Vol.2013-CSEC-62 No.22
Vol.2013-SPT-6 No.22
2013/7/18

Androidにおける実行コンテキストによるパーミッション切り替え手法の提案

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

概要：ソフトウェアに組み込んで動作を拡張及び支援するアドオン，モジュール，外部ライブラリ等が存在する．これらが原因でソフトウェア内部で障害が発生する場合がある．本稿では現在普及の進んでいる Android のアプリケーションに焦点を当てる．Android では障害防止の為にアプリケーションに対してパーミッションを設定して動作を制限している．しかし，パーミッションはアプリケーション全体に対してしか設定できないため，外部ライブラリにパーミッションが余分に与えられる可能性がある．本稿では，この問題を解決するために実行コンテキストに応じて開発者が自由にパーミッションを切り替えられるフレームワークを提案する．提案システムを導入することで，それぞれのコンテキスト毎に最小のパーミッションを与えることが可能となる．

キーワード：Android，Permission，権限分離，広告ライブラリ

A Permission Switching Method based on the Execution Context in Android

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

Abstract: There are modules, add-ons, and external libraries that extend and support the functionality of software. These sometimes cause incidents. In this paper, we focus on Android applications that are becoming popular. In Android, a permission mechanism limits behavior of applications against incidents. However, it has a possibility to give libraries too many permissions because it can set permissions only the entire application. We propose a novel framework that application developers can switch permissions based on execution context. In this framework, the developers can set least permissions to each context.

Keywords: Android, Permission, Privilege separation, Ad library

1. はじめに

ソフトウェアに組み込んでその動作を拡張することができるアドオン，モジュール，外部ライブラリ等の機能拡張ソフトウェアが多数存在している．例えば，Firefox 及び GoogleChrome 等のアドオン，Linux のカーネルモジュール，Android の広告ライブラリ等である．これらの機能拡張ソフトウェアは主となるソフトウェアとは別の開発者によって開発されることが多い．

しかし，機能拡張ソフトウェアが障害の原因となる場合が存在する．Apache ではマルウェアに感染したモジュールを導入したサーバが悪意ある html ファイルを配る事件 [1] が発生したり，Google Chrome では悪質な拡張機能に対してユーザに警告を行う仕組み [2] を提供している．Android では，アプリケーション（以下，アプリという）に対して広告ライブラリを組み込むことができ，この広告ライブラリを用いることで広告料を得ることができる．広告ライブラリがあることによって無料のアプリが開発費を回収する

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan University
[a)] hioki@matlab.nitech.ac.jp

ことが可能となる．しかし，Android では広告ライブラリが無断で個人情報を収集する事件 [3] が発生したり，広告ライブラリに対してセキュリティリスクが存在することが報告されている [4].

機能拡張ソフトウェアの不具合や悪意による異常動作が引き起こす問題は，プロセス（アプリ）に対する権限付与方法と，プロセス内部での権限制御が十分に機能していないことが原因の一つと考える．まず，プロセスの権限管理について考察すると，ケーパビリティがある．これは，Linux ではカーネルケーパビリティ，FreeBSD では Capsicum[5] として実装されている．Linux や FreeBSD では，ケーパビリティ機構を利用しない場合，root 権限を有するプロセスの動作に制限はない．そこでアプリプログラマは，各プロセスが必要最低限の権限のみを利用できるように，ソースコード中にケーパビリティ機構を記載する．これにより，当該プロセスの制御が乗っ取られた場合においても設計外の動作を抑制して被害を局所化することができる．ここでプロセス内部の権限分離に注目すると，Linux カーネルケーパビリティはプロセス全体を制御する機能であるため，プロセスの一部分の権限を制御することはできない．Capsicum では，ファイルディスクリプタ単位での制御が可能であり，これを利用してアプリ内の一部機能のみを対象にした権限制御が可能であるが，ディスクリプタをどのコンテキストで使用して良いかの制御までは行っていない．一方，Android では，パーミッション機構によりアプリの権限管理を行っている．パーミッション機構では，Android の機能や情報へのアクセス権としてパーミッションを設定する．このパーミッションの利用制御は，初期状態ではアプリはパーミッションを持たないが，アプリ開発者が必要なパーミッションを宣言して，利用者が承認することで，パーミッションがアプリプロセス全体に適用される．プロセスの一部機能に限定して制限することはできない．以上により，ケーパビリティや Capsicum は，プロセスが元々有する権限（root 権限）の一部を制限する仕組みであるのに対して，パーミッション機構は新たな権限を付与する仕組みであり，権限制御に対して逆の思想であると言える．さらに，プロセス内部の権限分離機能を有する Capsicum の実装も限定的であり，実行コンテキストを意識した権限制御は不十分といえる．

権限管理としてのケーパビリティは，権限を削る方向にのみ機能するため，セキュリティ強化として容易に適用できる．しかし，パーミッション機構は，新たな権限を付与する機能であるため，場合によってはセキュリティを低下させる．しかも，その判断はユーザである一般利用者に委ねられている．以上により，本稿では，安全なプロセスの権限管理機構を実現するために，権限付与を行う Android のパーミッション機構に対する権限分離方法と権限付与方法について提案する．

提案システムは，アプリ開発者を信頼することで，実行コンテキストに応じてパーミッションの組み合わせを切り替えることができる仕組みを提供する．これにより，ユーザの介在を必要とせずに，広告ライブラリによるアプリのパーミッションを悪用した動作を制限する．提案システムは，アプリ開発者がパーミッションの組み合わせを複数設定し，ソースコード内部で切り替えるメソッドを挿入することで実現する．パーミッションの切り替え時には実行コンテキストのスタックを用いることで，アプリの主たる処理と広告ライブラリによる処理を識別し，それぞれに適したパーミッションのみを利用可能とする．これにより，ユーザによる確認無しで広告ライブラリによるパーミッションの悪用がないセキュアなアプリを提供することが可能になる．また，アプリ開発者も自身のアプリをライブラリが原因となる障害から守ることができる．

以下，2 章で関連研究，3 章で Android のパーミッション機構とその問題点について述べ，4 章で広告ライブラリについて述べる．5 章と 6 章で提案システムとその実装について述べ，7 章で評価と考察を述べる．最後に 8 章でまとめる．

2. 関連研究

本章では，権限分離に関する研究と Android のパーミッション機構，広告ライブラリについての研究について述べる．

2.1 権限分離に関する研究

Linux ではケーパビリティによって root 権限で動作するプロセスの権限を部分的に制限する仕組みが存在する．このケーパビリティを実現した方式の一つに FreeBSD における Capsicum[5] がある．Capsicum ではファイルディスクリプタに対してケーパビリティを設定することでプロセス内部での権限分離を実現している．Capsicum ではまず，ケーパビリティモードと呼ばれるモードに入ることで新しいファイルディスクリプタを作成不可能にする．そして，あらかじめ作成しておいたファイルディスクリプタに対してケーパビリティを設定する．ケーパビリティは再設定することができるが，権限を減らす変更のみが可能である．Capsicum を用いることで，プロセスの制御が奪われた場合でもファイルへの被害を最小限にすることができる．また，機能拡張ソフトウェアに渡すファイルディスクリプタに対してケーパビリティを設定することで，機能拡張ソフトウェアの動作に制限を設けて安全な利用を実現できる．Capsicum はソースコードレベルのセキュリティであり，プログラムに制御用関数を挿入するだけで簡単にセキュリティを向上させることができる．

提案システムも同じソースコードレベルでのセキュリティであるといえる．しかし，Capsicum がファイルディ

スクリプタ単位での権限分離であるのに対して，提案システムは実行コンテキスト単位での権限分離である．例えば，書き込み可能なファイルディスクリプタに対しても，書き込むべきではない実行コンテキストにおける書き込みを禁止することができる．

Upro[6] はオブジェクトファイル単位でファイルへのアクセス権限を管理するシステムである．Upro では，ソースコードとは別にアクセス権限を設定するファイルを記述する．このアクセス制限設定ファイルとソースコードを一緒に独自コンパイラでコンパイルすることで保護ドメインが設定され，オブジェクトファイル単位でのアクセス権管理を実現している．この手法では，ライブラリに対しても個別に保護ドメインを設定することができるため，ライブラリの権限を小さくすることが可能である．

しかし，オブジェクトファイル単位の権限分離では，オブジェクトファイルの持つ機能が増加するにつれて，一つのオブジェクトファイルに与える権限が大きくなる．このため，細かな権限分離が困難である．一方，提案システムでは一つのオブジェクトファイル（パッケージ）が複数の機能を有する場合でも，それぞれの機能を利用する際に個別に権限を付与することができる．また，提案システムは Android における Framework 層の一部の書き換えとマニフェストファイルの書式変更のみでシステムを実現しており，コンパイラの変更は必要ない．

2.2 Android のパーミッションに関する研究

AndroidOS において，アプリのインストール後，実際にパーミッションを必要とする動作が発生した時にユーザに確認を行うことができる仕組み（API Manager）[7] が提案されている．API Manager ではアプリ動作時のユーザ確認機能に加えて，ユーザがアプリの要求するパーミッションに対して個別に許可・拒否を設定する機能を有している．しかし，ユーザがアプリのパーミッションに関する知識を十分持っていない場合には，動作時の確認やパーミッションの設定において正しい判断をすることは難しい．また，アプリ内のどのようなクラスからパーミッションを必要とする動作が呼び出されたか分からないため，呼び出し元に基づく判断ができない．

Android 標準のパーミッションに対してポリシを設定する研究 [8], [9] がある．これらはパーミッションに対してユーザが設定したポリシによってアプリの動作を制限する．しかし，このポリシ設定はユーザが行う必要があるため，ユーザの負担が大きい．また，これらの方式は，アプリ全体に対して設定を適用するため，広告ライブラリのパーミッションだけを取り消すことはできない．提案システムではアプリ開発者がパーミッションに関する設定を行うため，ユーザが行う処理は一切なく，さらに，特定のライブラリに限定した制限が可能な点が大きく異なっている．

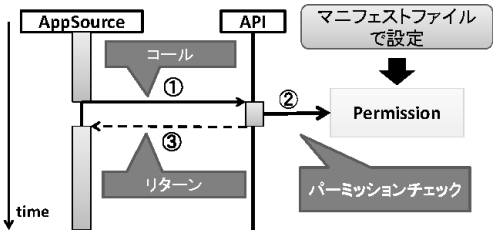

図 1 パーミッション機構の概要

2.3 広告ライブラリに関する研究

広告ライブラリに関する研究に AdDroid[10] と AdSplit[11] がある．AdDroid は広告ライブラリのためのパーミッションを定義し，アプリ内部で AdDroid が広告のためのネットワーク通信を代わりに行う仕組みを提案している．これにより，アプリ開発者は広告ライブラリのためのパーミッションを設定する必要がなくなり，ユーザはより安全なアプリを利用することができる．

AdSplit は完成したアプリを再度コンパイルすることで，広告ライブラリをアプリから分離し，別プロセス上で動作させる．アプリと広告ライブラリのプロセスが分かれるため，広告ライブラリが余分なパーミッションを利用することができない．

これらの研究は提案システムと同様にパーミッションの分離を行っているが，いずれもクラス単位での分離である．一方，提案システムはクラス単位ではなく実行コンテキスト単位で実行権限の分離を行っており，アプリ開発者が切り替えるタイミングを自由に設定できる点が異なっている．

3. Android のセキュリティ

本章では Android のセキュリティシステムの概要とその問題点について述べる．

3.1 概要と動作

Android ではすべてのアプリが DalvikVM 上で実行される．DalvikVM はサンドボックスとして働き，アプリは基本的に他のプロセスや外部の機能，情報にアクセスすることができない．それらの機能を利用するには，パーミッション機構から必要なパーミッションを得る必要がある．パーミッション機構の動作を図 1 に示す．

Android は，機能の利用や情報へのアクセスのために多種多様な API を提供している．API は，図 1 のようにアプリからコール（①）されて，その結果をアプリに対してリターン（③）する．API には，利用するためにパーミッションを必要とするものがある．例えばインターネット接続や電話番号等の情報の取得には対応したパーミッションが必要となる．このため，API 内部ではパーミッションチェック（②）が行われ，必要なパーミッションが与えられていなければその処理を失敗させる．アプリ開発者はマニフェストファイルに必要なパーミッションをあらかじめ記述して

おくことで，アプリインストール時にそのパーミッションがアプリに対して与えられる．マニフェストファイルはアプリの情報（パーミッションやパッケージ名等）が記述され，アプリがインストールされる前にシステムにアプリの情報を伝える働きを持っている．さらに，ユーザはこの情報をアプリインストール前に確認し，インストールするかしないかを判断する．パーミッション機構によって，ユーザの承認が無ければアプリに権限を与えない仕組みを実現している．なお，一度与えられたパーミッションはインストール後に変更することはできない．

3.2 問題点

パーミッション機構には 2 つの問題が存在する．1 つ目は，ユーザがパーミッションに関する十分な知識を持っていない場合があることである．ユーザはアプリに設定されているすべてのパーミッションを承認しなければアプリを利用することができないため，パーミッションをよく確認せずアプリをインストールする場合がある．パーミッション機構では，アプリのパーミッションを必要最低限に限定することでユーザを保護するが，一旦パーミッションが与られば，そのパーミッションをどのように利用するかに関しては感知しない．そのため，アプリは，本来必要ではないパーミッションを要求することで，情報漏えい等を行うことが可能である．しかし，この問題はユーザの意識改善が必要であり，解決することは困難である．

2 つ目は，パーミッション機構はパーミッションをアプリ単位でのみ設定できるため，広告ライブラリに余分なパーミッションが与えられる問題である．そのため，広告ライブラリがアプリ本体用のパーミッションを誤用または悪用することが原因で，障害が発生する場合がある．しかし，アプリ開発者は広告ライブラリの詳細を把握しているわけでは無いため，導入時に広告ライブラリの詳細な動作を判別することは困難である．そこで，提案システムでは 1 つ目の問題を考慮しつつ，2 つ目の問題を解決するために，権限分離機構の提案を行う．

4. Android の広告ライブラリ

本章では Android の広告ライブラリの概要について述べる．アプリ開発者が広告ライブラリを利用するためには，まず広告会社にアカウント登録等を行い，広告会社が公開している広告ライブラリの Software Development Kit（SDK）を入手する．そして，広告ライブラリをインポートし，マニフェストファイルに広告ライブラリに必要なパーミッションを記述する．最後にアプリ内部で広告ライブラリのクラスインスタンスを作成し，広告を表示するためのメソッドを呼び出すことで広告が表示される．広告ライブラリのインスタンスの作成方法は，ソースコード中に記述する方法とレイアウト xml ファイルに記述する方法の 2 つがある．1 つ目のソースコード中に記述する方法は，通常の Java クラスと同様に new 演算子を用いてオブジェクトを作成する．2 つ目のレイアウト xml ファイルに記述する方法は，Android では View クラスのインスタンスを xml ファイルから作成可能であることから，広告ライブラリが Veiw クラスを継承して xml ファイルからのインスタンス生成形式に対応していれば，xml ファイルからのインスタンス作成が可能である．インスタンスの作成後は，広告ライブラリのメソッドを呼び出して広告の操作ができる．なお，広告ライブラリの提供元によってサポートしている利用方法の詳細は異なる．

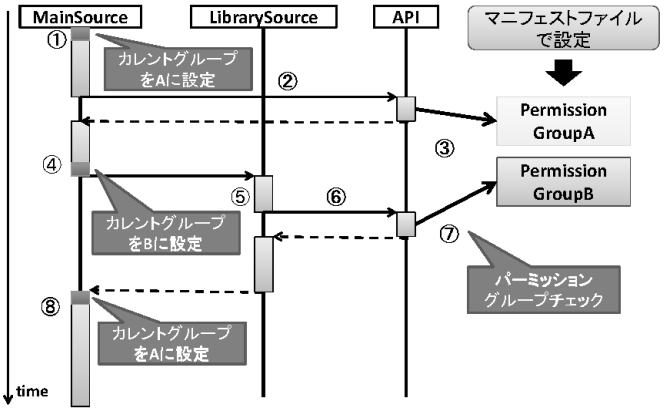

図 2 提案システムの動作概要

5. 提案システム

提案システムは，パーミッション機構の適用範囲がアプリ全体のみである問題を解決するために，アプリ開発者を信用することで，アプリ開発者がパーミッションの組み合わせを実行コンテキストに応じて変更することができるフレームワークを提案する．提案システムによって広告ライブラリには広告ライブラリ専用の，アプリ本体にはアプリ本体専用のパーミッションを割り当ててそれぞれを最小権限で実行することが可能になり，アプリ開発者はライブラリが障害が発生する危険性を減らすことができる．また，アプリ本体の内部においても細かくパーミッションを切り替えることができる．これを利用して，アプリが不正に利用された場合であっても被害を最小限にすることも可能である．提案システムは，ソースコード内部に簡単なメソッドを挿入するだけで実現できるため，アプリに大規模な変更が必要なく，アプリ開発者への負担は小さい．また，ユーザの負担を増やさずにより安全なアプリを利用できるようになる．

提案システムの動作概要を図 2 に示す．提案システムでは，アプリに与えるパーミッションの組み合わせをパーミッショングループという．アプリ開発者は，パーミッションが必要なコンテキスト毎に適切なパーミッショングループ（以下，カレントグループという）を設定する．さ

らに，ソースコード内部でカレントグループ切り替え用メソッドを呼び出すことで，カレントグループの切り替えを行う．図 2 の例ではアプリ本体用のパーミッショングループ A と広告ライブラリ用のパーミッショングループ B の 2 つを設定している．アプリ開発者は，カレントグループを A に設定する（①）ことで，アプリ本体からパーミッショングループ A のパーミッション群を利用できるようにする．この時，カレントグループ設定権限が呼び出し元コンテキストに存在するか否かを呼び出し元検証機能を用いて確認する．次に，実際に API が呼ばれた場合（②）には，API 内部で，パーミッション機構のチェックとは別に，必要となるパーミッションがカレントグループに含まれているかをチェック（③）し，含まれていない場合は API の処理を失敗させる．アプリ内部で広告ライブラリによる処理（⑤）を行う場合は，広告ライブラリの API を呼び出す前にカレントグループを B に設定（④）することで，広告ライブラリから呼び出される API（⑥）にはパーミッショングループ B が適用される（⑦）．再度アプリ本体に処理が戻ってきた際には再びカレントグループを A に戻す（⑧）ことで，以降の広告ライブラリ以外の処理にはパーミッショングループ A が適用される．

6. 実装

本章では，提案システムを実現するために実装した項目について述べる．まず 6.1 で実装の概要について述べ，6.2 以降で個別に述べる．なお，Android アプリは Java で記述されるため，提案システムの実装も Java である．

6.1 概要

提案システムを実現するために，以下に示す 4 つの実装が必要となる．

- カレントグループ管理クラス
- カレントグループ管理用メソッド
- マニフェストファイル記述部
- パーミッショングループ検証部

提案システムでは，カレントグループを管理するためにカレントグループ管理クラスを用いる．また，アプリ開発者には，カレントグループ管理用メソッドを提供することでカレントグループ管理クラスへのアクセスを可能にする．カレントグループを設定するメソッドは，広告ライブラリやアプリ開発者の意図しない場所から呼び出されることを防ぐために，特定のクラス（以下，許可クラスという）以外から呼び出せない仕組みを搭載する．マニフェストファイル記述部では，パーミッショングループと許可クラスを記述する拡張を行った．Android ではアプリの設定がマニフェストファイルで行われるため，その仕組みに準拠する．パーミッショングループ検証部では，マニフェストファイルで設定されたパーミッショングループと，処理中のカレントグループの情報を用いてパーミッショングループの検証処理を行う．この処理によって API の動作を制御する．

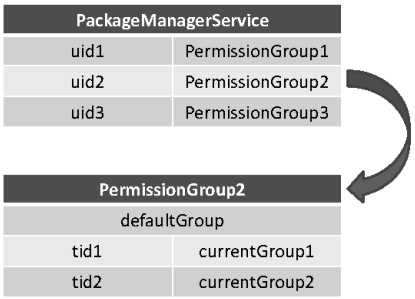

図 3 カレントグループ管理クラス

6.2 カレントグループ管理クラス

提案システムではカレントグループの情報を用いてパーミッショングループ検証を行うため，カレントグループを不正に操作されると正常に動作しない．そこで，カレントグループ管理クラスは PackageManagerService クラスの内部クラスとして実装した．PackageManagerService クラスはパッケージの情報を管理するクラスであり，PackageManager が呼び出された時に処理を行うクラスである．Android では API 呼び出し後，Binder という仕組みを用いてプロセス間通信を行い，実際の処理はアプリとは違ったプロセス上で行う．つまり，PackageManagerService クラスは，アプリとは別のプロセス上で動作するため，そこで管理されるパーミッショングループの情報にアプリがアクセスすることはできない．

図 3 に，PackageManagerService クラスと PermissionGroup クラスの関係を示す．PacakgeManagerService クラスは一つのプロセスであり，提案システムを利用したアプリが複数動作した場合でも，そのすべてのパーミッショングループを保管する．そこで，アプリにユニークに与えられる uid に対して 1 つずつ PermissionGroup クラスのオブジェクトを生成する．図 3 では，uid1，uid2，uid3，の 3 つのアプリが動作している様子を示している．し，それぞれの管理を行っている．PermissionGroup クラスでは，スレッド ID（tid）とカレントグループの組を格納する．これによりスレッド毎のカレントグループを管理している．PermissionGroup クラスの defaultGroup は,tid に対応するカレントグループが設定されていない場合に適用するパーミッショングループである．図 3 では，defaultGroup と，2 つのスレッド tid1，tid2 に，それぞれ currentGroup1 と currentGrup2 というカレントグループが設定されている様子を示している．

提案システムでは，スレッドを作成する Thread クラスを継承して，生成された際にスレッド作成元のカレントグループを新しいスレッドに継承させる PGroupThread クラスを実装した．提案システムを利用するアプリがスレッ

表 1　実装メソッド

setPermissionGroup	カレントグループを設定
getPermissionGroup	現在のカレントグループを確認
checkPermissionGroup	パーミッショングループの検証

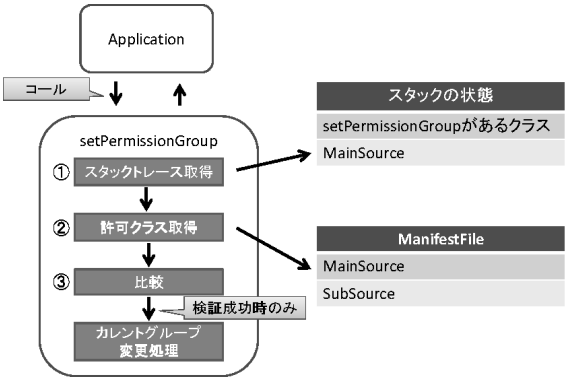

図 4　setPermissionGroup メソッドの呼び出し元検証

ドを作成する場合には，このクラスを使用する必要がある．しかし，広告ライブラリが PGroupThread クラス以外のクラスや，ネイティブ上でスレッドを作成した場合，カレントグループが継承されない．そこで，当該スレッドのカレントグループが存在しない場合のために defaultGroup を実装した．なお，PGroupThread クラスは，Java の Thread クラスをフックし，カレントグループを反映させる処理を追加することでも実現可能である．

6.3　カレントグループ管理用メソッド

図 3 で示した PermissionGroup クラスを，アプリ開発者が利用するためのメソッドを実装した．実装したメソッドとその機能を表 1 に示す．

setPermissionGroup メソッドは，カレントグループを設定するメソッドである．このメソッドは，引数で指定されたパーミッショングループをカレントグループとして設定する．提案システムでは実行コンテキストに合わせてカレントグループを切り替える必要があり，実行コンテキストは，setPermiissionGroup メソッドが呼び出された位置に基づいて決定する．そのため，アプリ開発者が指定した位置以外で setPermiissionGroup が呼び出されて，カレントグループが切り替えられることを防止する必要がある．

アプリ開発者が呼び出したことを確実に判断するために，setPermiissionGroup メソッドに呼び出し元を検証する機能を実装した（図 4 参照）．setPermissionGroup メソッドは呼び出されるとまず，スタックトレースを参照して setPermissionGroup メソッドを呼び出したクラスを取得する（①）．Java ではメソッドが呼び出される度にスタックに情報が格納されており，この情報は Thread クラスの getStackTrace メソッドを用いることで参照することができる．setPermissionGroup メソッド呼び出し時には，スタックは図 4 右上に示すように積まれているため，スタックトップの一つ下のスタックフレームを確認することで，setPermissionGroup メソッドを呼び出したクラスを特定できる．次に，許可クラス（図 4 の右下）を取得する（②）．許可クラスとは，setPermissionGroup メソッドを呼び出せるクラスを定義したものであり，パーミッショングループと同様にマニフェストファイルでアプリ開発者が設定する．そのため，マニフェストファイルの情報へアクセスすることで許可クラスを得ることができる．なお，アプリ開発者が setPermissionGroup メソッドを挿入するため，許可クラスの設定は容易である．最後に，得られた呼び出し元クラスが許可クラスに含まれているか確認し（③），含まれている場合のみ setPermissionGroup メソッドはカレントグループ変更処理を行う．また，defaultGroup の設定もこのメソッドで行う．

```
①<uses-permission android:name="パーミッション名"/>
②<uses-permission android:name="パーミッション名"
                  android:groupName="グループ名"/>
③<uses-class android:name="クラス名"/>
```

図 5　マニフェストファイルの書式

getPermissionGroup メソッドは現在のカレントグループを取得するメソッドで，checkPermissionGroup メソッドは引数に与えたパーミッションが現在の参照するグループに含まれているかどうかを検証するメソッドである．これらメソッドは無くても十分運用可能であるが，例えば「現在のカレントグループが A だった場合に B にする」等の条件を用いた処理を行いたい場合に利用することができる．

6.4　マニフェストファイルの記述

アプリ開発者はマニフェストファイルに，パーミッショングループの設定を行う．また許可クラスの設定もマニフェストファイルで行う．それぞれの書式を図 5 に示す．

Android の標準的なパーミッション設定は，図 5 の①に示すようにマニフェストファイルの uses-permission 要素にパーミッション名を記述することで行う．パーミッショングループを設定するために，②に示す書式を用いて，uses-permission 要素にパーミッション名に加えてパーミッショングループ名を記述可能とした．パーミッショングループ名にはアプリ開発者が自由な文字列を設定することができる．設定されたパーミッショングループの情報はパーミッションや他のマニフェストファイルの情報とともに PackageManagerService クラスで管理される．

次に許可クラスは，③に示すように uses-class 属性で定義し，その中の name 要素に setPermissionGroup メソッドを呼び出すクラスをフルパスで記述する．setPermissionGroup メソッド内部では，図 4 の②において，スタックトレースを用いて直前の呼び出し元クラスを取得した後，呼び出し元が uses-class 属性で設定されているかどうか検証

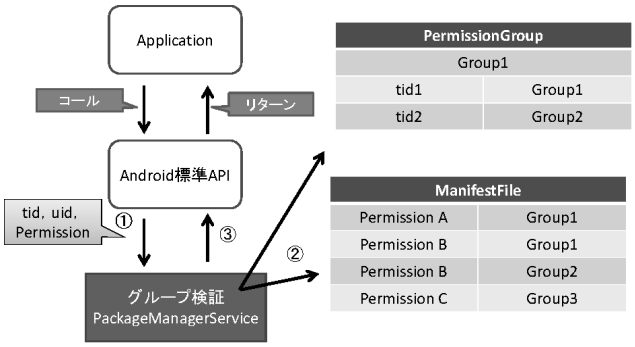

図 6 提案システムのパーミッショングループ検証手順

表 2 評価環境

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

を行うことで，広告ライブラリからの不正な呼び出しを防止する．

6.5 パーミッショングループの検証

パーミッションを必要とする API 利用時のパーミッショングループの検証手順を図 6 に示す．パーミッショングループの検証は，API のパーミッション検証処理とは別に行われる．API が呼び出されると API 自体の処理前にパーミッショングループ検証処理が呼び出される（①）．この時パーミッショングループ検証処理を呼び出す API からパーミッショングループ検証処理に対して，uid と tid，API 利用に必要となるパーミッション名が引数として渡される．パーミッショングループ検証処理（②）では，PackageManagerService クラスにおいて uid をキーとして PermissionGroup を特定し，tid をキーとしてカレントグループを特定する．なお，図 6 には PermissionGroup のみ記載している．そして PackageManager が管理しているマニフェストファイルに記述された情報へアクセスし，必要なパーミッションが現在のカレントグループに含まれているかどうか確認する．含まれていれば正常に動作を行い，含まれていなければその API の処理を失敗させる（③）．

7. 評価と考察

本章では提案システムの動作の確認と，導入によるオーバーヘッドについて評価し，安全性と Linux への応用の可能性について考察する．評価環境を表 2 に示す．

7.1 動作

提案システムの動作結果のログを図 7 に示す．図 7 は，Android 標準 API である getLine1Number を，1 度目はアプリ本体である MainSource 上から，2 度目は広告ライブラリを想定した自作ライブラリである LibrarySource

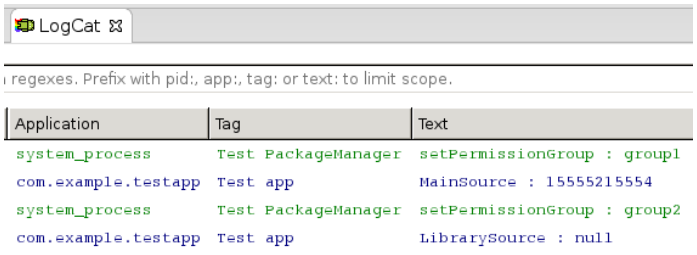

図 7 提案システムの動作結果

表 3 各メソッドの処理時間

API	時間（msec）
setPermissionGroup	0.742
getPermissionGroup	0.182
checkPermissionGroup（検証処理）	0.254
checkPermission（参考）	0.216

上から呼び出している．この API を利用するためには，*read_phone_state* というパーミッションが必要となる．既存のパーミッション機構ではパーミッションを与えれば，そのパーミッションを必要とする API を広告ライブラリからも呼び出すことができたが，図 7 に示すようにパーミッショングループを切り替えることで 2 度目の API の呼び出しを失敗させている．以上のようにパーミッショングループを切り替えることで，アプリ内であっても適切にパーミッションを適用することができる．

7.2 提案システムのオーバーヘッド

提案システムでは，カレントグループ管理用メソッドをソースコード内部に挿入するため，カレントグループ切り替えのための処理時間が増加する．また，パーミッションを必要とする API 実行時にはパーミッショングループの検証を行うための処理時間も増加する．各 API を 10000 回実行した時間を測定し，その平均処理時間を表 3 に示す．なお，パーミッショングループ検証の処理内容は，checkPermissionGroup メソッドにエラー処理を加えたものである．なお参考として，Android 標準 API である checkPermission メソッドの処理時間も測定した．

評価の結果，いずれも 1msec に満たない時間で処理が完了した．検証処理はアプリに設定されたパーミッションの数とパーミッショングループの数に応じて処理時間が増加すると考えられる．パーミッション数に関しては，パーミッション数に比例した処理時間がかかる標準 API の checkPermission メソッドと比較しても差は小さく，アプリに与える影響は小さいと言える．また，パーミッショングループ数に関しては，何百ものパーミッショングループが作成されることは考えにくいため影響は軽微である．他の 2 つのメソッドに関しては，スレッド数に比例した処理時間が必要である．しかし，何百ものスレッドが利用されることは考えにくいため，実際の処理時間への影響は軽微

であるといえる．

7.3 提案システムの安全性について

提案システムでは，カレントグループの切り替えをソースコード内のメソッドによって行う．そのため悪意ある外部ライブラリ作者が，不正にカレントグループを切り替える問題が考えられる．Java では，リフレクションを用いることでカプセル化を越えてメソッドの呼び出しやメンバの読み取り，ある条件下でのメンバへの書き込みが可能となる．これを防止するために，図 3 に示したクラスへ実装することで，別プロセス上でカレントグループの管理を行い，リフレクションを用いたメンバへのアクセスを防いでいる．また，リフレクションを用いて setPermiissionGroup メソッドが呼び出された場合でも，許可クラスとスタックトレースの確認によってカレントグループの切り替えを防止することができる．なお，許可クラス内でカレントグループ切り替えを行うためだけのメソッドを実装することは，そのメソッドを対象にしてリフレクションで呼び出される可能性があるため危険である．

7.4 提案システムの Linux への適用可能性について

提案システムを Linux へ適用することは可能である．しかし，Linux では，スタックが偽装される危険性があることが問題である．提案システムでは，実行コンテキストの切り替えとカレントグループの切り替えを一致させるために，setPermissionGroup メソッドの呼び出し時に呼び出し元を確認することで，広告ライブラリ等のアプリ開発者以外が開発したクラスからの呼び出しを防止している．この時，呼び出し元の確認にスタックトレースを用いている．提案システムではアプリが JavaVM の一種である DalvickVM 上で動作しており，VM で管理されたスタックを参照しているためスタックを偽造することは難しい．しかし，Linux システム上のスタックは偽造される可能性があるため，実行コンテキストが詐称される場合がある．そのため Linux ではケーパビリティのように権限を減らす方向のシステムは実現可能であるが，提案システムのように権限を与える方向のシステムの実現は難しいと考えられる．

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に対して余分なパーミッションが与えられる問題に対する提案を行った．多くの研究ではユーザに確認や設定を依頼することで与えすぎたパーミッションやその一部を取り上げる手法であったが，ユーザが十分な知識を持っていないため正確な判断をすることは困難である．そこで，提案システムでは，アプリ開発者を信頼することで十分な知識を持たないユーザへの確認を行うことなく，最小のパーミッションでアプリを提供できるフレームワークを提案した．提案システムにより，広告ライブラリを含む実行コンテキスト毎に適したパーミッションを与えることが可能となる．評価によって，提案システムの処理時間は軽微であり，アプリ開発者が行う設定も少量であるため提案システムの運用は十分可能であることを示した．

参考文献

[1] おさまらぬ Web 改ざん被害，Apache モジュールの確認を：http://www.atmarkit.co.jp/ait/articles/1303/25/news122.html(2013/6/10 アクセス).
[2] Google Chrome、悪質な拡張機能を「マルウェア」として警告：http://www.itmedia.co.jp/news/articles/1304/18/news080.html(2013/6/10 アクセス).
[3] ネットラジオ Pandora が個人情報を広告に無断活用？そのほか多数のアプリも関係か：http://japanese.engadget.com/2011/04/08/pandora/(2013/6/10 アクセス).
[4] Grace, M. C., Zhou, W., Jiang, X. and Sadeghi, A.-R.: Unsafe exposure analysis of mobile in-app advertisements, *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 101–112 (2012).
[5] Watson, R. N. M., Anderson, J., Laurie, B. and Kennaway, K.: Capsicum: practical capabilities for UNIX, *Proceedings of the 19th USENIX conference on Security* (2010).
[6] Niu, B. and Tan, G.: Enforcing user-space privilege separation with declarative architectures, *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pp. 9–20 (2012).
[7] 川端 秀明, 磯原 隆将, 窪田 歩, 可児 潤也, 上松 晴信, 西垣 正勝：Android OS における機能や情報へのアクセス制御機構の提案，コンピュータセキュリティシンポジウム 2011 論文集，pp.161-166 (2011).
[8] Nauman, M., Khan, S. and Zhang, X.: Apex: extending Android permission model and enforcement with user-defined runtime constraints, *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pp. 328–332 (2010).
[9] Ongtang, M., McLaughlin, S., Enck, W. and McDaniel, P.: Semantically rich application-centric security in Android, *Security and Communication Networks*, Vol. 5, No. 6, pp. 658–673 (2012).
[10] Pearce, P., Felt, A. P., Nunez, G. and Wagner, D.: AdDroid: privilege separation for applications and advertisers in Android, *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security* (2012).
[11] Shekhar, S., Dietz, M. and Wallach, D. S.: AdSplit: separating smartphone advertising from applications, *Proceedings of the 21st USENIX conference on Security symposium* (2012).
[12] PandaBoard ES: http://pandaboard.org/content/pandaboard-es.