Androidにおけるランタイムパーミッションのクラス別適用手法

岡野 圭史[1]　瀧本 栄二[2]　毛利 公一[2]　齋藤 彰一[1]

概要：Android では端末の情報や機能の利用をパーミッションという権限により制御する．しかしパーミッションはアプリ全体に適用されるため，広告をはじめとするライブラリが不要な権限を使用できる問題がある．この問題に対する先行研究として，クラスに基づくパーミッション制御機構によりホストアプリとライブラリの権限を分離する手法が存在する．Android 6.0 ではランタイムパーミッションが導入され，ユーザーは権限毎に許可の選択が可能になった．本研究では先行研究をランタイムパーミッションに適用する方式を提案する．提案システムにより，ホストアプリとライブラリそれぞれで権限の許可の選択が可能となる．

キーワード：Android, ランタイムパーミッション, 権限分離, 第三者ライブラリ

1. はじめに

Android のアプリケーション（以下，アプリとする）の開発の補助のため，第三者が作成したライブラリが広く利用されている．例えば，UI を提供するライブラリ，複雑な処理をまとめたライブラリ，広告を表示するライブラリが挙げられる．このようなライブラリはアプリ開発の効率化に貢献し，また，広告ライブラリは無料のアプリに広告を表示することで収入を得られる仕組みを持つため，多くのアプリに組み込まれている．しかし，広告ライブラリで端末内のデータを 1 日に 1.7 種類のサーバーに送信していることが明らかになっている [1]．また，第三者によるライブラリに脆弱性が存在した事例 [2] があり，もし広告ライブラリに脆弱性が存在すれば端末内のデータが盗まれる危険性が高くなる．

Android のアプリは個人情報などの機密データを扱う可能性が高いため，アプリに対するセキュリティの機能としてパーミッションと呼ばれる権限が搭載されている．パーミッションとは，アプリから端末内の情報や機能を使用する権限である．アプリで必要とする権限は，アプリ開発者が開発時にアプリに記載することで使用可能となる．しかし，パーミッションはアプリ全体に対して適用されるという問題がある．このため，アプリで第三者によるライブラリが使用されている場合，ホストアプリとライブラリで使用するパーミッションは区別されない．したがって，ライブラリはホストアプリが要求するパーミッションを使用可能となり，ホストアプリのパーミッションを使用して情報を漏洩させる可能性がある．

パーミッションの適用範囲の問題に対する研究として，クラスの集合を単位とするパーミッション制御機構 [3] がある．この機構では，ホストアプリに属するクラスの集合とライブラリに属するクラスの集合に対するパーミッションの設定を分離する．このパーミッションの分離により，ライブラリによるホストアプリのパーミッションの悪用をユーザーの判断で防止可能である．

また，パーミッションの問題点として，Android 5.1 以前ではパーミッションをアプリのインストール時に一括で要求する点も存在した．ユーザーはアプリをインストールする際にアプリに必要なパーミッションを確認し，すべてのパーミッションを許可することでアプリのインストールが可能であった．アプリのインストール時に一部のパーミッションを拒否したり，また，アプリのインストール後に一部のパーミッションの設定を変更したりすることは不可能であった．したがって，インストール時に 1 つでもパーミッションを拒否する場合はアプリをインストールできず，また，インストール後にパーミッションが不要と判断した場合もそのパーミッションを拒否する場合はアプリを削除する必要があった．すなわちパーミッションを拒否する場合はそのアプリを利用不可能であった．

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan University

パーミッションの要求タイミングの問題に対して，Android 6.0 以降ではランタイムパーミッションと呼ばれる新しいパーミッション制御機構が導入された．ランタイムパーミッションは，アプリの実行時にユーザーに対してパーミッションの要求を行う機構である．ランタイムパーミッションでは，アプリの実行中にパーミッションが必要になった場合に必要なパーミッションのみをユーザーに要求する．ユーザーは要求されたパーミッションに対して許可または拒否を選択可能である．一度設定したパーミッションは Android 標準の設定アプリにより変更が可能である．したがって，ランタイムパーミッションにより権限を細かく制御可能である．しかし，ランタイムパーミッションにおいても適用範囲はアプリ全体であり，ライブラリが使用されているアプリでホストアプリのパーミッションをライブラリが使用できるという問題は残されている．また，ライブラリで使用するパーミッションがアプリの本来の目的と無関係な場合があることも判明している [4].

本稿では，先行研究であるクラスの集合を単位とするパーミッション制御機構を適用したランタイムパーミッション制御機構を提案する．提案システムでは，アプリの開発者がランタイムパーミッションを指定する際にクラスの集合を単位としてパーミッションを指定する．これにより，ホストアプリとライブラリそれぞれでランタイムパーミッションの許可と拒否の選択が可能となる．

以下，2 章で Android のランタイムパーミッションについて述べる．3 章で先行研究であるクラスに基づくパーミッション制御機構について述べ，4 章で提案，5 章で実装の詳細を述べる．6 章で評価を述べ，最後に 7 章でまとめる．

2. Android のパーミッション

本章では，Android のパーミッションについて述べる．

2.1 パーミッションの概要

Android では，電話番号やインターネットなど端末内の情報や機能を使用する権限を管理するパーミッションと呼ばれる制御機構を搭載している．パーミッションが必要な情報や機能をアプリで利用するには，アプリ開発者がアプリの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．マニフェストファイルとはアプリに関する情報が記述されている AndroidManifest.xml である．パーミッションが必要な機能が使用される際に，そのパーミッションが許可されているか検証される．必要なパーミッションがない場合，その API を実行したアプリは強制終了される．アプリのパーミッションは，使用する情報や機能に応じてプロテクションレベルが設定され，プロテクションレベルは次の 4 種類が定義されている．

Dangerous　電話帳など，個人情報を扱うためユーザーに対してパーミッションが必要であることを明示する必要がある．Dangerous であるパーミッションを**表 1** に示す．Dangerous のパーミッションはパーミッショングループにより端末のどの機能を使用するかに応じて分類されている．

Normal　パーミッションが必要であることをユーザーに明示する必要はない．

Signature　アプリの署名が一致する場合のみ使用可能である．

SignatureOrSystem　Android にプリインストールされたアプリでのみ使用可能である．

表 1　Dangerous レベルのパーミッション [5]

パーミッショングループ	パーミッション名
CALENDAR	READ_CALENDAR
	WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS
	WRITE_CONTACTS
	GET_ACCOUNTS
LOCATION	ACCESS_FINE_LOCATION
	ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	READ_PHONE_STATE
	CALL_PHONE
	READ_CALL_LOG
	WRITE_CALL_LOG
	ADD_VOICEMAIL
	USE_SIP
	PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	SEND_SMS
	RECEIVE_SMS
	READ_SMS
	RECEIVE_WAP_PUSH
	RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE
	WRITE_EXTERNAL_STORAGE

2.2 パーミッションの要求タイミングとランタイムパーミッション

アプリに必要なパーミッションはユーザーに通知され許可を求めるが，そのタイミングは Android 5.1 までと Android 6.0 以降で異なる．Android 5.1 以前では，パーミッションはアプリのインストール時にユーザーに要求されていた．アプリのインストールを開始するとインストールの確認画面が表示され，アプリで使用するパーミッションの一覧が表示された．ユーザーは要求されたパーミッションをすべて許可することにより，アプリのインストールが可能だった．なお，アプリのパーミッションを 1 つでも拒否する場合はアプリのインストールは不可能である．

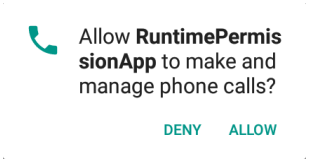

図 1　ランタイムパーミッションの要求ダイアログ

また，インストール後のパーミッションの拒否も不可能である．

Android 6.0 において，ランタイムパーミッションが導入された．ランタイムパーミッションはアプリの実行中に要求される．そのため，アプリのインストール直後のランタイムパーミッションは拒否の状態である．ランタイムパーミッションが必要な場合，**図 1** のようにダイアログが表示され，必要なランタイムパーミッションのみ要求される．ユーザーは表示されたランタイムパーミッションの要求に対して，許可または拒否の選択が可能である．拒否を選択した場合は，そのランタイムパーミッションの必要な情報や機能は使用できない．ランタイムパーミッションの対象となるパーミッションは表 1 に示した Dangerous のパーミッションであり，Dangerous 以外のパーミッションは従来と同様インストール時に要求される．なお，Android 6.0 より古いバージョンを対象とするアプリはランタイムパーミッションに対応していないため，すべてのパーミッションがインストール時に要求される．

2.3　ランタイムパーミッション要求の動作

ランタイムパーミッション要求の動作を**図 2** に示す．アプリがパーミッションを要求するには，ランタイムパーミッション名をアプリのインストールを行う PackageInstaller に送信し，PackageInstaller にランタイムパーミッションの要求を依頼する (図 2 の①)．PackageInstaller は要求元アプリのパーミッションの情報を PackageManagerService から取得し，要求を依頼したアプリがランタイムパーミッションを宣言しているか確認する (図 2 の②)．アプリが当該ランタイムパーミッションを宣言している場合，そのランタイムパーミッションが属するパーミッショングループを取得する (図 2 の③)．ランタイムパーミッションの要求処理はそのランタイムパーミッションが属するパーミッショングループ毎に行う．そのため，メッセージではパーミッショングループに対して許可を求めるように書かれている．

次に，要求するランタイムパーミッションが属するパーミッショングループが PackageManagerService から取得した情報にあれば，そのパーミッショングループに属するランタイムパーミッションへの許可または拒否をユーザーが選択しているかを確認し，選択がない場合にランタイムパーミッションの許可を要求するダイアログを表示する (図 2 の④)．すなわち，アプリが要求したランタイムパーミッションと同じパーミッショングループのランタイムパーミッションが既に許可またはあるいは拒否されている場合，要求したランタイムパーミッションは自動で許可または拒否に設定される．

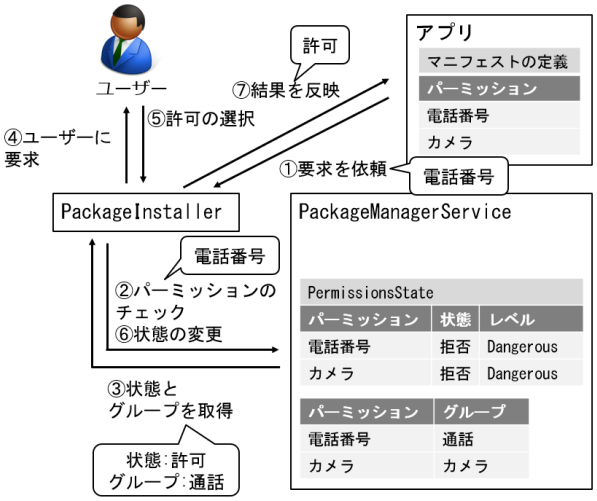

図 2　ランタイムパーミッション要求ダイアログ

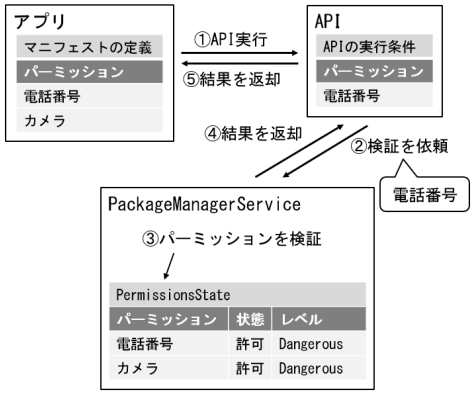

図 3　パーミッション検証

そしてユーザーがダイアログに従いランタイムパーミッションの許可または拒否を選択した後 (図 2 の⑤)，その選択が許可であればランタイムパーミッションの状態を許可に設定する (図 2 の⑥)．要求したすべてのランタイムパーミッションがそれぞれ属するパーミッショングループに対する許可あるいは拒否をユーザーが選択した後，ランタイムパーミッション毎の要求結果を要求元のアプリに返却する (図 2 の⑦)．

2.4　パーミッション検証の動作

パーミッション検証の動作を**図 3** に示す．アプリが API を呼び出すと (図 3 の①)，その API の実行にパーミッションが必要であれば，API は PackageManagerService に対してパーミッションの検証を依頼する (図 3 の②)．API は検証の際に必要なパーミッションの名前を PackageManagerService に渡す．PackageManagerService では，API

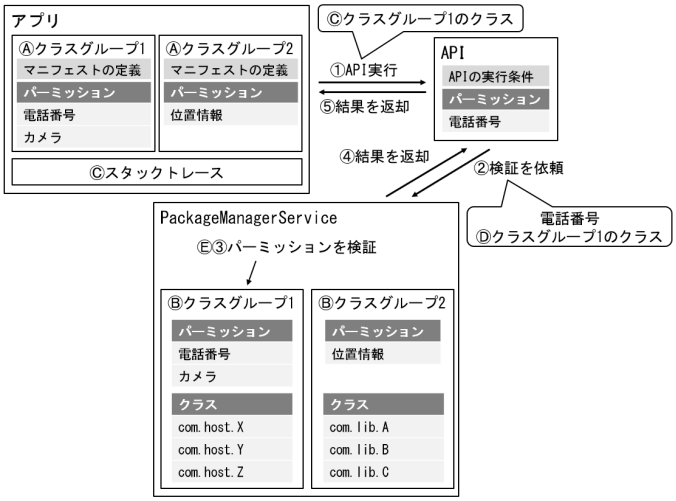

図 4 先行研究の概要

の要求元のアプリのパッケージにある PermissionsState を参照しパーミッションの検証を行い (図 3 の③)，その結果を返す (図 3 の④)．検証の結果，パーミッションが許可されていれば API を実行し (図 3 の⑤)，許可されていなければ例外が発生しアプリは強制終了する．PackageManagerService はアプリパッケージの情報を管理しており，使用するパーミッションの情報は PermissionsState に記憶される．PermissionsState にはアプリの持つパーミッションとそのパーミッションが許可されているか (以下，パーミッションの状態と呼ぶ)，パーミッションのプロテクションレベルの情報を持つ．アプリがインストールされる際に，マニフェストに書かれた情報が PackageManagerService 上にアプリのパッケージ毎に保存される．

2.5 ランタイムパーミッションの状態の変更

ランタイムパーミッションは一度許可または拒否した後でその状態を変更できる．ランタイムパーミッションの状態の変更には Android 標準の設定アプリ Settings を使用する．

3. 関連研究

本章では，先行研究であるクラスに基づくパーミッション制御機構 [3] の詳細を述べる．

3.1 概要

先行研究の概要を**図 4** に示す．Ⓐ〜Ⓔが先行研究で実装された箇所である．先行研究ではクラスの集合の単位をクラスグループと定義し，アプリ，パーミッションの必要な API，パーミッションの検証機能それぞれをクラスグループに対応するように拡張を行っている．アプリ開発者はアプリのマニフェストにクラスグループを記述し，クラスグループそれぞれに対してパーミッションを設定する (図 4 のⒶ)．クラスグループはホストアプリ，ライブラリ毎に分割し設定する．クラスグループを持つアプリをインストールすると，クラスグループ毎のパーミッションが PackageManagerService に記憶される (図 4 のⒷ)．クラスグループを持つアプリを実行しパーミッションの必要な API を実行すると，API はスタックフレームを取得しその API を呼び出したクラス名を取得する (図 4 のⒸ)．API はこのクラス名と必要なパーミッション名を PackageManagerService に渡してパーミッション検証を依頼する (図 4 のⒹ)．PackageManagerService は渡されたクラス名が属するクラスグループをパッケージから検索し，そのクラスグループに API の必要なパーミッションが許可されているかを検証する (図 4 のⒺ)．これにより，ホストアプリで必要だがライブラリで不要なパーミッションをライブラリから使用することは不可能になる．なお，この機構は Android 4.2 に実装されているため，ランタイムパーミッションには対応していない．また，Android 4.2 では PermissionsState は存在せず，パーミッション名のリストによって管理していた．

3.2 クラスグループ

アプリやライブラリは複数のクラスで構成されるため，複数のクラスをクラスグループという単位にまとめて定義している．クラスグループは同一のパーミッションを使用するクラスを 1 つ以上含む．複数のライブラリを使用する場合はライブラリの数だけクラスグループを設定可能である．図 4 の PackageManagerService にクラスグループの例を示す．この例ではホストアプリのクラスの集合 (com.host.*) をクラスグループ 1 とし，2 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の集合 (com.lib.*) をクラスグループ 2 とし，1 つのパーミッションを設定している．複数の外部ライブラリの使用や，ホストアプリ内でさらに細かくパーミッションを設定する場合はさらにクラスグループを追加可能である．

3.3 アプリのクラスグループ対応

アプリでクラスグループを使用するために，マニフェストファイルの書式を変更している．マニフェスト内のパーミッションの記述方法と先行研究におけるマニフェストの記述方法の比較を**図 5** に示す．Android の標準のマニフェストでは，uses-permission でパーミッションを設定する．先行研究では，マニフェストにクラスグループを設定する class-group，その中に記述する要素として必要なパーミッション名を記述する uses-class-permission とクラス名を記述する join-class を追加している．

3.4 パーミッションの必要な API のクラスグループ対応

クラスグループ毎のパーミッションの検証を依頼するために，API を呼び出したクラス名が必要である．その

```
<manifest...>
...
<!--Android 標準の記述-->
<uses-permission
  android:name="android.permission.READ_PHONE_STATE"/>

<!--先行研究での記述-->
<classgroup>
 <uses-class-permission
   android:name="android.permission.READ_PHONE_STATE"/>
 <join-class android:name="com.lib.A"/>
 <join-class android:name="com.lib.A"/>
...
</manifest>
```

図 5 マニフェストの記述例

ため，パーミッションを必要とする API で getStackTrace を使用してスタックフレームを取得し，スタックフレームから Android 標準のクラスを除くことでスタックフレームの最上位から API を呼び出したクラス名を取得している．このクラス名と API に必要なパーミッションを PackageManagerService に渡し，パーミッションの検証を依頼する．

3.5 PackageManagerService のクラスグループ対応

PackageManagerService には，クラスグループ毎に必要なパーミッションの記憶とクラス名に基づくパーミッションの検証を実装している．クラスグループとパーミッションの記憶にはハッシュテーブルを使用する．アプリをインストールした時にクラス名とパーミッション名の組をキーとして保存する．アプリに対応するテーブルがある場合にクラスグループ毎のパーミッションの検証を行う．検証では，API から渡されたクラス名とパーミッション名の組をキーとしてテーブルを検索する．検索に成功するとパーミッションは許可されている，失敗すると拒否されていると判断する．

4. 提案手法

本章では，先行研究を適用したランタイムパーミッション制御機構を提案する．提案手法では，先行研究で用いられたクラスグループ単位でランタイムパーミッションを要求する．その際にアプリのどのクラスグループがどのランタイムパーミッションを要求するかをユーザーが確認できるように，クラスグループをランタイムパーミッションの要求画面に表示する．ランタイムパーミッションの要求に対するユーザーの許可及び拒否の選択は要求元のクラスグループにのみ反映する．複数のクラスグループで同一のランタイムパーミッションが指定されている場合，あるクラスグループでランタイムパーミッションが許可されても，他のクラスグループはそのランタイムパーミッションを要求していないあるいは拒否された場合使用できない．ランタイムパーミッションの指定は，先行研究のパーミッションと同じくアプリ開発者がマニフェストに記述する．なお，ランタイムパーミッションの適用範囲の分割を目的としているため，設定アプリに関しては未実装である．ライブラリの脆弱性の存在が判明した場合に，許可しているランタイムパーミッションを拒否することでライブラリによる個人情報の漏洩を防止可能であり，設定アプリからのランタイムパーミッションの設定の拡張は今後の課題である．

提案手法により，ランタイムパーミッションの適用範囲が分割され，クラスグループで分割されたホストアプリ部分とライブラリ部分それぞれで権限の許可を選択できる．許可したランタイムパーミッションは要求元のクラスグループにのみ適用されるため，ホストアプリ用に許可したランタイムパーミッションをライブラリが悪用し端末内の個人情報を盗むのを防ぐ．また，ランタイムパーミッションの要求時に要求元のクラスグループと必要なランタイムパーミッションのみを表示するため，ランタイムパーミッションがホストアプリとライブラリのどちらで使用されるかの判断が容易になる．

5. 実装

本章では，提案手法の実装の詳細を述べる．

5.1 概要

実装の概要を**図 6** に示す．Android 標準のランタイムパーミッション機構に追加した機能を (i)〜(vi) で示す．また，先行研究と同様の実装の箇所は図 4 の対応するアルファベットを明記している．

実装の流れを始めに述べる．まず，アプリはクラス内のランタイムパーミッションの要求を PackageInstaller に依頼する (図 6 の (i))．PackageInstaller はその情報を元にクラスグループの情報をチェックする (図 6 の (ii))．クラスグループの情報は PackageManagerService に登録されており (図 6 の (iii))，アプリが指定したクラスが属するクラスグループのランタイムパーミッションの状態とグループを PackageManagerService から取得する．PackageInstaller はそのグループを用いてユーザーにランタイムパーミッションを要求し，その際に要求元のクラスグループを表示する (図 6 の (iv))．ユーザーがその要求に対して選択した許可および拒否の結果を PackageInstaller はアプリに返し，指定したクラスの属するクラスグループに対してランタイムパーミッションを許可する (図 6 の (v))．許可されたランタイムパーミッションが必要な API を使用する際は，先行研究と同様に検証を要求する．検証を依頼された PackageManagerService はクラスグループの PermissionsState を参照し，パーミッションの状態を検証する (図 6 の (vi))．

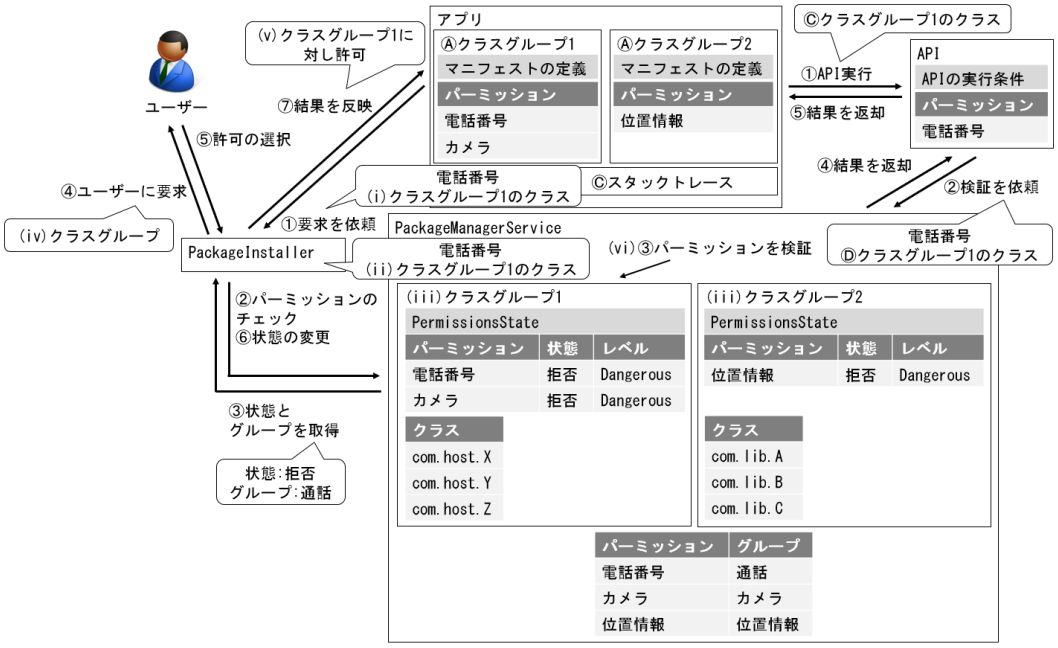

図 6 実装の概要

5.2 実装内容

実装はランタイムパーミッションの利用要求時の処理が主な対象である．一方，図 6 のⒶからⒹに至る API 実行時の処理は，PackageManagerService におけるパーミッションの検証処理を除き，先行研究 [3] と同じである．

ランタイムパーミッションの利用要求時の処理の実装は，アプリへの処理の記述，ランタイムパーミッションの管理，ランタイムパーミッションの要求の 3 点に対して行った．1 つ目に，クラスグループとそれに対応したランタイムパーミッションの処理を記述できるように関連機構の拡張を行った．まず，クラスグループについては先行研究と同様にマニフェストファイルの書式を拡張した．次に，ランタイムパーミッションの要求に関するメソッドでクラスグループを利用できるように，ランタイムパーミッションを使用するクラス名を指定する処理を追加した．2 つ目に，クラスグループ毎にランタイムパーミッションの状態を記憶し管理する処理を PackageManagerService に追加した．また，クラスグループ毎のランタイムパーミッションの状態を参照するようにランタイムパーミッションを検証する処理を拡張した．3 つ目に，ランタイムパーミッションの要求処理をクラスグループに対応するよう PackageInstaller を拡張した．

5.3 アプリのクラスグループ対応

アプリでクラスグループを利用する実装は先行研究の実装と同様に行った．また，ランタイムパーミッションの要求を PackageInstaller に依頼する処理をクラスグループに対応するよう拡張した．まず，アプリは PackageInstaller に依頼する際，当該ランタイムパーミッション名に加えそれを使用するクラス名を渡す．そして，ランタイムパーミッションの要求結果を受け取る処理に，要求時に指定したクラスも受け取るように拡張した．これにより，ランタイムパーミッションの要求に対するユーザーの選択に応じた処理をクラスグループ毎に設定可能になる．

なお，本来であれば利用要求時のスタックトレースからクラス名を取得するのが望ましい．理由は他のクラスグループのランタイムパーミッションの要求を防ぐためである．しかし，ランタイムパーミッション要求の依頼は Activity の機能であるため，Activity を継承したクラス以外ではランタイムパーミッション要求は依頼できない．また，スタックトレースを取得した場合，ランタイムパーミッション要求の依頼処理を実行した Activity がホストアプリ内であれば，ライブラリのクラスグループに属するクラスはスタックトレースに出現しないため，クラスグループ毎のランタイムパーミッションの要求が正しく動作しない．したがって，クラス名を直接引数に指定する形をとっている．ランタイムパーミッション要求を行うクラスの指定方法は今後の課題である．なお，API 実行時のクラス名を特定するためには，先行研究と同様にスタックトレースを利用している．

5.4 クラスグループ毎のランタイムパーミッションの状態

クラスグループ毎にランタイムパーミッションの許可および拒否の状態を管理するため，PackageManagerService にクラスグループ名とパーミッションの状態を管理する PermissionsState の組をキーとするテーブルを作成した．PermissionsState は Android 標準のシステムにおいてはアプリ毎に存在，すなわちランタイムパーミッションの適

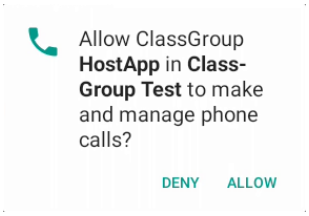

図 7 提案手法のランタイムパーミッションの要求ダイアログ

用範囲毎に存在する．そのため，提案手法のクラスグループの単位で PermissionsState を利用可能である．クラスグループが存在しない場合このテーブルは空である．

ランタイムパーミッションの検証の際にクラスグループ毎の PermissionsState を使用する．クラスグループ毎の PermissionsState が登録されていない場合は従来のアプリ単位でのランタイムパーミッション要求を行う．クラスグループが存在する場合，クラスグループ名を検索しその名前と組になっている PermissionsState を取得し，さらに PermissionsState に登録されているランタイムパーミッション名を検索してその状態を取得する．ランタイムパーミッションが PermissionsState に存在し，かつ許可の場合のみランタイムパーミッションは許可と判断する．ランタイムパーミッションが PermissionsState に存在しない場合，または存在しても拒否の場合はランタイムパーミッションは拒否と判断する．パーミッション名が存在してもそれが許可とは限らない点は先行研究のパーミッション検証の方法と異なる．

5.5 ランタイムパーミッション要求処理の拡張

PackageInstaller のランタイムパーミッションの要求処理をクラスグループに対応した．まず，アプリからランタイムパーミッション要求を依頼された際にランタイムパーミッション名とクラス名を取得し，クラス名を取得した場合はクラスグループに対するランタイムパーミッション要求を行う．次に PackageManagerService からクラスグループ毎の PermissionsState を取得し，指定されたクラスが属するクラスグループとランタイムパーミッションを検索する．要求するランタイムパーミッションが PermissionsState にない場合は要求元のアプリに戻る．ランタイムパーミッションの検索に成功すると，そのランタイムパーミッションの属するパーミッショングループを取得する．そのパーミッショングループが既に許可または拒否を選択された場合はその設定を要求されたランタイムパーミッションに適用する．パーミッショングループに対する要求がされていなかった場合は，**図 7** のようにランタイムパーミッションを要求するダイアログを表示する．この時，表示する内容にクラスグループ名を含める．ユーザーの許可あるいは拒否の選択に対する結果を，要求したランタイムパーミッション名と要求元のクラスグループ名とともに要求したアプリに返却する．

表 2 評価環境

エミュレータ	Android Emulator
OS	Android 7.0.0 に機能を追加

6. 評価

本章では提案手法に対して行った評価を述べる．評価環境を**表 2** に示す．

6.1 提案手法を実装したシステムの動作

提案手法を実装したシステムにおいて，ランタイムパーミッションの要求及び検証ができることを評価した．まず，ホストアプリとライブラリの 2 つのクラスグループを持つテストアプリを作成した．テストアプリはランタイムパーミッションをホストアプリにのみ設定したテストアプリ A とホストアプリとライブラリの両方に設定したテストアプリ B の 2 つを作成した．設定するランタイムパーミッションは READ_PHONE_STATE を用いる．テストアプリではクラスグループ毎のランタイムパーミッション要求と，READ_PHONE_STATE が必要で電話番号を取得する API である getLine1Number の実行のテストを行う．

提案手法を実装したシステムでテストアプリ A を実行すると，アプリがランタイムパーミッションを要求した場合，ホストアプリからの要求の場合では図 7 のようにダイアログが表示され，メッセージにホストアプリのクラスグループ名 “HostApp” が表示された．一方，ライブラリではダイアログが表示されなかった．また，テストアプリ B を実行した場合，ホストアプリとライブラリのいずれもダイアログが表示された．また，ホストアプリで許可を選択した後，ライブラリがランタイムパーミッションを要求するとダイアログが表示された．以上の動作より，ランタイムパーミッションの要求がクラスグループ毎に行われたことを確認した．

テストアプリ A のホストアプリからの要求に許可を選択した後，getLine1Number をホストアプリで実行すると電話番号を取得できた．一方，ライブラリで実行するとアプリが強制終了した．また，ホストアプリからの要求に拒否を選択した場合に getLine1Number を実行した場合もアプリが強制終了した．次に，テストアプリ B でホストアプリからの要求に対しては許可，ライブラリからの要求に対しては拒否を選択すると，getLine1Number の実行結果はホストアプリ側では電話番号の取得に成功，ライブラリ側ではアプリの強制終了であった．以上の結果から，ランタイムパーミッションはクラスグループ毎の要求結果を反映し，かつクラスグループ毎に制御されたことを確認した．

表 3 ランタイムパーミッション要求時間の評価結果

	実行時間 (msec)
従来手法	100.029
提案手法	109.399

表 4 API 実行時間の評価結果

	実行時間 (msec)
従来手法	2.517
提案手法	7.489
スタックトレース取得	3.727

6.2 オーバヘッド

提案手法では，ランタイムパーミッションの要求およびランタイムパーミッションの必要な API の処理を追加したため，オーバヘッドが発生する．そのため，ランタイムパーミッションの要求，API の実行に要する時間を測定し，評価を行った．測定に用いた API は getLine1Number であり，この API では READ_PHONE_STATE が必要である．また，クラスグループはホストアプリとライブラリの 2 つを設定し，ホストアプリのクラスは 1 つとした．ランタイムパーミッション要求と getLine1Number はホストアプリで実行した．

ランタイムパーミッションの要求時間の評価には，ランタイムパーミッション要求の依頼から要求結果を受信するまでの処理時間を 100 回実行した際の平均値を測定し，従来の手法と提案手法で比較を行った．測定した結果を**表 3** に示す．提案手法は従来の手法と比較して約 9.370msec のオーバヘッドが発生している．ランタイムパーミッションはアプリのインストール後に要求するが，ユーザーが許可を選択した後は行わない．したがって，ランタイムパーミッション要求はアプリで使用するパーミッショングループの数だけ実行され，そのグループは表 1 にある最大で 9 つのみである．同時に 9 つのパーミッショングループを要求してもオーバヘッドは約 81msec である．実際にすべてのパーミッショングループを必要とするアプリは少ないため，要求時におけるオーバヘッドは小さいといえる．

API の実行時間は，変更前の状態における API の実行時間，提案手法における API の実行時間，提案手法においてスタックトレース取得の処理時間を測定した．測定した結果を**表 4** に示す．提案手法は約 4.972msec のオーバヘッドが発生している．スタックトレース取得にかかる時間は約 3.727msec で，オーバヘッドの約 74.95%を占めている．端末の電話番号を高頻度または短時間に何度も取得することは考えられないためオーバヘッドは小さいといえる．しかし，高頻度かつ短時間に何度も実行する API において提案手法を適用した場合はオーバヘッドがアプリの動作に大きく影響するため，オーバヘッドの削減が必要である．そのため，オーバヘッドの大部分を占めるスタックトレースの処理時間の削減が課題である．

7. まとめ

本稿では第三者の作成した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問題がランタイムパーミッションにも残されているという問題に対し，先行研究であるクラスに基づくパーミッション制御手法をランタイムパーミッションに適用した制御機構を提案した．クラスグループを用いることで，ランタイムパーミッションの適用範囲をホストアプリと第三者のライブラリに分割し，それぞれでランタイムパーミッションの許可を選択可能となる．提案手法を評価し，ユーザーが許可したランタイムパーミッションが要求元のクラスグループ内でのみ適用され，ランタイムパーミッションが不要あるいはユーザーが許可しないクラスグループに反映されず，クラスグループを用いたランタイムパーミッションの制御が可能であることを確認した．また，ランタイムパーミッションの要求と API の実行時間のオーバヘッドが小さいことを確認した．

今後の課題として，他のクラスグループを要求元とするランタイムパーミッションの要求の防止，ランタイムパーミッションの設定変更のクラスグループ対応，スタックトレース処理のオーバヘッド削減が挙げられる．

参考文献

[1] Taylor, Vincent F and Beresford, Alastair R and Martinovic, Ivan: Intra-Library Collusion: A Potential Privacy Nightmare on Smartphones, arXiv preprint arXiv:1708.03520 (2017).

[2] CVE-2018-14774: Possible host header injection when using HttpCache (Symfony Blog): http://symfony.com/blog/cve-2018-14774-possible-host-header-injection-when-using-httpcache

[3] 日置将太, 高瀬拓歩, 齋藤彰一, 毛利公一, 松尾啓志: 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, 情報処理学会論文誌, Vol.56, No.2, pp.720-732(2015).

[4] 河合佑紀, 明田修平, 半井明大, 窪田歩, 瀧本栄二, 毛利公一: Android アプリケーションにおける Runtime Permission 要求時の挙動調査, 研究報告コンピュータセキュリティ（CSEC）, Vol.2017, No.31, pp.1-7 (2017).

[5] システム パーミッション | Android Developers: https://developer.android.com/guide/topics/security/permissions?hl=ja

[6] 実行時のパーミッション リクエスト | Android Developers: https://developer.android.com/training/permissions/requesting?hl=ja.