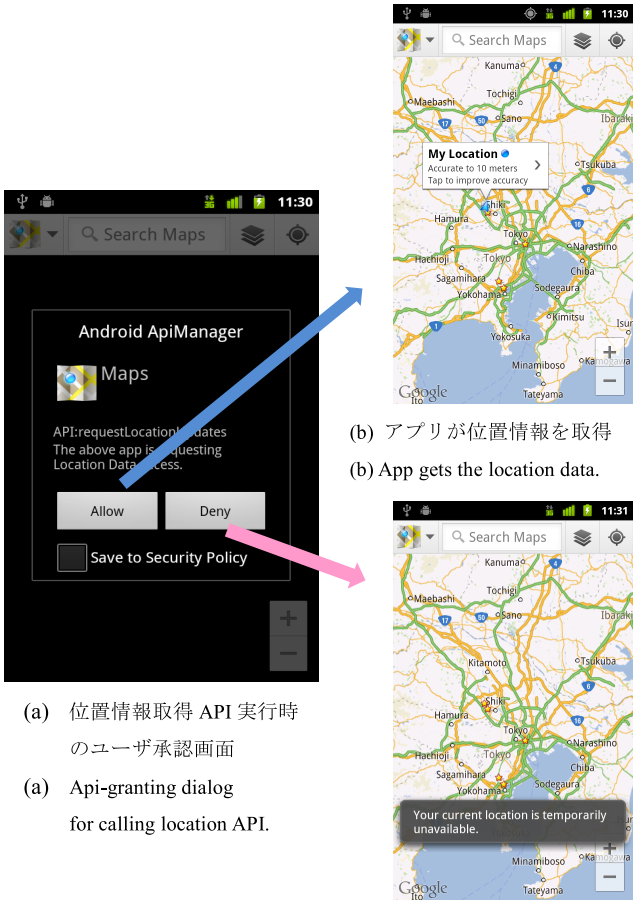
パーミッション透明性の課題を解決するため，提案方式では，APIコール時にスタックフレームを活用したユーザ承認を導入する．スタックフレームとは，APIコール元のメソッド，クラス，プログラム行番号の情報が再帰的に蓄積された情報である．スタックフレームを活用することで，APIコールの発行元がアプリケーション本体なのか第三者ライブラリなのか判定することや，APIコールのコンポーネントを判定することができる．これの詳細は3.2節で説明する．API単位でフックを行い，APIコール時にユーザ承認を導入することで，アプリケーションが「いつ」，「どのような機能や情報に」アクセスするのか把握することができる．また，スタックフレームを活用することで，APIコールの発行元が「だれか」をユーザから把握することができる．さらに，パーミッション利用の目的に関して，APIコールのタイミングやアプリケーション動作の特性から推測することができる．たとえば，SMSメールクライアントアプリケーションにおいてSMS送信時にAPI利用の確認画面が出現した場合，SMS送信のためにパーミッションが利用されるのだということを，APIコールのタイミングから判断できる．また，広告ライブラリから位置情報取得のAPIがコールされた場合，ターゲティング広告の目的のためにパーミッションが利用されるのだということを，アプリケーション動作の特性から推測することができる．もし，このアプリケーションのプライバシポリシの中に位置情報の利用に関する説明が謳われていなかった場合には，この位置情報の取得は不正と見なすことができるだろう．

上記のアクセス制御機構を実現するため，Androidを拡張し，スタックフレーム取得処理の追加，パーミッションに関係するAPIへフックメソッドを追加，ユーザ承認画面を提示する制御アプリケーションの作成，および，Androidフレームワークと制御アプリケーションを連携させるサービスの実装を行う．実装に関する詳細は4章で説明する．

図 2 は，提案方式を実装したAndroid上で，Google社が提供するMapsアプリケーション [9] を動作させた場合の例である．Mapsでは，起動時に位置情報を取得するAPIをコールする．このAPIをコールすると，図2(a)のようにユーザ承認画面を表示する制御アプリケーションが呼び出され，アプリが実行するAPIの動作をユーザの選択に基づき変更させることができる．ユーザがAPIのコールを許可した場合，図2(b)のようにアプリケーションは位置情報を利用した機能をユーザに提供することができる．ユーザが拒否した場合，図2(c)のようにアプリケーションは位置情報を取得できない．

3.2 スタックフレームを活用した細粒度制御

本方式は，スタックフレームを活用することによって，パーミッション利用に関する透明度を高め，細粒度のア



(a) 位置情報取得 API 実行時のユーザ承認画面

(a) Api-granting dialog for calling location API.

(b) アプリが位置情報を取得

(b) App gets the location data.

(c) アプリが位置情報を取得できない

(c) App can't get the location data.

図 2 マップアプリの動作例

Fig. 2 Example of Maps behavior.

クセス制御を達成する．Java では，getStacktrace という API で，スタックフレームを取得することができる．このスタックフレーム情報を利用することで，API コールの発行元の判定，アプリケーションのコンポーネント判定を行う．それぞれについて説明する．

- API コールの発行元判定

Android アプリケーションでは，アプリケーションを識別するパッケージ名を開発者が指定する．一般的にアプリケーションのソースコードは，パッケージ名配下のパスに作成する．一方，広告などの第三者ライブラリは，そのライブラリ事業者が設定したパスにソースコードを作成する．そのため，スタックトレースのクラス情報から，アプリケーションのパッケージ名が含まれていないクラスが API をコールしたことが分かった場合，第三者ライブラリからの API コールであると判断してその API コールを拒否するなど，必要に応じたアクセス制御を行うことができる．

- アプリケーションのコンポーネント判定

スタックフレームの情報を利用することで，API をコールしたコンポーネントが Activity か Service か識別することができる．API が ActivityThread クラスの handleCreateService から呼び出されている場合は，Service による API コールであると判定できる．Service はユーザから見えないバックグラウンド処理を実行するため，Service からの API コールは，Activity からの API コールと比べ，ユーザの意図とは異なる動作が実行されている可能性が強まる．たとえば，SMS を送信する sendTextMessage メソッドは，ユーザがメール本文を記述するため，画面を持つ Activity から送信されるケースが多い．よって，Service から sendTextMessage が実行された場合は，ユーザの意図しない SMS 送信であると判断し，ユーザに承認を問うなどのアクセス制御を行うことができる．このように，Service から通常利用されない API がコールされた場合に，ユーザ承諾を用いて API コールの許可/拒否することは有効である．

3.3 セキュリティポリシ

アプリケーションが API をコールするたびにユーザ承認画面が表示されると，ユーザの利便性が著しく低下する場合がある．これを緩和するため，本方式においては，セキュリティポリシによる自動承認機構を併設する．

本方式では，前節で説明したように，スタックフレームを活用した API コールに対する細粒度アクセス制御を実現する．これに合わせ，下記の粒度でセキュリティポリシを設定する機能を持たせる．

- API 単位でのコールの許可/拒否
- 第三者ライブラリからの API コールの許可/拒否
- アプリケーションコンポーネントに着目した API コールの許可/拒否
- パーミッション単位での API コールの許可/拒否

ここで，ユーザビリティを考慮し，アプリケーションごとに API を指定して許可/拒否を設定することや，インストールするアプリケーションのすべてに対して一括で API を指定して許可/拒否を設定できるようにする．これらの設定は，データベースとして制御アプリケーションが管理する．

本データベースは学習機能を有しており，アプリケーションが API をコールした際にデータベースが作成されていなかった場合には，いったん制御アプリケーションにユーザ承認画面を表示させてユーザ自身にその API コールの許可/拒否の判断を求め，そのつどのユーザ承認の結果を記録していくことでポリシを自動構築していく．当然ながら，制御アプリケーションには，セキュリティポリシをユーザ自身が編集する機能も設けられる．

4. 実装

提案方式を Google Nexus S，Android2.3.5 に実装した．3 章で説明したように，提案方式は，スタックフレーム取得処理，パーミッションで保護された API へのフックメソッ

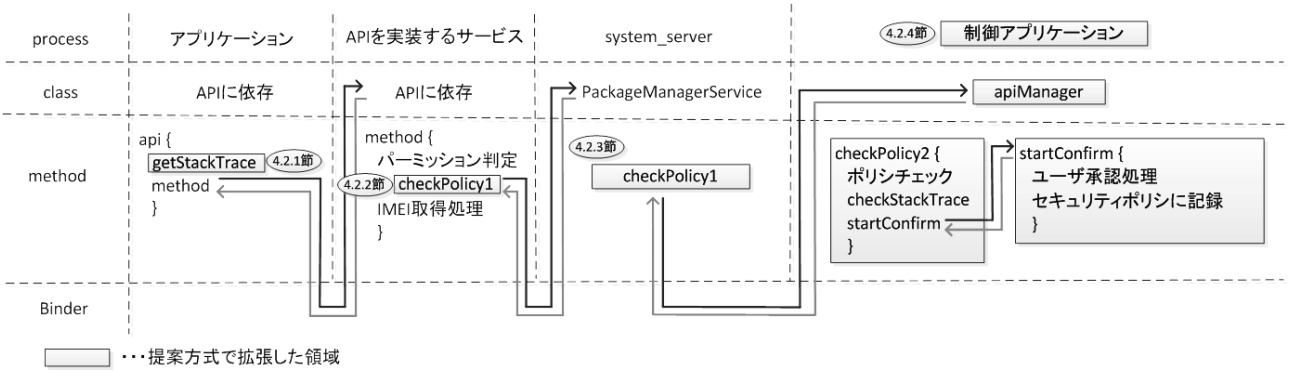


図 3 提案方式の API コールの遷移図
Fig. 3 Transition diagram of API call in proposed scheme.

ド，ユーザ承認を実行する制御アプリケーション，Android フレームワークと制御アプリケーションを連携するサービスを現行 OS に追加することで実現した．まず，実装方針について説明する．その後，具体的な細粒度アクセス制御機構の実装，フックポイントについて説明する．

4.1 実装方針

Android においてアプリケーションがパーミッションで保護された API をコールする場合の手順を以下に示す．

1. アプリケーションは Application Framework で提供される API をロードし，コールする．
2. API がコールされると，Binder を通して，その API を実装しているサービスが呼び出される．
3. API を実装するサービス内では，API コール元のアプリケーションがパーミッションを持つか判定するため，PackageManager に問合せを行う．
4. API コール元アプリケーションがパーミッションを持つ場合は，サービスの処理を実行しアプリケーションに返答する．パーミッションを持たない場合は，パーミッションエラーをアプリケーションに返答する．

アプリケーションによる API コールを制御する場合，上記の手順の中に何らかの処理を加える必要がある．API コールの制御だけを考えた場合，アプリケーションは Binder を利用して API を実装するサービスをコールすることから，Binder をフックし，API コールの制御機構を構築する方法が考えられる．この方法の利点として，Binder に対してフック処理を追加するだけで，API コールを制御できるため，既存のコードへの修正箇所を少なくすることができる．しかしながら，Binder をフックする方法では，API コールに関するスタックフレームを取得できないことや，Binder を利用したすべての API コールに制御の負荷がかかりパフォーマンス低下の懸念がある．そこで，本方式では API ごとにフック処理を追加する方針とした．このようにすることで，スタックフレームの取得が可能となり，制御したい API のみに制御の負荷がかかるためパフォーマンス低下の懸念を少なくすることができる．一方欠点として，フックしたい API ごとに制御処理を追加する必要があるが，API ごとに追加するコード量を小さくする設計を行い対処した．具体的な実装方法について次節で説明する．

4.2 細粒度アクセス制御機構の実装

本方式の実装を図 **3** に示す．四角で囲まれた箇所が新規に開発した要素である．以下，この図 3 をもとに順を追って説明する．

4.2.1 スタックフレーム取得処理の追加

アプリケーションは Application Framework で提供される API をコールする．本方式では，アプリケーションがコールする API を拡張して，getStackTrace のコールによってスタックフレームを取得し，API を実装するサービスに Binder 経由でスタックフレームを受け渡す処理を追加した．このようにすることで，アプリケーションのスタックフレームが API を実装するサービスに受け渡される．次項で，API を実装するサービスに追加した要素を説明する．

4.2.2 API を実装するサービスへのフックメソッドの追加

API を実装するサービスでは，まず API コール元がパーミッションを持つか判定を行い，サービスの処理を実行する．本方式では，パーミッション判定コードの直後にフックメソッドである checkPolicy1 を追加した．この checkPolicy1 は PackageManagerService クラスに実装した checkPolicy1 メソッドを Binder 経由でコールする．checkPolicy1 メソッドは，API コール元アプリケーションの UID，コールした API 名，スタックフレームを引数に持つ．メソッドの返り値は，API_GRANTED, API_DENIED であり，API_GRANTED の場合はサービス本来の処理を実行する．API_DENIED の場合は API コール元にエラーの返答を行う．また，値を取得するような API コールであった場合は，null を返答する．

このように，提案方式ではフックしたい API に対して，スタックフレームの取得処理を追加したうえで，API を

実装するサービスにおいて，パーミッション判定直後にcheckPolicy1 メソッドのコールを挿入することによって，その API コールに対する許可/拒否を制御する処理を実現する．そのため，フックしたい API ごとにこの処理を追加する必要があるが，少ないコードで実装することができる．checkPolicy1 メソッドの実体について次項で説明する．

4.2.3 Android フレームワークと制御アプリケーションを連携するサービス

インストールされたアプリケーションの管理やパーミッション判定などの役割を持つ PackageManager のサービス（PackageManagerService クラス）を拡張して，checkPolicy1 メソッドの実体を追加した．

checkPolicy1 メソッドは，API コール元アプリケーションの UID，スタックフレームを制御アプリケーションに伝達する役目を担う．すなわち，checkPolicy1 メソッドは，制御アプリケーションとして実装された apiManager クラスの checkPolicy2 メソッドを Binder 経由でコールする．checkPolicy2 メソッドについて次項で説明する．

4.2.4 制御アプリケーションの動作

制御アプリケーションは，セキュリティポリシを確認する機能，および，ユーザ承認画面を表示する機能からなる．このため，apiManager というクラスを新たに定義し，セキュリティポリシを確認するための checkPolicy2 メソッドと，スタックフレームを確認するための checkStackTrace メソッドと，ユーザ認証画面を表示するための startConfirm メソッドおよび画面インタフェースを作成した．セキュリティポリシを確認する機構は，他のアプリケーションからコールされる頻度が高いため，端末起動時に常駐する Service として実装した．ユーザ承認画面を表示する機構は，ユーザとインタラクトするため，Activity として実装した．データベースは，SQLite で作成しセキュリティポリシを管理する．

apiManager クラスの checkPolicy2 メソッドは，以下の 2 つの動作をする．

1. セキュリティポリシの確認

API コール元アプリケーションの UID，コールした API 名，スタックフレームをもとにセキュリティポリシが定義されているかデータベースに確認する．スタックフレームを活用した API 実体判定，コンポーネント判定についてもここで実行される．セキュリティポリシが定義されていた場合は，それに応じた応答を PackageManagerService クラスの checkPolicy1 メソッドに返答する．定義されていなかった場合は，ユーザ承認処理の 2 に手順を進める．

2. ユーザ承認処理

ユーザ承認画面を提示する startConfirm メソッドをコールする．startConfirm メソッドは，ユーザに対して API の使用許可を求めるダイアログを表示する．ユーザが許可（API_GRANTED）/拒否（API_DENIED）の返答をすると，checkPolicy2 メソッドは，その結果を PackageManagerService クラスの checkPolicy1 メソッドに返答する．また，ユーザの返答をセキュリティポリシ内のデータベースに，「アプリケーションの UID，API 名，API をコールしたアプリケーションのクラス名，API をコールしたアプリケーションのコード行番号，アプリケーションのコンポーネント種別，可否」の形式で記録する（ユーザは，今回の返答をセキュリティポリシに記憶しないというオプションを選択することもできる）．

4.2.5 ContentProvider に対する制御

Android において，アドレス情報，SMS 情報，ブラウザの閲覧履歴，カレンダ情報は，ContentProvider で管理される．アプリケーションから ContentProvider で管理される情報を取得する場合，ContentResolver クラスの query メソッドの第 1 引数に取得したいデータの URI を記述し取得する．そこで本方式では，ContentProvider へのアクセス制御として，ContentResolver クラスの query メソッドを拡張した．具体的には，query メソッドがアプリケーションからコールされた際に第 1 引数の URI を確認して，checkPolicy1 メソッドを呼び出し，アクセス制御を行う．

4.3 フックポイント

Android ではパーミッションで保護された API が多数存在する．本方式では，Android 利用者にアプリケーションを安心して利用してもらうため，利用者情報や，プライバシ情報の保護を目的とした．そこで，総務省の提言を参考に，利用者情報取得に注目してアクセス制御機構を構築した．本方式で注目したパーミッション，フックした API を**表 1** に示す．ContentProvider のアクセス制御は query メソッドの第 1 引数の URI から判定するため，第 1 引数の URI を記述した．もし API コールのフックポイントを増やしたい場合は，4.2 節で説明したように，フックしたい API に対して，スタックフレームの取得処理の追加，API を実装するサービスにフックメソッドを追加することで細粒度アクセス制御機構を構築することができる．

5. 評価

本章では，API のコール頻度に関する調査と提案方式の制御にかかるオーバヘッドの計測からパフォーマンス評価を行う．本方式では，現行 OS の API に制御を加えているため，制御にかかるオーバヘッドが発生する．まず 5.1 節で，API のコール頻度に関する調査をし，5.2 節で 1 回の API コールに加算される処理時間を評価し本方式の有効性を議論する．

5.1 API のコール頻度

本方式でフックした API のコール頻度が高い場合，制御にかかるオーバヘッドが積み重なり，パフォーマンスに影

表 1 提案方式のフックポイント
Table 1 Proposed scheme of hook placements.

パーミッション	クラス	API	説明
READ_PHONE_STATE	TelephonyManager	getDeviceSoftwareVersion, getDeviceId, getSimSerialNumber, getSubscriberId, getLine1Number, getLine1AlphaTag, getVoiceMailNumber, getVoiceMessageCount, getVoiceMailAlphaTag	端末固有 ID，契約者情報を取得する API
ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION	LocationManager	requestLocationUpdates, getLastKnownLocation, addNmeaListener	位置情報を取得する API
GET_ACCOUNTS	AccountManager	getAccounts, getAccountsByType, hasFeatures, getAccountsByTypeAndFeatures	Web サービスのアカウント情報を取得する API
GET_TASKS	ActivityManager	getRecentTasks, getRunningTasks	アプリケーション利用履歴情報を取得する API
SEND_SMS	SmsManager	sendTextMessage, sendMultipartTextMessage, sendDataMessage, copyMessageToIcc, updateMessageOnIcc, getAllMessageFromIcc	SMS 送信に関する API
READ_CONTACTS	ContentResolver	query 第一引数：com.android.contacts, contacts, call_log	アドレス情報，通話履歴を取得する API
READ_SMS	ContentResolver	query 第一引数：mms, sms	SMS 情報を取得する API
READ_HISTORY_BOOKMARK	ContentResolver	query 第一引数：browser	ブラウザ閲覧履歴を取得する API
READ_CALENDAR	ContentResolver	query 第一引数：com.android.calendar	カレンダー情報を取得する API

響を与える可能性がある．そこで著名な 3 つのアプリケーションについて，本方式でフックした API のコール頻度を調査した．具体的には，本方式でフックした API に対して，スタックフレームをデバッグログに出力するメソッドを追加した OS を作成し，その OS 上で室内で実際にアプリケーションを手動で操作し，デバッグログから API コールを解析した．調査端末は，Google Nexus S，Andorid2.3.5 である．調査したアプリケーションは，Maps，Facebook [10]，Angry Birds [11] であり，調査結果を**表 2** にまとめた．本方式を利用することで，アプリケーションがどのような API を，いつ，だれが，コールするか分かり，パーミッション利用の透明度が高まっていることが分かる．なお，今回調査したアプリケーションにおいては，定常的にコールされる API は存在しなかった．各アプリケーションの詳細について，それぞれ説明する．

5.1.1 Maps の API コール頻度

Maps は Google 社が提供する地図アプリケーションである．Google Play によると 2013 年 2 月 20 日の時点で，1〜5 億ダウンロードされている．Android では位置情報を取得する API として，requestLocationUpdates，addGpsStatusListner，getLastKnownLocation がある．requestLocationUpdates，addGpsStatusListner は，端末所有者が移動し，位置情報が変更されたときに OS から位置情報通知を受け取るイベントリスナを登録する API である．そのため，定期的にコールされる API ではなく，Maps でもアプリケーション起動時に 1 回コールされるだけであった．getLastKnownLocation は，端末で最後に取得した位置情報を取得する API であり，Maps では起動時に 1 回コールされた．getAccountByType は端末のアカウント情報を取得する API であり，Maps の起動時に 4 回コールされた．Maps では，お気に入りのお店などを地図に登録する機能があり，Google アカウントを利用して管理を行うため API を利用していると考えられる．

5.1.2 Facebook の API コール実態

Facebook は，Facebook 社が提供するソーシャルネットワークサービスのアプリケーションである．Google Play によると 2013 年 2 月 20 日の時点で，1〜5 億ダウンロードされている．Facebook では，初回起動のアカウント登録時，アカウント登録後の起動時，携帯の電話帳から友達検索時，付近の情報検索時に API コールが発生した．

- アプリケーション初回起動のアカウント登録時

Facebook のソーシャルネットワークサービスを利

表 2 著名な 3 アプリケーションの API コール頻度調査
Table 2 Survey of API call frequency in three famous applications.

App	API コールのタイミング	API のコール元	API	コール回数
Maps 6.14.2	アプリ起動時	アプリ本体	requestLocationUpdates	1
			addGpsStatusListener	1
			getLastKnownLocation	1
			getAccountsByType	4
Facebook 2.2.1	初回起動のアカウント登録時	アプリ本体	getLine1Number	1
			getAccounts	1
			getAccountsByType	4
			requestLocationUpdates	2
			getLastKnownLocation	2
	アカウント登録後の起動時	アプリ本体	requestLocationUpdates	2
	携帯の電話帳から友達検索実行時	アプリ本体	getLine1Number	1
			query(Contact)	4
			query(call_log)	1
	付近の情報実行時	アプリ本体	requestLocationUpdates	1
Angry Birds 3.0.0	アプリ起動時	com.burstly.lib.util.Utils	getDeviceId	2
		com.millennialmedia.android.MMAdViewSDK	getDeviceId	2
	ゲームを開始し広告表示時	com.flurry.android.FlurryAgent	requestLocationUpdates	1
			getLastKnownLocation	1
			getDeviceId	1

用するためには，Facebook の個人アカウントが必要である．Facebook 初回起動時にすでに作成済みの Facebook アカウントを入力すると，getLine1Number, getAccounts, getAccountsByType, requestLocationUpdates, getLastKnownLocation の API コールが発生した．Facebook アカウント管理のために利用していると考えられる．

- アカウント登録後のアプリケーション起動時

Facebook アカウント登録後のアプリケーション起動時には，requestLocationUpdates の API コールが 2 回発生した．

- 携帯の電話帳から友達検索実行時

Facebook では，友達とつながるため，携帯の電話帳から友達検索をする機能がある．この機能を利用した際に，getLine1Number，Contact，call_log の API コールが発生した．

- 付近の情報実行時

Facebook では，付近の情報を検索する機能がある．これを実行すると現在地近傍のお店などの情報を得ることができる．この機能利用時に requestLocationUpdates の API コールが 1 回発生した．

5.1.3 Angry Birds の API コール頻度

Angry Birds は Rovio Entertainment 社が提供するゲームアプリケーションである．Google Play によると 2013 年 2 月 20 日の時点で 1〜5 億ダウンロードされている．Angry Birds では，アプリケーション起動時と広告表示時に API コールが確認された．Angry Birds を起動すると，2 つの外部ライブラリから，それぞれ IMEI を取得する getDeviceId が 2 回コールされた．また，ゲームを開始し広告表示の際に，getDeviceID が 1 回コールされ，位置情報を取得する requestLocationUpdates，getLastKnownLocation がそれぞれ 1 回コールされた．広告表示のタイミングで API がコールされたため，利用者を特定する ID として IMEI を利用し，位置情報に基づく広告を表示するため位置情報を取得していると考えられる．

5.2 API の制御にかかるオーバヘッドの評価

本方式でフックした API がコールされると，セキュリティポリシを参照するため，現行 OS と比較し API コールのオーバヘッドが発生する．ここで，セキュリティポリシが定義されていない場合，定義されている場合のオーバヘッドについて評価した．なお，計測には Google Nexus S，Android2.3.5 に本方式を組み込んだ OS を利用した．それぞれについて説明する．

- セキュリティポリシが定義されていない場合

セキュリティポリシが定義されていない場合は，ユーザにそのつどの確認が求められる．このため，フックした API がコールされ，セキュリティポリシを参照し，ユーザ承認画面が表示されるまでの時間が，ユーザの待ち時間となり，利便性に影響を与えかねない．そこで，この時間を

表 3 getDeviceId をコールした際のオーバヘッド
Table 3 Overhead on getDeviceId API call.

	現行 OS	提案方式		
登録ポリシ数	−	1,000	10,000	100,000
所要時間 [msec]	0.352	1.158	1.159	1.166

1,000 回測定し待ち時間の平均値を求めた．結果として，約 0.641 sec と小さい時間でユーザに承認画面が表示されたため，ユーザの利便性に影響を与えない範囲で収まると考えられる．

- セキュリティポリシが定義されている場合

セキュリティポリシが定義されている場合は，API コールの可否が自動判定されるため，セキュリティポリシを参照するオーバヘッドがパフォーマンスの低下に直結する．そこで API コール許可のセキュリティポリシが事前に定義されているとし，アプリケーションから本方式でフックした API をコールした際の動作速度を測定する．具体的には，アプリケーションが IMEI を取得する getDeviceId をコールした際に，本方式の機構が働いてセキュリティポリシを参照したうえで API のコールが許可され，アプリケーションが IMEI の値を取得するまでの時間を計測した．また，セキュリティポリシ登録数が増加した際のデータベース検索処理時間も評価するため，本方式においては，セキュリティポリシに登録するレコード数を変化させながら計測した．getDeviceId を各 1,000 回コールし，その 1 回あたりの平均値を**表 3** に示す．

計測の結果，約 1.2 msec 以内と非常に小さい時間で API コールが完了することが判明した．データベースの検索処理時間に関しても，1,000 件と 100,000 件を比較して，8 μsec の差しかないため，データベースサイズによる検索処理時間は問題にならないと考えられる．データベース検索処理時間が 8 μsec という非常に小さい差異になったが，これは SQLite データベースがメモリ上に展開されたまま，検索処理が実行されたためだと考えられる．

5.1 節の調査結果より定常的にコールされる API がないこと，ユーザ承認画面表示までの待ち時間が約 0.6 sec と小さい時間であること，セキュリティポリシが設定されている場合の API コールの処理時間が約 1.2 msec と非常に小さいことから，本方式のパフォーマンスへの懸念は少ないと考えられる．なお，セキュリティポリシの登録件数は，1 アプリケーションあたり，アプリケーション本体からの API コールとアプリケーションに組み込まれた第三者ライブラリからの API コールの中で表 1 に該当するものの数である．端末にインストールされているアプリケーションの数を 100 個と想定し，すべてのアプリケーションが 3 つの第三者ライブラリを組み込んでいると仮定する．これらのアプリケーションおよび第三者ライブラリが表 1 のすべての API をコールすると仮定すると，端末に設定されるセキュリティポリシは，$100*(1+3)*31 = 12{,}400$ 件となる．

6. 考察

6.1 スタックフレーム情報の制限事項

getStacktrace は Java の API であるため，ネイティブアプリケーションが API を呼び出した場合には，スタックフレームが取得できず，API のコール元実体判定，アプリケーションのコンポーネント判定ができないという制限がある（ただし，その場合も，API 単位でのアクセス制御は行われるため，現行 OS よりも詳細なアクセス制御は達成される）．

6.2 アプリケーションに返答する値

現時点では，API コールについてユーザが拒否を選択した場合，null をアプリケーションに返答している．しかしながら，アプリケーションの中には null の返答を想定した設計がなされていないものがあり，アプリケーションが強制終了してしまう場合がある．これに対しては，ダミーの値をアプリケーションごとに生成して返答する手法が考えられる．このようにすることで情報漏えいを防ぎつつ，アプリケーションの動作に影響を与える可能性を小さくすることができる．

6.3 ユーザに提示する情報

本方式では，アプリケーションの API コールに関して，コールした API 名，API コールの発行元情報，アプリケーションのコンポーネント情報，など詳細な情報を取得することができる．しかしながら，これらの情報すべてをユーザに提示すると，ユーザの判断が難しくなる可能性がある．本方式の目的は，パーミッション利用の透明性を高めるため，「いつ」，「どのような機能や情報を」，「だれが」をユーザに伝えることである．そのためユーザに提示する情報として，API がコールされたときに，API 名，API の説明，API コール実体元，を提示することにした．また，本来 Service からコールされるべきでない API コールであった場合にのみ，ユーザに注意を促す情報を提示し，詳細なコンポーネント種別は提示しないことにした．

6.4 ユーザビリティ

提案方式は，ユーザビリティの観点で 2 つの課題をかかえている．

1. ユーザへの API コール可否の問合せが頻繁に発生すると，ユーザはつねに許可を選択する．
2. Android に慣れていないユーザは，API コールに関する承認画面が表示されても，許可/拒否のどちらを選択したらよいか判別つかない．

どちらの課題に関しても，セキュリティポリシを適切に

定義することが可能であれば問題を緩和できると考えられるが，ユーザごとにセキュリティポリシの調整を施すことは難しく，今後の検討が必要と考えられる．

7. 関連研究

スマートフォンのセキュリティに関して様々な研究がなされている．特にアプリケーションによる情報漏洩の危険性が指摘されている [12], [13], [14], [15], [16], [17], [18], [19] ことから，アプリケーションの解析手法や端末側の対策で情報漏洩を防ぐ仕組みが提案されている．アプリケーションによる情報漏洩の可能性を分析する方式として，データフロー解析を利用して動的解析する手法 [16], [17] や静的解析する手法 [18], [19] が提案されている．

スマートフォンのセキュリティに関する既存研究のもう1つの大きな潮流が，パーミッションに関する研究である．これまでに，アプリケーションの持つパーミッションの組合せから危険性を判断し，ユーザに提示する手法 [2]，特定の場所にいるときのみパーミッションが有効になるなどパーミッションに詳細な制限を持たせる手法 [3]，アプリケーションのインストール時にユーザからパーミッションを許可・拒否できる手法 [4]，パーミッションをユーザから変更する仕組みに加えて，アプリケーションがパーミッションにアクセスする順番に着目して不正行為を検知する仕組み [5]，アプリケーションに対して静的解析を実施し，過剰にパーミッションを利用していないか判定する手法 [20]，パーミッションモデルに関して一般ユーザの理解を調査した報告 [21]，パーミッションモデルにおいてどのようなユーザ承認が適切か調査した報告 [22] などがある．

提案方式は，アプリケーションのパーミッション利用に関して API コール時にユーザ承認を設けること，スタックフレームを活用し「だれが」API コールするか判定することでパーミッション利用の透明性と粒度の詳細化を達成している点で既存研究が解決していない課題に取り組んでいる．提案方式と同様のコンセプトで API フックによるアクセス制御機構を実現するアプリケーションとしては，LBE Privacy Guard [23] がある．しかし，LBE Privacy Guard は，スタックフレーム情報を利用しておらず，3.3.1 項で説明したような細粒度な API 制御が実現できていない．また，LBE Privacy Guard は root 権限を必要とする．1 度 root 権限をアプリケーションに与えてしまうと，悪用された場合のリスクが非常に大きいため，OS の基本機能としてアクセス制御機構を構築することが望ましいと考える．

Android では，端末の root 権限を奪取し不正を行うマルウェア [24] も存在する．Android の Kernel Layer や Middle Layer の脆弱性を突くコードを実行し，端末の root 権限を奪取するマルウェアである．文献 [15] によると 2010 年 8 月～2011 年 11 月に収集した 1,260 種のマルウェアの中で 36.7%のアプリケーションが root 権限を狙うアプリケーションであったという報告がある．提案方式は，パーミッションフレームワークに関する細粒度アクセス制御機構であるため，root 権限奪取型マルウェアは対象外である．この root 権限奪取型のマルウェアの対策としては，Security Enhanced Linux を Android に応用した SEAndroid（Security Enhanced Android）[25] の仕組みが有効である．SEAndroid では，一般権限から root 権限で動作するプロセスへの干渉を防ぐことで，マルウェアによる root 権限奪取の難易度を高め，さらに root 権限をマルウェアに奪取されたとしても，利用できる権限を Kernel Layer で制御することでマルウェアの動作を抑制する．

8. おわりに

本論文では，アプリケーションの機能や情報へのアクセスを制御するため，Android のパーミッションで保護されている API に対してフックを仕掛け，リアルタイムで動的制御を行う方式を提案した．API 単位でフックを行うことにより，アプリケーションの実行中にアプリケーションがどのようなパーミッションの機能や情報を要求したのか把握できるようになり，ユーザがアプリケーションの API 利用に対して動的に承認を与えるというアクセス制御が可能になる．特に，スタックフレームを活用することで，「だれが」API をコールするのか把握でき，パーミッション利用の透明性を高め，細かい粒度でアクセス制御を達成していることが本方式の特徴である．API コール頻度調査，セキュリティポリシが設定されている場合，設定されていない場合の提案方式のオーバヘッドに関する評価を行い，提案方式によるアプリケーションのパフォーマンス低下の懸念も少ないことを確認した．

参考文献

[1] Android, available from ⟨http://www.android.com⟩.
[2] Enck, W., Ongtang, M. and McDaniel, P.: On lightweight mobile phone application certification, *Proc. 16th ACM Conference on Computer and Communications Security*, *CSS'09* (2009).
[3] Ongtang, M., McLaughlin, S.E., Enck, W. and McDaniel, P.D.: Semantically Rich Application-Centric Security in Android, *Proc. 25th Annual Computer Security Applications Conference*, *ACSAC'09* (2009).
[4] Nauman, M., Khan, S. and Zhang, X.: Apex: Extending Android permission model and enforcement with user-defined runtime constraints, *5th ACM Symposium on Information Computer and Communications Security*, *ASIACCS'10* (2010).
[5] Banuri, H., Alam, M., Khan, S., Manzoor, J., Ali, B., Khan, Y., Yaseen, M., Tahir, M.N., Ali, T. and Zhaug, X.: Android Runtime Security Policy Enforcement Framework, *2010 International Workshop on Smartphone Applications and Services*, *Smartphone'10* (2010).
[6] AdMob, available from ⟨http://www.admob.com⟩.
[7] Millennial Media, available from ⟨http://www.

millennialmedia.com⟩.
[8] 総務省：「スマートフォン プライバシー イニシアティブ—利用者情報の適正な取扱いとリテラシー向上による新時代イノベーション—」の提言
[9] Maps, available from ⟨https://play.google.com/store/apps/details?id=com.google.android.apps.maps⟩.
[10] Facebook, available from ⟨https://play.google.com/store/apps/details?id=com.facebook.katana⟩.
[11] Angry Birds, available from ⟨https://play.google.com/store/apps/details?id=com.rovio.angrybirds⟩.
[12] Barrera, D., Kayacik, H.G., van Oorschot, P.C. and Somayaji, A.: A methodology for empirical analysis of permission based security models and its application to Android, *17th ACM Conference on Computer and Communications Security*, *CCS'10* (2010).
[13] 葛野弘樹：HTTP トラフィックを利用したクラスタリングによる Android アプリケーションの分類，研究報告コンピュータセキュリティ（CSEC），情報処理学会，2012-CSEC-56(19) (2012).
[14] Grace, M., Zhou, W., Jiang, X. and Sadeghi, A.-R.: Unsafe Exposure Analysis of Mobile In-App Advertisements, *Conference on Security and Privacy in Wireless and Mobile Networks*, *WiSec'12* (2012).
[15] Zhou, Y. and Jiang, X.: Dissecting Android Malware: Characterization and Evaluation, *2012 IEEE Symposium on Security and Privacy* (2012).
[16] 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 Symposium on Operating Systems Design and Implementation*, *OSDI'10* (2010).
[17] Schreckling, D., Posegga, J., Köstler, J. and Schaff, M.: Kynoid: Real-Time Enforcement of Fine-Grained, User-Defined and Data-Centric Security Policies for Android, *Proc. 6th Workshop in Information Security Theory and Practice*, *WISTP'12* (2012).
[18] Enck, W., Octeau, D., McDaniel, P. and Chaudhuri, S.: A study of Android application security, *20th USENIX Security Symposium* (2011).
[19] Mann, C. and Artem, S.: A framework for static detection of privacy leaks in android applications, *Proc. 27th Symposium on Applied Computing*, *SAC'12* (2012).
[20] Felt, A.P., Chin, E., Hanna, S., Song, D. and Wagner, D.: Android permissions demystified, *18th ACM Conference on Computer and Communication Security*, *CCS'11* (2011).
[21] Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E. and Wagner, D.: Android Permissions: User Attention, Comprehension, and Behavior, *Symposium on Usable Privacy and Security*, *SOUPS'12* (2012).
[22] Felt, A.P., Egelman, S., Finifter, M., Akhawe, D. and Wagner, D.: How TO Ask For Permission, *USENIX Workshop on Hot Topics in Security*, *HotSec'12* (2012).
[23] LBE Privacy Guard, available from ⟨https://play.google.com/store/apps/details?id=com.lbe.security.lite⟩.
[24] Ginger Master, available from ⟨http://www.cs.ncsu.edu/faculty/jiang/GingerMaster/⟩.
[25] SEAndroid, available from ⟨http://selinuxproject.org/page/SEAndroid⟩.

推薦文

Android を搭載した端末では，アプリケーションをインストールする際，利用する機能や情報をパーミッションという単位でユーザに通知し承認を得る．しかし，実行中のアプリケーションがパーミッションをどのように利用しているのか把握できないことや，いったんアプリケーションのパーミッションを承認すると，実行中はその制御ができないことが問題としてあげられる．本論文では，実行中のアプリケーションに対して，利用する機能や情報へのアクセスをリアルタイムでユーザに通知し，制御可能なフレームワークを提案しており，その実装と評価を行っている．本論文は，Android のセキュリティ向上に大きく貢献すると考えられるため，推薦論文として推薦する．

（コンピュータセキュリティシンポジウム 2011
プログラム委員長 四方順司）

川端 秀明 （正会員）

2010 年東海大学大学院工学研究科情報通信制御システム工学専攻前期博士課程修了．同年 KDDI 株式会社入社．現在，株式会社 KDDI 研究所にて，モバイル OS セキュリティ，ネットワークセキュリティに関する研究開発に従事．2011 年 CSS 優秀論文賞，2012 年 DICOMO 優秀論文賞・優秀プレゼンテーション賞，2013 年山下記念研究賞を各受賞．

磯原 隆将 （正会員）

2005 年慶應義塾大学理工学部情報工学科卒業．2007 年同大学院理工学研究科修了．同年 KDDI 株式会社入社．現在，KDDI 研究所にて，ネットワークセキュリティおよびスマートフォンセキュリティに関する研究開発に従事．電子情報通信学会会員．

竹森 敬祐 （正会員）

1994 年慶應義塾大学理工学部電気工学科卒業．1996 年同大学院修士課程修了．同年現 KDDI 入社．2004 年慶應義塾大学大学院博士課程修了．現在，KDDI 研究所に勤務．ネットワークセキュリティ，スマートフォンセキュリティに関する研究開発に従事．2002 年度 IEICE 学術奨励賞，2006 年 Interop グランプリ，2007 年 DICOMO 最優秀論文賞・最優秀プレゼンテーション賞，2009 年 MWS 優秀論文賞，2010 年山下記念研究賞等受賞．IEEE，電子情報通信学会各会員．日本スマートフォンセキュリティ協会アプリ WG リーダ．

窪田 歩 （正会員）

1995 年京都大学大学院工学研究科情報工学専攻前期博士課程修了．同年国際電信電話株式会社（現 KDDI）入社．2003 年米国カリフォルニア大学バークレイ校客員研究員．現在 KDDI 研究所にて，ネットワークセキュリティの研究開発に従事．

可児 潤也

2012 年静岡大学情報学部情報科学科卒業．現在，同大学院修士課程．情報セキュリティに関する研究に従事．

上松 晴信

2011 年静岡大学情報学部情報科学科卒業．2013 年同大学院修士課程修了．同年 KDDI 株式会社入社．在学中，情報セキュリティに関する研究に従事．

西垣 正勝 （正会員）

1990 年静岡大学工学部光電機械工学科卒業．1992 年同大学院修士課程修了．1995 年同博士課程修了．日本学術振興会特別研究員（PD）を経て，1996 年静岡大学情報学部助手．同講師，助教授の後，2006 年より同創造科学技術大学院助教授．2007 年同准教授，2010 年同教授，2013 年同大学院情報学研究科教授．博士（工学）．情報セキュリティ全般，特にヒューマニクスセキュリティ，メディアセキュリティ，ネットワークセキュリティ等に関する研究に従事．2013 年より情報処理学会コンピュータセキュリティ研究会主査．