Android OS における機能や情報へのアクセス制御機構の提案

川端 秀明† 磯原 隆将† 竹森 敬祐† 窪田 歩† 可児 潤也‡ 上松 晴信‡ 西垣 正勝‡
†KDDI 研究所
356-8502 埼玉県ふじみ野市大原 2 丁目 1 番 15 号
{ kawabata, ta-isohara, takemori, kubota }@kddilabs.jp
‡静岡大学
432-5011 浜松市中区城北 3 丁目 5 番 1 号
{cs08028, cs07003}@s.inf.shizuoka.ac.jp, nisigaki@inf.shizuoka.ac.jp

あらまし Android OSを搭載した端末では，アプリケーション（以下，アプリ）をインストールする際に，アプリ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，インストールの可否を仰いでいる．しかしながら，一度ユーザがアプリのパーミッションを許可してしまうと，そのアプリ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ず，制御が効かない問題がある．そこで本稿では，実行中のアプリに対して，利用する機能や情報へのアクセスをリアルタイムで制御す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，割り込み処理を組み込むことで，アプリが動作する中で，ユーザの承認機構を実現するものである．我々は，危険を伴うパーミッションに付随するAPIに割り込み処理を実装し，評価を行なったところ，処理負荷を殆ど要しないこと，そしてアプリが動作する中で機能や情報へのアクセスの様子を把握できる有効性を確認した．

Proposal of Fine-grained Access Control Framework in Android APIs

Hideaki Kawabata† Takamasa Isohara† Keisuke Takemori† Ayumu Kubota†
Jyunna Kani‡ Harunobu Agematsu‡ Masakatu Nishigaki‡

†KDDI R&D Laboratories
2-1-15 Ohara, Fujimino, Saitama 356-8502, JAPAN
{ kawabata, ta-isohara, takemori, kubota}@kddilabs.jp
‡Shizuoka University
3-5-1 Johoku, Naka-ku, Hamamatsu, Shizuoka 432-8011, JAPAN
{cs08028, cs07003}@s.inf.shizuoka.ac.jp, nisigaki@inf.shizuoka.ac.jp

Abstract In order to reduce the security risk caused by suspicious Android applications, users can verify the permissions requested by the application at the installation phase. The permissions define privileges to access certain APIs and the user has to grant them if she wishes to complete its installation. However, once the user grants these permissions, the user cannot check whether the application is properly using these permissions. In this paper, we propose the 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 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 application's runtime behavior.

1 はじめに

Android[1]の魅力の一つに，誰もが自由にアプリを開発・公開・インストールすることができ，スマートフォンの機能を自身の手でカスタマイズできる点がある．アプリ開発者は，携帯電話が持つ機能や情報をアプリから利用するApplication Programming Interface（API）を用いることで，利便性の高いアプリを開発することができる．そして，開発したアプリをAndroid Market[2]などのマーケットプレイスを通じて自由に販売や無料で公開をすることができる．しかしながら，マーケットプレイスで公開されているアプリの中には，Android OS の脆弱性を突いて管理者権限を奪うものや，端末に格納された個人情報を収集して外部に送信する，悪性アプリが存在する．特に，正規のアプリを装いつつ，不正な振る舞いを行う機能が組み込まれたトロイの木馬が現れ脅威となっている[3,4]．

Android の特徴として，安全性と利便性のトレードオフの判断をユーザに委ねるパーミッション機構[5]がある．パーミッション機構とは，端末内の情報や機能へのアクセス権をアプリに付与する承認を与えるものである．具体的には，アプリ開発者がマニフェストファイルにそのアプリで使用する機能やアクセスする情報をパーミッションとして定義しておき，アプリがインストールされる時に，Android OS からユーザへ使用許可を求める．しかしながら，このパーミッション機構には以下の課題がある．

（課題1）ユーザはアプリをインストールするためには，アプリが要求する全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．

（課題2）パーミッションにリンクする機能や情報を，いつどのように利用しているのか，ユーザは知ることができない．そのためアプリが成立するために必要なものとして承認したパーミッションが不正に利用された場合の検知は極めて難しい．

（課題3）パーミッションの粒度が大きく，パーミッションの中でもユーザが許可や不許可にしたい情報や機能が混在している．

先行研究の Apex[6]，Saint[7]，SEAF[8]では，前述の課題1の解決に取り組み，アプリインストール時，インストール後にユーザからパーミッションを変更可能とする方式を提案している．しかしながら，アプリがあるパーミッションを要求するという情報しか取得できないため，前述の課題 2，3 を解決することができていない．

そこで本稿では，Android フレームワークの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，リアルタイムで動的制御を行う方式を提案する．これは，Android OS における機能や情報を扱う API への割り込み処理の組み込みと，割り込みを受け取る制御アプリを Android OS に実装することで実現する．このようにすることで，アプリ実行時に，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することや，ユーザによって制御を行うことができる．また，パーミッションで制御される特定の情報，機能へのアクセス制御を可能とする．そして，提案方式の制御処理に要するパフォーマンス評価を行い制御による処理負荷の検討を行う．

2 背景

2.1 Android アプリ

Android アプリは通常 Java で記述される．Android OS では，アプリ毎に User ID（UID）が割り振られ，Dalvik 仮想マシンと呼ばれるレジスターベースの仮想マシンのサンドボックス上でアプリは実行される．初期状態では，機能や情報へのアクセス権が認められていないが，パーミッション機構を利用することで端末の機能や情報へのアクセス権を得ることができる．パーミッション機構とは，端末内の情報や機能へのアクセス権を，アプリのインストール時にユーザがアプリに承認を与えるものである．

Android アプリの構成要素として，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ーフェース（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.

2.2 Android マルウェア

Android マルウェアは，1. 管理者権限奪取マルウェア，2. パーミッション不正利用型マルウェアの 2 種類に分けられる.

管理者権限奪取型マルウェアとは，Linux カーネルや Android OS の脆弱性を突くコードを実行し，端末の root 権限（管理者権限）を奪取するマルウェアである. 実際にあるマーケットプレイスで Droid Dream[3],Ginger Master[4]と呼ばれるマルウェアが掲載され感染が広がった. アプリが root 権限を獲得してしまうと，サンドボックスの枠を超えることができ，他のアプリのリソースを参照することや，ユーザに気づかれないよう第二のマルウェアをインストールすることも可能であるため大きな危険となる.

パーミッション不正利用型マルウェアとは，アプリに与えられたパーミッションを用いて，アプリ本来の意図とは異なる不正行為を働くマルウェアである. 例えば，ユーザの同意を得ずに個人情報を外部に送信したり，ユーザ操作を問わずに電話の発信や SMS 送信することが不正行為にあたる.

本稿では，後者のパーミッション不正利用型のマルウェアを対象とし，新しいアクセス制御機構の提案を行う.

2.3 関連研究

Android のパーミッションに基づくセキュリティモデルについて様々な問題が指摘され改良に取り組む研究がなされている.

Kirin[9]は，アプリの持つパーミッションの組み合わせから危険性を判断し，アプリインストール時にユーザに提示する手法を提案している.

Apex[6]は，アプリのインストール時，インストール後にアプリの要求するパーミッションをユーザから変更できる仕組みを提案している.

Saint[7]は，アプリのパーミッションについて詳細な条件を設定できる仕組みとアプリ同士の連携を制御する仕組みの提案を行っている. 詳細な条件とは，時間，場所，インターネットに接続できない環境，パーミッションの使用回数，アプリのバージョン等によってパーミッションの有効・無効を動的に設定することができる.

SEAF[8]では，パーミッションをユーザから変更する仕組みに加えて，アプリがパーミッションにアクセスする順番に着目し不正行為を検知する仕組みを提案している.

2.4 課題

上記の通りパーミッションに基づくセキュリティモデルを改良する様々な手法が提案されているがいくつか課題がある.

課題 a：パーミッションにリンクする機能や情報を，いつどのように利用しているのか，ユーザは知ることができない. そのためアプリが成立するために必要なものとして承認したパーミッションが不正に利用された場合の検知は極めて難しい.

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 を宣言することは正しいと考えられる. しかしながら，メールクライアントアプリとして正規の振る舞いをしている中で，バックグラウンドでは不正に SMS を送信していた場合，ユーザが気づくことは極めて難しい. よって，正規の振る舞いをしている中で，不成功行為を働くアプリに関しては，ユーザがパーミッションを許可しているため，パーミションレベルの制御が働かない. アプリがどのようにパーミッションを使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.

課題 b:パーミッションの粒度が大きい

READ_PHONE_STATE と呼ばれるパーミッションでは，端末電話番号，端末識別子（International Mobile Equipment Identity:IMEI），SIM 識別子(International Mobile Subscriber Identity: IMSI)，端末ソフトウェアバージョンという端末の個人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紐付き難い情報は許可したいというケースも考えられる.

3 提案

前述の課題を解決するため，Android フレームワークの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，リアルタイムで動的制御を行う方式を提案する．これにより，アプリ実行時に利用する，機能や情報を具体的に把握できることや，ユーザによって動的に承認を与えることができる．また，アプリ毎に許可する API やアプリに一括で許可する API などを定義するセキュリティポリシーを設定することができる.

提案方式を Android に実装するため，Android OS における機能や情報を扱う API への割り込み処理の組み込みと，割り込みを受け取る制御アプリの開発を行った．図 1 は提案方式を Android に実装し Google 社が提供するマップアプリを動作させた場合の画像である．マップアプリでは，位置情報を取得する API を実行する．この時に提案するフレームワークから制御アプリが呼び出されアプリが実行する API を動的に制御している．以下，構成，セキュリティポリシー，API のフックポイントについて説明する.

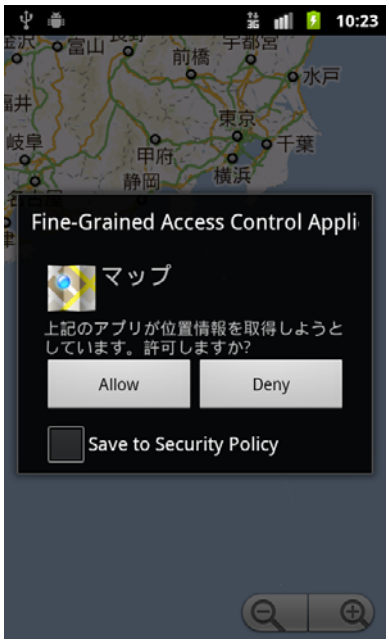

図 1 提案方式によってアプリの位置情報取得が制御される様子

3.1 構成

システムの概要を図 2 に示す．ここでは，TelephonyManager クラスに記述されている端末の設定情報を取得する API をフックする例で説明する．図中の ApiManager が Android フレームワークに新たに追加したクラスである．ApiManager と連携するアプリを Android に組み込むことで，アプリの動的制御を可能とする.

①一般アプリから端末識別子を取得する場合，TelephonyManager クラスの getDeviceId という API を使用する．通常の Android フレームワークでは要求に従いアプリに対して要求された情報を返答する.

②API がアプリによって呼び出された時，拡張したフレームワーククラスの ApiManager を実行するよう TelephonyManager にフックポイントを設ける.

③ApiManager では，まず API を呼び出したアプリの PID を取得する．その後，制御アプリが管理するセキュリティポリシーリポジトリを参照する.

④ApiManager は参照したセキュリティポリシーの情報を得る.

⑤ApiManager は，呼び出し元アプリのセキ

ュリティポリシーが設定されているか判定を行う.

⑥呼び出し元アプリのセキュリティポリシーが設定されていた場合は，その結果を TelephonyManager に返す．設定されていなかった場合は，制御アプリを起動し，端末 UI でユーザ確認を行うダイアログを出現させる.

⑦TelephonyManager は，ApiManager からの返答を受け取り，許可であった場合は，通常の API の動作を行い呼び出し元アプリに返答する．拒否であった場合は，API の動作を止め呼び出し元アプリに対して返答を行う.

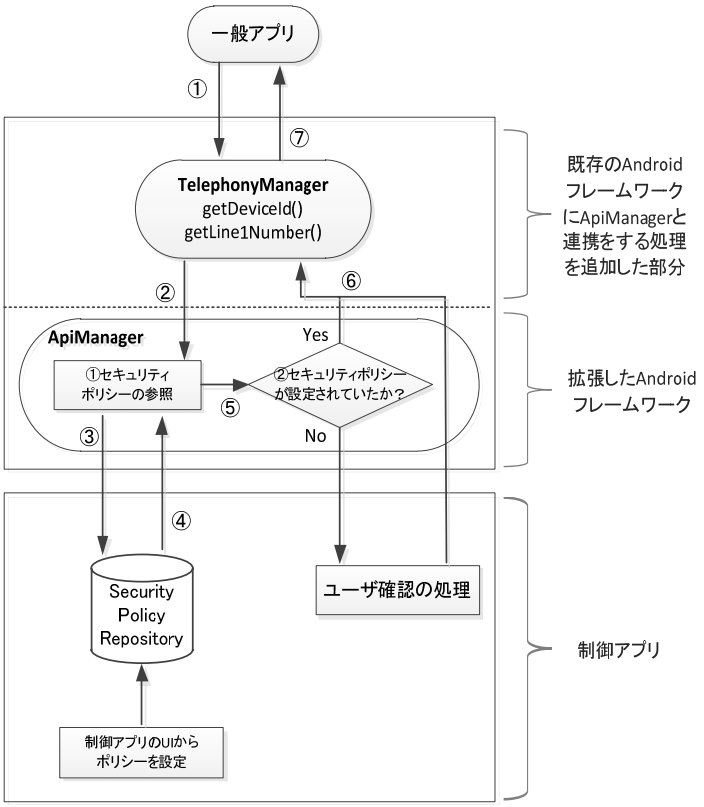

図1 システム概要

3.2 セキュリティポリシー

提案方式では，下記のセキュリティポリシーを設定することができる.

- API の実行を許可，拒否
- パーミション毎で実行を許可，拒否
- 端末に一括で API，パーミッションの実行を許可・拒否

ユーザビリティを考慮し，アプリ毎に API を指定して許可・拒否を行うことや，インストールするアプリに一括で API を指定して許可・拒否をできるようにする．これらの設定は，制御アプリのリポジトリに管理し，アプリが API を実行した際にアプリのリポジトリが作成されていない場合，ユーザに問い合わせを行い設定する．また，制御アプリをユーザが起動し設定することや，一度ユーザが設定した値は保持する設定もできる.

3.3 フックポイント

本稿では，端末の個人情報を取得する API，位置情報を取得する API，SMS で外部に送信する API のフックを行う．表 1 にフックした API の一例示す.

表1 Android フレームワークのフックポイント

クラス	メソッド
TelephonyManager	getDeviceId
	getDeviceSoftwareVersion
	getLine1Number
	getSimSerialNumber
	getSubscriberId
	getVoiceMailAlphaTag
	getVoiceMailNumber
LocationManager	getLastKnownLocation
	LocationListener
	requestLocationUpdates
SmsManager	SendTextMessage

4 評価

提案方式では，フックした API が呼び出される毎にセキュリティポリシーを参照するため，通常の Android フレームワークより，参照に関する時間が余分にかかる．セキュリティポリシーがアプリに設定されていない場合は，ユーザ確認の処理を行うため，ユーザ判断に関する時間が大きく影響する．そのため，本章ではセキュリティポリシーが事前に設定されているとし，アプリからフックした API を呼び出し，セキュリティポリシーを参照し，API の動作を行うという一連

の動作に要する時間を測定する. 具体的には, アプリが端末識別子を取得するgetDeviceIdを実行した際に, ApiManager がセキュリティポリシーを参照し, その結果を受け取りgetDeviceId の値をアプリが取得するまでの時間を計測した. セキュリティポリシーに登録するアプリ数を変化させながら, 各 5 回測定した時の平均値を表 2 に示す.

- セキュリティポリシーに登録したアプリ数: 1, 10, 20, 50, 100
- 測定端末: Nexus S
- OS：Andorid OS 2.3.5 を基に制御機構を組み込んだカスタム OS

表 2 セキュリティポリシーの参照に要する時間

	通常	提案方式				
登録アプリ数	－	1	10	20	50	100
所要時間 [msec]	0.49	9.36	9.78	10.18	10.08	10.06

評価の結果, 約 11msec 以内で処理が完了することが判明した. よって, 制御による処理負荷はアプリのプログラムに影響を与えない範囲で収まり, パフォーマンス低下の懸念は無視できると考えられる.

5 おわりに

本稿では, アプリの機能や情報へのアクセスを制御するフレームワークを提案した. Android フレームワークの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, 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, アプリ実行時に, 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ことや, ユーザがAPI 利用に対して動的に承認を与える制御が可能になる. セキュリティポリシーが事前に設定されている場合の一連の動作の処理時間に関する評価を行い, 約 11msec 以内で処理が可能なため, アプリのプログラムに影響を与えない範囲で収まり, パフォーマンス低下の懸念は無視できると考えられる.

参考文献

[1] Android , http://www.android.com

[2] Android Market, http://www.android.com/market/

[3] Droid Dream, http://blogs.computerworld.com/17929/google_android_market_kills_droid_dream_malware_in_trojans

[4] Ginger Master, http://www.cs.ncsu.edu/faculty/jiang/GingerMaster/

[5] Access permissions, http://developer.android.com/intl/ja/reference/android/Manifest.permission.html

[6] M.Nauman, S.Khan, and X.Zhang, “Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints”, 5th ACM Symposium on Information, Computer and Communications Security, pp 328-332,2010

[7] M. Ongtang, S. McLaughlin, W. Enck, and P. McDaniel, “Semantically Rich Application-Centric Security in Android”, ieee:Annual Computer Security Applications Conference,pp 340-349, 2009

[8] H.Banuri, M.Alam, S.Khan, J.Manzoor, B.Ali, Y.Khan, M.Yaseen, M.N.Tahir, T.Ali and X.Zhaug, “Android Runtime Security Policy Enforcement Framework” The 2010 International Workshop on Smartphone Applications and Services, 2010

[9] W. Enck, M. Ongtang, and P. McDaniel, “On lightweight mobile phone application certification”, in Proceedings of the 16th acm conference on Computer and communications security. acm, pp. 235-245, 2009