Vol.2013-DPS-154 No.57
Vol.2013-CSEC-60 No.57
2013/3/15

Android OS におけるマスカレーディングポイントを用いたプライバシー保護（その２）

上松晴信[†] 可児潤也[†] 米山裕太[†] 川端秀明[††] 磯原隆将[††] 竹森敬祐[††] 西垣正勝[†††]

近年，スマートフォンのセキュリティが重要視されてきている．特に，不正アプリによって引き起こされる情報漏洩，不正課金，ワンクリック詐欺などの問題が多発し，社会問題となっている．著者らは，Android OS 内にセキュリティマネージャと呼ばれる機構を設け，秘密情報をアプリから秘匿することによって機密情報管理を行う方式のコンセプトを提案した．本稿では，提案方式の詳細設計を進め，OS 内のフックポイントの検討と Java の stacktrace 機能の活用によって，セキュリティマネージャを具体化していく．

Privacy protection using masquerade pointer on Android OS (part 2)

HARUNOBU AGEMATSU[†] JUNYA KANI[†] YUTA YONEYAMA[†]
HIDEAKI KAWABATA[††] TAKAMASA ISOHARA[††] KEISUKE TAKEMORI[††]
MASAKATSU NISHIGAKI[†††]

Security of smart-phone is one of the most important issues. Especially the number of malicious Android applications that cause leakage of privacy information, incorrect billing, and one-click billing fraud has been drastically increasing recently. To cope with this problem, we have proposed a concept of sensitive information management mechanism called “security manager” and “masquerade pointer”, which is built in the Android OS to hide sensitive information from applications. This paper carries out a detailed design of the proposed mechanism by making use of “stacktrace” function of the Java and investigating the appropriate hook point.

1. はじめに

近年，Android OS が搭載された Android フォンが爆発的に普及している．Android フォンの特徴として，Android フォンを持つユーザは Google Play Store [1]にアクセスし，アプリケーションソフトウェア（以下，アプリ）を自由にダウンロードしてインストールできることが挙げられる．また，Android アプリの開発環境は無償で提供されているため，誰でも自由にアプリを作成し，Google Play Store にアップロードすることができる．

Google Play Store のような有名なマーケットは全てのアプリケーションをチェックし，不正アプリを削除している．しかし Android OS の場合，ネット上には信用できないマーケットも存在しており，そこではトロイの木馬と呼ばれる正規のアプリを装った不正アプリ（ワンクリックウェア [2], Geimini [3]等）が紛れ込み，様々な被害をもたらしている．これらの不正アプリはインストールされると，フォアグラウンドでは正規のアプリと同様に振舞いながら，バックグランドでユーザの個人情報（電話番号，端末番号，位置情報，SMS メッセージ，通話記録など）を取得しそれらを外部のサーバへ送信することで，ユーザに気づかれることなく情報漏洩を引き起こす[9].

不正アプリの被害を減らすためのアプローチとして，アプリ提供側で対策を行う方法と端末側で対策を行う方法が考えられる．また，端末側での対策は，ユーザによる対策，ツールによる対策，OS による対策に大別される．

アプリ提供側での対策としては，提供するアプリをあらかじめ検査しておき，安全であると判断されたアプリのみをマーケットにて提供する方法が考えられる[5][6][10]．アプリ提供側で対策を行うことにより，ユーザのセキュリティ意識などに依存せず，すべてのユーザに対して安全なアプリのみを提供することが可能となる．しかしながら，信用できないマーケットにおいては，このような対策を期待することはできない．

端末側でのユーザによる対策としては，既にこれまでに，Android OS のパーミッション機構の改善に関する多くの研究が報告されている[12][20-25]．しかし，Android フォンを利用しているユーザのセキュリティに対する意識や知識は様々であり，ユーザに完璧な対策を期待することは難しい場合がある．端末側でのツールによる対策としては，アンチウイルスソフトなどの活用が考えられる．しかし，文献[4]では，セキュリティ対策ソフトでの対策では限界があることが述べられている．

端末側での OS による対策としては，アプリからの機密情報の不正通信を制御する機構に関する研究が進められている（2 章にて詳述する）．しかし，一旦アプリ側に渡って

[†]静岡大学大学院情報学研究科，〒432-8011 浜松市中区城北 3-5-1，Graduate school of Informatics, Shizuoka University, 3-5-1 Johoku, Naka, Hamamatsu, 432-8011 Japan
[††]株式会社 KDDI 研究所，〒356-8502 埼玉県ふじみ野市大原 2-1-15 KDDI R&D Laboratories, Inc. 2-1-15 Ohara, Fujimino, Saitama, 356-8502 JAPAN
[†††]静岡大学創造科学技術大学院，〒432-8011 浜松市中区城北 3-5-1，Graduate School of Science and Technology, Shizuoka University, 3-5-1 Johoku, Naka, Hamamatsu, 432-8011 Japan

Vol.2013-DPS-154 No.57
Vol.2013-CSEC-60 No.57
2013/3/15

しまった機密情報を制御することは，OS にとって負荷の高いタスクとなる．そこで著者らは,「アプリに機密情報を渡さない」というコンセプトによって Android OS 内の機密情報を管理する「セキュリティマネージャ」という機構を提案している[7].

OS が管理している機密情報をアプリが利用したい場合，（i）アプリは OS に機密情報の取得要求を送り，これに対して（ii）OS が当該秘密情報をアプリに返送する．セキュリティマネージャは，（ii）の時点でアプリと OS の間に介在し，アプリに機密情報そのものではなく，機密情報を指し示す参照ポインタを返す．アプリが機密情報を出力する際には，セキュリティマネージャが参照ポインタを機密情報に復元する．機密情報の出力先が自端末内の OS 管理下リソースである場合は，参照ポインタは自動的に当該機密情報に変換される．機密情報が自端末 OS 外に出力される場合は，その操作を行う前にユーザに確認が求められる．参照ポインタは機密情報をマスクする役割を果たすため，この仕組みを「マスカレーディングポインタ」と名付ける．

この機構によって，不正アプリが機密情報を読み取るパーミッションを宣言し，ユーザがインストール時にこれを承認したとしても，アプリが取得できるのは参照ポインタのみとなり，不正アプリによる機密情報の漏洩を防ぐことができる．正規のアプリが機密情報を表示・送信する場合には，セキュリティマネージャによって参照ポインタは真の機密情報に自動変換されるので，アプリは正常にその動作を完了することができる．

本稿では，セキュリティマネージャの詳細設計を進める．不正アプリが提案方式に対して行うことができる攻撃としては，リフレクション，View 情報の読出し，端末画面のスクリーンショットが考えられる．OS 内の API コールのフックポイントの検討と Java の stacktrace 機能の活用によって，これらの攻撃に対する耐性を備えたセキュリティマネージャを具体化していく．

2. 関連研究

本章では，端末側での OS による対策に関する関連研究を説明する．

Enck らは，IMEI（International Mobile Equipment Identity や IMSI（International Mobile Subscriber Identity）等の機密情報を色付け（taint）し，この色付き情報の流れを捕捉することによって，OS からアプリに渡された機密情報を管理する TaintDroid という仕組みを提案している[8]．TaintDroid を実装して評価も行っており，そのオーバヘッドは最大 29%程度であったと報告している．色付き情報の捕捉には CPU の命令コードの拡張が必要であるため，TaintDroid の実装においては OS の改造が大掛かりなものになる点が懸念される．

川端らは，Android アプリによる機密情報の外部送信を制御する機構（Api Manager）を提案している[11]．Android フレームワークを拡張し，アプリが発行する機密情報送信に関する API コールに対してセキュリティポリシを強制的に適用する．ポリシの参照から API コールの制御までに要する時間を計測し，Api Manager 実装時のオーバヘッドが許容範囲内で抑えられていることを確かめている．しかし，セキュリティポリシを書き下すこと自体に困難性が残る．

SEAndroid においても，セキュリティポリシに関する問題が顕著となる．矢儀らは，SEAndroid のポリシ作成が困難なこと，使い方が難しいこと，必要な通信を拒否する可能性があることを指摘した上で，SEAndroid のパーミッション制御機構 Permission Revocation と通信制御機能 Tag Propagation を拡張する方法を提案している[13].

これらの既存方式は，アプリ側に渡された機密情報を制御するというコンセプトに基づく対策となっている．しかし，一旦アプリ側に渡ってしまった機密情報を制御することは，OS にとって負荷の高いタスクとなる．これに対し，著者らの提案しているセキュリティマネージャは,「アプリに機密情報を渡さない」というコンセプトに基づく対策であることに特長を有する．

3. Android OS

本章では，Android OS のセキュリティモデルとリフレクションに関して説明する．

3.1 セキュリティモデル

Android OS には Dalvik と呼ばれる仮想マシンが搭載されており（図 1），全てのアプリはこの仮想マシン上で実行される．Dalvik はサンドボックスとして機能し，それぞれのアプリには異なる UID，GID が割り当てられ，別々のメモリ空間で実行される．アプリが他のアプリと連携する際には，Binder によるメッセージパッシング機能を用いる．また，アプリが OS 内のリソースにアクセスするためには，アプリのインストール時にパーミッションを提示し，ユーザの承認を得る必要がある．

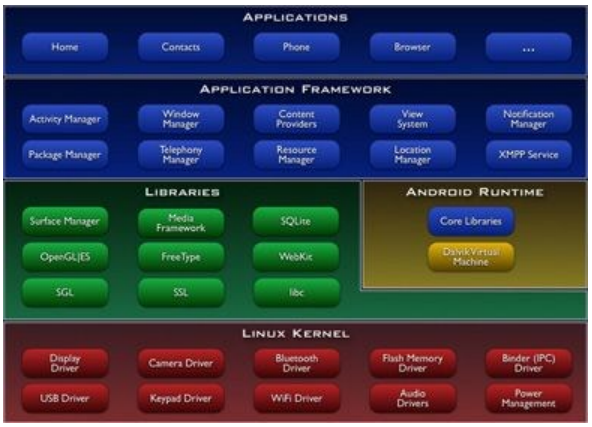

図 1. Android OS の構成[14]

アプリはパーミッションを得ることでさまざまな機能を提供する API（Application Program Interface）を使うことができる．API は図 1 の Application Framework 層に実装されている．例として，アプリが端末の機密情報を取得し表示するときの一連のフローを以下に示す（図 2）．

Step1） アプリが機密情報を取得する API をコールする．
Step2） OS が機密情報をアプリに返す．
Step3） アプリは受け取った機密情報を表示するために，当該機密情報を引数にして表示 API をコールする．
Step4） OS が機密情報を端末画面に表示する．

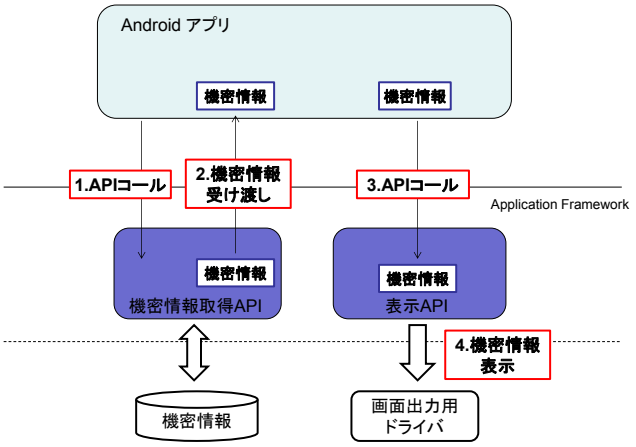

図 2. Android OS の機密情報管理

3.2 リフレクション

Android OS は図 1 に示されるようなレイヤ構造を有しているが，Java によって実装されているため，Java においてメソッドの動的実行を提供するための機能であるリフレクション[17][18]を用いることによって，レイヤを超えてのメソッド読み出しが可能となっている．

例えば，図 1 の APPLICATION FRAMEWORK 層には APPLICATION 層からは見えないメソッド(hide メソッド)が存在しているが，リフレクションを利用することによって，アプリがこれらのメソッドを呼ぶことができる．ただし，パーミッションによって保護されているメソッドに関しては呼び出すことはできない．

3.3 問題点

携帯電話には多くの個人情報が存在している．これらの機密情報は端末の OS 内に格納されており，パーミッション機構によってアプリから保護されている．しかし，パーミッション情報は抽象的であり，この情報だけではユーザは具体的にアプリがどのような機能を有し，OS 内のどのリソースにアクセスするのか理解することが難しい．このため，一般的なユーザにとって，アプリインストール時にアプリが有する潜在脅威を認識してパーミッション可否を正しく判断することは困難であるという問題が指摘されている．

最近では，電話番号，SIM 番号，端末 ID 情報などの漏洩によって引き起こされるワンクリック詐欺[15][16]の事例が複数報告されている．携帯端末画面に自身の電話番号や IMSI 番号が記載された不正請求書が表示されるため，ユーザがその不正請求を誤信しやすい．また，通話履歴，メール送受信履歴，電話帳データ，位置（GPS）情報の漏洩は，ユーザのプライバシーに強く関係する情報であるため，ユーザの精神的な苦痛は非常に大きいものとなる．Android は，アプリによる電話発信やメール送信についてもパーミッションによって制御される．不正アプリによる電話発信やメール送信は，機密情報を不正者に送信する手段として用いられるだけでなく，通話料およびネットワークアクセス課金を不正に請求する手段，あるいは，不正者の踏み台として外部へのリモート攻撃を実行する手段として利用される．

パーミッションに関する問題に対し，アプリによる機密情報に対するアクセスを制御するようなセキュリティ対策も提案されている．しかし，不正アプリの中には，リフレクションを巧みに悪用し，Linux Kernel やデバイスドライバを直接呼び出すことなどによって，これらのセキュリティ対策を回避するものが存在する．

4. セキュリティマネージャ

機密情報に対して参照ポインタ（マスカレーディングポインタ）を用意し，機密情報と参照ポインタを管理するセキュリティマネージャ[7]の機構を説明する．

セキュリティマネージャは，アプリからの機密情報の取得要求に対し，機密情報の代わりに参照ポインタをアプリに返し，機密情報そのものは OS 内のテーブルにて格納する．アプリが当該機密情報を出力する際には，セキュリティマネージャが参照ポインタから機密情報への復元の可否を動的に判断する．機密情報の出力先が自端末内の OS 管理下リソースである場合（例えば機密情報を端末画面に表示する場合など）は，セキュリティマネージャは自動的に参照ポインタを真の機密情報に変換する．機密情報が自端末 OS 外に出力される場合（例えば機密情報を自端末内アプリに転送する場合，外部端末に送信する場合，自端末の SD カードへ保存する場合など）は，その操作を行う前にユーザに確認を求め，ユーザの承諾があったときのみ，セキュリティマネージャが参照ポインタを機密情報へと変換した上で OS がこれを出力する．

不正なアプリが機密情報を読み取るパーミッション（例えば READ_PHONE_STATE 等）を宣言し，ユーザがインストール時にこれを承認してしまったとしても，このアプリが OS から機密情報を実際に読み取ろうとした際には，セ

キュリティマネージャによってそれらの機密情報はすべて参照ポインタに置き換えられる．この結果，不正アプリに機密情報そのものが渡ることはなく，不正アプリによる機密情報の漏洩を防ぐことができる．正規のアプリが機密情報を操作する場合には，セキュリティマネージャが動的に参照ポインタを真の機密情報に自動変換するので，機密情報を扱う正規アプリは正常にその動作を完了することができる．

3.1 節および図 2 にてアプリが端末の機密情報を取得し表示する際のフローを示したが，提案方式を実装した場合には，このフローが以下のように変わる（図 3）．

Step1） アプリが機密情報を取得する API をコールする．
Step2) セキュリティマネージャは Step1 の API コールをフックする．セキュリティマネージャは乱数等によって参照ポインタを生成し，参照ポインタと機密情報のペアを機密情報管理テーブルに格納する．
Step3） セキュリティマネージャは参照ポインタをアプリに返す．
Step4） アプリがこの機密情報を表示する場合は，参照ポインタを引数にして表示 API をコールする．
Step5) セキュリティマネージャは Step4 の API コールをフックする．セキュリティマネージャは機密情報管理テーブルを参照して，参照ポインタを対応する機密情報に変換した上で OS に渡す．
Step6） OS が機密情報を端末画面に表示する．

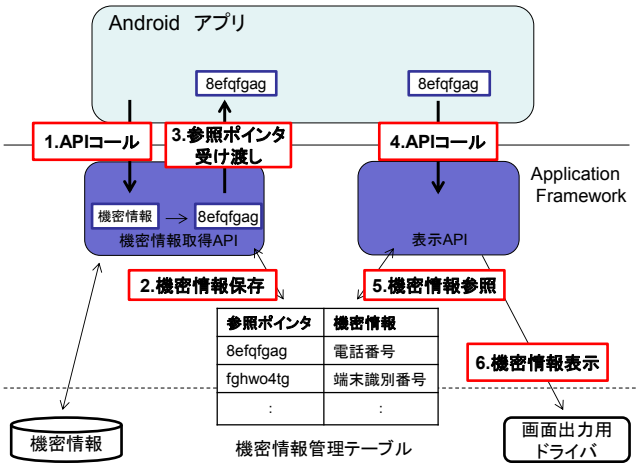

図 3. セキュリティマネージャによる機密情報管理

ここで，上記（図 3）は機密情報を自端末の画面に表示する場合の例であるため，Step 5 においてユーザへの許諾確認は行われない．例えば機密情報を SD カードに出力する場合は，Step 4 および Step 6 にてコールされる API が SD 保存用の API に代わるとともに，Step 5 においてユーザに確認ダイアログが表示されることになる．

5. 実装にあたっての注意点

図 3 のセキュリティマネージャを，実際の Android OS のフレームワーク上に実装する場合の構成図を図 4 に示す．

5.1 セキュリティマネージャの実装方針

Android OS における機密情報取得 API は複数存在する（表 3）ため，そのすべての API をフックすることになる．図 4 では，TelephonyManager クラス内の getLine1Number の例を示している．表示 API については，画面表示，SMS 送信，SD カード保存の API をフックする．図 4 では，画像表示 API である drawText の例を示している．

セキュリティマネージャは，機密情報を OS 外部に出力する場合には，ユーザに確認ダイアログを表示する．Android OS フレームワークにおいては，サービスが直接ダイアログを表示することができないため，ダイアログ表示用のアプリを別途用意し，セキュリティマネージャがこのアプリを起動するという実装形態となる．ここで，Android OS におけるアプリの起動が PackageManagerService によって担われることに鑑み，セキュリティマネージャは PackageManagerService クラス内に実装するようにした．すなわち，機密情報管理テーブルおよびテーブル操作用の API（表 2）は PackageManagerService クラス内に実装されることになる．

本章では，不正アプリによる機密情報の窃取を阻止するために，図 4 のセキュリティマネージャを実装するにあたっての注意点を述べる．不正アプリが図 4 の機構から機密情報を入手しようとした場合には，以下の 3 つの方法が考えられる．図 4 には，この 3 つの攻撃ポイントについても図示した．

(1) リフレクションを用いて，PackageManagerService 内のメソッドを使用し，機密情報を不正取得する方法．
(2) TextView にセットされた機密情報を不正取得する方法．
(3) ディスプレイに表示された機密情報をスクリーンショットで撮影する方法．

5.2 リフレクションによる機密情報の不正取得

今回作成する機密情報管理テーブル操作用の API（図 4 における secretInsert や secretSearch）は，通常のアプリケーションからは呼び出すこと（import すること）ができない hide メソッドとして実装する．

しかし，3.2 節で説明したとおり，リフレクションを用いることで不正アプリはこれらのメソッドも使用することが可能である．すなわち，不正アプリは，テーブル内の機密情報を検索するメソッド(secretSearch)を呼び出すことで，

Vol.2013-DPS-154 No.57
Vol.2013-CSEC-60 No.57
2013/3/15

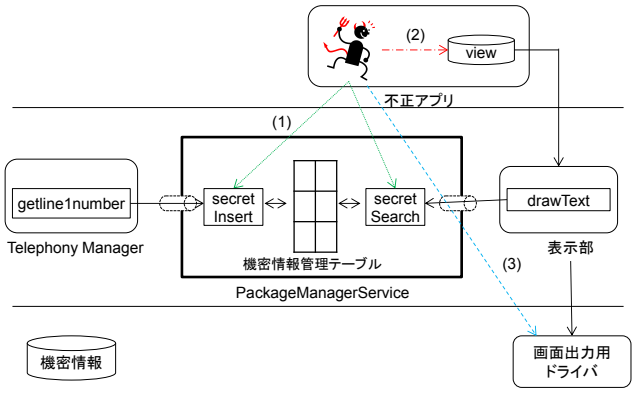

図 4. セキュリティマネージャの構成と不正アプリによる攻撃ポイント

機密情報を入手することができる．また，不正アプリは，機密情報をテーブルに新規格納するメソッド(secretInsert)を何度も呼び出すことで，機密情報管理テーブルのオーバーフローを引き起こすこともできてしまう．

この問題に対処するために，スタックフレーム情報を活用する．JAVA では，getStacktrace という API を実行すると，呼び出し元メソッドのクラス，メソッド名，プログラムの行番号を再帰的に辿るスタックフレーム情報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し，「getline1number のみが secretSearch を呼ぶことができる」という制限をかけ，アプリから secretSearch が直接呼ばれた場合にはこれを棄却することで不正アプリによる悪用を防ぐ．secretInsert やその他のテーブル管理用 API に対しても，同様の実装を施す．

5.3 TextView 内の機密情報の不正取得

Android OS では view と呼ばれるコンポーネントによって画面情報を構成する．view は図 5 のように階層的に管理されており，view tree と呼ばれる．view tree の葉には，図 5 に記された TextView や MapView 以外にも ListView や WebView などの様々な view エレメントを配置できる．view の情報は，アプリが所有する情報である．

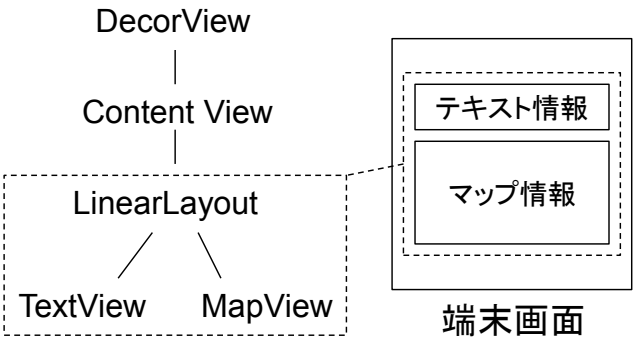

図 5. View Tree の例

セキュリティマネージャは，アプリが機密情報をディスプレイに表示する際には，参照ポインタを真の機密情報に変換する．しかし，前述のように view の情報はアプリ自身が所有しているため，例えば TextView 内にセットされている情報については，不正アプリが getText メソッドを呼び出すことによってこれを入手することができてしまう．したがって，setText の API コールのフックによって参照ポインタを機密情報に戻す方法（setText が呼ばれた時点でテキスト情報を検査し，それが参照ポインタであった場合には，機密情報に戻した上で TextView にその値をセットする方法）を採ることは適切ではない．

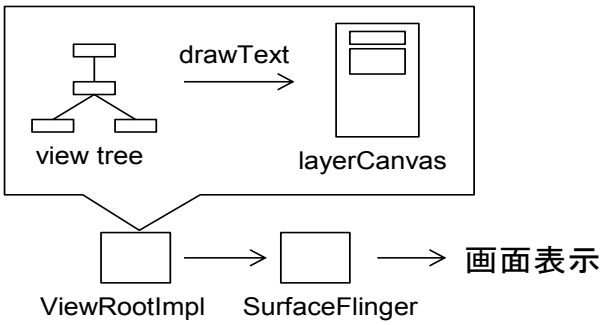

図 6. Android OS のグラフィックアーキテクチャ

Android OS (Android 4.1.2)のグラフィックアーキテクチャ[19]を図 6 に示す．図 5 で示した view tree は ViewRootImpl クラスが保持している．ViewRootImpl クラスは view の情報の全てを layerCanvas に書き込み，これを SurfaceFlinger に渡す．SurfaceFlinger は，openGL を用いて layerCanvas の情報をディスプレイに描画する．

ViewRootImpl クラスによる view 情報の layerCanvas への書き込みは，performTraversals メソッドによって実行される．performTraversals メソッドは，view tree のルート(DecorView)からリーフ（TextView や MapView）に向かって，順次それぞれの view に対する draw メソッドを呼ぶ．個々の draw メソッドがそれぞれの view 情報を layerCanvas に展開することによって，画面全体のビットマップ情報が構成される．draw メソッドは view のタイプごとにそれぞれ用意されており，TextView のテキスト情報を layerCanvas に書き込むメソッドは Canvas クラスの drawText である．

ここで，layerCanvas 変数は performTraversals 内のローカル変数であり，performTraversals メソッドが終了すると同時に消える．すなわち，Android OS の画面描画は，

(i) 画面描画のために performTraversals メソッドが呼ばれることによって，初めて layerCanvas 内に画面情報が構築される．

(ii) 画面情報が構築され次第，その画面がディスプレイに表示され，これによって performTraversals メソッドは終了し，それとともに layerCanvas 内に画面情報も消滅する．

という仕組みとなっており，layerCanvas 内に画面情報が保

持されている時間は(i)から(ii)までの刹那の瞬間のみとなる．よって，不正アプリが layerCanvas 内の画面情報を参照することは基本的に不可能であると考えらえる．

以上の理由により，参照ポインタを機密情報に戻すための操作を行うフックポイントとしては，drawText の API コールの時点が適当であろう．すなわち，「drawText が呼ばれた時点でテキスト情報を検査し，それが参照ポインタであった場合には，機密情報に戻した上で layerCanvas に view 情報を書き込む」という方法によって，図 4 における secretSearch を実装する．

5.4 スクリーンショットによる機密情報の不正取得

セキュリティマネージャは，（アプリに機密情報そのものを渡すことはしないが）アプリが機密情報をディスプレイに表示する際には，参照ポインタを真の機密情報に変換する．このため，不正アプリが画面をスクリーンショットで撮影し，これを外部に送信することによって，機密情報を漏洩させるという攻撃が考えられる．しかし，Android OS の現行バージョンにおいては，root 権限を持たない android アプリがスクリーンショットを撮ることは基本的には不可能であるという仕様になっている．

また，不正アプリは，getDrawingCache などの API を用いて，view のビットマップイメージを取得することもできる．しかし，6.3 節に述べたように，drawText の段階で機密情報の復元を行う方法であれば，view の中身は画像描画の直前の瞬間まで参照ポインタのままであるため，getDrawingCache による攻撃に対しては，実害はないと考えられる．

以上より，本稿では，スクリーンショットを利用しての攻撃に対しては，検討範囲から除外する．

6. 実装

5 章の注意点を考慮した上で，セキュリティマネージャを実装した．その全体図を図 7 に示す．図 7 は，「アプリが電話番号を SMS によって外部へ送信する」というシナリオに関する部分のみを示した図となっている．

6.1 セキュリティマネージャの実装と動作

Android OS の既存 API の内，機密情報の取得と出力に関する API（表 3）をフックし，機密情報操作のための自作 API（表 2）を挿入することによって，図 3 のフローを実現している．今回は，機密情報取得 API については，表 3 に示した 5 つの API のみに絞って実装をした．また，情報出力 API については，機密情報の端末画面への表示，SMS 送信，SD カードへの書き込みに絞って実装した．必要に応じて，他の機密情報取得 API および情報出力 API に対しても同様の改造が必要である．機密情報管理テーブルは Java の配列を用いて実装した．5.1 節で述べたように，機密情報管理テーブルおよびテーブル操作用の API（表 2）は PackageManagerService クラス内に実装されている．OS のビルド環境は表 1 のとおりである．

図 7 を用いて，セキュリティマネージャの動作を説明する．

(1) アプリが機密情報取得 API である getline1number を呼ぶと，getline1number に追加されたフックによって，処理がセキュリティマネージャ内の secretInsert メソッドに移る．
(2) secretInsert メソッドは，OS から機密情報を取得し，その内容を機密情報テーブルに格納するとともに，参照ポインタを生成し，参照ポインタの情報をアプリに返す．
(3) アプリが SMS 送信 API である sendTextMessage を呼ぶと，sendTextMessage に追加されたフックによって，処理がセキュリティマネージャ内の isWhite メソッドに移る．
(4) isWhite メソッドは，機密情報の SMS 送信に関するユーザへの承諾を得るために，画面にダイアログを表示する．5.1 節で述べたように，システム領域にプリインストールしたユーザ承諾画面表示用アプリを呼び出すことによって，このダイアログ表示が実現されている．
(5) ユーザの承諾が得られれば，セキュリティマネージャ内の secretSearch 関数を呼び出され，参照ポインタがこれに対応する機密情報に変換される．ユーザの承諾が得られなかった場合には，機密情報への変換を行わない．

表 1. Android OS のビルド環境

OS	Ubuntu12.04(precise) 64bit
memory	5.7GiB
CPU	Intel Core i7-2700K CPU @ 3.50 GHz * 8
Android version	Android4.1.2

表 2. 追加するクラスと API（クラスのディレクトリは，/frameworks/base/services/java/com/android/server/pm/*）

クラス	API	機能
PackageManagerService.java	secretInsert	テーブル格納
PackageManagerService.java	secretSearch	テーブル参照
PackageManagerService.java	isWhite	ユーザへの承諾画面表示

Vol.2013-DPS-154 No.57
Vol.2013-CSEC-60 No.57
2013/3/15

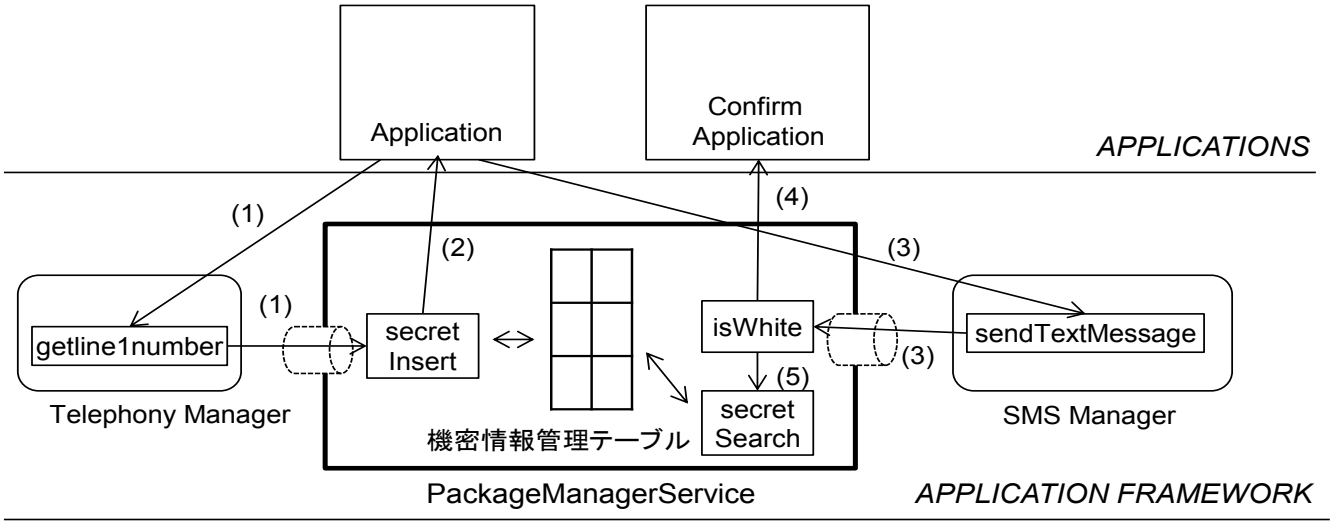

図 7. セキュリティマネージャの全体像

表 3. フックする API

	ディレクトリとクラス	API
機密情報取得 API	/frameworks/base/telephony/java/android/telephony/TelephonyManager.java	getLine1Number
		getDeviceId
		getSimCountryIso
		getSimOperator
		getSimSerialNumber
表示 API	/frameworks/base/graphics/java/android/graphics/Canvas.java	drawText
SMS 送信 API	/frameworks/base/telephony/java/android/telephony/SmsManager.java	sendTextMessage
SD 保存 API	/libcore/luni/src/main/java/java/io/BufferedWriter.java	Write

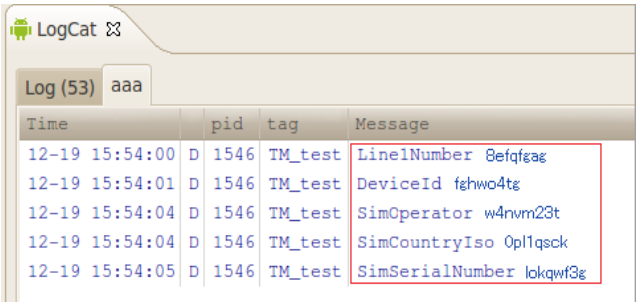

図 8. アプリ側で管理されている実際の値

図 9. 機密情報表示画面

6.2 機密情報の端末画面への表示

端末の電話番号，デバイス ID，SIM の国名，SIM の国コード，SIM 番号を取得し，これを画面に表示するアプリを作成し，セキュリティマネージャを実装した Android フォンにて実行した．

図 8 はアプリ側で保持している値を Android のログ機能（Logcat）によって出力した画面である．Message 欄に表示されている情報がアプリの所持している値である．電話番号，デバイス ID，SIM の国名，SIM の国コード，SIM 番号がそれぞれ参照ポインタ 8efqfgag, fghwo4tg, w4nvm23t, 0pl1qsck, lokqwf3g に置き換わっている．つまり，アプリが取得したのは端末情報そのものではなく参照ポインタの値であることが分かる．

一方，図 9 は，アプリがこれらの端末情報を端末画面に表示しているときのスクリーンショットである． OS 管理下のリソースである端末画面に機密情報が出力される場合は，セキュリティマネージャが動的に参照ポインタを真の情報に自動変換する．これによって，ユーザは端末情報を正しく閲覧可能であることが確認できる．

6.3 機密情報の SD カードへの出力

端末の電話番号，デバイス ID，SIM の国名，SIM の国コード，SIM 番号を取得し，これを端末に挿入されている SD カードに書き出すアプリを作成し，セキュリティマネージャを実装した Android フォンにて実行した．

OS の管理外となる SD カードに機密情報を保存するときには，ユーザに対してその可否が尋ねられる（図 10 左）．ユーザの承諾が得られた場合には，セキュリティマネージャが参照ポインタを機密情報に復元した上で，それを SD カードに書き込む（図 10 右上）．もし，承諾が得られなか

った場合は，参照ポインタがそのまま SD カードに書き込まれる（図 10 右下）．

なお，SD 保存 API である Write メソッドは，Android OS の frameworks 層（図 1 の第 2 層目のレイヤ）の API ではなく，Core Libraries 層（図 1 の第 3 層目のレイヤ）の API である．よって，通常は，Core Libraries 層の Write メソッドから frameworks 層に実装されているセキュリティマネージャ（機密情報管理テーブルおよびテーブル操作用 API）にアクセスすることはできない．そこで，今回は，リフレクションを用いることで, Write メソッドからテーブル操作用 API を呼び出せるよう実装した．これによって，「Write メソッドのフックによって，セキュリティマネージャにその処理を移す」という仕組みを実現している．

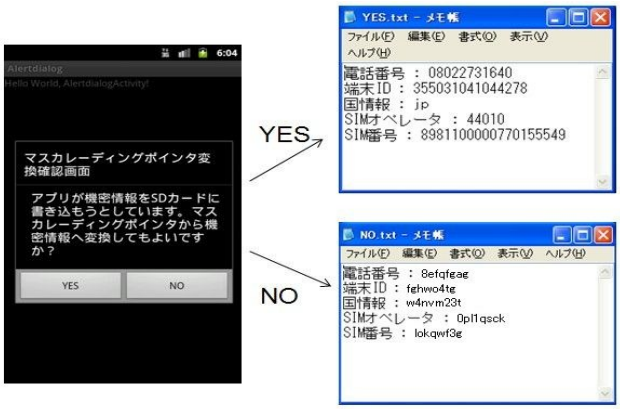

図 10. ユーザ承諾画面と SD カードに保存された機密情報

7. まとめ

本稿では，現在社会的問題になっているスマートフォンの情報漏洩に対する解決策として，機密情報をマスクするマスカレードポインタの機能を有したセキュリティマネージャを提案し実装した．提案方式は，端末側での OS による情報漏洩対策の一つであり,「アプリに機密情報を渡さない」というコンセプトに基づく点に特長を有する．

今後は，セキュリティマネージャのパフォーマンスに関する評価を行っていく予定である．また，提案方式においては, アプリ自身が機密情報そのものを有していないため，アプリが機密情報データを加工したり，機密情報の値に応じた処理を行う場合には，このままではアプリ内でそれらの操作を実行することができない．よって，秘匿計算を導入するなどの方法によって，アプリに機密情報を秘匿したままで，正規アプリ内での機密情報の加工や演算を可能とする仕組みを検討していきたい．

謝辞　岡山大学山内利宏先生，独立行政法人情報通信研究機構安藤類央様には，Android OS の内部構造に関する情報のご提供を頂いた．ここに謝意を表する．

参考文献

[1] Google Play Store:
https://play.google.com/store

[2] TrendLabs Security Blog:
http://blog.trendmicro.co.jp/archives/4714

[3] Yomiuri Online”Android 端末に「ボット」出現 ”:
http://www.yomiuri.co.jp/net/security/goshinjyutsu/20110107-OYT8T00678.htm

[4] ITmedia, Android OS から見たセキュリティ対策ソフトの制約:
http://www.itmedia.co.jp/enterprise/articles/1112/26/news015.html

[5] au Market:
http://www.au.kddi.com/seihin/ichiran/smartphone/app/index.html

[6] App Store Review Guidelines - App Store Resource Center:
http://developer.apple.com/jp/appstore/guidelines.html

[7] 上松晴信 可児潤也 名坂康平 川端秀明 磯原隆将 竹森敬祐 西垣正勝:” Android OS におけるマスカレーディングポインタを用いたプライバシー保護”, 情報処理学会研究報告，2012-CSEC, 2012.5

[8] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth : “TaintDroid: An Information - Flow Tracking System for Realtime Privacy Monitoring on Smartphones”, Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI’10), Canada, 2010

[9] Erika Chin, Adrienne Porter Felt, Kate Greenwood, David Wagner; “Analyzing inter-application communication in Android”, Proceedings of the 9th international conference on mobile systems, applications, and services, NY USA, 2011

[10] Seung-Hyun Seo, Dong-Guen Lee, Kangbin Yim : “Analysis on maliciousness for mobile applications”, 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Palermo Italy, 4-6 July 2012

[11] 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:“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, css2011, 2011.10

[12] 矢儀真也 山内利宏:“SEAndroid の拡張による AP の動的制御手法の提案”, コンピュータセキュリティシンポジウム 2012, css2012, 2012.10

[13] 松戸隆幸 児玉英一郎 王家宏 高田豊雄:”Android OS 上でのアプリケーション導入時におけるセキュリティ助言システムの提案”, 情報処理学会研究報告，2012-CSEC, 2012.2

[14] Android:http://developer.android.com/guide/basics/what-is-android.html

[15] ITPro，Android を狙う新たなワンクリ詐欺，ウイルスで料金請求:http://itpro.nikkeibp.co.jp/article/NEWS/20120113/378422/

[16] ITPro，スマホを狙うワンクリ詐欺の新手口，シャッター音や振動で脅かす:http://itpro.nikkeibp.co.jp/article/NEWS/20120312/385782/

[17] TechBooster，リフレクションを使ってメソッドを呼び出す: http://techbooster.org/android/hacks/13357/

[18] リフレクションおよびマッピングの使用について:
http://www.limy.org/program/java/coding/mapping_reflection.html

[19] AndroidGraphicsArchitecture I - himmele:https://himmele.googlecode.com

[20] 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, CSS’09, 2009.

[21] M. Ongtang, S. E. McLaughlin, W. Enck, and P. D. McDaniel. Semantically Rich Application-Centric Security in Android. In Proceedings of the 25th Annual Computer Security Applications Conference, ACSAC'09, 2009.

[22] M. Nauman, S. Khan, and X. Zhang. Apex: Extending Android permission model and enforcement with user-defined runtime constraints. In 5th ACM Symposium on Information Computer and Communications Security, ASIACCS'10, 2010.

[23] 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, In 2010 International Workshop on Smartphone Applications and Services, Smartphone’10, 2010.

[24] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In 18th ACM Conference on Computer and Communication Security, CCS'11, 2011.

[25] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin and D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior, In Symposium on Usable Privacy and Security, SOUPS’12, 2012.