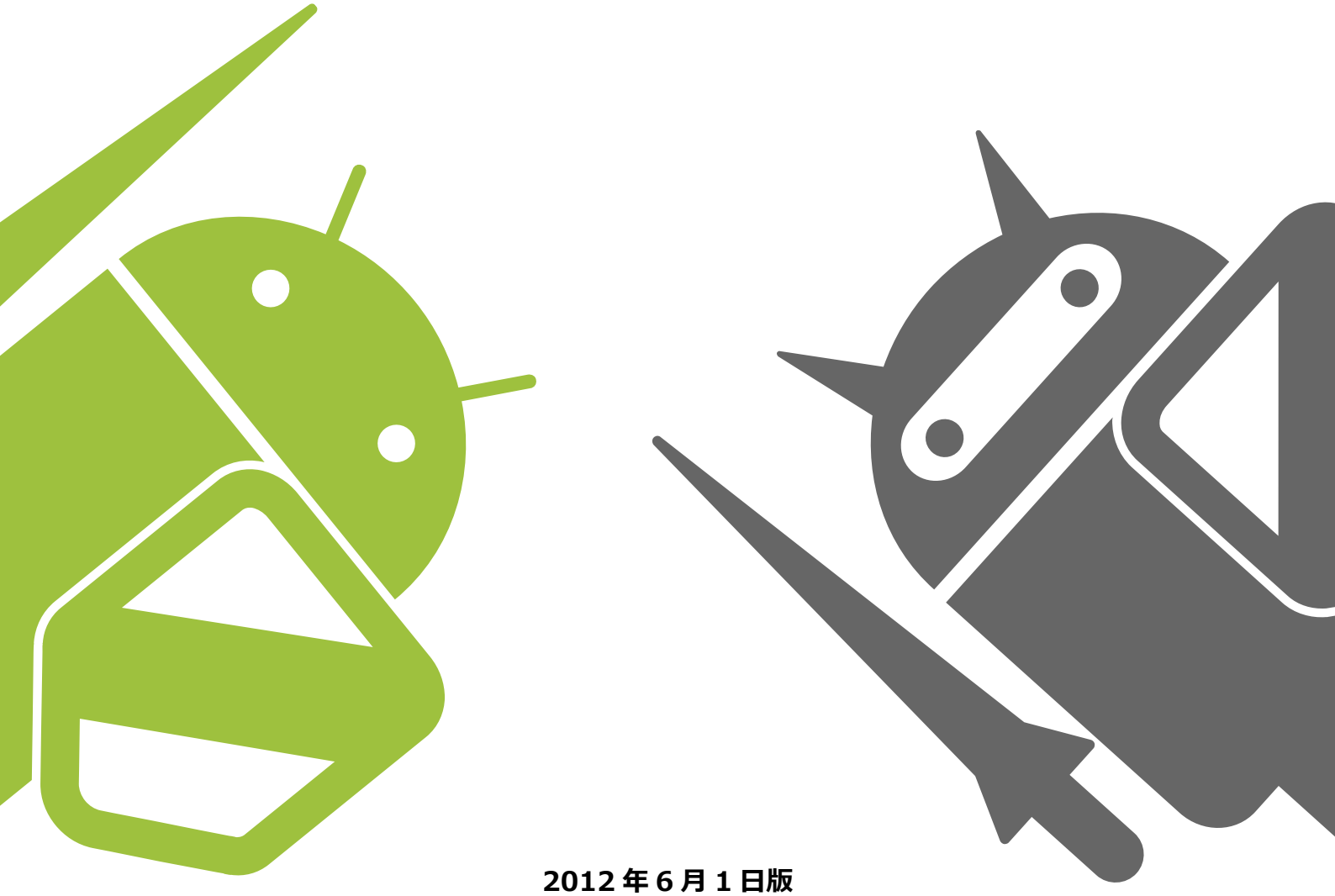
JAPAN SMARTPHONE SECURITY ASSOCIATION

Android アプリのセキュア設計
・
セキュアコーディングガイド

【みんなでスマフォが安全に使える世界へ！】



2012 年 6 月 1 日版

日本スマートフォンセキュリティ協会（JSSEC）

セキュアコーディンググループ

※　本ガイドの内容は執筆時点のものです。サンプルコードを使用する場合はこの点にあらかじめご注意ください。

※　JSSEC ならびに執筆関係者は、このガイド文書に関するいかなる責任も負うものではありません。全ては自己責任にてご活用ください。

Android アプリのセキュア設計・セキュアコーディングガイド

【β 版】
2012 年 6 月 1 日
日本スマートフォンセキュリティ協会（JSSEC）
セキュアコーディンググループ

目 次

更新履歴

日付	改訂内容
2012-06-01	• 初版

■制作■

日本スマートフォンセキュリティ協会（JSSEC）

技術部会　アプリケーションワーキンググループ　セキュアコーディンググループ

リーダー	松並　勝	ソニーデジタルネットワークアプリケーションズ株式会社
メンバー	佐藤　勝彦	Android セキュリティ部
	大内　智美	株式会社 SRA
	比嘉　陽一	株式会社 SRA
	星本　英史	株式会社 SRA
	武井　滋紀	エヌ・ティ・ティ・ソフトウェア株式会社
	久保　正樹	一般社団法人 JPCERT コーディネーションセンター（JPCERT/CC）
	熊谷　裕志	一般社団法人 JPCERT コーディネーションセンター（JPCERT/CC）
	戸田　洋三	一般社団法人 JPCERT コーディネーションセンター（JPCERT/CC）
	大園　通	シスコシステムズ合同会社
	谷田部 茂	シスコシステムズ合同会社
	田口　陽一	株式会社システムハウス. アイエヌジー
	坂本　昌彦	株式会社セキュアスカイ・テクノロジー
	安藤　彰	ソニーデジタルネットワークアプリケーションズ株式会社
	市川　茂	ソニーデジタルネットワークアプリケーションズ株式会社
	奥山　謙	ソニーデジタルネットワークアプリケーションズ株式会社
	佐藤　郁恵	ソニーデジタルネットワークアプリケーションズ株式会社
	西村　宗晃	ソニーデジタルネットワークアプリケーションズ株式会社
	山岡　一夫	ソニーデジタルネットワークアプリケーションズ株式会社
	倉永　英久	株式会社大和総研ビジネス・イノベーション
	谷口　岳	タオソフトウェア株式会社
	島野　英司	タオソフトウェア株式会社
	北村　久雄	タオソフトウェア株式会社
	佐藤　導吉	東京システムハウス株式会社
	服部　正和	トレンドマイクロ株式会社
	八津川　直伸	日本ユニシス株式会社
	千田　雅明	ネットエージェント株式会社
	藤井　茂樹	ユニアデックス株式会社

（執筆関係者、社名五十音順）

1. はじめに

1.1. スマートフォンを安心して利用出来る社会へ

本ガイドは Android アプリケーション開発者向けのセキュア設計、セキュアコーディングのノウハウをまとめた Tips 集です。できるだけ多くの Android アプリケーション開発者に活用していただきたく思い、ここに公開いたします。

昨今、スマートフォン市場は急拡大しており、さらにその勢いは増すばかりです。スマートフォン市場の急拡大は多種多彩なアプリケーション群によってもたらされています。従来の携帯電話ではセキュリティ制約によって利用できなかったさまざまな携帯電話の重要な機能がスマートフォンアプリケーションには解放され、従来の携帯電話では実現できなかった多種多彩なアプリケーション群がスマートフォンの魅力を引き立てています。

スマートフォンのアプリケーション開発者にはそれ相応の責任が生じています。従来の携帯電話ではあらかじめ課せられたセキュリティ制約によって、セキュリティについてあまり意識せずに開発したアプリケーションであっても比較的安全性が保たれていました。スマートフォンでは前述のとおり、携帯電話の重要な機能がアプリケーション開発者に解放されているため、アプリケーション開発者がセキュリティを意識して設計、コーディングをしなければ、スマートフォン利用者の個人情報が漏えいしたり、料金の発生する携帯電話機能をマルウェアに悪用されたりといった被害が生じます。

Android スマートフォンは iPhone に比べると、アプリケーション開発者のセキュリティへの配慮がより多く求められます。iPhone に比べ Android スマートフォンはアプリケーション開発者に開放された携帯電話機能が多く、App Store に比べ Google Play（旧 Android Market）は無審査でアプリケーション公開ができるなど、アプリケーションのセキュリティがほぼ全面的にアプリケーション開発者に任されているためです。

スマートフォン市場の急拡大にともない、様々な分野のソフトウェア技術者が一気にスマートフォンアプリケーション開発市場に流れ込んできており、スマートフォン特有のセキュリティを考慮したセキュア設計、セキュアコーディングのノウハウ集約、共有が急務となっています。

このような状況を踏まえ、日本スマートフォンセキュリティ協会はセキュアコーディンググループを立ち上げ、Android アプリケーションのセキュア設計、セキュアコーディングのノウハウを集めて、公開することにいたしました。それがこのガイド文書です。多くの Android アプリケーション開発者にセキュア設計、セキュアコーディングのノウハウを知っていただき、アプリケーション開発に活かしていただくことで、市場にリリースされる多くの Android アプリケーションのセキュリティを高めることを狙っています。その結果、安心、安全なスマートフォン社会づくりに貢献したいと考えています。

1.2. 常にβ版でタイムリーなフィードバックを

私たち JSSEC セキュアコーディンググループはこのガイド文書の内容について、できるだけ間違いがないように心がけておりますが、その正しさを保証するものではありません。私たちはタイムリーにノウハウを公開し共有していくことが第一と考え、最新かつその時点で正しいと思われることをできるだけ記載・公開し、間違いがあればフィードバックを頂いて常に正しい情報に更新し、タイムリーに提供するよう心がける、いわゆる常にβ版というアプローチをとっています。このアプローチはこのガイド文書をご利用いただく多くの Android アプリケーション開発者のみなさまにとって有意義であると私たちは信じています。

このガイド文書とサンプルコードの最新版はいつでも下記 URL から入手できます。

- http://www.jssec.org/dl/android_securecoding.pdf ガイド文書
- http://www.jssec.org/dl/android_securecoding.zip サンプルコード一式

1.3. 本文書の利用許諾

このガイド文書のご利用に際しては次の 2 つの注意事項に同意いただく必要がございます。

1. このガイド文書には間違いが含まれている可能性があり、ご自身の責任のもとでご利用ください。
2. このガイド文書に含まれる間違いを見つけた場合には、下記連絡先までメールにてご連絡ください。ただしお返事することや修正をお約束するものではありませんのでご了承ください。

一般社団法人　日本スマートフォンセキュリティ協会
メール宛先：　sec@jssec.org
※：タイトルに「セキュアコーディングガイド 2012 年〇月〇日版についての意見」とし、可能であれば、お名前、連絡先を明記してください。

2. ガイド文書の構成

2.1. 開発者コンテキスト

セキュアコーディング系のガイド文書は「こういうコーディングは危ない、だからこのようにコーディングすべき」といった内容で構成されることが多いのですが、このような構成はすでにコーディングされたソースコードをレビューするときには役立つ反面、これから開発者がコーディングしようというときには、どの記事を読んだらよいのか分かりにくいという問題があります。

このガイド文書では、開発者がいま何をしようとしているか？という開発者コンテキストに着目し、開発者コンテキストに合わせた切り口の記事を用意する方針をとっています。たとえば「Activity を作る」ですとか「SQLite を使う」という開発者が行うであろう作業単位ごとに記事を用意しています。

開発者コンテキストに合わせて記事を用意することにより、開発者は必要な記事を見つけやすく、業務にすぐ役立つようになると考えています。

2.2. サンプルコード、ルールブック、アドバンスト

それぞれの記事はサンプルコード、ルールブック、アドバンストの 3 つのセクションで構成されています。お急ぎの方はサンプルコードとルールブックをご覧ください。ある程度再利用可能なパターンに落とし込んだ内容にしてあります。サンプルコードセクションとルールブックセクションに収まらない課題をお持ちの方はアドバンストをご覧ください。個別課題の解決方法を検討するための考慮材料を記載してあります。

2.2.1. サンプルコード

サンプルコードセクションでは、その記事がテーマとする開発者コンテキストにおいて基本的なお手本となるサンプルコードを掲載しています。複数のパターンがある場合はその分類方法とそれぞれのパターンのサンプルコードを用意しています。解説は簡潔さを心がけており、本文中にセキュリティ上考慮すべきポイントを「**ポイント:**」部分に番号付き箇条書きで記載し、その箇条書き番号 N に対応するサンプルコード上のコード部分にも「**★ポイント N★**」と記載しコメントで解説しています。一つのポイントがサンプルコード上では複数個所に対応することにご注意ください。このようにセキュリティを考慮すべき個所はソースコード全体に対して僅かの量ですが、その個所は点在します。セキュリティの考慮が必要な個所を見渡せるように、サンプルコードは 1 クラスといった単位でまるごと掲載するようにしています。

このガイド文書で掲載しているサンプルコードは一部です。すべてのサンプルコードをまとめた圧縮ファイルも下記の URL に公開していますので、ダウンロードして eclipse に Import するなどしてご利用ください。サンプルコードは Apache License, Version 2.0 で公開していますので、自由にサンプルコードをコピーペーストしてご利用いただけます。ただしエラー処理についてはサンプルコードが長くなり過ぎないように最小限にしていますのでご注意ください。

- http://www.jssec.org/dl/android_securecoding.pdf ガイド文書
- http://www.jssec.org/dl/android_securecoding.zip サンプルコード一式

サンプルコードに添付する Projects/keystore ファイルは APK 署名用の開発者鍵を含んだキーストアファイルです。パスワードは「android」です。自社限定系のサンプルコードを APK 署名する際にご利用ください。

2.2.2. ルールブック

ルールブックセクションでは、その記事がテーマとする開発者コンテキストにおいて、セキュリティ観点から守るべきルールや考慮事項を掲載しています。ルールブックセクションの冒頭にはそのセクションで扱っているルールを表形式で一覧表示し、「必須」または「推奨」のレベル分けをしています。ルールには肯定文または否定文の 2 種類がありますので、必須の肯定文は「やらなきゃだめ」、推奨の肯定文は「やったほうがよい」、必須の否定文は「やったらだめ」、推奨の否定文は「やらないほうがよい」といったレベル感で表現しています。もちろんこのレベル分けは執筆者の主観に基づくものですので、参考程度としてお取扱いください。

サンプルコードセクションに掲載されているサンプルコードはこれらのルールや考慮事項が反映されたものとなっていますが、その詳しい説明はルールブックセクションに記載されています。また、サンプルコードセクションでは扱っていないルールや考慮事項についてもルールブックセクションでは扱っています。

2.2.3. アドバンスト

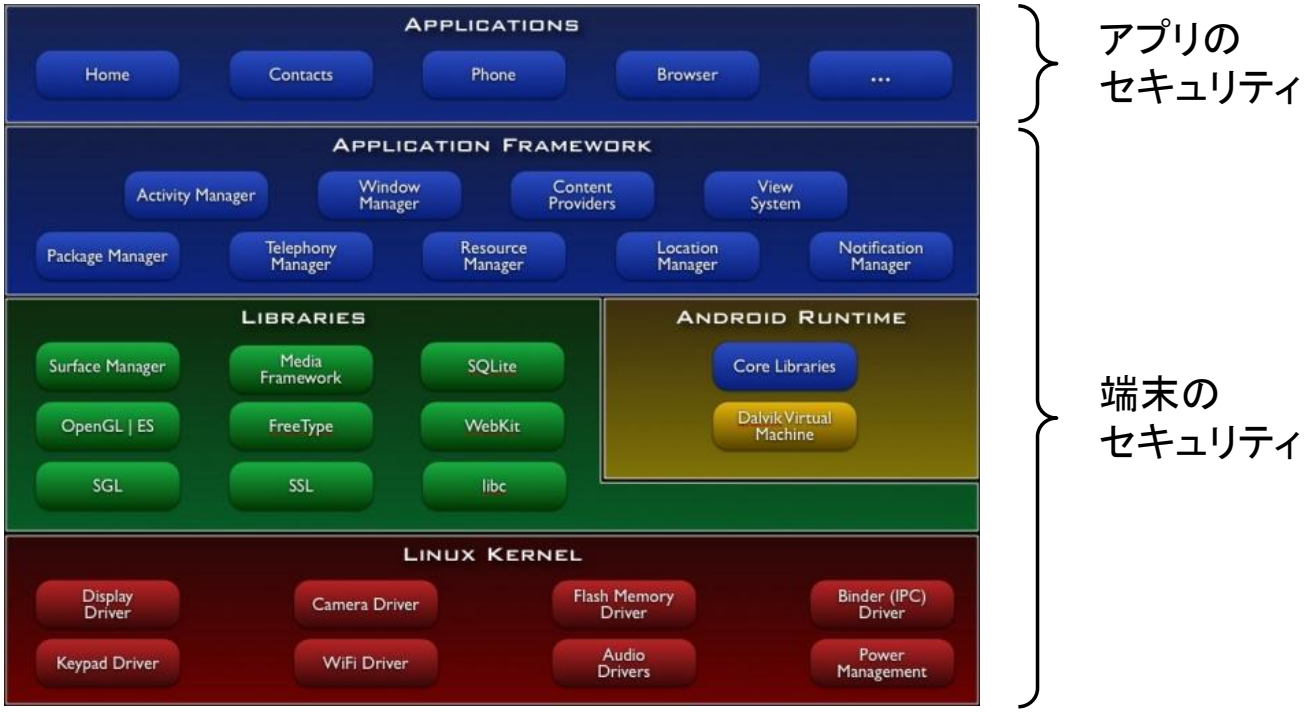
アドバンストセクションでは、その記事がテーマとする開発者コンテキストにおいて、サンプルコードセクションやルールブックセクションで説明できなかった、しかし注意を要する事項について記載しています。その記事がテーマとする開発者コンテキストにまつわる、コラム的な話題や Android OS の限界に関する話題など、サンプルコードセクションやルールブックセクションの内容で解決できなかった個別課題の解決方法を検討するための考慮材料として役立てることができます。

開発者のみなさんは常に多忙です。開発者の多くは、Android の深遠なるセキュリティの構造について深く理解することよりも、ある程度の Android セキュリティの知識を持って、迅速にかつ安全な Android アプリケーションをどんどん生産することが求められます。一方、セキュリティ設計が重要なアプリケーションもあります。このようなアプリケーションの開発者は Android のセキュリティについて深く理解している必要があります。

このようにスピード重視の開発者とセキュリティ重視の開発者の両方を支援するために、このガイド文書のすべての記事はサンプルコード、ルールブック、アドバンストの 3 つのセクションに分けて記述しています。サンプルコードとルールブックセクションは「そういうことがしたければ、これをしておけば安全ですよ」といった一般化できる内容が書いてあり、可能な限りソースコードのコピーペーストで自動的に安全なコーディングができることを狙っています。アドバンストセクションは「こんなときはこういう問題があって、こういう考え方をするとよい」といった考えるための材料が書いてあり、開発者が取り組んでいる個別のアプリケーションで最適なセキュア設計、セキュアコーディングを検討できることを狙っています。

2.3. ガイド文書のスコープ

このガイド文書は一般の Android アプリ開発者に必要なセキュリティ Tips を集めることを目的としています。そのため主にマーケット等で配布される Android アプリの開発におけるセキュリティ Tips（下図の「アプリのセキュリティ」）が主なスコープとなっています。



Android OS 層以下の Android 端末実装に関するセキュリティ（上図の「端末のセキュリティ」）はスコープ外です。また Android 端末にユーザーがインストールする一般の Android アプリと、Android 端末メーカーがプレインストールする Android アプリでは気を付けるべきセキュリティの観点で異なるところがありますが、特に初版においては前者のみを扱っており、後者については扱っていません。初版では Java により実装する Tips だけを記載しておりますが、JNI 実装についても今後の版で記載していく予定です。

root 権限が奪取される脅威についても今のところ扱っていません。基本的には root 権限が奪われていないセキュアな Android 端末を前提とし、Android OS のセキュリティモデルを活用したセキュリティ Tips をまとめています。一般に root 権限奪取の脅威についてセキュリティ対策を実施するには、まず root 権限が奪われたときにも死守しなければならない資産と、そうでない資産を区別します。一般に、root 権限が奪われたときにも死守すべき資産はシステム全体に比べるとほんの一部です。そのような資産を守るためには、Android OS のセキュリティモデルが活用できないため、暗号化、難読化、ハードウェア支援、サーバー支援など複数の手段を組み合わせて高度な防御設計をします。これはガイド文書に簡潔に書けるようなノウハウではありませんし、個別の状況に応じて適切な防御設計は異なります。もし root 権限奪取の脅威がビジネス上大きなインパクトを与える場合は、Android の耐タンパ設計に詳しいセキュリティ専門家にご相談されることをお勧めします。

2.4. Android セキュアコーディング関連書籍の紹介

このガイド文書では Android セキュアコーディングのすべてを扱うことはとてもできないので、下記で紹介する書籍を併用することをお勧めします。

- Android Security 安全なアプリケーションを作成するために
  著者:タオソフトウェア株式会社 ISBN978-4-8443-3134-6
  http://www.amazon.co.jp/dp/4844331345/
- Java セキュアコーディングスタンダード CERT/ Oracle 版
  著者:Fred Long, Dhruv Mohindra, Robert C. Seacord, Dean F. Sutherland, David Svoboda
  監修:歌代和正 翻訳:久保正樹, 戸田洋三 ISBN978-4-04-886070-3
  http://www.amazon.co.jp/dp/4048860704/

3. セキュアコーディングの基礎知識

このガイド文書は Android OS に関するセキュリティ Tips をまとめるものであるが、この章では Android OS に依存しない一般的なセキュアコーディングの基礎知識を扱う。なぜなら、後続の章から一般的なセキュアコーディングの解説が必要なときに本章の記事を参照するためである。後続の章を読み進める前に本章の内容に一通り目を通しておくことをお勧めする。

3.1. 入力データの安全性を確認する

入力データの安全性確認はもっとも基礎的で効果の高いセキュアコーディング作法である。プログラムに入力されるデータのうち、攻撃者が直接的、間接的にそのデータの値を操作可能であるものはすべて、入力データの安全性確認が必要である。以下、Activity をプログラムに見立て、Intent を入力データに見立てた場合を例にして、入力データの安全性確認の在り方について解説する。

Activity が受け取った Intent には攻撃者が細工したデータが含まれている可能性がある。攻撃者はプログラマが想定していない形式・値のデータを送り付けることで、アプリケーションに誤動作を誘発し、結果として何らかのセキュリティ被害を生じさせるのである。ユーザーも攻撃者の一人となり得ることも忘れてはならない。

Intent は action や data、extras などのデータで構成されるが、攻撃者が制御可能なデータはすべて気を付けなければならない。攻撃者が制御可能なデータを扱うコードでは、必ず次の事項を確認しなければならない。

(a) 受け取ったデータは、プログラマが想定した形式であって、値は想定の範囲内に収まっているか？
(b) 想定している形式、値のあらゆるデータを受け取っても、そのデータを扱うコードが想定外の動作をしないと保証できるか？

次の例は指定 URL の Internet 上の Web ページの HTML を取得し、画面上の TextView に表示するだけの簡単なサンプルである。 しかしこれには不具合がある。

Internet 上の Web ページの HTML を TextView に表示するサンプルコード

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { ...
```

(a)の観点で「urlstr が正しい URL である」ことを new URL()で MalformedURLException が発生しないことにより確認している。しかしこれは不十分であり、urlstr に「file://～」形式の URL が指定されると Internet 上の Web ペー

ジではなく、内部ファイルシステム上のファイルを開いて TextView に表示してしまう。プログラマが想定した動作を保証していないため、(b)の観点を満たしていない。

次は改善例である。(a)の観点で「urlstr は正規の URL であって、protocol は http または https に限定される」ことを確認している。これにより(b)の観点でも url.openConnection().getInputStream()では Internet 経由の InputStream を取得することが保証される。

Internet 上の Web ページの HTML を TextView に表示するサンプルコードの修正版

```
TextView tv = (TextView) findViewById(R.id.textview);
InputStreamReader isr = null;
char[] text = new char[1024];
int read;
try {
    String urlstr = getIntent().getStringExtra("WEBPAGE_URL");
    URL url = new URL(urlstr);
    String prot = url.getProtocol();
    if (!"http".equals(prot) && !"https".equals(prot)) {
        throw new MalformedURLException("invalid protocol");
    }
    isr = new InputStreamReader(url.openConnection().getInputStream());
    while ((read=isr.read(text)) != -1) {
        tv.append(new String(text, 0, read));
    }
} catch (MalformedURLException e) { ...
```

入力データの安全性確認は Input Validation と呼ばれる基礎的なセキュアコーディング作法である。Input Validation という言葉の語感から(a)の観点のみ気を付けて(b)の観点を忘れてしまいがちである。データはプログラムに入ってきたときではなく、プログラムがそのデータを「使う」ときに被害が発生することに気を付けていただきたい。下記 URL もぜひ参考にしていただきたい。

- IPA「セキュアプログラミング講座」
  http://www.ipa.go.jp/security/awareness/vendor/programmingv2/clanguage.html
- JPCERT CC「Java セキュアコーディングスタンダード CERT/Oracle 版」
  http://www.jpcert.or.jp/java-rules/
- JPCERT CC「Java Android アプリケーション開発へのルールの適用」
  http://www.jpcert.or.jp/java-rules/android-j.html

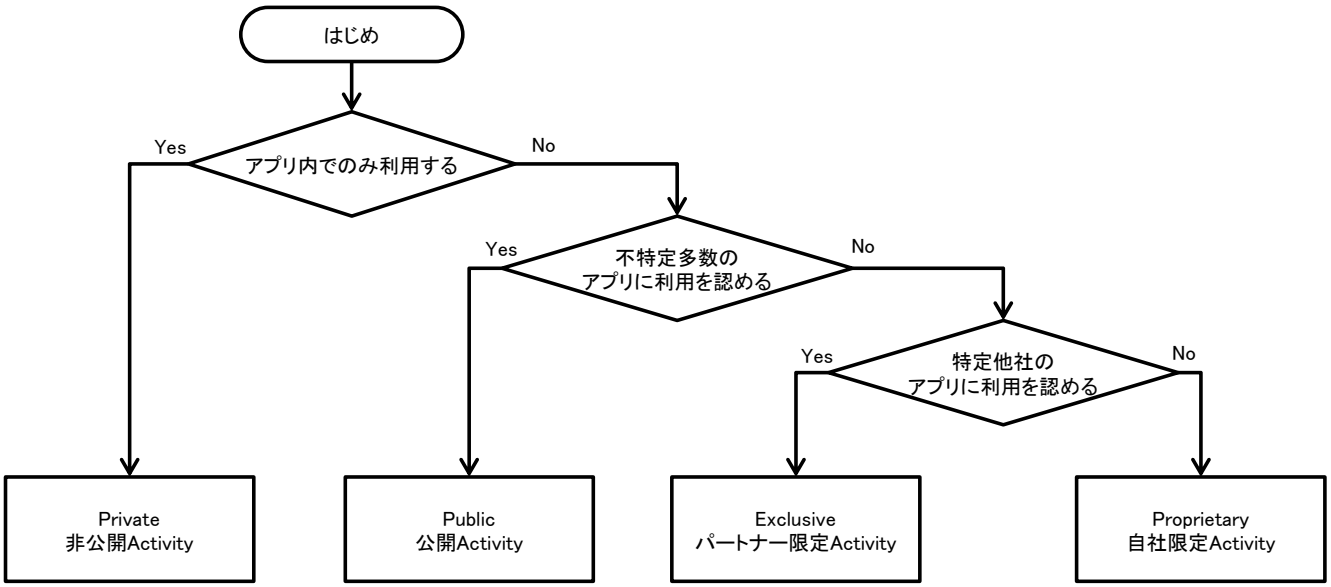
4. 安全にテクノロジーを活用する

Android で言えば Activity や SQLite など、テクノロジーごとにセキュリティ観点の癖というものがある。そうしたセキュリティの癖を知らずに設計、コーディングしていると思わぬ脆弱性をつくりこんでしまうことがある。この章では開発者が Android のテクノロジーを活用するシーンを想定した記事を扱う。

4.1. Activity を作る

4.1.1. サンプルコード

Activity がどのように利用されるかによって、Activity が抱えるリスクや適切な防御手段が異なる。次の判定フローによって作成する Activity がどのタイプであるかを判断できる。



4.1.1.1. 非公開 Activity を作る

非公開 Activity は、同一アプリ内でのみ利用される Activity であり、もっとも安全性の高い Activity である。

ポイント:

1. exported="false"により、明示的に非公開設定する
2. 同一アプリからの Intent であっても、受信 Intent の安全性を確認する
3. 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい

Activity を非公開設定するには、AndroidManifest.xml の activity 要素の exporeted 属性を false と指定する。

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.privateactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- 非公開 Activity -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
    </application>
</manifest>
```

PrivateActivity.java

```
package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateActivity extends Activity {

    @Override
```

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.private_activity);

        // ★ポイント 2★ 同一アプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param), Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ポイント 3★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "センシティブな情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```


4.1.1.2. 公開 Activity を作る

公開 Activity は、不特定多数のアプリに利用されることを想定した Activity である。マルウェアが送信した Intent を受信することがあることに注意が必要だ。

ポイント:

1. 受信 Intent の安全性を確認する
2. 結果を返す場合、センシティブな情報を含めない

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.publicactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- 公開 Activity -->
        <activity
            android:name=".PublicActivity"
            android:label="@string/app_name" >

            <!-- Action 指定による暗黙的 Intent を受信するように Intent Filter を定義 -->
            <intent-filter>
                <action android:name="org.jssec.android.activity.MY_ACTION" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

PublicActivity.java

```
package org.jssec.android.activity.publicactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PublicActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // ★ポイント 1★ 受信 Intent の安全性を確認する
        // 公開 Activity であるため利用元アプリがマルウェアである可能性がある。
```

```
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param), Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ポイント 2★ 結果を返す場合、センシティブな情報を含めない
        // 公開 Activity であるため利用元アプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。

        Intent intent = new Intent();
        intent.putExtra("RESULT", "センシティブではない情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```


4.1.1.3. パートナー限定 Activity を作る

パートナー限定 Activity は、特定のアプリだけから利用できる Activity である。パートナー企業のアプリと自社アプリが連携してシステムを構成し、パートナーアプリとの間で扱う情報や機能を守るために利用される。

ポイント:

1. Intent Filter は定義してはならない
2. 利用元アプリの証明書がホワイトリストに登録されていることを確認する
3. パートナーアプリからの Intent であっても、受信 Intent の安全性を確認する
4. パートナーアプリに開示してよい情報に限り返送してよい

なお、ホワイトリストに指定する利用先アプリの証明書ハッシュ値の確認方法は「5.2.1.3 アプリの証明書のハッシュ値を確認する方法」を参照すること。

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.exclusiveactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 1★ Intent Filter を定義してはならない -->
        <activity
            android:name=".ExclusiveActivity"
            android:exported="true" />

    </application>
</manifest>
```

ExclusiveActivity.java

```
package org.jssec.android.activity.exclusiveactivity;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class ExclusiveActivity extends Activity {

    // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
```

```
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナーアプリ org.jssec.android.activity.exclusiveuser の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.activity.exclusiveuser", isdebug ?
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // keystore の"partner key"の証明書ハッシュ値
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // 以下同様に他のパートナーアプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(this, getCallingPackage())) {
            Toast.makeText(this, "利用元アプリはパートナーアプリではない。", Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // ★ポイント 3★ パートナーアプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        Toast.makeText(this, "パートナーアプリからアクセスあり", Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ポイント 4★ パートナーアプリに開示してよい情報に限り返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "パートナーアプリに開示してよい情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
```



```
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // 0-9A-F 以外の文字がある

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
```

```
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```


4.1.1.4. 自社限定 Activity を作る

自社限定 Activity は、自社以外のアプリから利用されることを禁止する Activity である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。

ポイント:

1. 独自定義 Signature Permission を定義する
2. Activity 定義にて、独自定義 Signature Permission を要求宣言する
3. Activity 定義にて、Intent Filter を定義してはならない
4. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
5. 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
6. 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
7. 利用元アプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.proprietaryactivity"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント 1★ 独自定義 Signature Permission を定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 2★ 独自定義 Signature Permission を要求宣言する -->
        <!-- ★ポイント 3★ Intent Filter を定義してはならない -->
        <activity
            android:name=".ProprietaryActivity"
            android:exported="true"
            android:permission="org.jssec.android.permission.MY_PERMISSION" />
    </application>
</manifest>
```

ProprietaryActivity.java

```
package org.jssec.android.activity.proprietaryactivity;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.Toast;

public class ProprietaryActivity extends Activity {

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // ★ポイント 4★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "独自定義 Signature Permission が自社アプリにより定義されていない。", Toast.LENGT
H_LONG).show();
            finish();
            return;
        }

        // ★ポイント 5★ 自社アプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        String param = getIntent().getStringExtra("PARAM");
        Toast.makeText(this, String.format("パラメータ「%s」を受け取った。", param), Toast.LENGTH_LONG).show();
    }

    public void onReturnResultClick(View view) {

        // ★ポイント 6★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "センシティブな情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
```



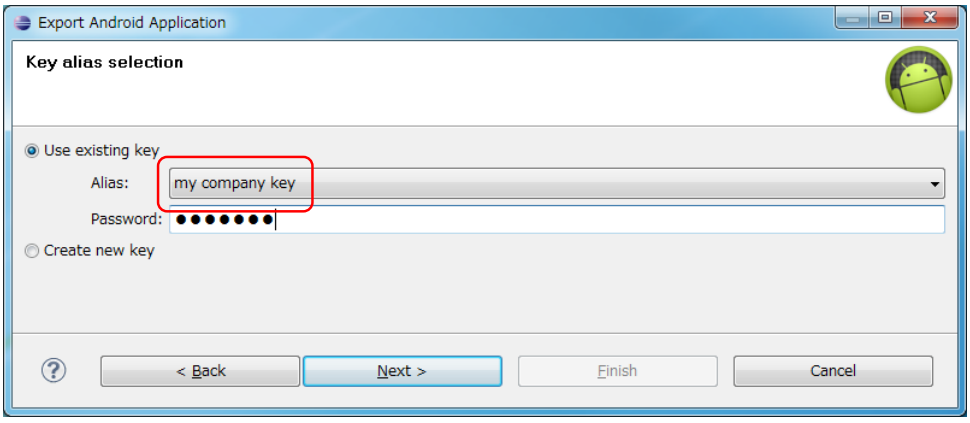
```
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // sigPermName を定義したアプリの証明書のハッシュ値を返す
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
```

```
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

★ポイント 7★ eclipse から APK を Export するときに、利用元アプリと同じ開発者鍵で APK を署名する。



4.1.2. ルールブック

Activity を作る際には以下のルールを守ること。

1. 内部使用の Activity は非公開設定する	（必須）
2. 受信 Intent の安全性を確認する	（必須）
3. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する	（必須）
4. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する	（必須）

4.1.2.1. 内部使用の Activity は非公開設定する （必須）

同一アプリ内からのみ利用される Activity は他のアプリから Intent を受け取る必要がないだけでなく、開発者も Activity を攻撃する Intent を受信する可能性を考慮しないことが多い。このような Activity は明示的に非公開設定し、非公開 Activity とすべきである。

AndroidManifest.xml
```
        <!-- 非公開 Activity -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <activity
            android:name=".PrivateActivity"
            android:label="@string/app_name"
            android:exported="false" />
```

このようなケースは少ないと思われるが、同一アプリ内からのみ利用する Activity であり、かつ Intent Filter を設置する場合は、次のように exported 属性を付加して明示的に非公開とする。

AndroidManifest.xml
```
        <!-- 非公開 Activity -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <activity
            android:name=".PictureActivity"
            android:label="@string/picture_name"
            android:exported="false" >
            <intent-filter>
                <action android:name="org.jssec.android.activity.OPEN" />
            </intent-filter>
        </activity>
```

4.1.2.2. 受信 Intent の安全性を確認する （必須）

Activity のタイプによって若干リスクは異なるが、基本的には受信 Intent のデータを処理する際には、まず最初に受信 Intent の安全性を確認しなければならない。

公開 Activity は不特定多数のアプリから Intent を受け取るため、マルウェアの攻撃 Intent を受け取る可能性がある。非公開 Activity は他のアプリから Intent を直接受け取ることはない。しかし同一アプリ内の公開 Activity が他

のアプリから受け取った Intent のデータを非公開 Activity に転送することがあるため、受信 Intent を盲目的に安全であると考えてはならない。パートナー限定 Activity や自社限定 Activity はその中間のリスクであるため、やはり受信 Intent の安全性を確認する必要がある。

「3.1 入力データの安全性を確認する」を参照すること。

4.1.2.3. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）

自社アプリだけから利用できる自社限定 Activity を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2 Permission と Protection Level」の「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

4.1.2.4. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する （必須）

Activity のタイプによって setResult()により結果情報の返送先アプリの信用度が異なる。公開 Activity が結果情報を返送する場合、結果返送先アプリがマルウェアである可能性があり、結果情報が悪意を持って使われる危険性がある。非公開 Activity や自社限定 Activity の場合は、結果返送先は自社アプリであるため結果情報の扱いをあまり心配する必要はない。パートナー限定 Activity の場合はその中間に位置する。

このように Activity から結果情報を返す場合には、返送先アプリからの結果情報の漏洩に配慮しなければならない。

結果情報を返送する場合の例

```
    public void onReturnResultClick(View view) {

        // ★ポイント 4★ パートナーアプリに開示してよい情報に限り返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "パートナーアプリに開示してよい情報");
        setResult(RESULT_OK, intent);
        finish();
    }
```


4.1.3. アドバンスト

4.1.3.1. exported 設定と intent-filter 設定の組み合わせ

このガイド文書では、Activity の用途から非公開 Activity、公開 Activity、パートナー限定 Activity、自社限定 Activity の 4 タイプの Activity について実装方法を述べている。各タイプに許されている AndroidManifest.xml の exported 属性と intent-filter 要素の組み合わせを次の表にまとめた。作ろうとしている Activity のタイプと exported 属性および intent-filter 要素の対応が正しいことを確認すること。

	exported 属性の値		
	true	false	無指定
intent-filter 定義がある	公開、パートナー限定	非公開	公開、パートナー限定
intent-filter 定義がない	公開、パートナー限定、自社限定	非公開	非公開

4.1.3.2. 利用元アプリを確認する

ここではパートナー限定 Activity の実装に関する技術情報を解説する。パートナー限定 Activity はホワイトリストに登録された特定のアプリからのアクセスを許可し、それ以外のアプリからはアクセスを拒否する Activity である。自社以外のアプリもアクセス許可対象となるため、Signature Permission による防御手法は利用できない。

基本的な考え方は、パートナー限定 Activity の利用元アプリの身元を確認し、ホワイトリストに登録されたアプリであればサービスを提供する、登録されていないアプリであればサービスを提供しないというものである。利用元アプリの身元確認は、利用元アプリが持つ証明書を取得し、その証明書のハッシュ値をホワイトリストのハッシュ値と比較することで行う。

ここでわざわざ利用元アプリの「証明書」を取得せずとも、利用元アプリの「パッケージ名」との比較で十分ではないか？と疑問を持たれた方もいるかと思う。しかしパッケージ名は任意に指定できるため他のアプリへの成りすましが簡単である。成りすまし可能なパラメータは身元確認用には使えない。一方、アプリの持つ証明書であれば身元確認に使うことができる。証明書に対応する署名用の開発者鍵は本物のアプリ開発者しか持っていないため、第三者が同じ証明書を持ち、尚且つ署名検証が成功するアプリを作成することはできないからだ。ホワイトリストはアクセスを許可したいアプリの証明書データを丸ごと保持してもよいが、サンプルコードではホワイトリストのデータサイズを小さくするために証明書データの SHA-256 ハッシュ値を保持することにしている。

このテクニックには次の二つの制約条件がある。

- 利用元アプリにおいて startActivity()ではなく startActivityForResult()を使用しなければならない
- 利用元アプリにおいて Activity 以外から呼び出すことはできない

2 つ目の制約事項はいわば 1 つ目の制約事項の結果として課される制約であるので、厳密には 1 つの同じ制約と言える。この制約は呼び出し元アプリのパッケージ名を取得する Activity.getCallingPackage()の制約により生じてい

る。Activity.getCallingPackage()は startActivityForResult()で呼び出された場合にのみ利用元アプリのパッケージ名を返すが、残念ながら startActivity()で呼び出された場合には null を返す仕様となっている。そのためここで紹介するテクニックは必ず利用元アプリが、たとえ戻り値が不要であったとしても、startActivityForResult()を使わなければならないという制約がある。さらに startActivityForResult()は Activity クラスでしか使えないため、利用元は Activity に限定されるという制約もある。

ExclusiveActivity.java

```
package org.jssec.android.activity.exclusiveactivity;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class ExclusiveActivity extends Activity {

    // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナーアプリ org.jssec.android.activity.exclusiveuser の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.activity.exclusiveuser", isdebug ?
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // keystore の"partner key"の証明書ハッシュ値
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // 以下同様に他のパートナーアプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // ★ポイント 2★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(this, getCallingPackage())) {
            Toast.makeText(this, "利用元アプリはパートナーアプリではない。", Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        // ★ポイント 3★ パートナーアプリからの Intent であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        Toast.makeText(this, "パートナーアプリからアクセスあり", Toast.LENGTH_LONG).show();
    }
```



```
    public void onReturnResultClick(View view) {

        // ★ポイント 4★ パートナーアプリに開示してよい情報に限り返送してよい
        Intent intent = new Intent();
        intent.putExtra("RESULT", "パートナーアプリに開示してよい情報");
        setResult(RESULT_OK, intent);
        finish();
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // 0-9A-F 以外の文字がある

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

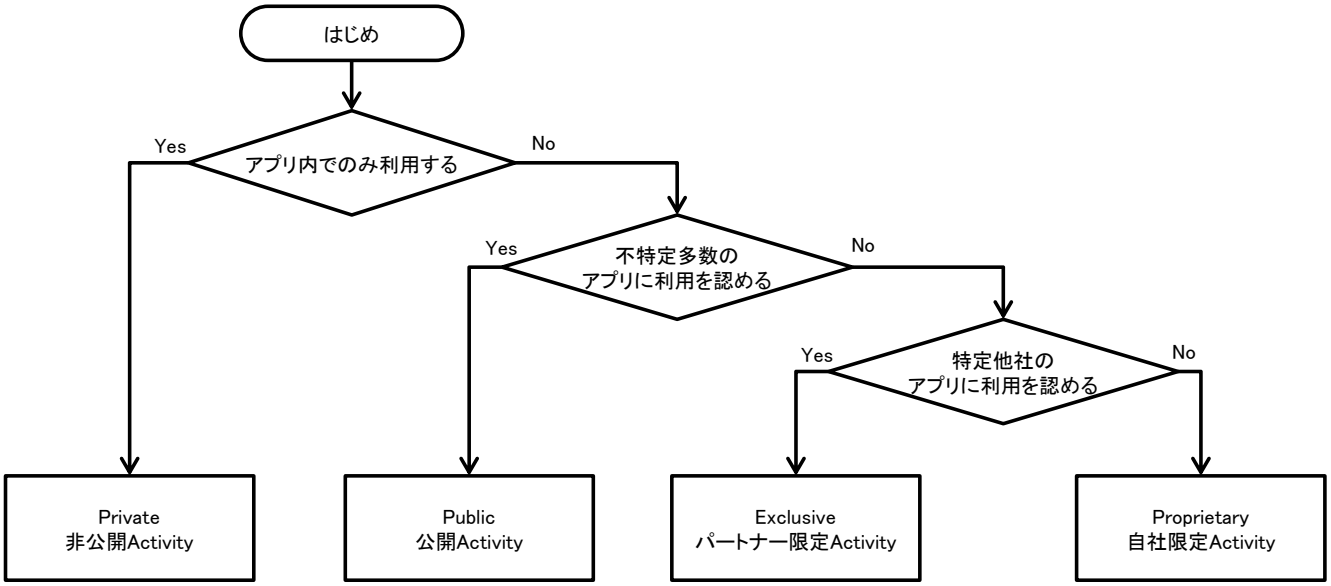
```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {
```

```
    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

4.2. Activity を利用する

4.2.1. サンプルコード

利用先 Activity がどのように利用されることを前提としているかによって、利用元 Component に発生するリスクや適切な利用方法が異なる。次の判定フローによって利用する Activity がどのタイプであるかを判断できる。



4.2.1.1. 非公開 Activity を利用する

同一アプリ内だけで利用される Activity(非公開 Activity)を利用するには、クラスを指定する明示的 Intent を使えば誤って外部アプリに Intent を送信してしまうことがない。

ポイント:

1. 同一アプリ内 Activity はクラス指定の明示的 Intent で呼び出す
2. 利用先アプリは同一アプリであるから、センシティブな情報を送信してもよい
3. 同一アプリ内 Activity からの結果情報であっても、受信データの安全性を確認する

PrivateUserActivity.java

```
package org.jssec.android.activity.privateactivity;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PrivateUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.user_activity);
    }

    public void onUseActivityClick(View view) {

        Intent intent = new Intent();

        // ★ポイント1★ 同一アプリ内 Activity はクラス指定の明示的 Intent で呼び出す
        intent.setClass(this, PrivateActivity.class);

        // ★ポイント2★ 利用先アプリは同一アプリであるから、センシティブな情報を送信してもよい
        intent.putExtra("PARAM", "センシティブな情報");

        startActivityForResult(intent, REQUEST_CODE);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // ★ポイント3★ 同一アプリ内 Activity からの結果情報であっても、受信データの安全性を確認する
            // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
            Toast.makeText(this, String.format("結果「%s」を受け取った。", result), Toast.LENGTH_LONG).show();
```

```
                break;
            }
        }
    }
```

4.2.1.2. 公開 Activity を利用する

公開 Activity を利用する場合、送信する Intent がマルウェアに受信されることがあることに注意が必要である。

ポイント:

1. センシティブな情報を送信してはならない
2. 結果を受け取る場合、結果データの安全性を確認する

PublicUserActivity.java

```
package org.jssec.android.activity.publicuser;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class PublicUserActivity extends Activity {

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        try {
            // ★ポイント 1★ センシティブな情報を送信してはならない
            Intent intent = new Intent("org.jssec.android.activity.MY_ACTION");
            intent.putExtra("PARAM", "センシティブではない情報");
            startActivityForResult(intent, REQUEST_CODE);
        } catch (ActivityNotFoundException e) {
            Toast.makeText(this, "利用先 Activity が見つからない。", Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        // ★ポイント 2★ 結果を受け取る場合、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        if (resultCode != RESULT_OK) return;
        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");
            Toast.makeText(this, String.format("結果「%s」を受け取った。", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```


4.2.1.3. パートナー限定 Activity を利用する

パートナー限定 Activity は、特定のアプリだけから利用できる Activity である。パートナー企業のアプリと自社アプリが連携してシステムを構成し、パートナーアプリとの間で扱う情報や機能を守るために利用される。ここではパートナー限定 Activity を呼び出す方法を説明する。

ポイント:

1. 利用先パートナー限定 Activity アプリの証明書がホワイトリストに登録されていることを確認する
2. 利用先パートナー限定アプリに開示してよい情報に限り送信してよい
3. 明示的 Intent によりパートナー限定 Activity を呼び出す
4. startActivityForResult()によりパートナー限定 Activity を呼び出す
5. パートナー限定アプリからの結果情報であっても、受信 Intent の安全性を確認する

なお、ホワイトリストに指定する利用先アプリの証明書ハッシュ値の確認方法は「5.2.1.3 アプリの証明書のハッシュ値を確認する方法」を参照すること。

ExclusiveUserActivity.java

```
package org.jssec.android.activity.exclusiveuser;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class ExclusiveUserActivity extends Activity {

    // ★ポイント 1★ 利用先パートナー限定 Activity アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナー限定 Activity アプリ org.jssec.android.activity.exclusiveactivity の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.activity.exclusiveactivity", isdebug ?
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // keystore の"my company key"の証明書ハッシュ値
                "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA");

        // 以下同様に他のパートナー限定 Activity アプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }
```

```
    private static final int REQUEST_CODE = 1;

    // 利用先のパートナー限定 Activity に関する情報
    private static final String TARGET_PACKAGE =  "org.jssec.android.activity.exclusiveactivity";
    private static final String TARGET_ACTIVITY = "org.jssec.android.activity.exclusiveactivity.ExclusiveActivity
";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        // ★ポイント 1★ 利用先パートナー限定 Activity アプリの証明書がホワイトリストに登録されていることを確認す
る
        if (!checkPartner(this, TARGET_PACKAGE)) {
            Toast.makeText(this, "利用先 Activity アプリはホワイトリストに登録されていない。", Toast.LENGTH_LONG).
show();
            return;
        }

        try {
            Intent intent = new Intent();

            // ★ポイント 2★ 利用先パートナー限定アプリに開示してよい情報に限り送信してよい
            intent.putExtra("PARAM", "パートナーアプリに開示してよい情報");

            // ★ポイント 3★ 明示的 Intent によりパートナー限定 Activity を呼び出す
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);

            // ★ポイント 4★ startActivityForResult()によりパートナー限定 Activity を呼び出す
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
            Toast.makeText(this, "利用先 Activity が見つからない。", Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // ★ポイント 5★ パートナー限定アプリからの結果情報であっても、受信 Intent の安全性を確認する
            // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
            Toast.makeText(this,
                    String.format("結果「%s」を受け取った。", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```


PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // 0-9A-F 以外の文字がある

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
```

```
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

4.2.1.4. 自社限定 Activity を利用する

自社限定 Activity は、自社以外のアプリから利用されることを禁止する Activity である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。ここでは自社限定 Activity を呼び出す方法を説明する。

ポイント:

1. 独自定義 Signature Permission を定義する
2. 独自定義 Signature Permission を利用宣言する
3. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
4. 利用先アプリの証明書が自社の証明書であることを確認する
5. 利用先アプリは自社アプリであるから、センシティブな情報を送信してもよい
6. 明示的 Intent により自社限定 Activity を呼び出す
7. 自社アプリからの結果情報であっても、受信 Intent の安全性を確認する
8. 利用先アプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.activity.proprietaryuser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント1★ 独自定義 Signature Permission を定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- ★ポイント2★ 独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.permission.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ProprietaryUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

ProprietaryUserActivity.java

```
package org.jssec.android.activity.proprietaryuser;
```

```
import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class ProprietaryUserActivity extends Activity {

    // 利用先の Activity 情報
    private static final String TARGET_PACKAGE =  "org.jssec.android.activity.proprietaryactivity";
    private static final String TARGET_ACTIVITY = "org.jssec.android.activity.proprietaryactivity.ProprietaryActi
vity";

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final int REQUEST_CODE = 1;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onUseActivityClick(View view) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "独自定義 Signature Permission が自社アプリにより定義されていない。", Toast.LENGT
H_LONG).show();
            return;
        }

        // ★ポイント 4★ 利用先アプリの証明書が自社の証明書であることを確認する
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this, "利用先アプリは自社アプリではない。", Toast.LENGTH_LONG).show();
            return;
        }
```



```
        try {
            Intent intent = new Intent();

            // ★ポイント 5★ 利用先アプリは自社アプリであるから、センシティブな情報を送信してもよい
            intent.putExtra("PARAM", "センシティブな情報");

            // ★ポイント 6★ 明示的 Intent により自社限定 Activity を呼び出す
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivityForResult(intent, REQUEST_CODE);
        }
        catch (ActivityNotFoundException e) {
            Toast.makeText(this, "利用先 Activity が見つからない。", Toast.LENGTH_LONG).show();
        }
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (resultCode != RESULT_OK) return;

        switch (requestCode) {
        case REQUEST_CODE:
            String result = data.getStringExtra("RESULT");

            // ★ポイント 7★ 自社アプリからの結果情報であっても、受信 Intent の安全性を確認する
            // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
            Toast.makeText(this, String.format("結果「%s」を受け取った。", result), Toast.LENGTH_LONG).show();
            break;
        }
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;
```

```
            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // sigPermName を定義したアプリの証明書のハッシュ値を返す
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

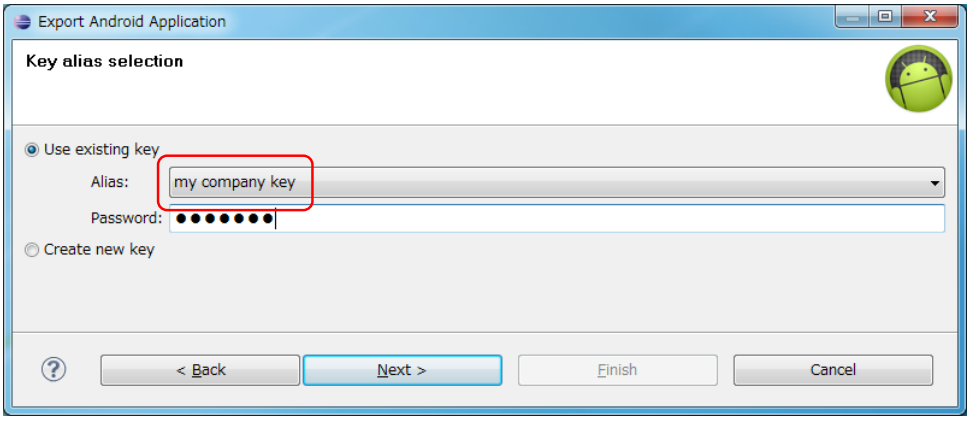
PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
```



```
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

★ポイント 8★ eclipse から APK を Export するときに、利用先アプリと同じ開発者鍵で APK を署名する。



4.2.2. ルールブック

Activity を利用する際には以下のルールを守ること。

1. 利用先 Activity が固定できる場合は明示的 Intent で Activity を利用する	（必須）
2. 利用先 Activity からの戻り Intent の安全性を確認する	（必須）
3. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する	（必須）
4. 他社の特定アプリと連携する場合は利用先 Activity を確認する	（必須）
5. センシティブな情報はできる限り送らない	（推奨）

4.2.2.1. 利用先 Activity が固定できる場合は明示的 Intent で Activity を利用する （必須）

暗黙的 Intent により Activity を利用すると、最終的にどの Activity に Intent が送信されるかは Android OS 任せになってしまう。もしマルウェアに Intent が送信されてしまうと情報漏洩が生じる。一方、明示的 Intent により Activity を利用すると、指定した Activity 以外が Intent を受信することはなく比較的安全である。

処理を任せるアプリ(の Activity)をユーザーに選択させるなど、利用先 Activity を実行時に決定したい場合を除けば、利用先 Activity はあらかじめ特定できる。このような Activity を利用する場合には明示的 Intent を利用すべきである。

同一アプリ内の Activity を明示的 Intent で利用する

```
Intent intent = new Intent(this, PictureActivity.class);
intent.putExtra("BARCODE", barcode);
startActivity(intent);
```

他のアプリの公開 Activity を明示的 Intent で利用する

```
Intent intent = new Intent();
intent.setClassName(
    "org.jssec.android.activity.publicactivity",
    "org.jssec.android.activity.publicactivity.PublicActivity");
startActivity(intent);
```

ただし他のアプリの公開 Activity を明示的 Intent で利用した場合も、相手先 Activity を含むアプリがマルウェアである可能性がある。宛先をパッケージ名で限定したとしても、相手先アプリが実は本物アプリと同じパッケージ名を持つ偽物アプリである可能性があるからだ。このようなリスクを排除したい場合は、パートナー限定 Activity や自社限定 Activity の使用を検討する必要がある。

4.2.2.2. 利用先 Activity からの戻り Intent の安全性を確認する （必須）

Activity のタイプによって若干リスクは異なるが、基本的には戻り値として受信した Intent のデータを処理する際には、まず最初に受信 Intent の安全性を確認しなければならない。

利用先 Activity が公開 Activity の場合、不特定のアプリから戻り Intent を受け取るため、マルウェアの攻撃 Intent

を受け取る可能性がある。利用先 Activity が非公開 Activity の場合、同一アプリ内から戻り Intent を受け取るのでリスクはないように考えがちだが、他のアプリから受け取った Intent のデータを間接的に戻り値として転送することがあるため、受信 Intent を盲目的に安全であると考えてはならない。利用先 Activity がパートナー限定 Activity や自社限定 Activity の場合、その中間のリスクであるため、やはり受信 Intent の安全性を確認する必要がある。

「3.1 入力データの安全性を確認する」を参照すること。

4.2.2.3. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）

自社アプリだけから利用できる自社限定 Activity を利用する場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 利用宣言だけでは保護が不十分であるため、「5.2 Permission と Protection Level」の「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

4.2.2.4. 他社の特定アプリと連携する場合は利用先 Activity を確認する （必須）

他社の特定アプリと連携する場合にはホワイトリストによる確認方法がある。自アプリ内に利用先アプリの証明書ハッシュを予め保持しておく。利用先の証明書ハッシュと保持している証明書ハッシュが一致するかを確認することで、なりすましアプリに Intent を発行することを防ぐことができる。具体的な実装方法についてはサンプルコードセクション「4.1.1.3 パートナー限定 Activity を作る」を参照すること。

4.2.2.5. センシティブな情報はできる限り送らない （推奨）

不特定多数のアプリと連携する場合にはセンシティブな情報を送ってはならない。特定のアプリと連携する場合においても、意図しないアプリに Intent を発行してしまった場合や第三者による Intent の盗聴などで情報が漏洩してしまうリスクがある。「4.2.3.2 Activity 利用時のログ出力ついて」を参照すること。

センシティブな情報を Activity に送付する場合、その情報の漏洩リスクを検討しなければならない。公開 Activity に送付した情報は必ず漏洩すると考えなければならない。また限定公開 Activity や自社限定 Activity に送付した情報もそれら Activity の実装に依存して情報漏洩リスクの大小がある。非公開 Activity に送付する情報に至っても、Intent の data に含めた情報は LogCat 経由で漏洩するリスクがある。Intent の extras は LogCat に出力されないので、センシティブ情報は extras で送付するとよい。

センシティブな情報はできるだけ送付しないように工夫すべきである。送付する場合も、利用先 Activity は信頼できる Activity に限定し、Intent の情報が LogCat へ漏洩しないように配慮しなければならない。

4.2.3. アドバンスト

4.2.3.1. Activity 利用時の Intent の取得について

タスクのルート Activity を起動した Intent は、タスク履歴として他のアプリから取得できてしまう。つまりマルウェアが他のアプリのルート Activity を起動した Intent 内の情報を盗み取ることができてしまうのだ。以下の Activity がタスクのルート Activity となり、それを起動した Intent 内の情報にはセンシティブな情報を含めてはならない。

- ランチャーから起動された Activity
- android:launchMode=”singleTask”と指定された Activity
- android:launchMode=”singleInstance”と指定された Activity
- Intent に FLAG_ACTIVITY_NEW_TASK が指定して起動された Activity

タスク履歴からルート Activity 起動時の Intent を取得する例

MainActivity.java

```
// ActivityManager を取得する
ActivityManager activityManager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);
// RecentTask リストを 10 件取得する
List<RecentTaskInfo> recentTaskInfoList = activityManager
        .getRecentTasks(10, ActivityManager.RECENT_WITH_EXCLUDED);
for (int i = 0; i < recentTaskInfoList.size(); i++) {
    RecentTaskInfo recentTaskInfo = recentTaskInfoList.get(i);

    // baseIntent の内容を Log 表示する
    Log.v("baseIntent", recentTaskInfo.baseIntent.toString());
}
```

4.2.3.2. Activity 利用時のログ出力ついて

Activity を利用する際に ActivityManager が Intent の内容を logcat に出力する。以下の内容は logcat に出力されるため、センシティブな情報が含まれないように注意すべきだ。

- 利用先パッケージ名
- 利用先クラス名
- Intent#setData()で設定した URI

例えば、メール送信する場合、URI にメールアドレスを指定して Intent を発行するとメールアドレスが logcat に出力されてしまう。Intent#putExtra()で設定した値は logcat に出力されないため、Extras に設定して送るようにした方が良い。

次のようにメール送信すると logcat にメールアドレスが表示されてしまう

MainActivity.java

```
// URI は logcat に出力される
Uri uri = Uri.parse("mailto:test@gmail.com");
Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
```



```
        startActivity(intent);
```

次のように Extra を使用すると logcat にメールアドレスが表示されなくなる

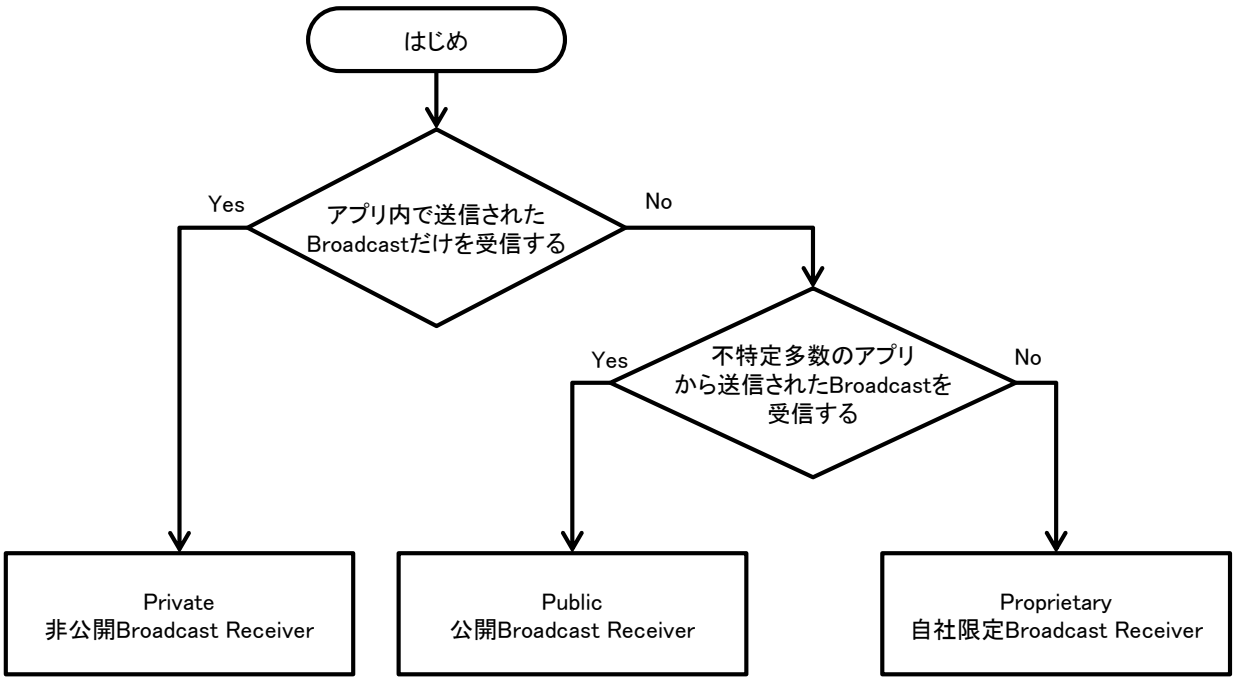
MainActivity.java

```
        // Extra に設定した内容は logcat に出力されない
        Uri uri = Uri.parse("mailto:");
        Intent intent = new Intent(Intent.ACTION_SENDTO, uri);
        intent.putExtra(Intent.EXTRA_EMAIL, new String[] {"test@gmail.com"});
        startActivity(intent);
```

4.3. Broadcast を受信する

4.3.1. サンプルコード

Broadcast を受信するには Broadcast Receiver を作る必要がある。どのような Broadcast を受信するかによって、Broadcast Receiver が抱えるリスクや適切な防御手段が異なる。次の判定フローによって作成する Broadcast Receiver がどのタイプであるかを判断できる。なお、Broadcast の送信元アプリを確認する手段がないため、パートナー限定 Broadcast Receiver を作ることはできない。



また Broadcast Receiver にはその定義方法により、静的 Broadcast Receiver と動的 Broadcast Receiver との 2 種類があり、下表のような特徴の違いがある。サンプルコード中では両方の実装方法を紹介している。

	定義方法	特徴
静的 Broadcast Receiver	AndroidManifest.xml に <receiver> 要素を記述することで定義する	● システムから送信される一部の Broadcast（ACTION_BATTERY_CHANGED など）を受信できない制約がある ● アプリが初回起動してからアンインストールされるまでの間、Broadcast を受信できる
動的 Broadcast Receiver	プログラム中で registerReceiver() および unregisterReceiver() を呼び出すことにより、動的に Broadcast Receiver を登録／登録解除する	● 静的 Broadcast Receiver では受信できない Broadcast でも受信できる ● Activity が前面に出ている期間だけ Broadcast を受信したいなど、Broadcast の受信可能期間をプログラムで制御できる ● 非公開の Broadcast Receiver を作ることはできない

4.3.1.1. 非公開 Broadcast Receiver を作る

非公開 Broadcast Receiver は、同一アプリ内から送信された Broadcast だけを受信できる Boradcast Receiver であり、もっとも安全性の高い Broadcast Receiver である。動的 Broadcast Receiver を非公開で登録することはできないため、非公開 Broadcast Receiver では静的 Broadcast Receiver だけで構成される。

ポイント:

1. exported="false"により、明示的に非公開設定する
2. 同一アプリ内から送信された Broadcast であっても、受信 Intent の安全性を確認する
3. 結果を返す場合、送信元は同一アプリ内であるから、センシティブな情報を返送してよい

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.privatereceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- 非公開 Broadcast Receiver を定義する -->
        <!-- ★ポイント 1★ exported="false"により、明示的に非公開設定する -->
        <receiver
            android:name=".PrivateReceiver"
            android:exported="false" />

        <activity
            android:name=".PrivateSenderActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

PrivateReceiver.java

```
package org.jssec.android.broadcast.privatereceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PrivateReceiver extends BroadcastReceiver {
```

```
    @Override
    public void onReceive(Context context, Intent intent) {

        // ★ポイント 2★ 同一アプリ内から送信された Broadcast であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        String param = intent.getStringExtra("PARAM");
        Toast.makeText(context,
                String.format("「%s」を受信した。", param),
                Toast.LENGTH_SHORT).show();

        // ★ポイント 3★ 送信元は同一アプリ内であるから、センシティブな情報を返送してよい
        setResultCode(Activity.RESULT_OK);
        setResultData("センシティブな情報 from Receiver");
        abortBroadcast();
    }
}
```


4.3.1.2. 公開 Broadcast Receiver を作る

公開 Broadcast Receiver は、不特定多数のアプリから送信された Broadcast を受信できる Broadcast Receiver である。マルウェアが送信した Broadcast を受信することがあることに注意が必要だ。

ポイント:

1. 受信 Intent の安全性を確認する
2. 結果を返す場合、センシティブな情報を含めない

公開 BroadcastReceiver のサンプルコードである PublicReciver は、静的 Broadcast Receiver および動的 Broadcast Receiver の両方で利用される。

PublicReceiver.java

```
package org.jssec.android.broadcast.publicreceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class PublicReceiver extends BroadcastReceiver {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    public boolean isDynamic = false;
    private String getName() {
        return isDynamic ? "公開動的 Broadcast Receiver" : "公開静的 Broadcast Receiver";
    }

    @Override
    public void onReceive(Context context, Intent intent) {

        // ★ポイント 1★ 受信 Intent の安全性を確認する
        // 公開 Broadcast Receiver であるため利用元アプリがマルウェアである可能性がある。
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        if (MY_BROADCAST_PUBLIC.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context,
                    String.format("%s:¥n「%s」を受信した。", getName(), param),
                    Toast.LENGTH_SHORT).show();
        }

        // ★ポイント 2★ 結果を返す場合、センシティブな情報を含めない
        // 公開 Broadcast Receiver であるため、
        // Broadcast の送信元アプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("センシティブではない情報 from %s", getName()));
        abortBroadcast();
    }
}
```

静的 Broadcast Receiver は AndroidManifest.xml で定義する。

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.publicreceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- 公開静的 Broadcast Receiver を定義する -->
        <receiver android:name=".PublicReceiver" >
            <intent-filter>
                <action android:name="org.jssec.android.broadcast.MY_BROADCAST_PUBLIC" />
            </intent-filter>
        </receiver>

        <service
            android:name=".DynamicReceiverService"
            android:exported="false" />

        <activity
            android:name=".PublicReceiverActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

動的 Broadcast Receiver はプログラム中で registerReceiver()および unregisterReceiver()を呼び出すことにより登録／登録解除する。ボタン操作により登録／登録解除を行うために PublicReceiverActivity 上にボタンを配置している。動的 Broadcast Receiver インスタンスは PublicReceiverActivity より生存期間が長いため PublicReceiverActivity のメンバー変数として保持することはできない。そのため DynamicReceiverService のメンバー変数として動的 Broadcast Receiver のインスタンスを保持させ、DynamicReceiverService を PublicReceiverActivity から開始／終了することにより動的 Broadcast Receiver を間接的に登録／登録解除している。

DynamicReceiverService.java

```
package org.jssec.android.broadcast.publicreceiver;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
```

```
import android.widget.Toast;

public class DynamicReceiverService extends Service {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    private PublicReceiver mReceiver;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        // 公開動的 Broadcast Receiver を登録する
        mReceiver = new PublicReceiver();
        mReceiver.isDynamic = true;
        IntentFilter filter = new IntentFilter();
        filter.addAction(MY_BROADCAST_PUBLIC);
        filter.setPriority(1);  // 静的 BroadcastReceiver より動的 BroadcastReceiver を優先させる
        registerReceiver(mReceiver, filter);
        Toast.makeText(this,
                "動的 Broadcast Receiver を登録した。",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();

        // 公開動的 Broadcast Receiver を登録解除する
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                "動的 Broadcast Receiver を登録解除した。",
                Toast.LENGTH_SHORT).show();
    }
}
```

PublicReceiverActivity.java

```
package org.jssec.android.broadcast.publicreceiver;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class PublicReceiverActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
```

```
    public void onRegisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        startService(intent);
    }

    public void onUnregisterReceiverClick(View view) {
        Intent intent = new Intent(this, DynamicReceiverService.class);
        stopService(intent);
    }
}
```

4.3.1.3. 自社限定 Broadcast Receiver を作る

自社限定 Broadcast Receiver は、自社以外のアプリから送信された Broadcast を一切受信しない Broadcast Receiver である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。

ポイント:

1. 独自定義 Signature Permission を定義する
2. Ordered Broadcast に結果を返す場合、独自定義 Signature Permission を利用宣言する
3. 静的 Broadcast Receiver 定義にて、独自定義 Signature Permission を要求宣言する
4. 動的 Broadcast Receiver を登録するとき、独自定義 Signature Permission を要求宣言する
5. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
6. 自社アプリからの Broadcast であっても、受信 Intent の安全性を確認する
7. Broadcast 送信元は自社アプリであるから、センシティブな情報を返送してよい
8. Broadcast 送信元アプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.proprietaryreceiver"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント1★ 独自定義 Signature Permission を定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- ★ポイント2★ Ordered Broadcast に結果を返す場合、独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.permission.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント3★ 独自定義 Signature Permission を要求宣言する -->
        <receiver
            android:name=".ProprietaryReceiver"
            android:permission="org.jssec.android.permission.MY_PERMISSION">
            <intent-filter>
                <action android:name="org.jssec.android.broadcast.MY_BROADCAST_PROPRIETARY" />
            </intent-filter>
        </receiver>

        <service
            android:name=".DynamicReceiverService"
            android:exported="false" />

        <activity
            android:name=".ProprietaryReceiverActivity"
            android:label="@string/app_name" >
```

```
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

DynamicReceiverService.java

```
package org.jssec.android.broadcast.proprietaryreceiver;

import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.IBinder;
import android.widget.Toast;

public class DynamicReceiverService extends Service {

    private static final String MY_BROADCAST_PROPRIETARY =
        "org.jssec.android.broadcast.MY_BROADCAST_PROPRIETARY";

    private ProprietaryReceiver mReceiver;

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();

        mReceiver = new ProprietaryReceiver();
        mReceiver.isDynamic = true;
        IntentFilter filter = new IntentFilter();
        filter.addAction(MY_BROADCAST_PROPRIETARY);
        filter.setPriority(1);  // 静的 BroadcastReceiver より動的 BroadcastReceiver を優先させる

        // ★ポイント 4★ 動的 BroadcastReceiver を登録するとき、独自定義 Signature Permission を要求宣言する
        registerReceiver(mReceiver, filter, "org.jssec.android.permission.MY_PERMISSION", null);

        Toast.makeText(this,
                "動的 Broadcast Receiver を登録した。",
                Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        unregisterReceiver(mReceiver);
        mReceiver = null;
        Toast.makeText(this,
                "動的 Broadcast Receiver を登録解除した。",
                Toast.LENGTH_SHORT).show();
    }
}
```


ProprietaryReceiver.java

```
package org.jssec.android.broadcast.proprietaryreceiver;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

public class ProprietaryReceiver extends BroadcastReceiver {

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final String MY_BROADCAST_PROPRIETARY =
        "org.jssec.android.broadcast.MY_BROADCAST_PROPRIETARY";

    public boolean isDynamic = false;
    private String getName() {
        return isDynamic ? "自社限定動的 Broadcast Receiver" : "自社限定静的 Broadcast Receiver";
    }

    @Override
    public void onReceive(Context context, Intent intent) {

        // ★ポイント 5★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(context, MY_PERMISSION, myCertHash(context))) {
            Toast.makeText(context, "独自定義 Signature Permission が自社アプリにより定義されていない。", Toast.LE
NGTH_LONG).show();
            return;
        }

        // ★ポイント 6★ 自社アプリからの Broadcast であっても、受信 Intent の安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        if (MY_BROADCAST_PROPRIETARY.equals(intent.getAction())) {
            String param = intent.getStringExtra("PARAM");
            Toast.makeText(context,
                    String.format("%s:¥n「%s」を受信した。", getName(), param),
                    Toast.LENGTH_SHORT).show();
```

```
        }

        // ★ポイント 7★ 送信元は自社アプリであるから、センシティブな情報を返送してよい
        setResultCode(Activity.RESULT_OK);
        setResultData(String.format("センシティブな情報 from %s", getName()));
        abortBroadcast();
    }
}
```

SigPerm.java

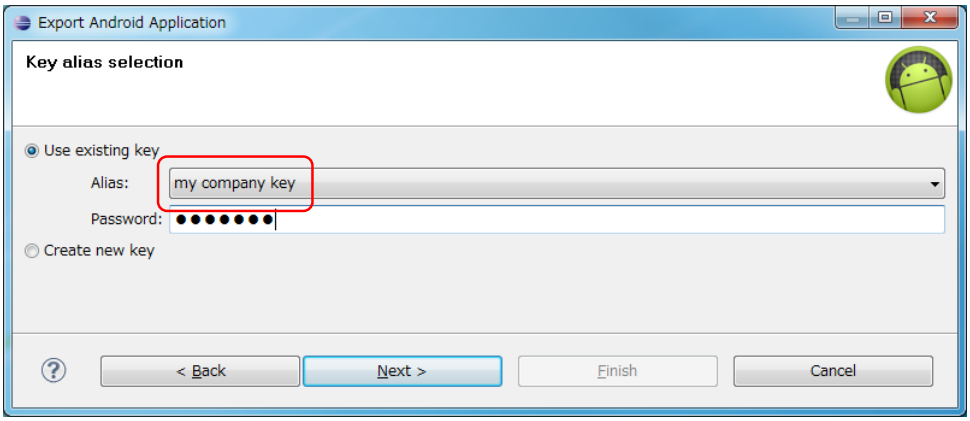
```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // sigPermName を定義したアプリの証明書のハッシュ値を返す
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;
```

```
public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

★ポイント 8★ eclipse から APK を Export するときに、Broadcast 送信元アプリと同じ開発者鍵で APK を署名する。



4.3.2. ルールブック

Broadcast を受信する際には以下のルールを守ること。

1. 内部使用の Broadcast Receiver は非公開設定する	（必須）
2. 受信 Intent の安全性を確認する	（必須）
3. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する	（必須）
4. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する	（必須）

4.3.2.1. 内部使用の Broadcast Receiver は非公開設定する （必須）

アプリ内でのみ使用される Broadcast Receiver は非公開設定する。これにより、他のアプリから意図せず Broadcast を受け取ってしまうことがなくなり、アプリの機能を利用されたり、アプリの動作に異常をきたしたりするのを防ぐことができる。Intent Filter を設置した場合はデフォルトで exported="true"となるので明示的に exported="false"と非公開設定する必要がある。

```
<!-- ポイント 1: 外部アプリに非公開とする BroadcastReceiver -->
<!-- exported=false とする -->
<receiver android:name=".PrivateReceiver"
    android:exported="false" >
    <intent-filter>
        <action android:name="org.jssec.android.broadcast.MY_ACTION" />
    </intent-filter>
</receiver>
```

4.3.2.2. 受信 Intent の安全性を確認する （必須）

Broadcast Receiver のタイプによって若干リスクは異なるが、基本的には受信 Intent のデータを処理する際には、まず最初に受信 Intent の安全性を確認しなければならない。

公開 Broadcast Receiver は不特定多数のアプリから Intent を受け取るため、マルウェアの攻撃 Intent を受け取る可能性がある。非公開 Broadcast Receiver は他のアプリから Intent を直接受け取ることはない。しかし同一アプリ内の公開 Component が他のアプリから受け取った Intent のデータを非公開 Broadcast Receiver に転送することがあるため、受信 Intent を盲目的に安全であると考えてはならない。自社限定 Broadcast Receiver はその中間のリスクであるため、やはり受信 Intent の安全性を確認する必要がある。

「3.1 入力データの安全性を確認する」を参照すること。

4.3.2.3. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）

自社のアプリから送信された Broadcast だけを受信し、それ以外の Broadcast を一切受信しない自社限定 Broadcast Receiver を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2

Permission と Protection Level」の「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

4.3.2.4. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する （必須）

Broadcast Receiver のタイプによって setResult()により結果情報を返すアプリの信用度が異なる。公開 Broadcast Receiver の場合は、結果返送先のアプリがマルウェアである可能性もあり、結果情報が悪意を持って使われる危険性がある。非公開 Broadcast Receiver や自社限定 Broadcast Receiver の場合は、結果返送先は自社開発アプリであるため結果情報の扱いをあまり心配する必要はない。

このように Broadcast Receiver から結果情報を返す場合には、返送先アプリからの結果情報の漏洩に配慮しなければならない。

4.3.3. アドバンスト

4.3.3.1. Android 3.1 以降はアプリを起動しないと Receiver が登録されない

Android 3.1 以降では、AndroidManifest.xml に静的に定義した Broadcast Receiver は、インストールしただけでは有効にならないので注意が必要である。アプリを 1 回起動することで、それ以降の Broadcast を受信できるようになる。インストール後に Broadcast 受信をトリガーにして処理を起動させることはできなくなった。ただし Broadcast の送信側で Intent に Intent.FLAG_INCLUDE_STOPPED_PACKAGES を設定して Broadcast 送信した場合は、一度も起動していないアプリであってもこの Broadcast を受信することができる。

4.3.3.2. 同じ UID を持つアプリから送信された Broadcast は、非公開 Broadcast Receiver でも受信できる

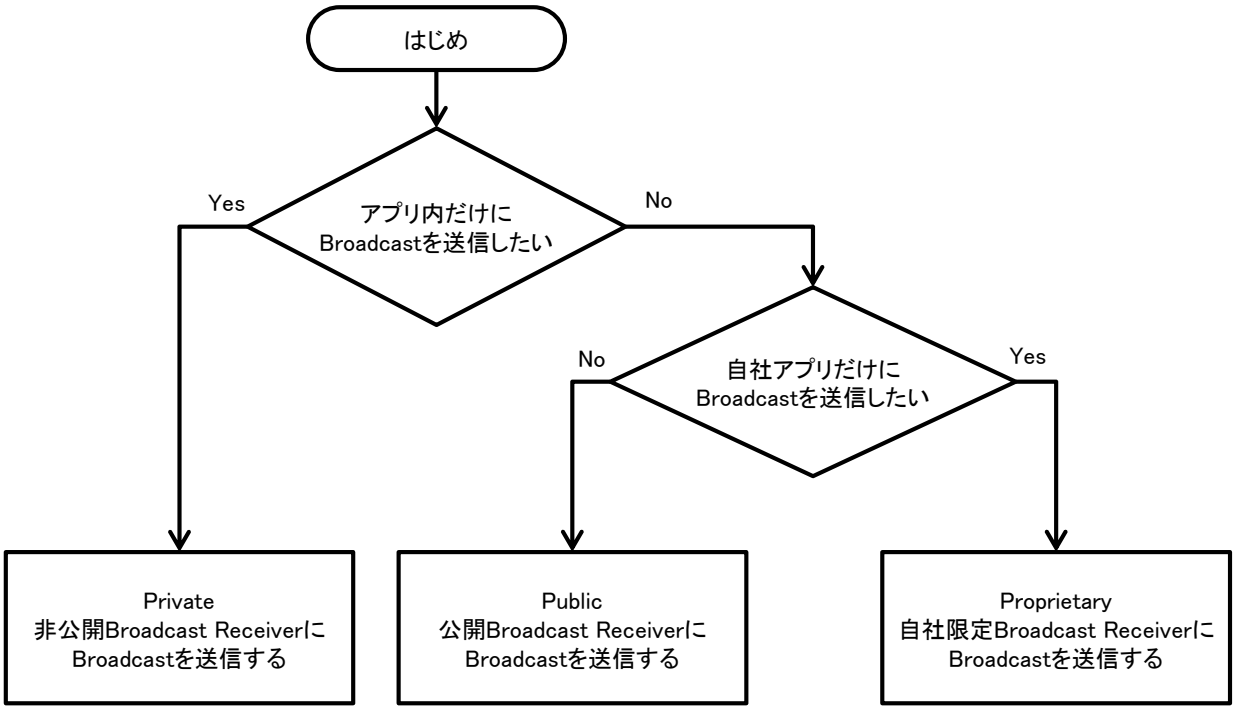
複数のアプリに同じ UID を持たせることができる。この場合、たとえ非公開 Broadcast Receiver であっても、同じ UID のアプリから送信された Broadcast は受信してしまう。

しかしこれはセキュリティ上問題となることはない。同じ UID を持つアプリは APK を署名する開発者鍵が一致することが保証されており、非公開 Broadcast Receiver が受信するのは自社アプリから送信された Broadcast に限定されるからである。

4.4. Broadcast を送信する

4.4.1. サンプルコード

どのような相手に Broadcast を送信するかによって、送信する情報の適切な防御手段が異なる。次の判定フローによって Broadcast の送信方法を分類できる。それぞれの送信方法についてサンプルコードを用意している。



4.4.1.1. 非公開 Broadcast Receiver に Broadcast を送信する

ここで説明する非公開 Broadcast Receiver への Broadcast 送信方法は、セキュリティ面では安全であるものの、Sticky が使えないという制約があることに注意が必要である。

ポイント:

1. 同一アプリ内 Receiver はクラス指定の明示的 Intent で Broadcast 送信する
2. 送信先は同一アプリ内 Receiver であるため、センシティブな情報を送信してよい
3. 同一アプリ内 Receiver からの結果情報であっても、受信データの安全性を確認する

PrivateSenderActivity.java

```
package org.jssec.android.broadcast.privatereceiver;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateSenderActivity extends Activity {

    public void onSendNormalClick(View view) {
        // ★ポイント1★ 同一アプリ内 Receiver はクラス指定の明示的 Intent で Broadcast 送信する
        Intent intent = new Intent(this, PrivateReceiver.class);

        // ★ポイント2★ 送信先は同一アプリ内 Receiver であるため、センシティブな情報を送信してよい
        intent.putExtra("PARAM", "センシティブな情報 from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // ★ポイント1★ 同一アプリ内 Receiver はクラス指定の明示的 Intent で Broadcast 送信する
        Intent intent = new Intent(this, PrivateReceiver.class);

        // ★ポイント2★ 送信先は同一アプリ内 Receiver であるため、センシティブな情報を送信してよい
        intent.putExtra("PARAM", "センシティブな情報 from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            // ★ポイント3★ 同一アプリ内 Receiver からの結果情報であっても、受信データの安全性を確認する
            // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
            String data = getResultData();
            PrivateSenderActivity.this.logLine(
                    String.format("結果「%s」を受信した。", data));
        }
    };

    private TextView mLogView;
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```


4.4.1.2. 公開 Broadcast Receiver に Broadcast を送信する

公開 Broadcast Receiver に Broadcast を送信する場合、送信する Broadcast がマルウェアに受信されることがあることに注意が必要である。

ポイント:

1. センシティブな情報を送信してはならない
2. 結果を受け取る場合、結果データの安全性を確認する

PublicSenderActivity.java

```
package org.jssec.android.broadcast.publicsender;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicSenderActivity extends Activity {

    private static final String MY_BROADCAST_PUBLIC =
        "org.jssec.android.broadcast.MY_BROADCAST_PUBLIC";

    public void onSendNormalClick(View view) {
        // ★ポイント1★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        sendBroadcast(intent);
    }

    public void onSendOrderedClick(View view) {
        // ★ポイント1★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        sendOrderedBroadcast(intent, null, mResultReceiver, null, 0, null, null);
    }

    public void onSendStickyClick(View view) {
        // ★ポイント1★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        sendStickyBroadcast(intent);
    }

    public void onSendStickyOrderedClick(View view) {
        // ★ポイント1★ センシティブな情報を送信してはならない
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        intent.putExtra("PARAM", "センシティブではない情報 from Sender");
        sendStickyOrderedBroadcast(intent, mResultReceiver, null, 0, null, null);
    }

    public void onRemoveStickyClick(View view) {
        Intent intent = new Intent(MY_BROADCAST_PUBLIC);
        removeStickyBroadcast(intent);
```

```
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            // ★ポイント 2★ 結果を受け取る場合、結果データの安全性を確認する
            // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
            String data = getResultData();
            PublicSenderActivity.this.logLine(
                    String.format("結果「%s」を受信した。", data));
        }
    };

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```


4.4.1.3. 自社限定 Broadcast Receiver に Broadcast を送信する

自社限定 Broadcast Receiver に Broadcast を送信する場合、Broadcast Receiver 側に独自定義 Signature Permission を要求する必要があるため、Sticky が使えないという制約があることに注意が必要である。

ポイント:

1. 独自定義 Signature Permission を定義する
2. 独自定義 Signature Permission を利用宣言する
3. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
4. Receiver は自社アプリ限定であるから、センシティブな情報を送信してもよい
5. Receiver に独自定義 Signature Permission を要求する
6. 結果を受け取る場合、結果データの安全性を確認する
7. Receiver 側アプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.broadcast.proprietarysender"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />
    <uses-permission android:name="android.permission.BROADCAST_STICKY"/>

    <!-- ★ポイント1★ 独自定義 Signature Permission を定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- ★ポイント2★ 独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.permission.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ProprietarySenderActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

ProprietarySenderActivity.java

```
package org.jssec.android.broadcast.proprietarysender;

import org.jssec.android.shared.SigPerm;
```

```
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

public class ProprietarySenderActivity extends Activity {

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private static final String MY_BROADCAST_PROPRIETARY =
        "org.jssec.android.broadcast.MY_BROADCAST_PROPRIETARY";

    public void onSendNormalClick(View view) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "独自定義 Signature Permission が自社アプリにより定義されていない。", Toast.LENGT
H_LONG).show();
            return;
        }

        // ★ポイント 4★ Receiver は自社アプリ限定であるから、センシティブな情報を送信してもよい
        Intent intent = new Intent(MY_BROADCAST_PROPRIETARY);
        intent.putExtra("PARAM", "センシティブな情報 from Sender");

        // ★ポイント 5★ Receiver に独自定義 Signature Permission を要求する
        sendBroadcast(intent, "org.jssec.android.permission.MY_PERMISSION");
    }

    public void onSendOrderedClick(View view) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "独自定義 Signature Permission が自社アプリにより定義されていない。", Toast.LENGT
H_LONG).show();
            return;
        }
```



```
        // ★ポイント 4★ Receiver は自社アプリ限定であるから、センシティブな情報を送信してもよい
        Intent intent = new Intent(MY_BROADCAST_PROPRIETARY);
        intent.putExtra("PARAM", "センシティブな情報 from Sender");

        // ★ポイント 5★ Receiver に独自定義 Signature Permission を要求する
        sendOrderedBroadcast(intent, "org.jssec.android.permission.MY_PERMISSION",
                mResultReceiver, null, 0, null, null);
    }

    private BroadcastReceiver mResultReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {

            // ★ポイント 6★ 結果を受け取る場合、結果データの安全性を確認する
            // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
            String data = getResultData();
            ProprietarySenderActivity.this.logLine(String.format("結果「%s」を受信した。", data));
        }
    };

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
```

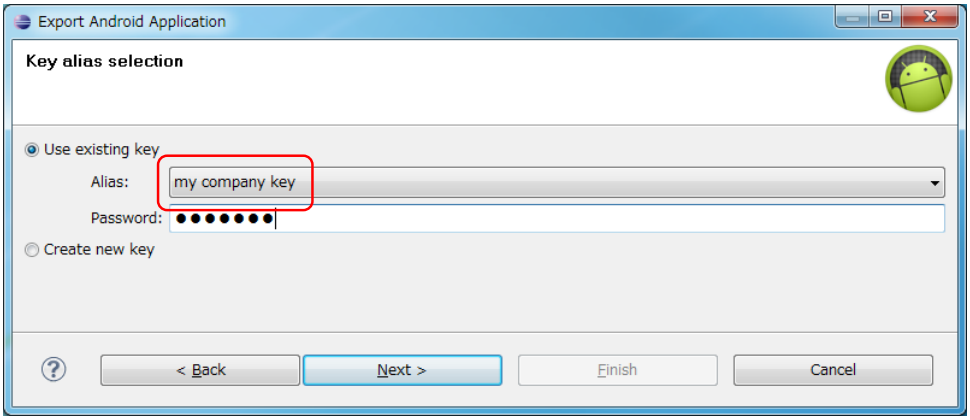
```
            String pkgname = pi.packageName;

            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // sigPermName を定義したアプリの証明書のハッシュ値を返す
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
```

```
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

★ポイント 7★ eclipse から APK を Export するときに、Receiver 側アプリと同じ開発者鍵で APK を署名する。



4.4.2. ルールブック

Broadcast を送信する際には以下のルールを守ること。

1. センシティブな情報を Broadcast 送信する場合は、受信可能な Receiver を制限する	（必須）
2. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する	（必須）
3. Sticky Broadcast にはセンシティブな情報を含めない	（必須）
4. receiverPermission を指定しない Ordered Broadcast は届かないことがあることに注意	（必須）
5. Broadcast Receiver からの返信データの安全性を確認する	（必須）

4.4.2.1. センシティブな情報を Broadcast 送信する場合は、受信可能な Receiver を制限する （必須）

Broadcast という名前が表すように、そもそも Broadcast は不特定多数のアプリに情報を一斉送信したり、タイミングを通知したりすることを意図して作られた仕組みである。そのためセンシティブな情報を Broadcast 送信する場合には、マルウェアに情報を取得されないような注意深い設計が必要となる。

センシティブな情報を Broadcast 送信する場合、信頼できる Broadcast Receiver だけが受信可能であり、他の Boradcast Receiver は受信不可能である必要がある。そのための Broadcast 送信方法には以下のようなものがある。

- 明示的 Intent で Broadcast 送信することで宛先を固定し、意図した信頼できる Broadcast Receiver だけに Broadcast を届ける方法。この方法には次の 2 つのパターンがある。
  - 同一アプリ内 Broadcast Receiver 宛てであれば Intent#setClass(Context, Class)により宛先を限定する。具体的なコードについてはサンプルコードセクション「4.4.1.1 非公開 Broadcast Receiver に Broadcast を送信する」を参考にすること。
  - 他のアプリの Broadcast Receiver 宛てであれば Intent#setClassName(String, String)により宛先を限定するが、Broadcast 送信に先立ち宛先パッケージの APK 署名の開発者鍵をホワイトリストと照合して許可したアプリであることを確認してから Broadcast を送信する。実際には暗黙的 Intent を利用できる次の方法が実用的である。
- receiverPermission に独自定義 Signature Permission を指定して Broadcast 送信し、信頼する Broadcast Receiver に当該 Signature Permission を利用宣言してもらう方法。具体的なコードについてはサンプルコードセクション「4.4.1.3 自社限定 Broadcast Receiver に Broadcast を送信する」を参考にすること。またこの Broadcast 送信方法を実装するにはルール「4.4.2.2 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）」も適用しなければならない。

4.4.2.2. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）

独自定義 Signature Permission を receiverPermission に指定して Broadcast 送信する場合、独自定義 Signature Permission を AndroidManifest.xml において定義、利用宣言するだけでは独自定義 Signature Permission を信用できない問題がある。さらにその独自定義 Signature Permission が自社アプリによって定義されていることをプログラム中で確認する必要がある。詳細については「5.2 Permission と Protection Level」の

「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

4.4.2.3. Sticky Broadcast にはセンシティブな情報を含めない （必須）

通常の Broadcast は、Broadcast 送信時に受信可能状態にある Broadcast Receiver に受信処理されると、その Broadcast は消滅してしまう。一方 Sticky Broadcast（および Sticky Ordered Broadcast、以下同様）は、送信時に受信状態にある Broadcast Receiver に受信処理された後もシステム上に存在しつづけ、その後 registerReceiver() により受信できることが特徴である。不要になった Sticky Broadcast は removeStickyBroadcast()により任意のタイミングで削除できる。

Sticky Broadcast は暗黙的 Intent による使用が前提であり、receiverPermission を指定した Broadcast 送信はできない。したがって Sticky Broadcast で送信した情報はマルウェアを含む不特定対数のアプリから取得できてしまう。したがってセンシティブな情報を Sticky Broadcast で送信してはならない。

4.4.2.4. receiverPermission を指定しない Ordered Broadcast は届かないことがあることに注意 （必須）

receiverPermission を指定せずに送信された Ordered Broadcast は、マルウェアを含む不特定多数のアプリが受信可能である。Ordered Broadcast は Receiver からの返り情報を受け取るため、または複数の Receiver に順次処理をさせるために利用される。優先度の高い Receiver から順次 Broadcast が配送されるため、優先度を高くしたマルウェアが最初に Broadcast を受信し abortBroadcast()すると、後続の Receiver に Broadcast が配信されなくなる。

4.4.2.5. Broadcast Receiver からの返信データの安全性を確認する （必須）

返信データを送り返してきた Broadcast Receiver のタイプによって若干リスクは異なるが、基本的には戻り値として受信したデータを処理する際には、まず最初に受信データの安全性を確認しなければならない。

返信元 Broadcast Receiver が公開 Broadcast Receiver の場合、不特定のアプリから戻りデータを受け取るため、マルウェアの攻撃データを受け取る可能性がある。返信元 Broadcast Receiver が非公開 Broadcast Receiver の場合、同一アプリ内からの戻りデータであるのでリスクはないように考えがちだが、他のアプリから受け取ったデータを間接的に戻り値として転送することがあるため、戻りデータを盲目的に安全であると考えてはならない。返信元 Broadcast Receiver が自社限定 Broadcast Receiver の場合、その中間のリスクであるため、やはり戻りデータの安全性を確認する必要がある。

「3.1 入力データの安全性を確認する」を参照すること。

4.4.3. アドバンスト

4.4.3.1. Broadcast の種類とその特徴

送信する Broadcast は Ordered かそうでないか、Sticky かそうでないかの組み合わせにより 4 種類の Broadcast が存在する。Broadcast 送信用メソッドに応じて、送信する Broadcast の種類が決まる。

Broadcast の種類	送信用メソッド	Ordered?	Sticky?
Normal Broadcast	sendBroadcast()	No	No
Ordered Broadcast	sendOrderedBroadcast()	Yes	No
Sticky Broadcast	sendStickyBroadcast()	No	Yes
Sticky Ordered Broadcast	sendStickyOrderedBroadcast()	Yes	Yes

それぞれの Broadcast の特徴は次のとおりである。

Broadcast の種類	Broadcast の種類ごとの特徴
Normal Broadcast	Normal Broadcast は送信時に受信可能な状態にある Broadcast Receiver に配送されて消滅する。Ordered Broadcast と異なり、複数の Broadcast Receiver が同時に Broadcast を受信するのが特徴である。特定の Permission を持つアプリケーションの Broadcast Receiver だけに Broadcast を受信させることもできる。
Ordered Broadcast	Ordered Broadcast は送信時に受信可能な状態にある Broadcast Receiver が一つずつ順番に Broadcast を受信することが特徴である。より priority 値が大きい Broadcast Receiver が先に受信する。すべての Broadcast Receiver に配送完了するか、途中の Broadcast Receiver が abortBroadcast()を呼び出した場合に、Broadcast は消滅する。特定の Permission を利用宣言したアプリケーションの Broadcast Receiver だけに Broadcast を受信させることもできる。また Ordered Broadcast では送信元が Broadcast Receiver からの結果情報を受け取ることもできる。SMS 受信通知 Broadcast（SMS_RECEIVED）は Ordered Broadcast の代表例である。
Sticky Boradcast	Sticky Broadcast は送信時に受信可能な状態にある Broadcast Receiver に配送された後に消滅することはなくシステムに残り続け、後に registerReceiver()を呼び出したアプリケーションが Sticky Broadcast を受信することができることが特徴である。Sticky Broadcast は他の Broadcast と異なり自動的に消滅することはないので、Sticky Broadcast が不要になったときに、明示的に removeStickyBroadcast()を呼び出して Sticky Broadcast を消滅させる必要がある。他の Broadcast と異なり、特定の Permission を持つアプリケーションの Broadcast Receiver だけに Broadcast を受信させることはできない。バッテリー状態変更通知 Broadcast

	(ACTION_BATTERY_CHANGED)は Sticky Broadcast の代表例である。
Sticky Ordered Broadcast	Ordered Broadcast と Sticky Broadcast の両方の特徴を持った Broadcast である。Stiky Broadcast と同様、特定の Permission を持つアプリケーションの Broadcast Receiver だけに Broadcast を受信させることはできない。

Broadcast の特徴的な振る舞いの視点で、上表を逆引き的に再整理すると下表になる。

Broadcast の特徴的な振る舞い	Normal Broadcast	Ordered Broadcast	Sticky Broadcast	Sticky Ordered Broadcast
受信可能なBroadcast Receiverを Permission により制限する	○	○	−	−
Broadcast Receiver からの処理結果を取得する	−	○	−	○
順番に Broadcast Receiver に Broadcast を処理させる	−	○	−	○
既に送信されている Broadcast を後から受信する	−	−	○	○

4.4.3.2. Broadcast 送信した情報が logcat に出力される場合がある

Broadcast の送受信は基本的に logcat に出力されない。しかし、受信側の Permission 不足によるエラーや、送信側の Permission 不足によるエラーの際に logcat にエラーログが出力される。エラーログには Broadcast で送信する Intent 情報も含まれるので、エラー発生時には Broadcast 送信する場合は logcat に表示されることに注意してほしい。

送信側の Permission 不足時のエラー

```
W/ActivityManager(266): Permission Denial: broadcasting Intent { act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION } from org.jssec.android.broadcast.sending (pid=4685, uid=10058) requires org.jssec.android.permission.MY_PERMISSION due to receiver org.jssec.android.broadcastreceiver.creating/org.jssec.android.broadcastreceiver.creating.CreatingType3Receiver
```

受信側の Permission 不足時のエラー

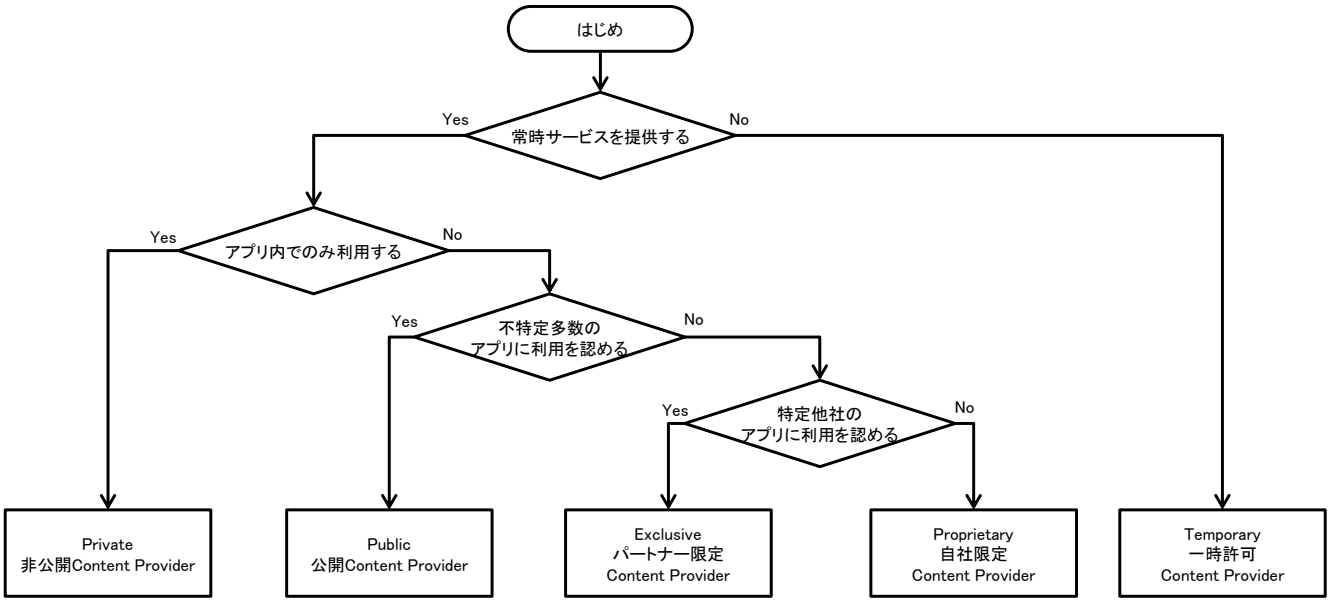
```
W/ActivityManager(275): Permission Denial: receiving Intent { act=org.jssec.android.broadcastreceiver.creating.action.MY_ACTION } to org.jssec.android.broadcastreceiver.creating requires org.jssec.android.permission.MY_PERMISSION due to sender org.jssec.android.broadcast.sending (uid 10158)
```

4.5. Content Provider を作る

ContentResolver と SQLiteDatabase のインタフェースが似ているため、Content Provider は SQLiteDatabase と密接に関係があると勘違いされることが多い。しかし実際には Content Provider はアプリ間データ共有のインタフェースを規定するが、データ保存の形式は一切問わないことに注意が必要だ。作成する Content Provider 内部でデータの保存に SQLiteDatabase を使うこともできるし、XML ファイルなどの別の保存形式を使うこともできる。そのため、ここで紹介するサンプルコードではデータ保存に関わるコードは含んでいない。

4.5.1. サンプルコード

Content Provider がどのように利用されるかによって、Content Provider が抱えるリスクや適切な防御手段が異なる。次の判定フローによって作成する Content Provider がどのタイプであるかを判断できる。



4.5.1.1. 非公開 Content Provider を作る

非公開 Content Provider は、同一アプリ内だけで利用される Content Provider である。Content Provider の非公開設定は Android 2.2（API Level 8）以前では機能しないことに注意が必要だ。

ポイント:

1. Android 2.2（API Level 8）以前では非公開 Content Provider を実装しない（できない）
2. exported="false"により、明示的に非公開設定する
3. 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
4. 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.privateprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <!-- ★ポイント 1★ Android 2.2 (API Level 8) 以前では非公開 Content Provider を実装しない（できない） -->
    <uses-sdk android:minSdkVersion="9" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".PrivateUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- ★ポイント 2★ exported="false"により、明示的に非公開設定する -->
        <provider
            android:name=".PrivateProvider"
            android:authorities="org.jssec.android.provider.privateprovider"
            android:exported="false" />
    </application>
</manifest>
```

PrivateProvider.java

```
package org.jssec.android.provider.privateprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class PrivateProvider extends ContentProvider {
```

```
public static final String AUTHORITIY = "org.jssec.android.provider.privateprovider";
public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

// Content Provider が提供するインタフェースを公開
public interface Download {
    public static final String PATH = "downloads";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
}
public interface Address {
    public static final String PATH = "addresses";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
}

// UriMatcher
private static final int DOWNLOADS_CODE = 1;
private static final int DOWNLOADS_ID_CODE = 2;
private static final int ADDRESSES_CODE = 3;
private static final int ADDRESSES_ID_CODE = 4;
private static UriMatcher sUriMatcher;
static {
    sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    sUriMatcher.addURI(AUTHORITIY, Download.PATH, DOWNLOADS_CODE);
    sUriMatcher.addURI(AUTHORITIY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
    sUriMatcher.addURI(AUTHORITIY, Address.PATH, ADDRESSES_CODE);
    sUriMatcher.addURI(AUTHORITIY, Address.PATH + "/#", ADDRESSES_ID_CODE);
}

// DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "pref" });
static {
    sAddressCursor.addRow(new String[] { "1", "北海道" });
    sAddressCursor.addRow(new String[] { "2", "青森" });
    sAddressCursor.addRow(new String[] { "3", "岩手" });
}
private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
static {
    sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
    sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
}

@Override
public boolean onCreate() {
    return true;
}

@Override
public String getType(Uri uri) {

    // ★ポイント 3★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // 「3.1 入力データの安全性を確認する」を参照。
    // ★ポイント 4★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
    // ただし getType の結果がセンシティブな意味を持つことはあまりない。
    switch (sUriMatcher.match(uri)) {
    case DOWNLOADS_CODE:
    case ADDRESSES_CODE:
        return CONTENT_TYPE;
```



```
        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // ★ポイント 3★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 4★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        // query の結果がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // ★ポイント 3★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 4★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        // Insert 結果、発番される ID がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // ★ポイント 3★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 4★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
```

```
        // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // 5 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 15;  // 15 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // ★ポイント 3★ 同一アプリ内からのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 4★ 利用元アプリは同一アプリであるから、センシティブな情報を返送してよい
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // 10 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 20;  // 20 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }
}
```


4.5.1.2. 公開 Content Provider を作る

公開 Content Provider は、不特定多数のアプリに利用されることを想定した Content Provider である。マルウェアからも利用されることがあることに注意が必要だ。

ポイント:

1. リクエストパラメータの安全性を確認する
2. センシティブな情報を返送してはならない

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.publicprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <provider
            android:name=".PublicProvider"
            android:authorities="org.jssec.android.provider.publicprovider" />
    </application>
</manifest>
```

PublicProvider.java

```
package org.jssec.android.provider.publicprovider;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;

public class PublicProvider extends ContentProvider {

    public static final String AUTHORITIY = "org.jssec.android.provider.publicprovider";
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Content Provider が提供するインタフェースを公開
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }
```

```
// UriMatcher
private static final int DOWNLOADS_CODE = 1;
private static final int DOWNLOADS_ID_CODE = 2;
private static final int ADDRESSES_CODE = 3;
private static final int ADDRESSES_ID_CODE = 4;
private static UriMatcher sUriMatcher;
static {
    sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    sUriMatcher.addURI(AUTHORITIY, Download.PATH, DOWNLOADS_CODE);
    sUriMatcher.addURI(AUTHORITIY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
    sUriMatcher.addURI(AUTHORITIY, Address.PATH, ADDRESSES_CODE);
    sUriMatcher.addURI(AUTHORITIY, Address.PATH + "/#", ADDRESSES_ID_CODE);
}

// DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "pref" });
static {
    sAddressCursor.addRow(new String[] { "1", "北海道" });
    sAddressCursor.addRow(new String[] { "2", "青森" });
    sAddressCursor.addRow(new String[] { "3", "岩手" });
}
private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
static {
    sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
    sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
}

@Override
public boolean onCreate() {
    return true;
}

@Override
public String getType(Uri uri) {

    switch (sUriMatcher.match(uri)) {
    case DOWNLOADS_CODE:
    case ADDRESSES_CODE:
        return CONTENT_TYPE;

    case DOWNLOADS_ID_CODE:
    case ADDRESSES_ID_CODE:
        return CONTENT_ITEM_TYPE;

    default:
        throw new IllegalArgumentException("Invalid URI : " + uri);
    }
}

@Override
public Cursor query(Uri uri, String[] projection, String selection,
        String[] selectionArgs, String sortOrder) {

    // ★ポイント 1★ リクエストパラメータの安全性を確認する
    // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
    // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
    // ★ポイント 2★ センシティブな情報を返送してはならない
    // query の結果がセンシティブな意味を持つかどうかはアプリ次第。
    // リクエスト元のアプリがマルウェアである可能性がある。
```



```
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // ★ポイント 1★ リクエストパラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 2★ センシティブな情報を返送してはならない
        // Insert 結果、発番される ID がセンシティブな意味を持つかどうかはアプリ次第。
        // リクエスト元のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // ★ポイント 1★ リクエストパラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 2★ センシティブな情報を返送してはならない
        // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        // リクエスト元のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // 5 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 15;  // 15 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定
```

```
        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // ★ポイント 1★ リクエストパラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // その他のパラメータの確認はサンプルにつき省略。「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 2★ センシティブな情報を返送してはならない
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        // リクエスト元のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であれば結果として返してもよい。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // 10 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 20;  // 20 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }
}
```


4.5.1.3. パートナー限定 Content Provider を作る

パートナー限定 Content Provider は、特定のアプリだけから利用できる Content Provider である。パートナー企業のアプリと自社アプリが連携してシステムを構成し、パートナーアプリとの間で扱う情報や機能を守るために利用される。

ポイント:

1. 利用元アプリの証明書がホワイトリストに登録されていることを確認する
2. パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
3. パートナーアプリに開示してよい情報に限り返送してよい

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.exclusiveprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <provider
            android:name=".ExclusiveProvider"
            android:authorities="org.jssec.android.provider.exclusiveprovider" />
    </application>
</manifest>
```

ExclusiveProvider.java

```
package org.jssec.android.provider.exclusiveprovider;

import java.util.List;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.ActivityManager;
import android.app.ActivityManager.RunningAppProcessInfo;
import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;
import android.os.Binder;

public class ExclusiveProvider extends ContentProvider {

    public static final String AUTHORITIY = "org.jssec.android.provider.exclusiveprovider";
```

```
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Content Provider が提供するインタフェースを公開
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
    private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "pref" });
    static {
        sAddressCursor.addRow(new String[] { "1", "北海道" });
        sAddressCursor.addRow(new String[] { "2", "青森" });
        sAddressCursor.addRow(new String[] { "3", "岩手" });
    }
    private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
    }

    // ★ポイント 1★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナーアプリ org.jssec.android.provider.exclusiveuser の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.provider.exclusiveuser", isdebug ?
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // keystore の"partner key"の証明書ハッシュ値
                "1F039BB5 7861C27A 3916C778 8E78CE00 690B3974 3EB8259F E2627B8D 4C0EC35A");

        // 以下同様に他のパートナーアプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
        return sWhitelists.test(context, pkgname);
    }
    // 利用元アプリのパッケージ名を取得
```



```
    private String getCallingPackage(Context context) {
        String pkgname = null;
        ActivityManager am = (ActivityManager)context.getSystemService(Context.ACTIVITY_SERVICE);
        List<RunningAppProcessInfo> procList = am.getRunningAppProcesses();
        int callingPid = Binder.getCallingPid();
        if (procList != null) {
            for (RunningAppProcessInfo proc : procList) {
                if (proc.pid == callingPid) {
                    pkgname = proc.pkgList[proc.pkgList.length - 1];
                    break;
                }
            }
        }
        return pkgname;
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // ★ポイント 1★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }

        // ★ポイント 2★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 3★ パートナーアプリに開示してよい情報に限り返送してよい
        // query の結果がパートナーアプリに開示してよい情報かどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;
```

```
        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // ★ポイント 1★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }

        // ★ポイント 2★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 3★ パートナーアプリに開示してよい情報に限り返送してよい
        // Insert 結果、発番される ID がパートナーアプリに開示してよい情報かどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // ★ポイント 1★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }

        // ★ポイント 2★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 3★ パートナーアプリに開示してよい情報に限り返送してよい
        // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // 5 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 15;  // 15 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
```



```
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // ★ポイント 1★ 利用元アプリの証明書がホワイトリストに登録されていることを確認する
        if (!checkPartner(getContext(), getCallingPackage(getContext()))) {
            throw new SecurityException("利用元アプリはパートナーアプリではない。");
        }

        // ★ポイント 2★ パートナーアプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 3★ パートナーアプリに開示してよい情報に限り返送してよい
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // 10 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 20;  // 20 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }
}
```

4.5.1.4. 自社限定 Content Provider を作る

自社限定 Content Provider は、自社以外のアプリから利用されることを禁止する Content Provider である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。

ポイント:

1. 独自定義 Signature Permission を定義する
2. Content Provider 定義にて、独自定義 Signature Permission を要求宣言する
3. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
4. 自社アプリからのリクエストであっても、パラメータの安全性を確認する
5. 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.proprietaryprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント1★ 独自定義 Signature Permission を定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント2★ 独自定義 Signature Permission を要求宣言する -->
        <provider
            android:name=".ProprietaryProvider"
            android:authorities="org.jssec.android.provider.proprietaryprovider"
            android:permission="org.jssec.android.permission.MY_PERMISSION" />
    </application>
</manifest>
```

ProprietaryProvider.java

```
package org.jssec.android.provider.proprietaryprovider;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.Context;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.MatrixCursor;
import android.net.Uri;
```



```
public class ProprietaryProvider extends ContentProvider {

    public static final String AUTHORITIY = "org.jssec.android.provider.proprietaryprovider";
    public static final String CONTENT_TYPE = "vnd.android.cursor.dir/vnd.org.jssec.contenttype";
    public static final String CONTENT_ITEM_TYPE = "vnd.android.cursor.item/vnd.org.jssec.contenttype";

    // Content Provider が提供するインタフェースを公開
    public interface Download {
        public static final String PATH = "downloads";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }
    public interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    // UriMatcher
    private static final int DOWNLOADS_CODE = 1;
    private static final int DOWNLOADS_ID_CODE = 2;
    private static final int ADDRESSES_CODE = 3;
    private static final int ADDRESSES_ID_CODE = 4;
    private static UriMatcher sUriMatcher;
    static {
        sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH, DOWNLOADS_CODE);
        sUriMatcher.addURI(AUTHORITIY, Download.PATH + "/#", DOWNLOADS_ID_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH, ADDRESSES_CODE);
        sUriMatcher.addURI(AUTHORITIY, Address.PATH + "/#", ADDRESSES_ID_CODE);
    }

    // DB を使用せずに固定値を返す例にしているため、query メソッドで返す Cursor を事前に定義
    private static MatrixCursor sAddressCursor = new MatrixCursor(new String[] { "_id", "pref" });
    static {
        sAddressCursor.addRow(new String[] { "1", "北海道" });
        sAddressCursor.addRow(new String[] { "2", "青森" });
        sAddressCursor.addRow(new String[] { "3", "岩手" });
    }
    private static MatrixCursor sDownloadCursor = new MatrixCursor(new String[] { "_id", "path" });
    static {
        sDownloadCursor.addRow(new String[] { "1", "/sdcard/downloads/sample.jpg" });
        sDownloadCursor.addRow(new String[] { "2", "/sdcard/downloads/sample.txt" });
    }

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
```

```
    }

    @Override
    public boolean onCreate() {
        return true;
    }

    @Override
    public String getType(Uri uri) {

        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case ADDRESSES_CODE:
            return CONTENT_TYPE;

        case DOWNLOADS_ID_CODE:
        case ADDRESSES_ID_CODE:
            return CONTENT_ITEM_TYPE;

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        // ★ポイント 4★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match() と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 5★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // query の結果がパートナーアプリに開示してよい情報かどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
        case DOWNLOADS_ID_CODE:
            return sDownloadCursor;

        case ADDRESSES_CODE:
        case ADDRESSES_ID_CODE:
            return sAddressCursor;

        default:
            throw new IllegalArgumentException("Invalid URI : " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }
```



```
        // ★ポイント 4★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 5★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // Insert 結果、発番される ID がパートナーアプリに開示してよい情報かどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return ContentUris.withAppendedId(Download.CONTENT_URI, 3);

        case ADDRESSES_CODE:
            return ContentUris.withAppendedId(Address.CONTENT_URI, 4);

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        // ★ポイント 4★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
        // ★ポイント 5★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // Update されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 5;   // 5 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 15;  // 15 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(getContext(), MY_PERMISSION, myCertHash(getContext()))) {
            throw new SecurityException("独自定義 Signature Permission が自社アプリにより定義されていない。");
        }

        // ★ポイント 4★ 自社アプリからのリクエストであっても、パラメータの安全性を確認する
        // ここでは uri が想定の範囲内であることを、UriMatcher#match()と switch case で確認している。
        // 「3.1 入力データの安全性を確認する」を参照。
```

```
        // ★ポイント 5★ 利用元アプリは自社アプリであるから、センシティブな情報を返送してよい
        // Delete されたレコード数がセンシティブな意味を持つかどうかはアプリ次第。
        switch (sUriMatcher.match(uri)) {
        case DOWNLOADS_CODE:
            return 10;  // 10 レコードが update されたという設定

        case DOWNLOADS_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        case ADDRESSES_CODE:
            return 20;  // 20 レコードが update されたという設定

        case ADDRESSES_ID_CODE:
            return 1;   // 1 レコードが update されたという設定

        default:
            throw new IllegalArgumentException("Invalid URI :" + uri);
        }
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // sigPermName を定義したアプリの証明書のハッシュ値を返す
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```


PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

4.5.1.5. 一時許可 Content Provider を作る

一時許可 Content Provider は、基本的には非公開の Content Provider であるが、特定のアプリに対して一時的に特定 URI へのアクセスを許可する Content Provider である。特殊なフラグを指定した Intent を対象アプリに送付することにより、そのアプリに一時的なアクセス権限が付されるようになっている。Content Provider 側アプリが能動的に他のアプリにアクセス許可を与えることもできるし、一時的なアクセス許可を求めてきたアプリに Content Provider 側アプリが受動的にアクセス許可を与えることもできる。

ポイント:

1. Android 2.2(API Level 8)以前では一時許可 Content Provider を実装しない
2. exported="false"により、一時許可する Path 以外を非公開設定する
3. grant-uri-permission により、一時許可する Path を指定する
4. 一時的にアクセスを許可する URI を Intent に指定する
5. 一時的に許可するアクセス権限を Intent に指定する
6. 一時的にアクセスを許可するアプリに明示的 Intent を送信する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.temporaryprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <!-- ★ポイント 1★ Android 2.2 (API Level 8) 以前では一時許可 Content Provider を実装しない (推奨) -->
    <uses-sdk android:minSdkVersion="9" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <activity
            android:name=".TemporaryActiveGrantActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- 一時許可 Content Provider -->
        <!-- ★ポイント 2★ exported="false"により、一時許可する Path 以外を非公開設定する -->
        <provider
            android:name=".TemporaryProvider"
            android:authorities="org.jssec.android.provider.temporaryprovider"
            android:exported="false" >

            <!-- ★ポイント 3★ grant-uri-permission により、一時許可する Path を指定する -->
            <grant-uri-permission android:path="/addresses" />

        </provider>

        <activity
```



```
            android:name=".TemporaryPassiveGrantActivity"
            android:label="@string/app_name"
            android:exported="true" />
    </application>
</manifest>
```

TemporaryActiveGrantActivity.java

```
package org.jssec.android.provider.temporaryprovider;

import android.app.Activity;
import android.content.ActivityNotFoundException;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class TemporaryActiveGrantActivity extends Activity {

    // User Activity に関する情報
    private static final String TARGET_PACKAGE =  "org.jssec.android.provider.temporaryuser";
    private static final String TARGET_ACTIVITY = "org.jssec.android.provider.temporaryuser.TemporaryUserActivity
";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.active_grant);
    }

    // Content Provider 側アプリが能動的に他のアプリにアクセス許可を与えるケース
    public void onSendClick(View view) {
        try {
            Intent intent = new Intent();

            // ★ポイント 4★ 一時的にアクセスを許可する URI を Intent に指定する
            intent.setData(TemporaryProvider.Address.CONTENT_URI);

            // ★ポイント 5★ 一時的に許可するアクセス権限を Intent に指定する
            intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

            // ★ポイント 6★ 一時的にアクセスを許可するアプリに明示的 Intent を送信する
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);

        } catch (ActivityNotFoundException e) {
            Toast.makeText(this, "User Activity が見つからない。", Toast.LENGTH_LONG).show();
        }
    }
}
```

TemporaryPassiveGrantActivity.java

```
package org.jssec.android.provider.temporaryprovider;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
public class TemporaryPassiveGrantActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.passive_grant);
    }

    // 一時的なアクセス許可を求めてきたアプリに Content Provider 側アプリが受動的にアクセス許可を与えるケース
    public void onGrantClick(View view) {
        Intent intent = new Intent();

        // ★ポイント 4★ 一時的にアクセスを許可する URI を Intent に指定する
        intent.setData(TemporaryProvider.Address.CONTENT_URI);

        // ★ポイント 5★ 一時的に許可するアクセス権限を Intent に指定する
        intent.setFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

        // ★ポイント 7★ 一時許可の要求元アプリに Intent を返信する
        setResult(Activity.RESULT_OK, intent);
        finish();
    }

    public void onCloseClick(View view) {
        finish();
    }
}
```


4.5.2. ルールブック

Content Provider を作る際は以下のルールを守ること。

1. Android 2.2（API Level 8）以前では内部使用の Content Provider は作らない	（必須）
2. 内部使用の Content Provider は非公開設定する	（必須）
3. リクエストパラメータの安全性を確認する	（必須）
4. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する	（必須）
5. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する	（必須）

4.5.2.1. Android 2.2（API Level 8）以前では内部使用の Content Provider は作らない （必須）

Android 2.2（API Level 8）以前では Content Provider の非公開設定は Android 2.2（API Level 8）以前では機能しない。同一アプリ内でのデータ共有のためならContent Providerを使わず、DBなどのデータ格納先へ直接アクセスすることで代用できる。

4.5.2.2. 内部使用の Content Provider は非公開設定する （必須）

同一アプリ内からのみ利用される Content Provider は他のアプリからアクセスできる必要がないだけでなく、開発者も Content Provider を攻撃するアクセスを考慮しないことが多い。Content Provider はデータ共有するための仕組みであるため、デフォルトでは公開扱いになってしまう。同一アプリ内からのみ利用される Content Provider は明示的に非公開設定し、非公開 Content Provider とすべきである。Android 2.3.1（API Level 9）以降では、provider 要素に android:exported="false"と指定することで、Content Provider を非公開にできる。

AndroidManifest.xml

```
    <!-- ★ポイント 1★ Android 2.2 (API Level 8) 以前では非公開 Content Provider を実装しない（できない） -->
    <uses-sdk android:minSdkVersion="9" />

～省略～

        <!-- ★ポイント 2★ exported="false"により、明示的に非公開設定する -->
        <provider
            android:name=".PrivateProvider"
            android:authorities="org.jssec.android.provider.privateprovider"
            android:exported="false" />
```

4.5.2.3. リクエストパラメータの安全性を確認する （必須）

Content Provider のタイプによって若干リスクは異なるが、基本的にはリクエストパラメータを処理する際には、まず最初にその安全性を確認しなければならない。

Content Provider の各メソッドは SQL 文の構成要素パラメータを受け取ることを想定したインタフェースになってはいるものの、仕組みの上では単に任意の文字列を受け渡すだけのものであり、Content Provider 側では想定外のパラメータが与えられるケースを想定しなければならないことに注意が必要だ。

公開 Content Provider は不特定多数のアプリからリクエストを受け取るため、マルウェアの攻撃リクエストを受け取る可能性がある。非公開 Content Provider は他のアプリからリクエストを直接受け取ることはない。しかし同一アプリ内の公開 Activity が他のアプリから受け取った Intent のデータを非公開 Content Provider に転送するといったケースも考えられるため、リクエストを盲目的に安全であると考えてはならない。その他の Content Provider についても、やはりリクエストの安全性を確認する必要がある。

「3.1 入力データの安全性を確認する」を参照すること。

4.5.2.4. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）

自社アプリだけから利用できる自社限定 Content Provider を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2 Permission と Protection Level」の「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

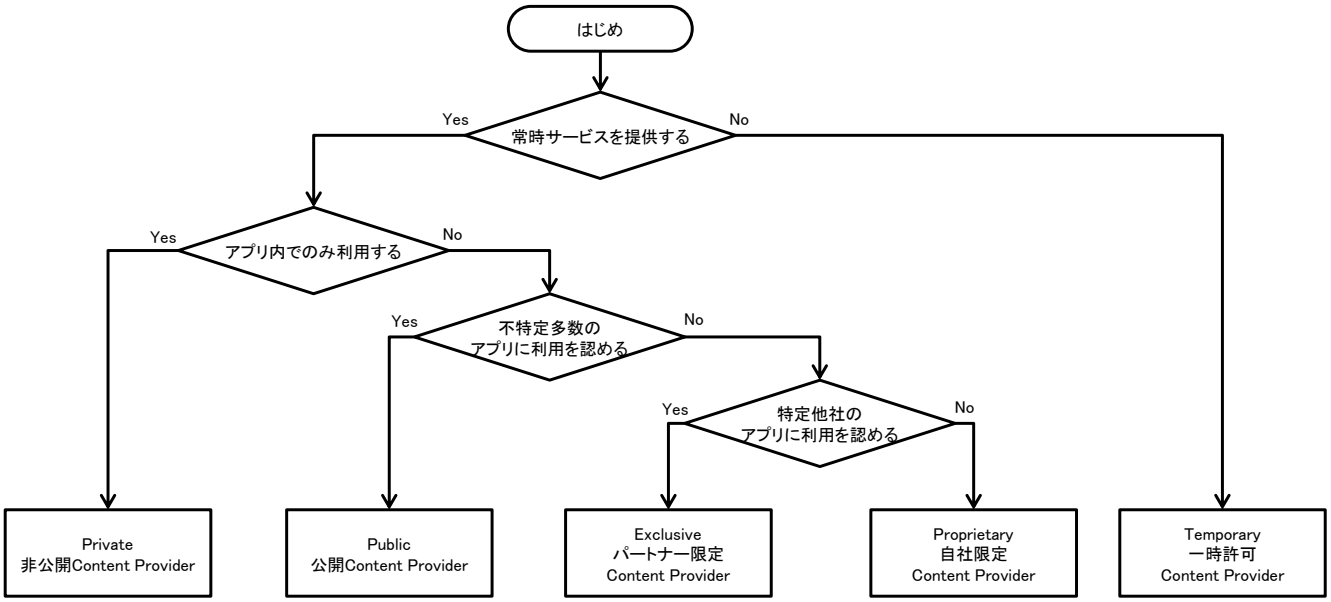
4.5.2.5. 結果情報を返す場合には、返送先アプリからの結果情報漏洩に注意する （必須）

query()や insert()ではリクエスト要求元アプリに結果情報として Cursor や Uri が返送される。結果情報にセンシティブな情報が含まれる場合、返送先アプリから情報漏洩する可能性がある。また update()や delete()では更新または削除されたレコード数がリクエスト要求元アプリに結果情報として返送される。まれにアプリ仕様によっては更新または削除されたレコード数がセンシティブな意味を持つ場合があるので注意すべきだ。

4.6. Content Provider を利用する

4.6.1. サンプルコード

利用先 Content Provider がどのように利用されることを前提としているかによって、利用元 Component に発生するリスクや適切な利用方法が異なる。次の判定フローによって利用する Content Provider がどのタイプであるかを判断できる。



4.6.1.1. 非公開 Content Provider を利用する

同一アプリ内だけで利用される Content Provider(非公開 Content Provider)を利用する場合、比較的安全である。

ポイント:

1. 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよい
2. 同一アプリ内からの結果情報であっても、受信データの安全性を確認する

PrivateUserActivity.java

```
package org.jssec.android.provider.privateprovider;

import android.app.Activity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PrivateUserActivity extends Activity {

    public void onQueryClick(View view) {

        logLine("[Query]");

        // ★ポイント1★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよい
        Cursor cursor = getContentResolver().query(
                PrivateProvider.Download.CONTENT_URI, null, null, null, null);

        // ★ポイント2★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        if (cursor == null) {
            logLine("  null cursor");
        } else {
            boolean moved = cursor.moveToFirst();
            while (moved) {
                logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                moved = cursor.moveToNext();
            }
            cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント1★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよい
        Uri uri = getContentResolver().insert(PrivateProvider.Download.CONTENT_URI, null);

        // ★ポイント2★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {
```



```
        logLine("[Update]");

        // ★ポイント 1★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよい
        int count = getContentResolver().update(PrivateProvider.Download.CONTENT_URI, null, null, null);

        // ★ポイント 2★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // ★ポイント 1★ 同一アプリ内へのリクエストであるから、センシティブな情報をリクエストに含めてよい
        int count = getContentResolver().delete(
                PrivateProvider.Download.CONTENT_URI, null, null);

        // ★ポイント 2★ 同一アプリ内からの結果情報であっても、受信データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

4.6.1.2. 公開 Content Provider を利用する

Android OS 既定ではない独自作成の公開 Content Provider を利用する場合、その公開 Content Provider に成り済ましたマルウェアにリクエストパラメータを受信されることがあること、および、攻撃結果データを受け取ることがあることに注意が必要である。Android OS 既定の Contacts や MediaStore 等も公開 Content Provider であるが、マルウェアはそれら Content Provider に成り済ましできない。ここでは独自作成の公開 Content Provider を利用する場合について述べる。

ポイント:

1. センシティブな情報をリクエストに含めてはならない
2. 結果データの安全性を確認する

PublicUserActivity.java

```
package org.jssec.android.provider.publicuser;

import android.app.Activity;
import android.content.ContentValues;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class PublicUserActivity extends Activity {

    // 利用先の Content Provider 情報
    private static final String AUTHORITIY = "org.jssec.android.provider.publicprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider が不在");
            return;
        }

        // ★ポイント 1★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        Cursor cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

        // ★ポイント 2★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        if (cursor == null) {
            logLine("  null cursor");
        } else {
            boolean moved = cursor.moveToFirst();
            while (moved) {
```



```
                logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                moved = cursor.moveToNext();
            }
            cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider が不在");
            return;
        }

        // ★ポイント 1★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // ★ポイント 2★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider が不在");
            return;
        }

        // ★ポイント 1★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // ★ポイント 2★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        if (!providerExists(Address.CONTENT_URI)) {
            logLine("  Content Provider が不在");
            return;
        }
```

```
        // ★ポイント 1★ センシティブな情報をリクエストに含めてはならない
        // リクエスト先のアプリがマルウェアである可能性がある。
        // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // ★ポイント 2★ 結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records deleted", count));
    }

    private boolean providerExists(Uri uri) {
        ProviderInfo pi = getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        return (pi != null);
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```


4.6.1.3. パートナー限定 Content Provider を利用する

パートナー限定 Content Provider は、特定のアプリだけから利用できる Content Provider である。パートナー企業のアプリと自社アプリが連携してシステムを構成し、パートナーアプリとの間で扱う情報や機能を守るために利用される。ここではパートナー限定 Content Provider を利用する方法を説明する。

ポイント:

1. 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されていることを確認する
2. パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
3. パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する

ExclusiveUserActivity.java

```
package org.jssec.android.provider.exclusiveuser;

import org.jssec.android.shared.PkgCertWhitelists;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ExclusiveUserActivity extends Activity {

    // 利用先の Content Provider 情報
    private static final String AUTHORITIY = "org.jssec.android.provider.exclusiveprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    // ★ポイント 1★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されていることを確認する
    private static PkgCertWhitelists sWhitelists = null;
    private static void buildWhitelists(Context context) {
        boolean isdebug = Utils.isDebuggable(context);
        sWhitelists = new PkgCertWhitelists();

        // パートナー限定 Content Provider アプリ org.jssec.android.provider.exclusiveprovider の証明書ハッシュ値を登録
        sWhitelists.add("org.jssec.android.provider.exclusiveprovider", isdebug ?
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255" :
                // keystore の"my company key"の証明書ハッシュ値
                "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA");

        // 以下同様に他のパートナー限定 Content Provider アプリを登録...
    }
    private static boolean checkPartner(Context context, String pkgname) {
        if (sWhitelists == null) buildWhitelists(context);
```

```
        return sWhitelists.test(context, pkgname);
    }
    // uri を AUTHORITY とする Content Provider のパッケージ名を取得
    private String providerPkgname(Uri uri) {
        String pkgname = null;
        ProviderInfo pi = getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // ★ポイント 1★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されていること
を確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 2★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        Cursor cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

        // ★ポイント 3★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        if (cursor == null) {
            logLine("  null cursor");
        } else {
            boolean moved = cursor.moveToFirst();
            while (moved) {
                logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                moved = cursor.moveToNext();
            }
            cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 1★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されていること
を確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 2★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // ★ポイント 3★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {
```



```
        logLine("[Update]");

        // ★ポイント 1★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されていること
を確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 2★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // ★ポイント 3★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // ★ポイント 1★ 利用先パートナー限定 Content Provider アプリの証明書がホワイトリストに登録されていること
を確認する
        if (!checkPartner(this, providerPkgname(Address.CONTENT_URI))) {
            logLine("  利用先 Content Provider アプリはホワイトリストに登録されていない。");
            return;
        }

        // ★ポイント 2★ パートナー限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // ★ポイント 3★ パートナー限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;
```

```
import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // 0-9A-F 以外の文字がある

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
```



```
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

4.6.1.4. 自社限定 Content Provider を利用する

自社限定 Content Provider は、自社以外のアプリから利用されることを禁止する Content Provider である。複数の自社製アプリでシステムを構成し、自社アプリが扱う情報や機能を守るために利用される。ここでは自社限定 Content Provider を利用する方法を説明する。

ポイント:

1. 独自定義 Signature Permission を定義する
2. 独自定義 Signature Permission を利用宣言する
3. 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
4. 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
5. 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
6. 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.provider.proprietaryuser"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント 1★ 独自定義 Signature Permission を定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- ★ポイント 2★ 独自定義 Signature Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.permission.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ProprietaryUserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

ProprietaryUserActivity.java

```
package org.jssec.android.provider.proprietaryuser;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
```



```
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class ProprietaryUserActivity extends Activity {

    // 利用先の Content Provider 情報
    private static final String AUTHORITIY = "org.jssec.android.provider.proprietaryprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    // 利用先 Content Provider のパッケージ名を取得
    private static String providerPkgname(Context context, Uri uri) {
        String pkgname = null;
        PackageManager pm = context.getPackageManager();
        ProviderInfo pi = pm.resolveContentProvider(uri.getAuthority(), 0);
        if (pi != null) pkgname = pi.packageName;
        return pkgname;
    }

    public void onQueryClick(View view) {

        logLine("[Query]");

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // ★ポイント 4★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
```

```
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, myCertHash(this))) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 5★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        Cursor cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

        // ★ポイント 6★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        if (cursor == null) {
            logLine("  null cursor");
        } else {
            boolean moved = cursor.moveToFirst();
            while (moved) {
                logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                moved = cursor.moveToNext();
            }
            cursor.close();
        }
    }

    public void onInsertClick(View view) {

        logLine("[Insert]");

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // ★ポイント 4★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 5★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        Uri uri = getContentResolver().insert(Address.CONTENT_URI, values);

        // ★ポイント 6★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine("  uri:" + uri);
    }

    public void onUpdateClick(View view) {

        logLine("[Update]");

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
```



```
        }

        // ★ポイント 4★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 5★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        ContentValues values = new ContentValues();
        values.put("pref", "東京都");
        String where = "_id = ?";
        String[] args = { "4" };
        int count = getContentResolver().update(Address.CONTENT_URI, values, where, args);

        // ★ポイント 6★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records updated", count));
    }

    public void onDeleteClick(View view) {

        logLine("[Delete]");

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        String correctHash = myCertHash(this);
        if (!SigPerm.test(this, MY_PERMISSION, correctHash)) {
            logLine("  独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // ★ポイント 4★ 利用先 Content Provider アプリの証明書が自社の証明書であることを確認する
        String pkgname = providerPkgname(this, Address.CONTENT_URI);
        if (!PkgCert.test(this, pkgname, correctHash)) {
            logLine("  利用先 Content Provider は自社アプリではない。");
            return;
        }

        // ★ポイント 5★ 自社限定 Content Provider アプリに開示してよい情報に限りリクエストに含めてよい
        int count = getContentResolver().delete(Address.CONTENT_URI, null, null);

        // ★ポイント 6★ 自社限定 Content Provider アプリからの結果であっても、結果データの安全性を確認する
        // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
        logLine(String.format("  %s records deleted", count));
    }

    private TextView mLogView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
```

```
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // sigPermName を定義したアプリの証明書のハッシュ値を返す
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;

public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
```



```
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

4.6.1.5. 一時許可 Content Provider を利用する

一時許可 Content Provider は、基本的には非公開の Content Provider であるが、特定のアプリに対して一時的に特定 URI へのアクセスを許可する Content Provider である。ここでは一時許可 Content Provider を利用する方法を説明する。

ポイント:

1. センシティブな情報をリクエストに含めてはならない
2. 結果データの安全性を確認する

TemporaryUserActivity.java

```
package org.jssec.android.provider.temporaryuser;

import android.app.Activity;
import android.content.Intent;
import android.content.pm.ProviderInfo;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

public class TemporaryUserActivity extends Activity {

    // Provider Activity に関する情報
    private static final String TARGET_PACKAGE =  "org.jssec.android.provider.temporaryprovider";
    private static final String TARGET_ACTIVITY = "org.jssec.android.provider.temporaryprovider.TemporaryPassiveG
rantActivity";

    // 利用先の Content Provider 情報
    private static final String AUTHORITIY = "org.jssec.android.provider.temporaryprovider";
    private interface Address {
        public static final String PATH = "addresses";
        public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITIY + "/" + PATH);
    }

    private static final int REQUEST_CODE = 1;

    public void onQueryClick(View view) {

        logLine("[Query]");

        try {
            if (!providerExists(Address.CONTENT_URI)) {
                logLine("  Content Provider が不在");
                return;
            }

            // ★ポイント 1★ センシティブな情報をリクエストに含めてはならない
            // リクエスト先のアプリがマルウェアである可能性がある。
            // マルウェアに取得されても問題のない情報であればリクエストに含めてもよい。
            Cursor cursor = getContentResolver().query(Address.CONTENT_URI, null, null, null, null);

            // ★ポイント 2★ 結果データの安全性を確認する
            // サンプルにつき割愛。「3.1 入力データの安全性を確認する」を参照。
```



```
            if (cursor == null) {
                logLine("  null cursor");
            } else {
                boolean moved = cursor.moveToFirst();
                while (moved) {
                    logLine(String.format("  %d, %s", cursor.getInt(0), cursor.getString(1)));
                    moved = cursor.moveToNext();
                }
                cursor.close();
            }
        } catch (SecurityException ex) {
            logLine("  例外:" + ex.getMessage());
        }
    }

    // このアプリが一時的なアクセス許可を要求し、Content Provider 側アプリが受動的にアクセス許可を与えるケース
    public void onGrantRequestClick(View view) {
        Intent intent = new Intent();
        intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
        startActivityForResult(intent, REQUEST_CODE);
    }

    private boolean providerExists(Uri uri) {
        ProviderInfo pi = getPackageManager().resolveContentProvider(uri.getAuthority(), 0);
        return (pi != null);
    }

    private TextView mLogView;

    // Content Provider 側アプリが能動的にこのアプリにアクセス許可を与えるケース
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mLogView = (TextView)findViewById(R.id.logview);
    }

    private void logLine(String line) {
        mLogView.append(line);
        mLogView.append("¥n");
    }
}
```

4.6.2. ルールブック

Content Provider 利用時には以下のルールを守ること。

1. Content Provider の結果データの安全性を確認する	（必須）
2. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する	（必須）

4.6.2.1. Content Provider の結果データの安全性を確認する （必須）

Content Provider のタイプによって若干リスクは異なるが、基本的には結果データを処理する際には、まず最初に結果データの安全性を確認しなければならない。

利用先 Content Provider が公開 Content Provider の場合、公開 Content Provider に成り済ましたマルウェアが攻撃結果データを返送してくる可能性がある。利用先 Content Provider が非公開 Content Provider の場合、同一アプリ内から結果データを受け取るのでリスクは少ないが、結果データを盲目的に安全であると考えてはならない。その他の Content Provider についても、やはり結果データの安全性を確認する必要がある。

「3.1 入力データの安全性を確認する」を参照すること。

4.6.2.2. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）

自社アプリだけから利用できる自社限定 Content Provider を利用する場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 利用宣言だけでは保護が不十分であるため、「5.2 Permission と Protection Level」の「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

4.7. Service を作る

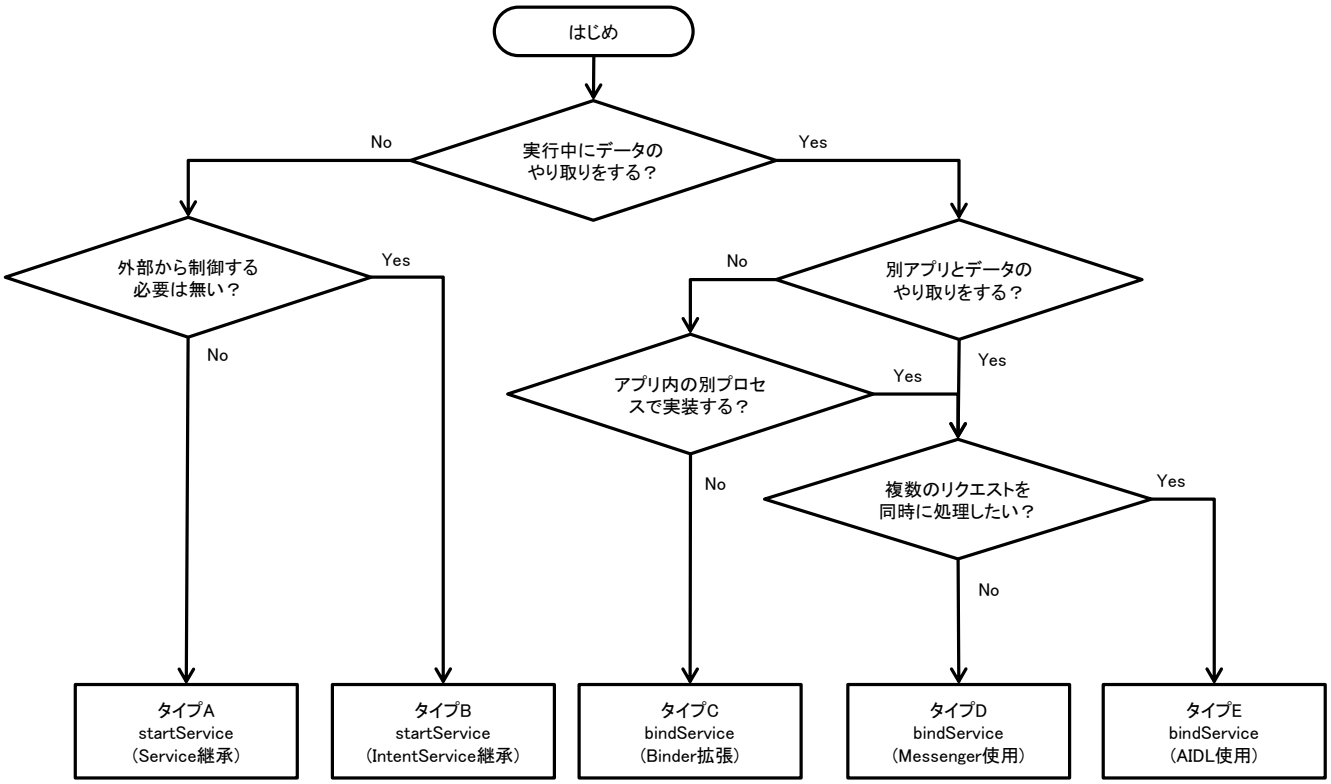
4.7.1. サンプルコード

Service の実装方法は複数あり、その中から目的に合った方法を選択することになる。Service に関するセキュリティについても本章で記載するが、サンプルコードでは Service の実装方法に重みを置いて説明する。AndroidManifest.xml に指定できる exported や permission の考え方は、Activity 等のコンポーネントと同様であるので、本章では詳細の説明を行わない。

Service の実装方法は、startService を利用する場合と bindService を利用する場合とに大きく分かれる。startService と bindService の両方で利用できる Service を作成することも可能ではあるが、デバッグのしやすさ等を考えると個別に作成した方が良い。

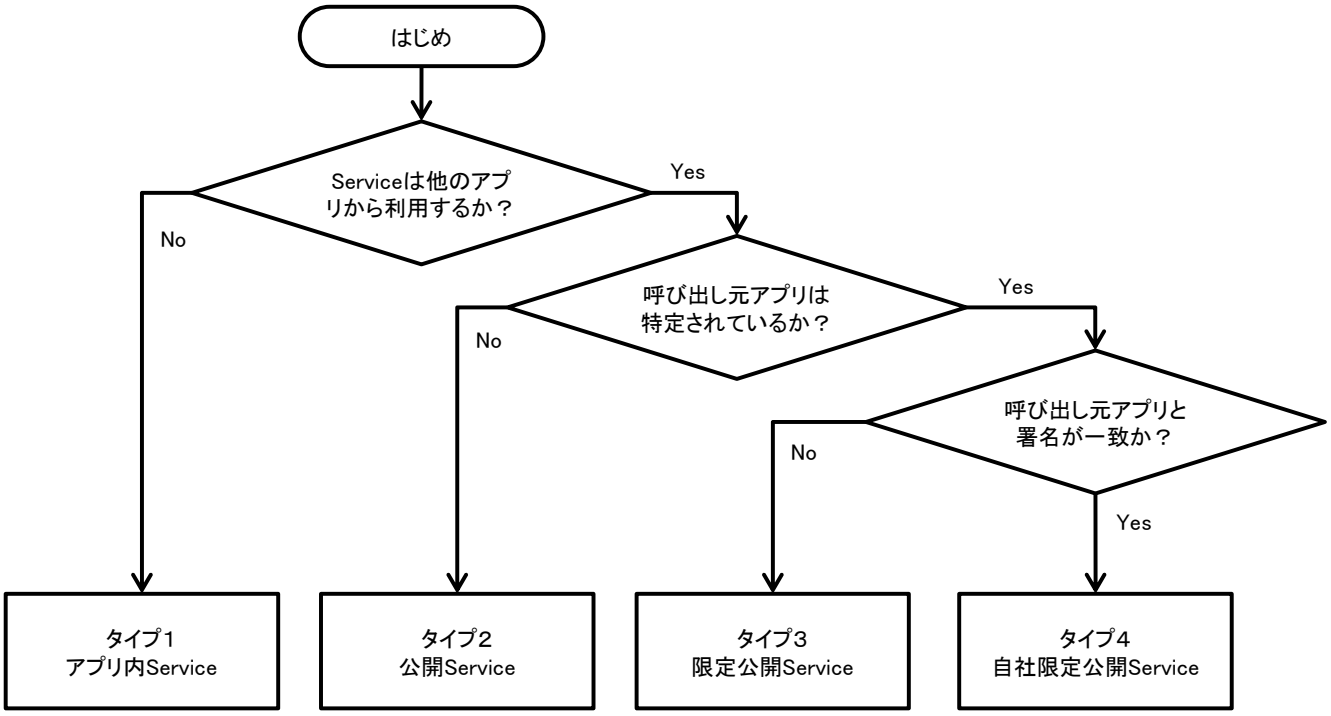
Service の実装方法を決定するために、以下の様な項目について検討を行うことになる。

- 実行中にデータのやり取りを行うか
- 外部から制御する必要が無いか（処理が Service 内で完結するか）
- 別アプリとデータのやり取りを行うか
- 同一アプリ内に Service を持つ場合でも、別プロセスとして実行するか
- 複数のリクエストを同時に処理するか



備考:

Service のサンプルコードではコンポーネントの公開、非公開について詳細な説明は行わないが、一般的に以下のフローで決定できる。公開、非公開を決定する、exported と permission の詳細な説明は、Activity の章などを参照してほしい。



上記フローにおいて、タイプ3は呼び出し元アプリを確認するための特殊な実装が必要になるため、本文書では使用を推奨しない。AndroidManifest.xml に記載する exported と permission を纏めると次の様になる。

タイプ ＼ 要素	exported	permission
タイプ 1	false	無し
タイプ 2	true	無し
タイプ 3	true	Protection Level = normal (Protection Level = dangerous)
タイプ 4	true	Protection Level = signature

タイプ4における AndroidManifest.xml への記載例は以下になる。

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.type4"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />
```



```
    <permission
        android:name="org.jssec.android.service.type4.permission.USE_SERVICE"
        android:protectionLevel="signature" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- Permission を持つ外部アプリに公開する例 -->
        <service android:name=".MyService"
            android:exported="true"
            android:permission="org.jssec.android.service.type4.permission.USE_SERVICE" />

    </application>

</manifest>
```

4.7.1.1. タイプ A – startService(Service 継承)

最も標準的な Service である。Service 実行中はノーティフィケーションバーにアイコンを表示するアプリを作成する。

ポイント:

1. onStartCommand においてセキュリティチェックを行うこと

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.typea"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ServiceTypeAActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- 最も標準的な Service -->
        <!-- intent-filter を付けない場合は、exported=false になる -->
        <service android:name=".ServiceTypeAService" />

    </application>

</manifest>
```

ServiceTypeAService.java

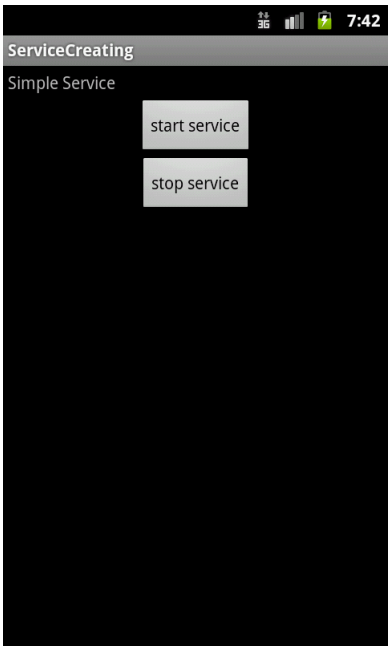
```
package org.jssec.android.service.typea;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class ServiceTypeAService extends Service{

    public static final String EXTRA_USER_ID = "EXTRA_USER_ID";

    private NotificationManager mNM;

    // ノーティフィケーション ID は、アプリ内で重複しないようにする。
    // この ID を利用して、ノーティフィケーションの開始と終了を制御する。
```



```
    private int NOTIFICATION_ID = 1;

    // Service が起動するときに１回だけ呼び出される
    @Override
    public void onCreate() {
        mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

        // サービスが生存している間はノーティフィケーションに表示する
        showNotification();

        Toast.makeText(this, "Service - onCreate()", Toast.LENGTH_SHORT).show();
    }

    // startService()が呼ばれた回数だけ呼び出される
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        // ★ポイント 1★  onStartCommand()内で各種セキュリティチェックを行う

        // Service 呼び出し側からのパラメータを受け取る
        String userId = intent.getStringExtra(EXTRA_USER_ID);
        if( isValidUserId(userId) ){
            // OK
            // TODO Service で実施したい処理

        }else{
            // パラメータ異常時は Service 終了
            stopSelf();
        }

        // サービスは明示的に終了させる
        // stopSelf や stopService を実行したときにサービスを終了する
        // START_NOT_STICKY は、メモリが少ない等で kill された場合に自動的には復帰しない
        return Service.START_NOT_STICKY;
    }

    // Service が終了するときに１回だけ呼び出される
    @Override
    public void onDestroy() {

        // ノーティフィケーションを非表示にする
        mNM.cancel(NOTIFICATION_ID);

        Toast.makeText(this, "Service - onDestroy()", Toast.LENGTH_SHORT).show();
    }

    /**
     * Service が実行中であることをノーティフィケーションバーに表示する
     */
    private void showNotification() {

        // ノーティフィケーションバーに登場する時のアイコン、テキストを指定
        Notification notification = new Notification(
                R.drawable.ic_launcher, "Start Service.",
                System.currentTimeMillis());

        // ノーティフィケーションをタップしたときに起動する Activity を指定
        Intent intent = new Intent(this, ServiceTypeAActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);
```

```
        // ノーティフィケーションの表示項目を指定
        notification.setLatestEventInfo(this, "Service is active.",
                        "If you tap, showing Activity.", contentIntent);

        // ノーティフィケーション表示
        mNM.notify(NOTIFICATION_ID, notification);
    }

    /**
     * ユーザーIDが正常であることを確認する
     * @param userId
     * @return
     */
    private boolean isValidUserId(String userId){
        // TODO ユーザーIDの正当性確認
        return true;
    }

    @Override
    public IBinder onBind(Intent intent) {
        // startServiceで起動するServiceでは使わない
        return null;
    }
}
```

Service を起動するための Activity を用意する。画面のイメージは以下の様になる。



type_a_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="vertical" android:p
adding="4dip"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```



```
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="4dip"
        android:text="Simple Service" />

    <!-- Service を開始するボタン -->
    <Button android:id="@+id/start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="start service" />

    <!-- Service を終了するボタン -->
    <Button android:id="@+id/stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="stop service" />

</LinearLayout>
```

ServiceTypeAActivity.java

```
package org.jssec.android.service.typea;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

public class ServiceTypeAActivity extends Activity {

    private Context mContext;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.type_a_activity);

        mContext = this;

        // サービス開始ボタン
        findViewById(R.id.start).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(mContext, ServiceTypeAService.class);
                // サービスに userId を通知する
                intent.putExtra(ServiceTypeAService.EXTRA_USER_ID, "abc");
                startService(intent);
            }
        });

        // サービス停止ボタン
        findViewById(R.id.stop).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(mContext, ServiceTypeAService.class);
                stopService(intent);
```

```
            }
        });
    }

    @Override
    public void onStop(){
        super.onStop();

        // サービスが終了していない場合は終了する
        Intent intent = new Intent(mContext, ServiceTypeAService.class);
        stopService(intent);
    }
}
```


4.7.1.2. タイプ B - startService(IntentService 継承)

IntentService は Service を継承して作られているクラスで、以下の特徴がある。

- 別スレッドで実行される
- 処理がキューイングされる

キューイング機構のためマルチスレッドによるバグの混入を排除できる。また、別スレッドが生成されており実装が簡略化される。

Service 実行中はノーティフィケーションバーにアイコンを表示する。Activity には Service を開始するボタンを配置するが、IntentService を継承した Service は外部から制御しない場合に適しているので、停止ボタンは用意しない。

ポイント:

1. onHandleIntent においてセキュリティチェックを行うこと

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.typeb"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ServiceTypeBActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- IntentService を継承した Service -->
        <!-- intent-filter を付けない場合は、exported=false になる -->
        <service android:name=".ServiceTypeBService" />

    </application>

</manifest>
```

ServiceTypeBService.java

```
package org.jssec.android.service.typeb;

import android.app.IntentService;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
```

```
public class ServiceTypeBService extends IntentService{

    public static final String EXTRA_USER_ID = "EXTRA_USER_ID";

    private NotificationManager mNM;

    // ノーティフィケーション ID は、アプリ内で重複しないようにする。
    // この ID を利用して、ノーティフィケーションの開始と終了を制御する。
    private int NOTIFICATION_ID = 1;

    /**
     * IntentService を継承した場合、引数無しのコンストラクタを必ず用意する。
     * これが無い場合、エラーになる。
     */
    public ServiceTypeBService() {
        super("CreatingTypeBService");
    }

    // Service で行いたい処理をこのメソッドに記述する
    @Override
    protected void onHandleIntent(Intent intent) {

        // ★ポイント 1★  onHandleIntent()内でセキュリティチェックを行う

        // Intent パラメータの確認
        String userId = intent.getStringExtra(EXTRA_USER_ID);
        if(isValidUserId(userId)){
            // OK
        }else{
            // 処理が終わったら stopSelf は自動的に呼ばれるので管理しなくて良い
            return;
        }


        mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
        showNotification();

        // 別スレッドになっているので、時間のかかる処理を行っても良い
        try {
            // 5 秒間スリープ
            Thread.sleep(5000);
        } catch (InterruptedException e) {
        }

        // ノーティフィケーションを非表示にする
        mNM.cancel(NOTIFICATION_ID);
    }

    /**
     * ユーザーID が正常であることを確認する
     * @param userId
     * @return
     */
    private boolean isValidUserId(String userId){
        // TODO userId の正当性を確認
        return true;
    }

    /**
```



```
     * Service が実行中であることをノーティフィケーションバーに表示する
     */
    private void showNotification() {

        // ノーティフィケーションバーに登場する時のアイコン、テキストを指定
        Notification notification = new Notification(
                R.drawable.ic_launcher, "Start Service.",
                System.currentTimeMillis());

        // ノーティフィケーションをタップしたときに起動する Activity を指定
        Intent intent = new Intent(this, ServiceTypeBActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);

        // ノーティフィケーションの表示項目を指定
        notification.setLatestEventInfo(this, "Service is active.",
                        "If you tap, showing Activity.", contentIntent);

        // ノーティフィケーション表示
        mNM.notify(NOTIFICATION_ID, notification);
    }
}
```

Service を開始するための Activity には、ボタンを1つだけレイアウトする。

type_b_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="vertical" android:p
adding="4dip"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
        android:paddingBottom="4dip"
        android:text="IntentService" />

    <!-- Service を開始するボタン -->
    <Button android:id="@+id/start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="start service" />

</LinearLayout>
```

ServiceTypeBActivity.java

```
package org.jssec.android.service.typeb;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;

public class ServiceTypeBActivity extends Activity {

    private Context mContext;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.type_b_activity);

        mContext = this;

        // サービス開始ボタン
        findViewById(R.id.start).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(mContext, ServiceTypeBService.class);
                // サービスに userId を通知する
                intent.putExtra(ServiceTypeBService.EXTRA_USER_ID, "abc");
                startService(intent);
            }
        });

    }
}
```


4.7.1.3. タイプ C – bindService(Binder 拡張)

Binder クラスを拡張して Service で実装した機能を呼び出し元に提供できるようにする。bindService を利用する中では、この方法が最もシンプルな実装になる。

Activity から Service が実装したインタフェースのメソッドを利用するアプリを作成する。また、Service 実行中はノーティフィケーションバーにアイコンを表示する。

ポイント:

1. onBind においてセキュリティチェックを行うこと

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.typec"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ServiceTypeCActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Binder クラスを拡張した Service -->
        <!-- intent-filter を付けない場合は、exported=false になる -->
        <service android:name=".ServiceTypeCService" />

    </application>

</manifest>
```

ICreatingTypeCService.java

```
package org.jssec.android.service.typec;

/**
 * Service が提供するメソッドを定義する。
 * 本クラス内のメソッドは、Activity から呼び出す。
 */
public interface ICreatingTypeCService {

    /**
     * ステータス（文字列）をセットする
     * @param status
     */
```

```
    public void setStatus(String status);

    /**
     * ステータス（文字列）を取得する
     * @return
     */
    public String getStatus();

}
```

ServiceTypeCService.java

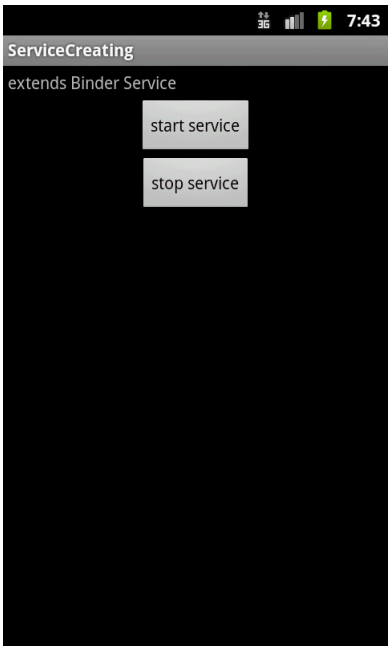
```
package org.jssec.android.service.typec;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.IBinder;
import android.widget.Toast;

public class ServiceTypeCService extends Service
implements ICreatingTypeCService{

    public static final String EXTRA_USER_ID = "EXTRA_USER_ID";

    // インタフェースを通してセットしたり取得する値
    private String mStatus = "initial state";

    private NotificationManager mNM;

    // ノーティフィケーション ID は、アプリ内で重複しないようにする。
    // この ID を利用して、ノーティフィケーションの開始と終了を制御する。
    private int NOTIFICATION_ID = 1;

    /**
     * Service に接続するためのクラス
     */
    public class LocalBinder extends Binder {
        ServiceTypeCService getService() {
            return ServiceTypeCService.this;
        }
    }

    @Override
    public IBinder onBind(Intent intent) {

        // ★ポイント 1★  onBind()内でセキュリティチェックを行う

        // Service 呼び出し側からのパラメータを受け取る
        String userId = intent.getStringExtra(EXTRA_USER_ID);
        if( isValidUserId(userId) ){
            // OK
            // Service に接続するためのオブジェクトを返す
            return new LocalBinder();
        }
```



```
        return null;
    }

    /**
     * ユーザーID が正常であることを確認する
     * @param userId
     * @return
     */
    private boolean isValidUserId(String userId){
        // TODO ユーザーID の正当性確認
        return true;
    }

    @Override
    public void onCreate() {
        mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

        // サービスが生存している間はノーティフィケーションに表示する
        showNotification();

        Toast.makeText(this, "Service - onCreate()", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {

        // ノーティフィケーションを非表示にする
        mNM.cancel(NOTIFICATION_ID);

        Toast.makeText(this, "Service - onDestroy()", Toast.LENGTH_SHORT).show();
    }

    /**
     * Service が実行中であることをノーティフィケーションバーに表示する
     */
    private void showNotification() {

        // ノーティフィケーションバーに登場する時のアイコン、テキストを指定
        Notification notification = new Notification(
                R.drawable.ic_launcher, "Start Service.",
                System.currentTimeMillis());

        // ノーティフィケーションをタップしたときに起動する Activity を指定
        Intent intent = new Intent(this, ServiceTypeCActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);

        // ノーティフィケーションの表示項目を指定
        notification.setLatestEventInfo(this, "Service is active.",
                        "If you tap, showing Activity.", contentIntent);

        // ノーティフィケーション表示
        mNM.notify(NOTIFICATION_ID, notification);
    }

    // 用意したインタフェース
    @Override
    public void setStatus(String status) {
        // セットした文字列は、setStatus()で取得できる
        mStatus = status;
    }
```

```
    // 用意したインタフェース
    @Override
    public String getStatus() {
        // getStatus()でセットした文字列を取得する
        return mStatus;
    }
}
```

Service を開始するための Activity には、bindService と unbindService を実施するボタンをレイアウトする。



type_c_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="vertical" android:p
adding="4dip"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="4dip"
        android:text="extends Binder Service" />

    <!-- Service を開始するボタン -->
    <Button android:id="@+id/start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="start service" />

    <!-- Service を終了するボタン -->
    <Button android:id="@+id/stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```



```
        android:text="stop service" />

</LinearLayout>
```

ServiceTypeCActivity.java

```
package org.jssec.android.service.typec;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;

public class ServiceTypeCActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    // Service に実装するインタフェースは、ICreatingTypeCService クラスとして定義ている
    private ICreatingTypeCService mServiceInterface;

    // Service と接続する時に利用するコネクション。bindService で実装する場合は必要になる。
    private ServiceConnection mConnection = new ServiceConnection() {

        // Service に接続された場合に呼ばれる
        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {
            mServiceInterface = ((ServiceTypeCService.LocalBinder)service).getService();
            Toast.makeText(mContext, "Connect to service", Toast.LENGTH_SHORT).show();

            // インタフェースを通して、Service に実装されたメソッドを呼ぶ
            Log.d("dbg","status = " + mServiceInterface.getStatus());

            mServiceInterface.setStatus("running");

            Log.d("dbg","status = " + mServiceInterface.getStatus());
        }

        // Service が異常終了して、コネクションが切断された場合に呼ばれる
        @Override
        public void onServiceDisconnected(ComponentName className) {
            // Service は利用できないので null をセット
            mServiceInterface = null;
            Toast.makeText(mContext, "Disconnected from service", Toast.LENGTH_SHORT).show();
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.type_c_activity);
```

```
        mContext = this;

        // サービス開始ボタン
        findViewById(R.id.start).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // bindService を実行する
                doBindService();
            }
        });

        // サービス停止ボタン
        findViewById(R.id.stop).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                doUnbindService();
            }
        });

    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        doUnbindService();
    }

    /**
     * Service に接続する
     */
    void doBindService() {
        if (! mIsBound){
            Intent intent = new Intent(this, ServiceTypeCService.class);
            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
            mIsBound = true;
        }
    }

    /**
     * Service への接続を切断する
     */
    void doUnbindService() {
        if (mIsBound) {
            unbindService(mConnection);
            mIsBound = false;
        }
    }
}
```


4.7.1.4. タイプ D – bindService(Messanger 使用)

Messengerクラスから、Binderを取得する実装例を示す。この方法は、プロセス間通信が可能なため、外部アプリと連携する Service の実装にも利用できる。また、処理はキューイングされるため、スレッドセーフに動作する特徴がある。プロセス間通信を可能とする実装方法に、AIDL(Android Interface Definition Language)を利用することも出来るが、可能ならば実装が単純でスレッドセーフな Messenger を利用した方が良い。

Activity で生成した Messenger オブジェクトを Service に渡し、そのオブジェクトのメソッドを Service 側で呼び出すアプリを作成する。また、Service 実行中はノーティフィケーションバーにアイコンを表示する。

ポイント:

1. onBind においてセキュリティチェックを行うこと

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.typed"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- 同じ署名のアプリケーションがアクセス可能な Permission を作成 -->
    <!-- Protection Level は、signature -->
    <permission
        android:name="org.jssec.android.service.typed.permission.SIGNATURE"
        android:protectionLevel="signature" />

    <!-- Permission 使用の宣言をしないと自分の Service 呼び出しでも SecurityException になる -->
    <uses-permission
        android:name="org.jssec.android.service.typed.permission.SIGNATURE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ServiceTypeDActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Messenger を利用した Service -->
        <!-- exported=true になっているので、別アプリケーションからアクセスできる -->
        <service android:name=".ServiceTypeDService"
            android:permission="org.jssec.android.service.typed.permission.SIGNATURE">
            <intent-filter>
                <action android:name="org.jssec.android.service.creating.action.TYPE_D" />
            </intent-filter>
        </service>

    </application>
```

```
</manifest>
```

ServiceTypeDService.java

```
package org.jssec.android.service.typed;

import java.util.ArrayList;
import java.util.Iterator;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.util.Log;
import android.widget.Toast;

public class ServiceTypeDService extends Service{

    public static final String EXTRA_USER_ID = "EXTRA_USER_ID";

    private NotificationManager mNM;

    // ノーティフィケーション ID は、アプリ内で重複しないようにする。
    // この ID を利用して、ノーティフィケーションの開始と終了を制御する。
    private int NOTIFICATION_ID = 1;

    /**
     * クライアントとして登録する時のコマンド。
     * Message.replyTo フィールドにクライアントの Messenger をセットする。
     */
    public static final int MSG_REGISTER_CLIENT = 1;

    /**
     * クライアントの登録解除を行う場合のコマンド。
     * Message.replyTo フィールドにクライアントの Messenger をセットする。
     */
    public static final int MSG_UNREGISTER_CLIENT = 2;

    /**
     * Service の保持している値を、登録されているクライアントに送信するコマンド。
     */
    public static final int MSG_SET_VALUE = 3;

    // Service のクライアント(データ送信先)をリストで管理する
    private ArrayList<Messenger> mClients = new ArrayList<Messenger>();

    // クライアントからのデータを受信するときに利用する Messenger
    private final Messenger mMessenger = new Messenger(new ServiceSideHandler());
```

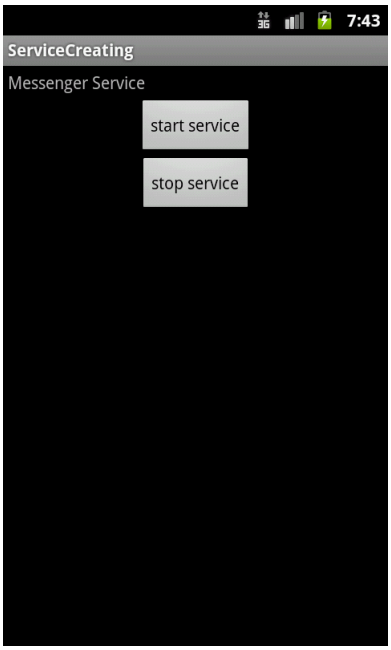


```
    // クライアントから受け取った Message を処理する Handler
    private class ServiceSideHandler extends Handler{
        @Override
        public void handleMessage(Message msg){
            switch(msg.what){
            case MSG_REGISTER_CLIENT:
                // クライアントから受け取った Messenger を追加
                mClients.add(msg.replyTo);
                break;
            case MSG_UNREGISTER_CLIENT:
                mClients.remove(msg.replyTo);
                break;
            case MSG_SET_VALUE:
                // クライアントにデータを送る
                sendMessageToClients();
                break;
            default:
                super.handleMessage(msg);
               break;
            }
        }
    }

    /**
     * クライアントにデータを送る
     */
    private void sendMessageToClients(){

        // クライアントに送信する値
        String sendValue = "I am Service.";

        // 登録されているクライアントへ、順番に送信する
        // ループ途中で remove しても全てのデータにアクセスしたいので Iterator を利用する
        Iterator<Messenger> ite = mClients.iterator();
        while(ite.hasNext()){
            try {
                // Message.obj フィールドに文字列をセット
                Message sendMsg = Message.obtain(null, MSG_SET_VALUE, sendValue);
                Messenger next = ite.next();
                next.send(sendMsg);

            } catch (RemoteException e) {
                // クライアントが存在しない場合は、リストから取り除く
                ite.remove();
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {

        // ★ポイント 1★  onBind()内でセキュリティチェックを行う

        // protectionLevel=signature で保護されていることの確認を行う
        if( ! isProtectionLevelSignature() ){
            Log.d("dbg", "The protection level is not signature.");
            return null;
        }

        // Messenger から Binder を取得
```

```
        return mMessenger.getBinder();
    }

    @Override
    public void onCreate() {
        mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

        // サービスが生存している間はノーティフィケーションに表示する
        showNotification();

        Toast.makeText(this, "Service - onCreate()", Toast.LENGTH_SHORT).show();
    }

    @Override
    public void onDestroy() {

        // ノーティフィケーションを非表示にする
        mNM.cancel(NOTIFICATION_ID);

        Toast.makeText(this, "Service - onDestroy()", Toast.LENGTH_SHORT).show();
    }

    /**
     * Permission の Protection Level が signature であることを確認する
     * @return
     */
    private boolean isProtectionLevelSignature(){

        try {
            // Permission 情報の取得
            PermissionInfo permissionInfo = getPackageManager()
            .getPermissionInfo("org.jssec.android.service.typed.permission.SIGNATURE",
                    PackageManager.GET_META_DATA);

            // Protection Level を確認
            if(permissionInfo.protectionLevel == PermissionInfo.PROTECTION_SIGNATURE){
                return true;
            }

        } catch (NameNotFoundException e) {
            // Permission が見つからない場合
        }

        return false;
    }

    /**
     * Service が実行中であることをノーティフィケーションバーに表示する
     */
    private void showNotification() {

        // ノーティフィケーションバーに登場する時のアイコン、テキストを指定
        Notification notification = new Notification(
                R.drawable.ic_launcher, "Start Service.",
                System.currentTimeMillis());

        // ノーティフィケーションをタップしたときに起動する Activity を指定
        Intent intent = new Intent(this, ServiceTypeDActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);
```

```
        // ノーティフィケーションの表示項目を指定
        notification.setLatestEventInfo(this, "Service is active.",
                        "If you tap, showing Activity.", contentIntent);

        // ノーティフィケーション表示
        mNM.notify(NOTIFICATION_ID, notification);
    }
}
```

Service を開始するための Activity には、bindService と unbindService を実施するボタンをレイアウトしている。



type_d_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="vertical" android:p
adding="4dip"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="4dip"
        android:text="Messenger Service" />

    <!-- Service を開始するボタン -->
    <Button android:id="@+id/start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="start service" />

    <!-- Service を終了するボタン -->
    <Button android:id="@+id/stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="stop service" />

</LinearLayout>
```

ServiceTypeDActivity.java

```
package org.jssec.android.service.typed;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.Messenger;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;

public class ServiceTypeDActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    // Service からデータを受信するときに利用する Messenger
    private Messenger mServiceMessenger = null;

    // Service にデータを送信するときに利用する Messenger
    private final Messenger mActivityMessenger = new Messenger(new ActivitySideHandler());

    // Service から受け取った Message を処理する Handler
    private class ActivitySideHandler extends Handler {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case ServiceTypeDService.MSG_SET_VALUE:
                    // Service では、Message.obj フィールドに値をセットしたので、obj フィールドを取得する
                    String value = (String)msg.obj;
                    Toast.makeText(mContext, "get string : " + value,
                            Toast.LENGTH_SHORT).show();
                    break;
                default:
                    super.handleMessage(msg);
            }
        }
    }

    // Service と接続する時に利用するコネクション。bindService で実装する場合は必要になる。
    private ServiceConnection mConnection = new ServiceConnection() {

        // Service に接続された場合に呼ばれる
        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {
            mServiceMessenger = new Messenger(service);
            Toast.makeText(mContext, "Connect to service", Toast.LENGTH_SHORT).show();
```



```
            try {
                // Service に自分の Messenger を渡す
                Message msg = Message.obtain(null, ServiceTypeDService.MSG_REGISTER_CLIENT);
                msg.replyTo = mActivityMessenger;
                mServiceMessenger.send(msg);
            } catch (RemoteException e) {
                // Service が異常終了していた場合
            }

            // 5 秒後に Service から値を取得する Message を送信する
            Runnable r = new Runnable() {
                @Override
                public void run() {
                    try {
                        Message msg = Message.obtain(null, ServiceTypeDService.MSG_SET_VALUE);
                        mServiceMessenger.send(msg);
                    } catch (RemoteException e) {
                        // Service が異常終了していた場合
                    }
                }
            };

            Handler h = new Handler();
            h.postDelayed(r, 5000);
        }

        // Service が異常終了して、コネクションが切断された場合に呼ばれる
        @Override
        public void onServiceDisconnected(ComponentName className) {
            Toast.makeText(mContext, "Disconnected from service", Toast.LENGTH_SHORT).show();
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.type_d_activity);

        mContext = this;

        // サービス開始ボタン
        findViewById(R.id.start).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // bindService を実行する
                doBindService();
            }
        });

        // サービス停止ボタン
        findViewById(R.id.stop).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                doUnbindService();
            }
        });
    }

    @Override
```

```
    protected void onDestroy() {
        super.onDestroy();
        doUnbindService();
    }

    /**
     * Service に接続する
     */
    void doBindService() {
        if (! mIsBound){
            // アプリケーション間で呼び出すことを想定して、Action で実施している
            Intent intent =
                new Intent("org.jssec.android.service.creating.action.TYPE_D");
            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
            mIsBound = true;
        }
    }

    /**
     * Service への接続を切断する
     */
    void doUnbindService() {
        if (mIsBound) {
            unbindService(mConnection);
            mIsBound = false;
        }
    }
}
```


4.7.1.5. タイプ E – bindService(AIDL 使用)

AIDL ファイルを作成し、インタフェースを作成する実装例を示す。この方法は、プロセス間通信が可能なため、外部アプリと連携する場合にも利用可能である。ただし、Messenger を使用する場合とは異なり、スレッドセーフではないので注意する。

Activity 側で生成したコールバックを Service に渡し、Service からコールバックメソッドを呼び出すアプリを作成する。Service 実行中はノーティフィケーションにアイコンを表示する。なお、eclipse の ADT(Android Development Tools)プラグインを利用して開発している場合は、AIDL ファイルを作成すると自動的にインタフェースが生成される。以下の例では、自動生成したインタフェースを使用したため、インタフェースクラスを java コードでは記載していない。

ポイント:

1. onBind においてセキュリティチェックを行うこと

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.service.typee"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- 同じ署名のアプリケーションがアクセス可能な Permission を作成 -->
    <!-- Protection Level は、signature -->
    <permission
        android:name="org.jssec.android.service.typee.permission.SIGNATURE"
        android:protectionLevel="signature" />

    <!-- Permission 使用の宣言をしないと自分の Service 呼び出しでも SecurityException になる -->
    <uses-permission
        android:name="org.jssec.android.service.typee.permission.SIGNATURE" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".ServiceTypeEActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- AIDL を利用した Service -->
        <!-- exported=true になっているので、別アプリケーションからアクセスできる -->
        <service android:name=".ServiceTypeEService"
            android:permission="org.jssec.android.service.typee.permission.SIGNATURE">
            <intent-filter>
                <action android:name="org.jssec.android.service.creating.action.TYPE_E" />
            </intent-filter>
        </service>
```

```
    </application>

</manifest>
```

実装例では AIDL ファイルを2つ作成する。1つは、Service から Activity にデータを渡すためのコールバックインタフェースで、もう1つは Activity 側で用意したコールバックを Service に渡すためのインタフェースである。なお、AIDL ファイルに記述するパッケージ名は、java ファイルに記述するパッケージ名と同様に、AIDL ファイルを作成するディレクトリ階層に一致させる必要がある。

ICreatingTypeEServiceCallback.aidl

```
package org.jssec.android.service.typee;

interface ICreatingTypeEServiceCallback {
    /**
     * 値が変わった時に呼び出される
     */
    void valueChanged(int value);
}
```

ICreatingTypeEService.aidl

```
package org.jssec.android.service.typee;

import org.jssec.android.service.typee.ICreatingTypeEServiceCallback;

interface ICreatingTypeEService {

    /**
     * コールバックを登録する
     */
    void registerCallback(ICreatingTypeEServiceCallback cb);

    /**
     * コールバックを解除する
     */
    void unregisterCallback(ICreatingTypeEServiceCallback cb);
}
```

ServiceTypeEService.java

```
package org.jssec.android.service.typee;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.os.RemoteCallbackList;
import android.os.RemoteException;
```



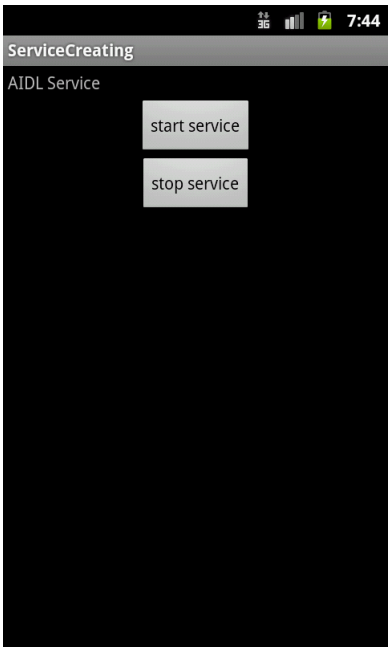
```
import android.util.Log;
import android.widget.Toast;

public class ServiceTypeEService extends Service {

    public static final String EXTRA_USER_ID = "EXTRA_USER_ID";

    private NotificationManager mNM;

    // ノーティフィケーション ID は、アプリ内で重複しないようにする。
    // この ID を利用して、ノーティフィケーションの開始と終了を制御する。
    private int NOTIFICATION_ID = 1;

    private static final int REPORT_MSG = 1;

    // Service からクライアントに通知する値
    private int mValue = 0;

    // コールバックを登録するオブジェクト。
    // RemoteCallbackList の提供するメソッドはスレッドセーフになっている。
    private final RemoteCallbackList<ICreatingTypeEServiceCallback> mCallbacks =
        new RemoteCallbackList<ICreatingTypeEServiceCallback>();

    // コールバックに対して Service からデータを送信するための Handler
    private final Handler mHandler = new Handler() {
        @Override public void handleMessage(Message msg) {
            switch (msg.what) {
            case REPORT_MSG: {
                // 通知を開始する
                // beginBroadcast()は、getBroadcastItem()で取得可能なコピーを作成している
                final int N = mCallbacks.beginBroadcast();
                for (int i = 0; i < N; i++) {
                    ICreatingTypeEServiceCallback target = mCallbacks.getBroadcastItem(i);
                    try {
                        // Service で管理している値をクライアントに通知
                        target.valueChanged(++mValue);

                    } catch (RemoteException e) {
                        // RemoteCallbackList がコールバックを管理しているので、ここでは unregeister しない
                        // RemoteCallbackList.kill()によって全て解除される
                    }
                }
                // finishBroadcast()は、beginBroadcast()と対になる処理
                mCallbacks.finishBroadcast();

                // 5 秒後に繰り返す
                sendMessageDelayed(obtainMessage(REPORT_MSG), 5000);
                break;
            }
            default:
                super.handleMessage(msg);
                break;
            } // switch
        }
    };

    // AIDL で定義したインタフェース
    private final ICreatingTypeEService.Stub mBinder = new ICreatingTypeEService.Stub() {
        public void registerCallback(ICreatingTypeEServiceCallback cb) {
            if (cb != null) mCallbacks.register(cb);
```

```
        }
        public void unregisterCallback(ICreatingTypeEServiceCallback cb) {
            if (cb != null) mCallbacks.unregister(cb);
        }
    };

    @Override
    public IBinder onBind(Intent intent) {

        // ★ポイント 1★  onBind()内でセキュリティチェックを行う

        // protectionLevel=signature で保護されていることの確認を行う
        if( ! isProtectionLevelSignature() ){
            Log.d("dbg", "The protection level is not signature.");
            return null;
        }

        return mBinder;
    }

    @Override
    public void onCreate() {
        mNM = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

        // サービスが生存している間はノーティフィケーションに表示する
        showNotification();

        Toast.makeText(this, "Service - onCreate()", Toast.LENGTH_SHORT).show();

        // Service が実行中の間は、定期的にインクリメントした数字を通知する
        mHandler.sendEmptyMessage(REPORT_MSG);
    }

    @Override
    public void onDestroy() {

        // ノーティフィケーションを非表示にする
        mNM.cancel(NOTIFICATION_ID);

        Toast.makeText(this, "Service - onDestroy()", Toast.LENGTH_SHORT).show();

        // コールバックを全て解除する
        mCallbacks.kill();

        mHandler.removeMessages(REPORT_MSG);
    }

    /**
     * Permission の Protection Level が signature であることを確認する
     * @return
     */
    private boolean isProtectionLevelSignature(){

        try {
            // Permission 情報の取得
            PermissionInfo permissionInfo = getPackageManager()
            .getPermissionInfo("org.jssec.android.service.typee.permission.SIGNATURE",
                    PackageManager.GET_META_DATA);

            // Protection Level を確認
```



```
            if(permissionInfo.protectionLevel == PermissionInfo.PROTECTION_SIGNATURE){
                return true;
            }

        } catch (NameNotFoundException e) {
            // Permission が見つからない場合
        }

        return false;
    }

    /**
     * Service が実行中であることをノーティフィケーションバーに表示する
     */
    private void showNotification() {

        // ノーティフィケーションバーに登場する時のアイコン、テキストを指定
        Notification notification = new Notification(
                R.drawable.ic_launcher, "Start Service.",
                System.currentTimeMillis());

        // ノーティフィケーションをタップしたときに起動する Activity を指定
        Intent intent = new Intent(this, ServiceTypeEActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);

        // ノーティフィケーションの表示項目を指定
        notification.setLatestEventInfo(this, "Service is active.",
                        "If you tap, showing Activity.", contentIntent);

        // ノーティフィケーション表示
        mNM.notify(NOTIFICATION_ID, notification);
    }
}
```

Service を開始するための Activity には、bindService と unbindService を実施するボタンをレイアウトしている。



type_e_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android" android:orientation="vertical" android:p
adding="4dip"
    android:gravity="center_horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:paddingBottom="4dip"
        android:text="AIDL Service" />

    <!-- Service を開始するボタン -->
    <Button android:id="@+id/start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="start service" />

    <!-- Service を終了するボタン -->
    <Button android:id="@+id/stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="stop service" />

</LinearLayout>
```

ServiceTypeEActivity.java

```
package org.jssec.android.service.typee;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Toast;

public class ServiceTypeEActivity extends Activity {

    private boolean mIsBound;
    private Context mContext;

    // AIDL で定義したインタフェース。Service からの通知を受け取る。
    private final ICreatingTypeEServiceCallback.Stub mCallback =
        new ICreatingTypeEServiceCallback.Stub() {
            @Override
            public void valueChanged(int value) throws RemoteException {
                Toast.makeText(mContext, "value = " + value, Toast.LENGTH_SHORT).show();
            }
    };

    // AIDL で定義したインタフェース。Service へ通知する。
```



```
    private ICreatingTypeEService mService = null;

    // Service と接続する時に利用するコネクション。bindService で実装する場合は必要になる。
    private ServiceConnection mConnection = new ServiceConnection() {

        // Service に接続された場合に呼ばれる
        @Override
        public void onServiceConnected(ComponentName className, IBinder service) {
            mService = ICreatingTypeEService.Stub.asInterface(service);

            try{
                // Service に接続
                mService.registerCallback(mCallback);

            }catch(RemoteException e){
                // Service が異常終了した場合
            }

            Toast.makeText(mContext, "Connected to service", Toast.LENGTH_SHORT).show();
        }

        // Service が異常終了して、コネクションが切断された場合に呼ばれる
        @Override
        public void onServiceDisconnected(ComponentName className) {
            Toast.makeText(mContext, "Disconnected from service", Toast.LENGTH_SHORT).show();
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.type_e_activity);

        mContext = this;

        // サービス開始ボタン
        findViewById(R.id.start).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // bindService を実行する
                doBindService();
            }
        });

        // サービス停止ボタン
        findViewById(R.id.stop).setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                doUnbindService();
            }
        });
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        doUnbindService();
    }
```

```
    /**
     * Service に接続する
     */
    private void doBindService() {
        if (! mIsBound){
            // アプリケーション間で呼び出すことを想定して、Action で実施している
            Intent intent =
                new Intent("org.jssec.android.service.creating.action.TYPE_E");
            bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
            mIsBound = true;
        }
    }

    /**
     * Service への接続を切断する
     */
    private void doUnbindService() {
        if (mIsBound) {
            // 登録していたレジスタがある場合は解除
            if(mService != null){
                try{
                    mService.unregisterCallback(mCallback);
                }catch(RemoteException e){
                    // Service が異常終了していた場合
                    // 処理無し
                }
            }

            unbindService(mConnection);
            mIsBound = false;
        }
    }
}
```


4.7.2. ルールブック

Service 実装時には以下のルールを守ること。

1. 内部使用の Service は非公開設定する	（必須）
2. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する	（必須）
3. 連携するタイミングで Service の機能を提供するかを判定する	（必須）

4.7.2.1. 内部使用の Service は非公開設定する （必須）

アプリ内（または、同じ UID）でのみ使用される Service は非公開設定する。これにより、他のアプリから意図せず Intent を受け取ってしまうことがなくなり、アプリの機能を利用される、アプリの動作に異常をきたす等の被害を防ぐことができる。

その方法は簡単で、AndroidManifest.xml に Service を定義する際に、exported 属性を false にするだけである。intent-filter を梱包していない場合は、exported 属性のデフォルト値として false が適用される。しかし、intent-filter を梱包している場合は、デフォルト値が true になるので注意が必要である。

```
<!-- 最も標準的な Service -->
<!-- intent-filter を付けない場合は、exported=false になる -->
<service android:name=".ServiceTypeAService" />
```

```
<!-- Messenger を利用した Service -->
<!-- exported=true になっているので、別アプリケーションからアクセスできる -->
<service android:name=".ServiceTypeDService"
    android:permission="org.jssec.android.service.typed.permission.SIGNATURE">
    <intent-filter>
        <action android:name="org.jssec.android.service.creating.action.TYPE_D" />
    </intent-filter>
</service>
```

4.7.2.2. 独自定義 Signature Permission は、自社アプリが定義したことを確認して利用する （必須）

自社アプリだけから利用できる自社限定 Service を作る場合、独自定義 Signature Permission により保護しなければならない。AndroidManifest.xml での Permission 定義、Permission 要求宣言だけでは保護が不十分であるため、「5.2 Permission と Protection Level」の「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

4.7.2.3. 連携するタイミングで Service の機能を提供するかを判定する （必須）

Intent パラメータの確認や独自定義 Signature Permission の確認といったセキュリティチェックを onCreate に入れてはいけない。その理由は、Service が起動中に新しい要求を受けたときに onCreate の処理が実施されないためである。したがって、startService によって開始される Service を実装する場合は、onStartCommand（IntentService を利用する場合は onHandleIntent）で判定を行わなければならない。bindService で開始する

Service を実装する場合も同様のことが言えるので、onBind で判定をしなければならない。

4.8. SQLite を使う

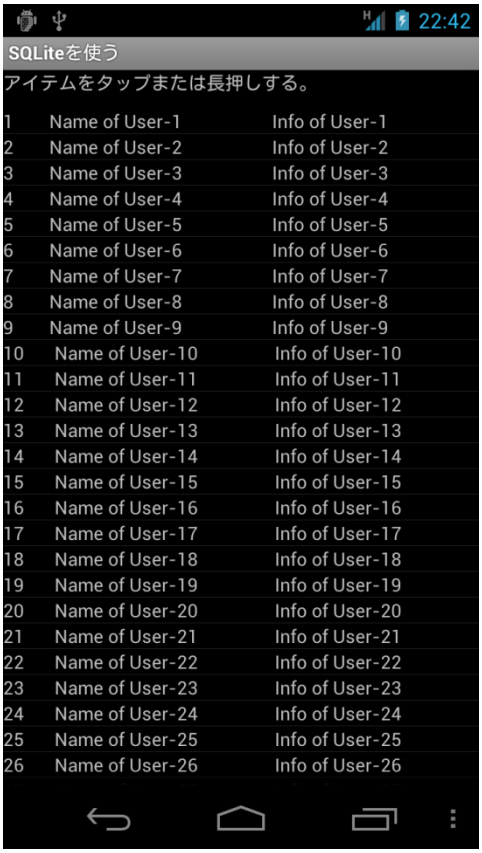
本文書では SQLite を使用してデータベースの作成および操作を行う際にセキュリティ上で注意すべき点をまとめる。主なポイントは、データベースファイルのアクセス権の適切な設定と SQL インジェクションに対する対策である。

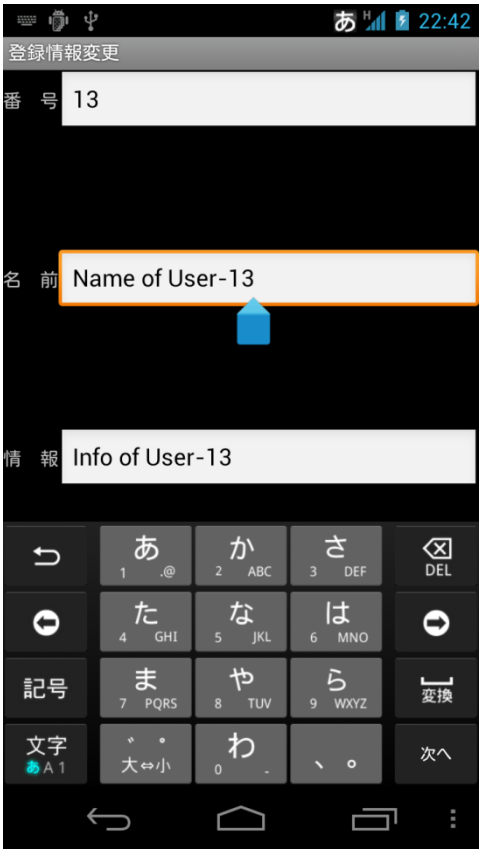
ここでは、直接外部からデータベースファイルの読み書きを許す(複数アプリで共有する)ようなデータベースはここでは想定せず Content Provider のバックエンドやアプリ単体での使用を前提とする。また、ある程度センシティブな情報を扱っていることを想定しているが、そうでない場合も他アプリからの想定外の読み書きを避けるためにもここで挙げる対策を適用することをお勧めする。

4.8.1. サンプルコード

4.8.1.1. データベースの作成と操作

Android のアプリでデータベースを扱う場合、SQLiteOpenHelper を使用することでデータベースファイルの適切な配置およびアクセス権の設定(他のアプリがアクセスできない設定)ができる[1]。ここでは、アプリ起動時にデータベースを作成し、UI 上からデータの検索・追加・変更・削除を行う簡単なアプリを例に、外部からの入力に対して不正な SQL が実行されないように SQL インジェクション対策したサンプルコードを示す。





[1] ファイルの配置に関しては、SQLiteOpenHelper のコンストラクタの第 2 引数(name)にファイルの絶対パスも指定できる。例えば、SD カードのような外部メディアを直接指定した場合には他のアプリからの読み書きが可能になる場合もあるので注意が必要である。

ポイント:

2. データベース作成には SQLiteOpenHelper を使用する
3. SQL インジェクションの対策として入力値を SQL 文に使用する場合にはプレースホルダを利用する
4. SQL インジェクションの保険的な対策としてアプリ要件に従って入力値をチェックする

SampleDbOpenHelper.java

```
package org.jssec.android.sqlite;

import org.jssec.android.sqlite.R;

import android.content.Context;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;
import android.widget.Toast;

public class SampleDbOpenHelper extends SQLiteOpenHelper {
    private SQLiteDatabase           mSampleDb;             //取り扱うデータを格納するデータベース

    public static SampleDbOpenHelper newHelper(Context context)
    {
        //★ポイント 1★ DB 作成には SQLiteOpenHelper を使用する
        return new SampleDbOpenHelper(context);
    }

    public SQLiteDatabase getDb() {
        return mSampleDb;
    }

    //Writable モードで DB を開く
    public void openDatabaseWithHelper() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (!mSampleDb.isReadOnly())// 既に読み書き可能でオープン済み
                    return;
                mSampleDb.close();
              }
            mSampleDb  = getWritableDatabase(); //この段階でオープンされる
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(mContext.getClass().toString(), mContext.getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            Toast.makeText(mContext, R.string.DATABASE_OPEN_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return;
        }
    }

    //ReadOnly モードで DB を開く
    public void openDatabaseReadOnly() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                if (mSampleDb.isReadOnly())// 既に ReadOnly でオープン済み
                    return;
                mSampleDb.close();
            }
            SQLiteDatabase.openDatabase(mContext.getDatabasePath(CommonData.DBFILE_NAME).getPath(), null, SQLiteD
atabase.OPEN_READONLY);
        } catch (SQLException e) {
```



```
            //データベース構築に失敗した場合ログ出力
            Log.e(mContext.getClass().toString(), mContext.getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            Toast.makeText(mContext, R.string.DATABASE_OPEN_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return;
        }
    }

    //Database Close
    public void closeDatabase() {
        try {
            if (mSampleDb != null && mSampleDb.isOpen()) {
                mSampleDb.close();
            }
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(mContext.getClass().toString(), mContext.getString(R.string.DATABASE_CLOSE_ERROR_MESSAGE));
            Toast.makeText(mContext, R.string.DATABASE_CLOSE_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return;
        }
    }

    //Context を覚えておく
    private Context mContext;

    //テーブル作成コマンド
    private static final String CREATE_TABLE_COMMANDS
            = "CREATE TABLE " + CommonData.TABLE_NAME + " ("
            + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + "idno INTEGER, "
            + "name VARCHAR(" + CommonData.TEXT_DATA_LENTGH_MAX + ") NOT NULL, "
            + "info VARCHAR(" + CommonData.TEXT_DATA_LENTGH_MAX + ")"
            + ");";

    public SampleDbOpenHelper(Context context) {
        super(context, CommonData.DBFILE_NAME, null, CommonData.DB_VERSION);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(CREATE_TABLE_COMMANDS);  //DB 構築コマンドの実行
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(this.getClass().toString(), mContext.getString(R.string.DATABASE_CREATE_ERROR_MESSAGE));
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
        // TODO データベースのバージョンアップ時に実行される、データ移行などの処理を記述する
    }

}
```

DataSearchTask.java （SQLite Database プロジェクト）

```
package org.jssec.android.sqlite.task;
```

```
import org.jssec.android.sqlite.CommonData;
import org.jssec.android.sqlite.DataValidator;
import org.jssec.android.sqlite.MainActivity;
import org.jssec.android.sqlite.R;

import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.AsyncTask;
import android.util.Log;


//データ検索タスク
public class DataSearchTask extends AsyncTask<String, Void, Cursor> {
    private MainActivity    mActivity;
    private SQLiteDatabase      mSampleDB;

    public DataSearchTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Cursor doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];
        String  cols[]  =   {"_id", "idno","name","info"};

        Cursor cur;

        //★ポイント 3★ アプリケーション要件に従って入力値をチェックする
        if (!DataValidator.validateData(idno, name, info))
        {
            return null;
        }

        //引数が全部 null だったら全件検索する)
        if ((idno == null || idno.length() == 0) &&
                (name == null || name.length() == 0) &&
                (info == null || info.length() == 0) ) {
            try {
                cur = mSampleDB.query(CommonData.TABLE_NAME, cols, null, null, null, null, null);
            } catch (SQLException e) {
                Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
                return null;
            }
            return cur;
        }

        //No が指定されていたら No で検索
        if (idno != null && idno.length() > 0) {
            String selectionArgs[] = {idno};

            try {
                //★ポイント 2★ プレースホルダを使用する
                cur = mSampleDB.query(CommonData.TABLE_NAME, cols, "idno = ?", selectionArgs, null, null, null);
            } catch (SQLException e) {
                Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
                return null;
```



```
            }
            return cur;
        }

        //Name が指定されていたら Name で完全一致検索
        if (name != null && name.length() > 0) {
            String selectionArgs[] = {name};
            try {
                //★ポイント 2★ プレースホルダを使用する
                cur = mSampleDB.query(CommonData.TABLE_NAME, cols, "name = ?", selectionArgs, null, null, null);
            } catch (SQLException e) {
                Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
                return null;
            }
            return cur;
        }

        //それ以外の場合は info を条件にして部分一致検索
        String argString = info.replaceAll("@", "@@"); //入力として受け取った info 内の$をエスケープ
        argString = argString.replaceAll("%", "@%"); //入力として受け取った info 内の%をエスケープ
        argString = argString.replaceAll("_", "@_"); //入力として受け取った info 内の_をエスケープ
        String selectionArgs[] = {argString};

        try {
            //★ポイント 2★ プレースホルダを使用する
            cur = mSampleDB.query(CommonData.TABLE_NAME, cols, "info LIKE '%' || ? || '%' ESCAPE '@'", selectionA
rgs, null, null, null);
        } catch (SQLException e) {
            Log.e(DataSearchTask.class.toString(), mActivity.getString(R.string.SEARCHING_ERROR_MESSAGE));
            return null;
        }
        return cur;
    }

    @Override
    protected void onPostExecute(Cursor resultCur) {
        mActivity.updateCursor(resultCur);
    }
}
```

DataValidator.java

```
package org.jssec.android.sqlite;

public class DataValidator {
    //入力値をチェックする
    //数字チェック
    public static boolean validateNo(String idno) {
        //null、空文字は OK
        if (idno == null || idno.length() == 0) {
           return true;
        }

        //数字であることを確認する
        try {
            if (!idno.matches("[1-9][0-9]*")) {
                //数字以外の時はエラー
                return false;
            }
```

```
        } catch (NullPointerException e) {
            //バグ
            return false;
        }

        return true;
    }

    // 文字列の長さを調べる
    public static boolean validateLength(String str, int max_length) {
       //null、空文字は OK
       if (str == null || str.length() == 0) {
              return true;
       }

       //文字列の長さが MAX 以下であることを調べる
       try {
           if (str.length() > max_length) {
               //MAX より長い時はエラー
               return false;
           }
       } catch (NullPointerException e) {
           //バグ
           return false;
       }

       return true;
    }

   // 入力値チェック
    public static boolean validateData(String idno, String name, String info) {
       if (!validateNo(idno)) {
           return false;
        }
       if (!validateLength(name, CommonData.TEXT_DATA_LENTGH_MAX)) {
           return false;
        }
       if (!validateLength(info, CommonData.TEXT_DATA_LENTGH_MAX)) {
           return false;
        }
       return true;
    }
}
```


4.8.2. ルールブック

SQLite を使用する際には以下のルールを守ること。

1. DB ファイルの配置場所、アクセス権限を正しく設定する	（必須）
2. DB 操作時に可変パラメータを扱う場合はプレースホルダを使用する	（必須）
3. 他アプリと DB データを共有する場合は Content Provider でアクセス制御する	（必須）

4.8.2.1. DB ファイルの配置場所、アクセス権限を正しく設定する （必須）

DB ファイルのデータの保護を考えた場合、DB ファイルの配置場所とアクセス権限の設定は合わせて考慮すべき重要な要素である。

例えば、ファイルのアクセス権を正しく設定したつもりでも、SD カードなどアクセス権の設定を行えない場所に配置している場合には、誰からでもアクセス可能な DB ファイルになってしまう。また、アプリケーションフォルダに配置した場合でも、アクセス権限を正しく設定しないと意図しないアクセスを許してしまうことになる。ここでは、配置場所とアクセス権限設定について守るべき点を挙げた後、それを実現するための方法について説明する。

まず配置場所とアクセス権限設定については、BD ファイル(データ)を保護する観点から考えると、以下の 2 点を実施する必要がある。

1. （配置場所）Context#getDatabasePath(String name)で取得できるファイルパスや場合によっては Context#getFilesDir() で取得できるフォルダの場所に配置する[2]
2. （アクセス権限）MODE_PRIVATE（＝ファイルを作成したアプリのみがアクセス可能）モードに設定する

この 2 点を実施することで、他のアプリからアクセスできない DB ファイルの作成を行うことができる。

次に、これらを実施するための方法である。SQLite の DB ファイルを作成する手段としては次のものがあり、それぞれ正しく使うことで上記を実現することができる。

1. SQLiteOpenHelper を使用する
2. SQLiteDatabase#openOrCreateDatabase 関数を使用する
3. Context#openOrCreateDatabase 関数を使用する

方法に関して特徴があるので 1 つずつ説明を加えていく。

SQLiteOpenHelper を使用する

[2] どちらの関数も該当するアプリだけが読み書き権限を与えられ、他のアプリからはアクセスができないフォルダ（パッケージフォルダ）のサブフォルダ以下のパスが取得できる。

SQLiteOpenHelper を使用する場合、開発者はあまり多くのことを考えなくてもよい。SQLiteOpenHelper を派生したクラスを作成し、コンストラクタの引数に BD の名前(ファイル名に使われる)[3]を指定すれば、自動的に上記のセキュリティ要件を満たす DB ファイルを作成してくれる。

SampleDbOpenHelper.java

```
//ポイント１：データベースを構築するには SQLiteOpenHelper を使用する
public class SampleDbOpenHelper extends SQLiteOpenHelper {
    // データベースファイル名
    private static final String DBFILE_NAME = "Sample.db";
    // データベースのバージョン
    private static final int DB_VERSION = 3;

    //Context を覚えておく
    private Context mContext;

    //テーブル作成コマンド
    private static final String CREATE_TABLE_COMMANDS
            = "create table SampleTable ("
            + "_id integer primary key autoincrement, "
            + "idno integer unique not null, "
            + "name text not null, "
            + "info text"
            + ");";

    public SampleDbOpenHelper(Context context) {
        super(context, DBFILE_NAME, null, DB_VERSION);
        mContext = context;
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //ポイント２：DB 初回アクセス時に onCreate が実行される
        try {
            db.execSQL(CREATE_TABLE_COMMANDS);  //DB 構築コマンドの実行
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(this.getClass().toString(), mContext.getString(R.string.DATABASE_CREATE_ERROR_MESSAGE));
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
        // TODO データベースのバージョンアップ時に実行される、データ移行などの処理を記述する
    }
}
```

MainActivity.java

```
public class MainActivity extends Activity {
    SQLiteDatabase      mSampleDb;          //取り扱うデータを格納するデータベース
    SampleDbOpenHelper  mSampleDbOpenHelper; //ポイント１：データベースを構築するには SQLiteOpenHelper を使用する

    ・・・ 省略 ・・・
```

[3] (ドキュメントに記述はないが)SQLiteOpenHelper の実装では DB の名前にはファイルのフルパスを指定できるので、SD カードなどアクセス権限の設定できない場所のパスが意図せず入力されないように注意が必要である。


```
    /** Activity 構築時の処理 */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //ポイント１：データベースを構築するには SQLiteOpenHelper を使用する
        mSampleDbOpenHelper = new SampleDbOpenHelper(this);
        try {
            //ポイント２：DB 初回アクセス時に onCreate が実行される
            mSampleDb  = mSampleDbOpenHelper.getReadableDatabase(); //この段階でオープンされる
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(this.getClass().toString(), getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            return;
        }

        //ボタンに対するリスナの設定
        Button okButton = (Button)findViewById(R.id.ok_button);
        okButton.setOnClickListener(new OnClickListener(){

            @Override
            public void onClick(View v) {
                //アプリケーションを終了する
                finish();
            }
        });

        //DB が作成された場所を画面に表示する
        TextView infoView = (TextView)findViewById(R.id.info_view);
        String infoText = getString(R.string.info_prefix)
                        + mSampleDb.getPath()
                        + getString(R.string.info_suffix);
        infoView.setText(infoText);
    }

     ・・・ 省略 ・・・

}
```

SQLiteDatabase#openOrCreateDatabase を使用する

SQLiteDatabase#openOrCreateDatabase 関数を使用して DB の作成を行う場合、開発者は DB ファイルの指定をフルパスで行う必要がある。そのため、前述の(配置場所)の要件を満たすようにファイルパスを設定するべきである。

アクセス権限に関しては、MODE_PRIVATE と同等以上に制限されるため、ファイルの配置が正しければアプリ開発者に考慮すべき点はない。

SQLiteDatabase#openOrCreateDatabase() の使用例

```
SQLiteDatabase db;
try {
    // "/data/data/<package>/databases/Sample.db" にファイルを作成する
    db = SQLiteDatabase.openOrCreateDatabase(getDatabasePath("Sample.db", null);
} catch (SQLException e) {
```

```
    //データベースオープンに失敗した場合ログ出力
    Log.e(this.getClass().toString(), getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
    return;
}
```

Context#openOrCreateDatabase を使用する

SQLiteDatabase#openOrCreateDatabase()関数を使用して DB の作成を行う場合、ファイルのアクセス権限をオプションで指定する必要があり、明示的に MODE_PRIVATE を指定する。

ファイルの配置に関しては、DB 名（ファイル名に使用される）の指定を SQLiteOpenHelper と同様に行えるので、自動的に前述のセキュリティ要件を満たすファイルパスにファイルが作成される。ただし、フルパスも指定できるので SD カードなどを指定した場合、MODE_PRIVATE を指定しても他アプリからアクセス可能になってしまうので注意が必要である。

DB に対して明示的にアクセス許可設定を行う例：MainActivity.java

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //データベースの構築
        try {
           //MODE_PRIVATE を設定して DB を作成
           db = Context.openOrCreateDatabase("Sample.db",
                                                MODE_PRIVATE, null);
        } catch (SQLException e) {
            //データベース構築に失敗した場合ログ出力
            Log.e(this.getClass().toString(), getString(R.string.DATABASE_OPEN_ERROR_MESSAGE));
            return;
        }
        //省略 その他の初期化処理
    }
```

ちなみに、アクセス権限の設定は MODE_PRIVATE と合わせて以下の 3 種類があり、MODE_WORLD_READABLE と MODE_WORLD_WRITABLE は OR 演算で同時指定することもできる。MODE_PRIVATE 以外を使う場合はアプリの要件に照らし合わせて注意深く慎重に検討することをお勧めする。

- MODE_PRIVATE　　作成アプリのみ読み書き可能
- MODE_WORLD_READABLE　　作成アプリは読み書き可能、他は読み込みのみ
- MODE_WORLD_WRITABLE　　作成アプリは読み書き可能、他は書き込みのみ

4.8.2.2. 他アプリと DB データを共有する場合は Content Provider でアクセス制御する （必須）

他のアプリと DB データを共有する手段として、DB ファイルを WORLD_READABLE、WORLD_WRITABLE として作成し、他のアプリから直接アクセスできるようにするという方法がある。しかし、この方法では DB にアクセスするアプリや

DB への操作を制限できないため、意図しない相手（アプリ）にデータを読み書きされることもある。結果として、データの機密性や整合性に問題が生じたり、マルウェアの攻撃対象となったりする可能性も考えられる。

以上のことから、Android において DB データを他のアプリと共有する場合は、Content Provider を使うことを強くお勧めする。Content Provider を使うことにより、DB に対するアクセス制御を実現できるというセキュリティの観点からのメリットだけでなく、DB スキーマ構造を Content Provider 内に隠ぺいできるといった設計観点のメリットもある。

4.8.2.3. DB 操作時に可変パラメータを扱う場合はプレースホルダを使用する （必須）

SQL インジェクションを防ぐという意味で、任意の入力値を SQL 文に組み込む時はプレースホルダを使用するべきである。プレースホルダを使用した SQL の実行方法としては以下の 2 つの方法を挙げることができる。

1. SQLiteDatabase#compileStatement() を使用して SQLiteStatement を取得する。その後、SQLiteStatement#bindString()、bindLong()などを使用してパラメータをプレースホルダに配置する
2. SQLiteDatabese クラスの execSQL()、insert()、update()、delete()、query()、rawQuery()、replace()などを呼び出す際にプレースホルダを持った SQL 文を使用する

なお、SQLiteDatabase#compileStatement()を使用して、SELECT コマンドを実行する場合、
「SELECT コマンドの結果として先頭の 1 要素(1 行 1 列目)しか取得できない」
という制限があるので用途が限られる。

どちらの方式を使う場合でも、プレースホルダに与えるデータの内容は事前にアプリケーション要件に従ってチェックされていることが望ましい。

以下で、それぞれの方法について説明する。

SQLiteDatabase#compileStatement()を使用する場合:

いわゆるプリペアードステートメントである。以下の手順でプレースホルダへデータを渡す。

1. SQLiteDatabase#compileStatement()を使用してプレースホルダを含んだ SQL 文を SQLiteStatement として取得する。
2. 作成した SQLiteStatement オブジェクトに対して、bindLong()bindString()などのメソッドを使用してプレースホルダに設定する。
3. SQLiteStatement オブジェクトの execute()などのメソッドによって SQL を実行する。

プリペアードステートメント使用例：DataInsertTask.java（抜粋）

```
//データ追加タスク
public class DataInsertTask extends AsyncTask<String, Void, Void> {
    private MainActivity    mActivity;
    private SQLiteDatabase  mSampleDB;
```

```
    private ProgressDialog  mProgressDialog;

    public DataInsertTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Void doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];

        //データ追加処理
        //プレースホルダを使用する
        String commandString = "insert into SampleTable (idno, name, info) values (?, ?, ?)";
        SQLiteStatement sqlStmt = mSampleDB.compileStatement(commandString);
        sqlStmt.bindString(1, idno);
        sqlStmt.bindString(2, name);
        sqlStmt.bindString(3, info);
        try {
            sqlStmt.executeInsert();
        } catch (SQLException e) {
            Toast.makeText(mActivity, R.string.UPDATING_ERROR_MESSAGE, Toast.LENGTH_LONG);
        }
        return null;
    }

    ・・・ 省略 ・・・
}
```

あらかじめ実行する SQL 文をオブジェクトとして作成しておきパラメータを当てはめる形である。実行する処理が確定しているので、SQL インジェクションが発生する余地はない。また、SQLiteStatement オブジェクトを再利用することで処理効率を高めることができるというメリットもある。

SQLiteDatabase が提供する各処理用のメソッドを使用する場合:

各処理用に提供されているメソッドを使用する場合は、以下の手順でデータを渡す。

1. プレースホルダを含んだ SQL 文を用意する。
2. プレースホルダに割り当てるデータを ContentValues に登録する。
3. 各処理用のメソッド(ここでは insert())に SQL 文と ContentValues を渡して実行する。

各処理用メソッドを使用する例

```
    private void addUserData(String idno, String name, String info) {
        String commandString = "insert into SampleTable (idno, name, info) values (?, ?, ?)";

       //値の妥当性（型、範囲）チェック、エスケープ処理
       if (!validateInsertData(idno, name, info)) {
           //バリデーションを通過しなかった場合、ログ出力
           Log.e(this.getClass().toString(), getString(R.string.VALIDATION_ERROR_MESSAGE));
           return
       }
```



```
        //インサートするデータの準備
        ContentValues insertValues = new ContentValues();
        insertValues.put("idno", idno);
        insertValues.put("name", name);
        insertValues.put("info", info);

        //Insert 実行
        try {
            mSampleDb.insert("SampleTable", null, insertValues);
        } catch (SQLException e) {
            Log.e(this.getClass().toString(), getString(R.string.DB_INSERT_ERROR_MESSAGE));
            return;
        }
    }
```

この例では、SQL コマンドを直接記述せず、SQLiteDatabase が提供するインサート処理用のメソッドを使用している。SQL コマンドを直接使用しないため、この方法も SQL インジェクションの余地はないと言える。なお、同種のメソッドとして execSQL()、update()、delete()、query()、rawQuery()、replace()があり、同じように使用することができる。

4.8.3. アドバンスト

4.8.3.1. SQL 文の LIKE 述語でワイルドカードを使用する際にエスケープ処理を施す

LIKE 述語のワイルドカード(%、_)を含む文字列をプレースホルダの入力値として使用した場合、そのままだとワイルドカードとして機能するため、必要に応じて事前にエスケープ処理を施す必要がある。必要なケースとしてはワイルドカードを単体の文字("%"や"_")として扱いたい場合が当てはまる。

実際にエスケープ処理は、以下のサンプルコードのように ESCAPE 句を使用して行うことができる。

LIKE を利用した場合のエスケープ処理の例

```
//データ検索タスク
public class DataSearchTask extends AsyncTask<String, Void, Cursor> {
    private MainActivity        mActivity;
    private SQLiteDatabase      mSampleDB;
    private ProgressDialog      mProgressDialog;

    public DataSearchTask(SQLiteDatabase db, MainActivity activity) {
        mSampleDB = db;
        mActivity = activity;
    }

    @Override
    protected Cursor doInBackground(String... params) {
        String  idno = params[0];
        String  name = params[1];
        String  info = params[2];
        String  cols[]  =   {"_id", "idno","name","info"};

        Cursor cur;

    ・・・ 省略 ・・・

        //info を条件にして like 検索 (部分一致)
        //ポイント：ワイルドカードに相当する文字はエスケープ処理する
        String argString = info.replaceAll("@", "@@"); //入力として受け取った info 内の$をエスケープ
        argString = argString.replaceAll("%", "@%"); //入力として受け取った info 内の%をエスケープ
        argString = argString.replaceAll("_", "@_"); //入力として受け取った info 内の_をエスケープ
        String selectionArgs[] = {argString};

        try {
            //ポイント：プレースホルダを使用する
            cur = mSampleDB.query("SampleTable", cols, "info LIKE '%' || ? || '%' ESCAPE '@'",
                                  selectionArgs, null, null, null);
        } catch (SQLException e) {
            Toast.makeText(mActivity, R.string.SERCHING_ERROR_MESSAGE, Toast.LENGTH_LONG).show();
            return null;
        }
        return cur;
    }

    @Override
    protected void onPostExecute(Cursor resultCur) {
        mProgressDialog.dismiss();
        mActivity.updateCursor(resultCur);
```



```
    }
}
```

4.8.3.2. プレースホルダを使用できない SQL コマンドに対して外部入力を使う

テーブルの作成や削除などの DB オブジェクトを処理対象とした SQL 文を実行する場合、テーブル名などの値に対してプレースホルダを使うことはできない。基本的には、プレースホルダの使用できない値に対して、外部から入力された任意の文字列を使用するようなデータベースの設計はすべきでない。

仕様や機能上の制限でプレースホルダを使用できない場合は、入力値に危険が無いかどうか実行前に確認し、必要な処理を施すことが必須となる。
基本的には、

1. 文字列パラメータとして使用する場合、文字のエスケープやクォート処理を施す
2. 数値パラメータとして使用する場合、数字以外の文字が混入していないことを確認する
3. 識別子、コマンドとして使用する場合、1. に加え、使用できない文字が含まれていないことを確認する

を実施する。

参照:
http://www.ipa.go.jp/security/vuln/documents/website_security_sql.pdf

4.8.3.3. 不用意にデータベースの書き換えが行われないための対策を行う

SQLiteOpenHelper#getReadableDatabase、getWritableDatabase を使用して DB のインスタンスを取得した場合、どちらのメソッドを利用しても DB は読み書き可能な状態でオープンされる[4]。また、Context#openOrCreateDatabase、SQLiteDatabase#openOrCreateDatabase なども同様である。

これは、アプリ操作や実装の不具合により意図せず DB の中身を書き換えてしまう(書き換えられてしまう)可能性を意味している。基本的にはアプリの仕様と実装の範囲で対応できると考えられるが、アプリの検索機能など、読み取りしか必要のない機能を実装する場合は、データベースを読み取り専用でオープンすることで、設計や検証の簡素化ひいてはアプリ品質の向上に繋がる場合があるので、状況に応じて検討をお勧めする。

具体的には、SQLiteDatabase#openDatabase に OPEN_READONLY を指定してデータベースをオープンする。

読み取り専用でデータベースをオープンする

```
    ・・・ 省略 ・・・
```

[4] getReableDatabase は基本的には getWritableDatabase で取得するのと同じオブジェクトを返す。ディスクフルなどの状況で書き込み可能オブジェクトを生成できない場合にリードオンリーのオブジェクトを返すという仕様である(getWritableDatabase はディスクフルなどの状況では実行エラーとなる)。

```
// データベースのオープン(データベースは作成済みとする)
SQLiteDatabase db
        = SQLiteDatabase.openDatabase(SQLiteDatabase.getDatabasePath("Sample.db"), null, OPEN_READONLY);
```

参照:http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html - getReadableDatabase()

4.8.3.4. アプリケーションの要件に従って DB の入出力データの妥当性をチェックする

SQLite は型に寛容なデータベースであり、DB 上で Integer として宣言されているカラムに対して文字型のデータを格納することが可能である。DB 内のデータは、数値型を含む全てのデータが平文の文字データとして DB 内に格納されている。このため、Integer 型のカラムに対して文字列型の検索(LIKE '%123%' など)を行うことも可能である。また、VARCHAR(100) のようにデータの最大長を記述してもそれ以上の長さのデータが入力可能であるなど、SQLite での値の制限(正当性確認)は期待できない。

このため、SQLite を使用するアプリケーションは、このような DB の特性に注意して予期せぬデータを DB に格納したり取得したりしないようにアプリケーションの要件にしたがって対処する必要がある。対処の方法としては次の 2 つがある。

1. データをデータベースに格納する際、型や長さなどの条件が一致しているか確認する
2. データベースから値を取得した際、データが想定外の型や長さでないか確認する

以下では、例として入力値が 1 以上の数字であることを検証する示す。

例:入力データが 1 以上の数字であることを確認する (MainActivity.java より抜粋)

```
public class MainActivity extends Activity {

    ・・・ 省略 ・・・

    //追加処理
    private void addUserData(String idno, String name, String info) {
        //No のチェック
        if (!validateNo(idno, CommonData.REQUEST_NEW)) {
            return;
        }

        //データ追加処理
        DataInsertTask task = new DataInsertTask(mSampleDb, this);
        task.execute(idno, name, info);
    }

    ・・・ 省略 ・・・

    private boolean validateNo(String idno, int request) {
        if (idno == null || idno.length() == 0) {
            if (request == CommonData.REQUEST_SEARCH) {
                //検索処理の時は未指定を OK にする
                return true;
```



```
            } else {
                //検索処理以外の時は null、空文字はエラー
                Toast.makeText(this, R.string.IDNO_EMPTY_MESSAGE, Toast.LENGTH_LONG).show();
                return false;
            }
        }

        //数字であることを確認する
        try {
            // 1 以上の値
            if (!idno.matches("[1-9] [0-9]+")) {
                //数字以外の時はエラー
                Toast.makeText(this, R.string.IDNO_NOT_NUMERIC_MESSAGE, Toast.LENGTH_LONG).show();
                return false;
            }
        } catch (NullPointerException e) {
            //今回のケースではあり得ない
            return false;
        }

        return true;
    }

    ・・・ 省略 ・・・
}
```

4.8.3.5. DB に格納するデータについての考察

SQLite では、データをファイルに格納する際に以下のような実装になっている。

- 数値型を含む全てのデータが平文の文字データとして DB ファイル内に格納される
- DB に対してデータの削除を行ってもデータ自体は DB ファイルから削除されない（削除マークが付くのみ）
- データを更新した場合も DB ファイル内には更新前のデータも削除されず残っている

よって、削除された「はず」の情報が DB ファイル内に残ったままの状態になっている可能性がある。この場合でも、本文書に従って対策を施し、Android のセキュリティ機能が有効であれば、他アプリを含む第三者からデータファイルに直接アクセスされる心配はない。ただし、ルート化など Android の保護機構を迂回してファイルを抜き出される可能性を考えると、ビジネスに大きな影響を与えるデータが格納されている場合には、Android 保護機構に頼らないデータ保護も検討しなければならない。

これらの理由により、端末がルート化された場合でも守る必要があるような重要なデータは SQLite の DB にそのまま格納すべきではない。どうしても重要なデータを格納せざるを得ない場合には暗号化したデータを格納する、DB 全体を暗号化する、などの対策が必要となる。

実際に暗号化が必要な場合、暗号化に使う鍵の扱いやコードの難読化など本文書の範囲を超える課題が多いので、現時点でビジネスインパクトの大きなデータを扱うアプリの開発には専門家への相談をお勧めする。

以下には、参考としてデータベースを暗号化するライブラリを紹介しておく。

4.8.3.6. [参考]SQLite データベースを暗号化する(SQLCipher for Android)

SQLCipher は、データベースファイルの透過的な 256 ビット AES の暗号化を提供する SQLite 拡張である。現在は、オープンソース(BSD ライセンス)化され、Zetetic LLC によって維持・管理されている。モバイルの世界では、SQLCipher は、ノキア/QT 、アップルの iOS で広く使用されている。

SQLCipher for Android プロジェクトは、Android 環境における SQLite データベースの標準の統合化された暗号化をサポートすることを目的としている。標準の SQLite の API を SQLCipher 用に作成することで、開発者は通常と同じコーディングで暗号化されたデータベースを利用できるようになっている。

参照:https://guardianproject.info/code/sqlcipher/

使い方

アプリ開発者は以下の3つの作業をすることで SQLCipher の利用が可能になる。

1. アプリケーションの lib ディレクトリに sqlcipher.jar および、libdatabase_sqlcipher.so 、libsqlcipher_android.so、libstlport_shared.so を配置する。
2. 全てのソースファイルについて、import で指定されている android.database.sqlite.* を全て info.guardianproject.database.sqlite.* に変更する。なお、android.database.Cursor はそのまま使用可能である。
3. onCreate()の中でデータベースを初期化し、データベースをオープンする際にパスワードを設定する。

簡単なコード例

```
SQLiteDatabase.loadLibs(this);                  //まず ライブラリを Context を使用して初期化する
SQLiteOpenHelper.getWritableDatabase(passwoed);  //引数はパスワード（String 型 セキュアに取得したものと仮定）
```

SQLCipher for Android は執筆時点でバージョン 1.1.0 であり、2.0.0 版が開発進行中で RC4 が公開されている状況である。Android における使用実績や API の安定性という点で今後検証が必要となるが、現時点で Android で利用可能な SQLite の暗号化ソリューションとして検討する余地はある。

ライブラリ構成

SQLCipher を使用するためには SDK として含まれている以下のファイルが必要となる。

- assets/icudt46l.zip 2,252KB
  端末の /system/usr/icu/ 以下に icudt46l.dat が存在しない場合に必要となる。
  icudt46l.dat が見つからない場合、この zip が解凍されて使用される。
- libs/armeabi/libdatabase_sqlcipher.so 44KB

- libs/armeabi/libsqlcipher_android.so 1,117KB
- libs/armeabi/libstlport_shared.so 555KB

  Native ライブラリ。
  SQLCipher の初期ロード時（SQLiteDatabase#loadLibs()呼び出し時）に読み込まれる。

- libs/commons-codec.jar 46KB
- libs/guava-r09.jar 1,116KB
- libs/sqlcipher.jar 102KB

  Native ライブラリを呼び出す Java ライブラリ。
  sqlcipher.jar がメイン。あとは sqlcipher.jar から参照されている。

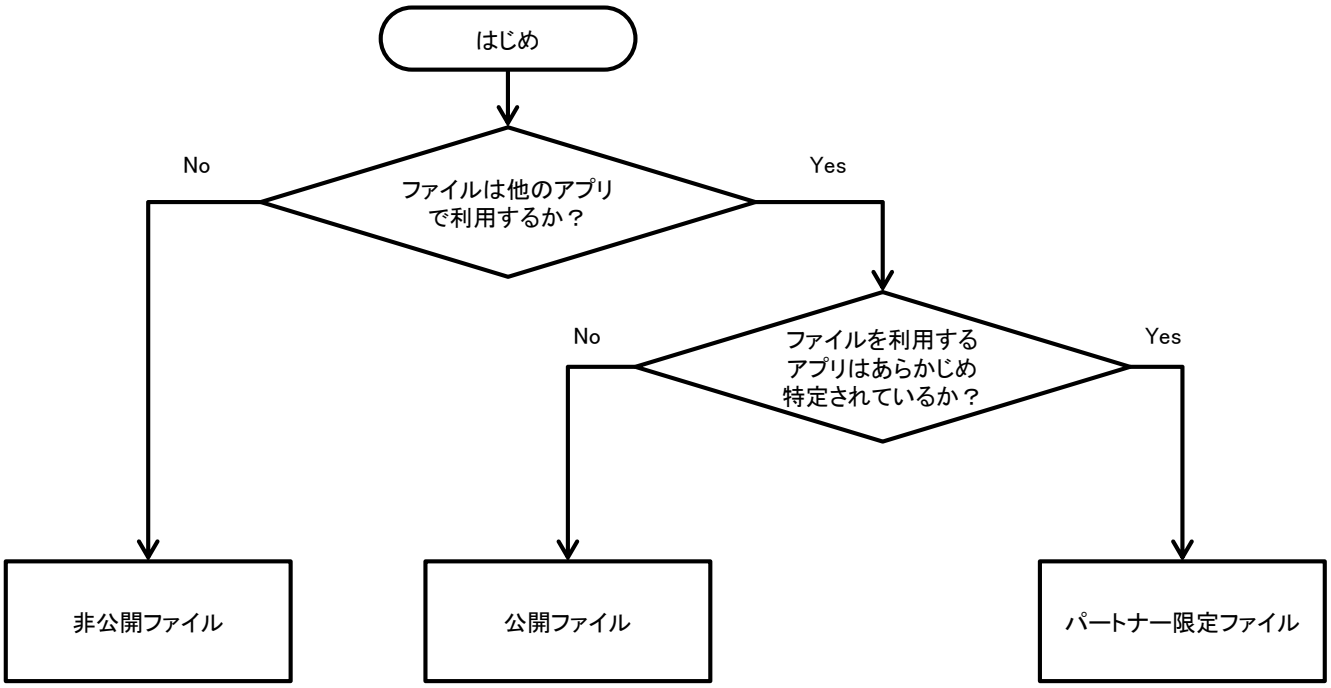
合計：約 5.12MB
ただし、icudt46l.zip は解凍されると 7MB 程度になる。

4.9. ファイルを扱う

ファイルを扱う際、セキュリティ上考慮すべきポイントについて述べる。
考慮すべきポイントは、ファイルがどのような場面で使用されるかによって異なるので、まず自分が使用するファイルのタイプを把握する事が肝要である。

4.9.1. サンプルコード

ファイルのタイプは以下のフローで判定できる。



4.9.1.1. 非公開ファイルを扱う

同一アプリ内だけで使用されるファイルである。次のポイントに気を付けて実装すること。

ポイント:

1. 他のアプリが利用できないようにプライベートモードでファイルを作る
2. SD カード等の外部記憶装置を使用してはならない
3. 悪意のあるデータを扱う可能性は低いが、ファイルの読み書き時に適切なチェックを実施する

UsingType1Activity.java

```
package org.jssec.android.file.using.type1;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class UsingType1Activity extends Activity {

    // ファイル名
    private static final String FILE_NAME = "sample_type1.txt";

    // ファイル書き込み内容を入力する EditText
    private EditText mWriteEdit;

    // ファイル読み込み内容を表示する TextView
    private TextView mReadText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.using_type_1_activity);

        // View を取得
        mWriteEdit = (EditText) findViewById(R.id.write_edit);
        mReadText = (TextView) findViewById(R.id.read_text);
    }

    /**
     * 書き込みボタンの処理
     *
     * @param view
     */
    public void onClickWriteButton(View view) {

        boolean isError = false;
```

```
        try {
            // ファイルを開く
            // ★ポイント 1★ アプリ内限定ファイルは MODE_PRIVATE を指定する
            // ★ポイント 2★ 内部記憶装置にファイルを保存する
            //  (外部記憶装置にファイルを保存するとアクセス制限が設定できない)
            FileOutputStream fos = openFileOutput(FILE_NAME, MODE_PRIVATE);
            PrintWriter pw = new PrintWriter(fos);

            // ★ポイント 3★ 書き込む値のチェック
            // ユーザー入力値を使用する場合は悪意のあるデータの可能性がある
            if (!validateText(mWriteEdit.getText().toString())) {
                // チェックエラーの場合
                // Toast 表示して終了する例
                Toast.makeText(this, "validate error.", Toast.LENGTH_LONG).show();
                return;
            }

            // ファイルに書き込む
            // EditText で入力された内容を書き込む例
            pw.println(mWriteEdit.getText().toString());

            pw.close();
        } catch (FileNotFoundException e) {
            // エラー時の処理
            // Toast 表示する例
            isError = true;
        }

        // 処理完了メッセージを Toast 表示する
        String text = "write completed.";
        if (isError) {
            // 異常終了
            text = "write error.";
        }
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }

    /**
     * 読み込みボタンの処理
     *
     * @param view
     */
    public void onClickReadButton(View view) {

        boolean isError = false;

        try {
            // ファイルを開く
            FileInputStream fis = openFileInput(FILE_NAME);
            BufferedReader br = new BufferedReader(new InputStreamReader(fis));

            // ファイルを読み込む
            StringBuffer buff = new StringBuffer();
            String line = null;
            while ((line = br.readLine()) != null) {
                buff.append(line).append("¥n");
            }
            br.close();
```



```
            // ★ポイント 3★ 悪意のあるデータを読み取る可能性は低いが、データの安全を確認すること
            if (!validateText(buff.toString())) {
                // チェックエラーの場合
                // Toast 表示して終了する例
                Toast.makeText(this, "validate error.", Toast.LENGTH_LONG).show();
                return;
            }

            // 読み込んだ内容を表示する
            mReadText.setText(buff);
        } catch (FileNotFoundException e) {
            // エラー時の処理
            // Toast 表示する例
            isError = true;
        } catch (IOException e) {
            // エラー時の処理
            // Toast 表示する例
            isError = true;
        }

        // 処理完了メッセージを Toast 表示する
        String text = "read completed.";
        if (isError) {
            // 異常終了
            text = "read error.";
        }
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }

    // TODO 以下は引数のチェックメソッド定義
    // チェックメソッド本体は、セキュアコーディングスタンダードに則って実装する

    private boolean validateText(String text) {
        // TODO 値のチェックを実装する
        return true;
    }
}
```

4.9.1.2. 公開ファイルを扱う

不特定多数のアプリがアクセスすることができるファイルである。ファイルの保存先を SD カード等の外部記憶装置にした場合はアクセス制限ができないため、必然的にこのタイプとなる。次のポイントに気を付けて実装すること。

ポイント:

1. センシティブな情報は記録しない(書き込む必要がある場合は暗号化する)
2. 可能な限り読み取り専用、または書き込み専用の指定をしてファイルを作成する
3. ファイルの読み書き時にデータが安全であることを確認する

内部記憶装置に記録するサンプルコード

UsingType2InternalActivity.java

```
package org.jssec.android.file.using.type2;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class UsingType2InternalActivity extends Activity {

    // ファイル名
    private static final String FILE_NAME = "sample_type2.txt";

    // ファイル書き込み内容を入力する EditText
    private EditText mWriteEdit;

    // ファイル読み込み内容を表示する TextView
    private TextView mReadText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.using_type_2_internal_activity);

        // View を取得
        mWriteEdit = (EditText) findViewById(R.id.write_edit);
        mReadText = (TextView) findViewById(R.id.read_text);
    }

    /**
     * 書き込みボタンの処理
     *
     * @param view
     */
```



```
    public void onClickWriteButton(View view) {

        boolean isError = false;

        try {
            // ファイルを開く
            // ★ポイント 2★ 公開ファイルは MODE_WORLD_READABLE や MODE_WORLD_WRITEABLE を指定して作成
            // 不要なアクセス許可は設定しないこと
            // 内部記憶装置または外部記憶装置にファイルを保存する
            // MODE_WORLD_READABLE を設定し、内部記憶装置に保存する例
            FileOutputStream fos = openFileOutput(FILE_NAME,
                    MODE_WORLD_READABLE);
            PrintWriter pw = new PrintWriter(fos);

            // ★ポイント 3★ 書き込む値のチェック
            // ユーザー入力値を使用する場合は悪意のあるデータの可能性がある
            if (!validateText(mWriteEdit.getText().toString())) {
                // チェックエラーの場合
                // Toast 表示して終了する例
                Toast.makeText(this, "validate error.", Toast.LENGTH_LONG).show();
                return;
            }

            // ファイルに書き込む
            // ★ポイント 1★ センシティブな情報は記録しない（書き込む必要がある場合は暗号化する）
            // EditText で入力された内容を書き込む例
            pw.println(mWriteEdit.getText().toString());

            pw.close();
        } catch (FileNotFoundException e) {
            // エラー時の処理
            // Toast 表示する例
            isError = true;
        }

        // 処理完了メッセージを Toast 表示する
        String text = "write completed.";
        if (isError) {
            // 異常終了
            text = "write error.";
        }
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }

    /**
     * 読み込みボタンの処理
     *
     * @param view
     */
    public void onClickReadButton(View view) {

        boolean isError = false;

        try {
            // ファイルを開く
            FileInputStream fis = openFileInput(FILE_NAME);
            BufferedReader br = new BufferedReader(new InputStreamReader(fis));

            // ファイルを読み込む
            StringBuffer buff = new StringBuffer();
```

```
            String line = null;
            while ((line = br.readLine()) != null) {
                buff.append(line).append("¥n");
            }
            br.close();

            // ★ポイント 3★ データの安全を確認すること
            if (!validateText(buff.toString())) {
                // チェックエラーの場合
                // Toast 表示して終了する例
                Toast.makeText(this, "validate error.", Toast.LENGTH_LONG).show();
                return;
            }

            // 読み込んだ内容を表示する
            mReadText.setText(buff);
        } catch (FileNotFoundException e) {
            // エラー時の処理
            // Toast 表示する例
            isError = true;
        } catch (IOException e) {
            // エラー時の処理
            // Toast 表示する例
            isError = true;
        }

        // 処理完了メッセージを Toast 表示する
        String text = "read completed.";
        if (isError) {
            // 異常終了
            text = "read error.";
        }
        Toast.makeText(this, text, Toast.LENGTH_LONG).show();
    }

    // TODO 以下は引数のチェックメソッド定義
    // チェックメソッド本体は、セキュアコーディングスタンダードに則って実装する

    private boolean validateText(String text) {
        // TODO 値のチェックを実装する
        return true;
    }
}
```

外部記憶装置に記録するサンプルコード

UsingType2ExternalActivity.java

```
package org.jssec.android.file.using.type2;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;

import android.app.Activity;
```



```
import android.os.Bundle;
import android.os.Environment;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class UsingType2ExternalActivity extends Activity {

    // ディレクトリパス
    private static final String DIR_PATH = "/org/jssec/android";

    // ファイル名
    private static final String FILE_NAME = "sample.txt";

    // ファイル書き込み内容を入力する EditText
    private EditText mWriteEdit;

    // ファイル読み込み内容を表示する TextView
    private TextView mReadText;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.using_type_2_external_activity);

        // View を取得
        mWriteEdit = (EditText) findViewById(R.id.write_edit);
        mReadText = (TextView) findViewById(R.id.read_text);
    }

    /**
     * 書き込みボタンの処理
     *
     * @param view
     */
    public void onClickWriteButton(View view) {

        boolean isError = false;

        try {
            // 外部記憶装置のステータスを取得する
            String state = Environment.getExternalStorageState();

            // 外部記憶装置が利用可能か
            if (!Environment.MEDIA_MOUNTED.equals(state)) {
                // 利用不可の場合
                // Toast 表示して終了する例
                Toast.makeText(this, "no mounted external storage.",
                        Toast.LENGTH_LONG).show();
                return;
            }

            // 外部記憶装置のルートディレクトリを取得する
            File rootDir = Environment.getExternalStorageDirectory();

            // ディレクトリが無い場合は作成する
            String dirPath = rootDir.getPath() + DIR_PATH;
            File dir = new File(dirPath);
            if (!dir.exists()) {
```

```
            if (!dir.mkdirs()) {
                // ディレクトリを作成できなかった場合
                // Toast 表示して終了する例
                Toast.makeText(this, "make dir error.",
                        Toast.LENGTH_LONG).show();
                return;
            }
        }

        // ファイルを開く
        // すべてのアプリからアクセス可能なファイルとなる
        File file = new File(dir, FILE_NAME);
        FileOutputStream fos = new FileOutputStream(file);
        PrintWriter pw = new PrintWriter(fos);

        // ★ポイント 2★ 書き込む値のチェック
        // ユーザー入力値を使用する場合は悪意のあるデータの可能性がある
        if (!validateText(mWriteEdit.getText().toString())) {
            // チェックエラーの場合
            // Toast 表示して終了する例
            Toast.makeText(this, "validate error.",
                    Toast.LENGTH_LONG).show();
            return;
        }

        // ファイルに書き込む
        // ★ポイント 1★ センシティブな情報は記録しない（書き込む必要がある場合は暗号化する）
        // EditText で入力された内容を書き込む例
        pw.println(mWriteEdit.getText().toString());

        pw.close();
    } catch (FileNotFoundException e) {
        // エラー時の処理
        // Toast 表示する例
        isError = true;
    }

    // 処理完了メッセージを Toast 表示する
    String text = "write completed.";
    if (isError) {
        // 異常終了
        text = "write error.";
    }
    Toast.makeText(this, text, Toast.LENGTH_LONG).show();
}

/**
 * 読み込みボタンの処理
 *
 * @param view
 */
public void onClickReadButton(View view) {

    boolean isError = false;

    try {
        // 外部記憶装置のステータスを取得する
        String state = Environment.getExternalStorageState();

        // 外部記憶装置が利用可能か
```



```
        if (!Environment.MEDIA_MOUNTED.equals(state)) {
            // 利用不可の場合
            // Toast 表示して終了する例
            Toast.makeText(this, "no mounted external storage.",
                    Toast.LENGTH_LONG).show();
            return;
        }

        // 外部記憶装置のルートディレクトリを取得する
        File rootDir = Environment.getExternalStorageDirectory();

        // ファイルを開く
        String dirPath = rootDir.getPath() + DIR_PATH;
        File file = new File(dirPath, FILE_NAME);
        FileInputStream fis = new FileInputStream(file);
        BufferedReader br = new BufferedReader(new InputStreamReader(fis));

        // ファイルを読み込む
        StringBuffer buff = new StringBuffer();
        String line = null;
        while ((line = br.readLine()) != null) {
            buff.append(line).append("¥n");
        }
        br.close();

        // ★ポイント 2★ データの安全を確認すること
        if (!validateText(buff.toString())) {
            // チェックエラーの場合
            // Toast 表示して終了する例
            Toast.makeText(this, "validate error.",
                    Toast.LENGTH_LONG).show();
            return;
        }

        // 読み込んだ内容を表示する
        mReadText.setText(buff);
    } catch (FileNotFoundException e) {
        // エラー時の処理
        // Toast 表示する例
        isError = true;
    } catch (IOException e) {
        // エラー時の処理
        // Toast 表示する例
        isError = true;
    }

    // 処理完了メッセージを Toast 表示する
    String text = "read completed.";
    if (isError) {
        // 異常終了
        text = "read error.";
    }
    Toast.makeText(this, text, Toast.LENGTH_LONG).show();
}

// TODO 以下は引数のチェックメソッド定義
// チェックメソッド本体は、セキュアコーディングスタンダードに則って実装する

private boolean validateText(String text) {
    // TODO 値のチェックを実装する
```

```
        return true;
    }
}
```

4.9.1.3. パートナー限定ファイルを扱う

特定のアプリにだけアクセス許可する必要があるファイルである。複数社の製品を連携させてサービスを構築するような場合に使用される。次のポイントに気を付けて実装すること。

残念ながら、特定のアプリのみがアクセス可能なファイルを作成することはできない。アクセス制限的には公開と同様になるため、センシティブな情報は記録しないことが望ましい。どうしても記録する必要がある場合は暗号化を施す必要がある。暗号化については第二版以降にて別途記事を設ける予定である。

4.9.2. ルールブック

ファイル読み書きの実装時には以下のルールを守ること。

1. ファイルの保存先を意識する	（必須）
2. ファイルへのアクセス許可をできる限り制限する	（推奨）
3. センシティブな情報はできる限り扱わない	（必須）

4.9.2.1. ファイルの保存先を意識する （必須）

ファイルの保存先（内部記憶装置か、SD カード等の外部記憶装置か）によって、アクセス制限の設定可否が変わるため、ファイルの保存先を意識することが重要となる。アクセス制限が可能なのは内部記憶装置のみとなるので、センシティブな情報を扱う場合は内部記憶装置へ保存することを推奨する（外部記憶装置へ保存したファイルはすべてのアプリからアクセスできる）。

ただし、内部記憶装置の場合、端末変更を行うとデータを引き継ぐことができない、アプリケーション情報画面で「データ削除」を選択するとファイルが削除されるといった制限がある。そのため、機能要件に応じた選択が必要となる。例えば、アプリの設定情報をエクスポート、インポートする機能がある場合、端末変更の際にそれらの機能を利用して、新しい端末にデータを引き継ぐ必要がある。そのような目的で利用するため、エクスポートした設定情報ファイルを SD カードに保存できる必要がある。

4.9.2.2. ファイルへのアクセス許可をできる限り制限する （推奨）

ファイルの保存先が内部記憶装置であればファイル作成時に他アプリのアクセス制御設定ができる。可能なアクセス制御設定は以下の 4 種類である。

1. 自アプリのみ読み書き可能（他アプリは読み書き不可）
2. 他アプリから読み取りのみ可能
3. 他アプリから書き込みのみ可能
4. 他アプリから読み書き可能

他アプリから読み取りまたは書き込みができる場合、そのファイルはマルウェアからも読み取りまたは書き込みができるということである。ファイルに付与するアクセス許可は必要最小限に定めるべきある。例えば、自アプリでデータの書き込みを行い、他アプリでは読み取りだけを行う場合は、ファイルには読み取りのみ可能のアクセス制限を設定する。

自アプリのみ読み書き可能を設定する場合

MyActivity.java

```
// MODE_PRIVATE を指定して、内部記憶装置にファイルを保存する
// 自アプリからは読み取りも書き込みもできる
// 他のアプリからは読み取りも書き込みもできない
FileOutputStream fos = openFileOutput(FILE_NAME, MODE_PRIVATE);
```


他アプリから読み取りのみ可能を設定する場合

MyActivity.java
```
// MODE_WORLD_READABLE を指定して、内部記憶装置にファイルを保存する
// 自アプリからは読み取りも書き込みもできる
// 他のアプリからは読み取りはできるが、書き込みはできない
FileOutputStream fos = openFileOutput(FILE_NAME,
        MODE_WORLD_READABLE);
```

他アプリから書き込みのみ可能を設定する場合

MyActivity.java
```
// MODE_WORLD_WRITEABLE を指定して、内部記憶装置にファイルを保存する
// 自アプリからは読み取りも書き込みもできる
// 他のアプリからは読み取りはできないが、書き込みはできる
FileOutputStream fos = openFileOutput(FILE_NAME,
        MODE_WORLD_WRITEABLE);
```

他アプリから読み書き可能を設定する場合

MyActivity.java
```
// MODE_WORLD_READABLE と MODE_WORLD_WRITEABLE を指定して、
// 内部記憶装置にファイルを保存する
// 自アプリからは読み取りも書き込みもできる
// 他のアプリからも読み取りも書き込みもできる
FileOutputStream fos = openFileOutput(FILE_NAME,
        MODE_WORLD_READABLE | MODE_WORLD_WRITEABLE);
```

4.9.2.3. センシティブな情報はできる限り扱わない （必須）

不特定多数のアプリに公開するファイルの場合にはセンシティブな情報を扱ってはならない。特定のアプリと連携する場合においても、特定のアプリのみファイルにアクセスを可能にするといったアクセス制限がかけられないため、センシティブな情報はできる限り扱わないことが重要である。どうしても扱う必要がある場合は漏洩のリスクを考慮して暗号化を施して扱うべきである。暗号化については第二版以降にて別途記事を設ける予定である。

4.9.3. アドバンスト

4.9.3.1. プリファレンスやデータベースファイルのアクセス制限について

プリファレンスやデータベースもファイルで構成されるため、アクセス制限については同じことが言える。したがって、他のファイルと同様に保存先や書き込む内容には注意が必要となる。

プリファレンスファイルにアクセス制限を設定する例

UsingTypeOtherActivity.java

```
package org.jssec.android.file.using;

import android.app.Activity;
import android.content.SharedPreferences;
import android.content.SharedPreferences.Editor;
    /**
     * プリファレンス書き込みボタンの処理
     */
    public void onClickWritePreferenceButton(View view) {

        // プリファレンスを取得する（なければ作成される）
        // ポイント：他のファイルと同様に MODE_PRIVATE、MODE_WORLD_READABLE、
        // MODE_WORLD_WRITEABLE を使用してモード指定する
        SharedPreferences preference = getSharedPreferences(
                PREFERENCE_FILE_NAME, MODE_PRIVATE);

        // key、value が"sample_key"、"abc"のデータを書き込む例
        Editor editor = preference.edit();
        editor.putString("sample_key", "abc");
        editor.commit();

        // 完了メッセージを表示する例
        Toast.makeText(getApplicationContext(), "completed.", Toast.LENGTH_LONG)
                .show();
    }
```

データベースファイルにアクセス制限を設定する例

UsingTypeOtherActivity.java

```
    /**
     * データベースを開くボタンの処理
     */
    public void onClickOpenDatabaseButton(View view) {

        // データベースを開く（なければ作成される）
        // ポイント：他のファイルと同様に MODE_PRIVATE、MODE_WORLD_READABLE、
        // MODE_WORLD_WRITEABLE を使用してモード指定する
        SQLiteDatabase database = openOrCreateDatabase(DATABASE_FILE_NAME,
                MODE_PRIVATE, null);

        // テーブル一覧を取得して Toast 表示する例
        Cursor cursor = database.query("sqlite_master",
                new String[] { "name" }, " type = ?", new String[] { "table" },
                null, null, "name asc");
        StringBuffer buff = new StringBuffer("tables not found.");
        if (cursor.moveToFirst()) {
            buff.delete(0, buff.length());
```



```
            buff.append(cursor.getString(0));
            while (cursor.moveToNext()) {
                buff.append(cursor.getString(0)).append("¥n");
            }
        }
        Toast.makeText(getApplicationContext(), buff, Toast.LENGTH_LONG).show();
        cursor.close();

        // データベースを閉じる
        database.close();
    }
```

4.9.3.2. 自動生成ディレクトリについて

アプリのデータディレクトリとして内部記憶装置には以下のディレクトリが用意される。

/data/data/<パッケージ名>

また、データディレクトリ内には以下のサブディレクトリが自動で生成される。

lib ディレクトリ

lib ディレクトリはアプリのインストール時に自動的に作成される。JNI(Java Native Interface)を利用するアプリの場合、ライブラリがディレクトリに展開される。JNI を利用しない場合は空となる。なお、アプリケーション情報画面の「データ消去」を選択しても削除されない。

files ディレクトリ

files ディレクトリは Context クラスの openFileOutput()メソッドでファイルが作成された際にディレクトリも作成される。このディレクトリは、アプリで使用するデータファイルを格納するために使う。なお、このディレクトリは、アプリケーション情報画面の「データ消去」を選択したときにファイルごと削除される。

cache ディレクトリ

cache ディレクトリは Context クラスの getCacheDir()メソッドを呼び出した際に作成される。アプリを動作させる上でキャッシュしておきたいデータファイルを格納するために使う。なお、このディレクトリは、アプリケーション情報画面の「キャッシュを消去」を選択したときにファイルごと削除される。

databases ディレクトリ

databases ディレクトリはデータベースの作成時に作成される。データベースファイルを格納するために使う。なお、このディレクトリは、アプリケーション情報画面の「データ消去」を選択したときにファイルごと削除される。

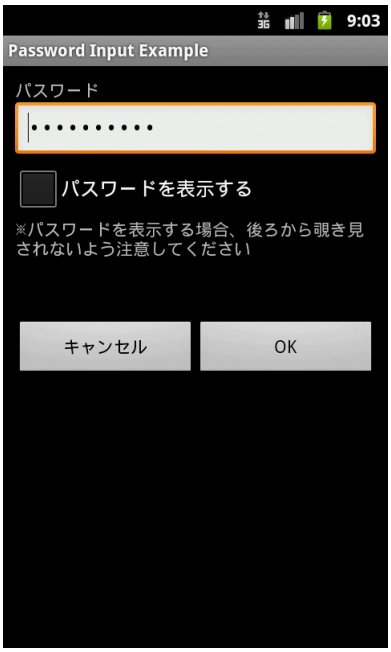
5. セキュリティ機能の使い方

暗号や電子署名、Permission など、Android にはさまざまなセキュリティ機能が用意されている。これらのセキュリティ機能は取り扱いを間違えるとセキュリティ機能が十分に発揮されず抜け道ができてしまう。この章では開発者がセキュリティ機能を活用するシーンを想定した記事を扱う。

5.1. パスワード入力画面を作る

5.1.1. サンプルコード

パスワード入力画面を作る際、セキュリティ上考慮すべきポイントについて述べる。ここではパスワードの入力に関する内容のみとする。パスワードの保存方法については第二版以降にて別途記事を設ける予定である。



ポイント:

1. 入力したパスワードはマスク表示(* で表示)する
2. パスワードを平文表示するオプションを用意する
3. パスワード平文表示時の危険性を注意喚起する

ポイント: 前回入力したパスワードを扱う場合には上記ポイントに加え、下記ポイントにも気を付けること

4. Activity 初期表示時に前回入力したパスワードがある場合、前回入力パスワードの桁数を推測されないよう固定桁数の * 文字でダミー表示する
5. 前回入力パスワードをダミー表示しているとき、「パスワードを表示」した場合、前回入力パスワードをクリアして、新規にパスワードを入力できる状態とする
6. 前回入力パスワードをダミー表示しているとき、ユーザーがパスワードを入力しようとした場合、前回入力パスワードをクリアし、ユーザーの入力を新たなパスワードとして扱う

password_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:padding="10dp" >

    <!-- パスワード項目のラベル -->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/password" />

    <!-- パスワード入力項目 -->
    <!-- ★ポイント 1★ android:password を true に設定する -->
    <EditText
        android:id="@+id/password_edit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/hint_password"
        android:password="true" />

    <!-- ★ポイント 2★ パスワード表示のオプションを用意する -->
    <CheckBox
        android:id="@+id/password_display_check"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/display_password" />

    <!-- ★ポイント 3★ パスワード表示時の危険性について注意喚起を行う-->
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/alert_password" />

    <!-- キャンセル、OK ボタン -->
    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:gravity="center"
        android:orientation="horizontal" >

        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:onClick="onClickCancelButton"
            android:text="@android:string/cancel" />

        <Button
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:onClick="onClickOkButton"
            android:text="@android:string/ok" />
    </LinearLayout>

</LinearLayout>
```


次の PasswordActivity.java の最後に配置した 3 つの関数は用途に合わせて実装内容を調整すること。

- private String getPreviousPassword()
- private void onClickCancelButton(View view)
- private void onClickOkButton(View view)

PasswordActivity.java

```
package org.jssec.android.password.passwordinputui;

import android.app.Activity;
import android.os.Bundle;
import android.text.Editable;
import android.text.InputType;
import android.text.TextWatcher;
import android.view.View;
import android.widget.CheckBox;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.EditText;
import android.widget.Toast;

public class PasswordActivity extends Activity {

    // 状態保存用のキー
    private static final String KEY_DUMMY_PASSWORD = "KEY_DUMMY_PASSWORD";

    // Activity 内の View
    private EditText mPasswordEdit;
    private CheckBox mPasswordDisplayCheck;

    // パスワードがダミー表示かを表すフラグ
    private boolean mIsDummyPassword;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.password_activity);

        // View の取得
        mPasswordEdit = (EditText) findViewById(R.id.password_edit);
        mPasswordDisplayCheck = (CheckBox) findViewById(R.id.password_display_check);

        // 前回入力パスワードがあるか
        if (getPreviousPassword() != null) {
            // ★ポイント 4★ 前回入力パスワードがある場合はダミーパスワードを表示する

            // 表示はダミーパスワードにする
            mPasswordEdit.setText("**********");
            // パスワード入力時にダミーパスワードをクリアするため、テキスト変更リスナーを設定
            mPasswordEdit.addTextChangedListener(new PasswordEditTextWatcher());
            // ダミーパスワードフラグを設定する
            mIsDummyPassword = true;
        }

        // パスワードを表示するオプションのチェック変更リスナーを設定
        mPasswordDisplayCheck
                .setOnCheckedChangeListener(new OnPasswordDisplayCheckedChangeListener());
```

```
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        // TODO 画面の縦横変更で Activity が再生成されないよう指定した場合には不要
        // Activity の状態保存
        outState.putBoolean(KEY_DUMMY_PASSWORD, mIsDummyPassword);
    }

    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        // TODO 画面の縦横変更で Activity が再生成されないよう指定した場合には不要
        // Activity の状態の復元
        mIsDummyPassword = savedInstanceState.getBoolean(KEY_DUMMY_PASSWORD);
    }

    /**
     * パスワードを入力した場合の処理
     */
    private class PasswordEditTextWatcher implements TextWatcher {

        public void beforeTextChanged(CharSequence s, int start, int count,
                int after) {
            // 未使用
        }

        public void onTextChanged(CharSequence s, int start, int before,
                int count) {
            // ★ポイント 6★ ダミー表示時にパスワードを再入力した場合は入力内容に応じた表示にする
            if (mIsDummyPassword) {
                // ダミーパスワードフラグを設定する
                mIsDummyPassword = false;
                // パスワードを入力した文字だけにする
                CharSequence work = s.subSequence(start, start + count);
                mPasswordEdit.setText(work);
                // カーソル位置が最初に戻るので最後にする
                mPasswordEdit.setSelection(work.length());
            }
        }

        public void afterTextChanged(Editable s) {
            // 未使用
        }

    }

    /**
     * パスワードの表示オプションチェックを変更した場合の処理
     */
    private class OnPasswordDisplayCheckedChangeListener implements
            OnCheckedChangeListener {

        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
            // ★ポイント 5★ ダミー表示時は空表示にする
            if (mIsDummyPassword && isChecked) {
```



```
            // ダミーパスワードフラグを設定する
            mIsDummyPassword = false;
            // パスワードを空表示にする
            mPasswordEdit.setText(null);
        }

        // カーソル位置が最初に戻るので今のカーソル位置を記憶する
        int pos = mPasswordEdit.getSelectionStart();

        // ★ポイント 2★ チェックに応じてパスワードを平文表示する
        // InputType の作成
        int type = InputType.TYPE_CLASS_TEXT;
        if (isChecked) {
            // チェック ON 時は平文表示
            type |= InputType.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD;
        } else {
            // チェック OFF 時はマスク表示
            type |= InputType.TYPE_TEXT_VARIATION_PASSWORD;
        }

        // パスワード EditText に InputType を設定
        mPasswordEdit.setInputType(type);

        // カーソル位置を設定する
        mPasswordEdit.setSelection(pos);
    }

}

// TODO 以下のメソッドはアプリに合わせて実装すること

/**
 * 前回入力パスワードを取得する
 *
 * @return 前回入力パスワード
 */
private String getPreviousPassword() {
    // 保存パスワードを復帰させたい場合にパスワード文字列を返す
    // パスワードを保存しない用途では null を返す
    return "hirake5ma";
}

/**
 * キャンセルボタンの押下処理
 *
 * @param view
 */
public void onClickCancelButton(View view) {
    // Activity を閉じる
    finish();
}

/**
 * OK ボタンの押下処理
 *
 * @param view
 */
public void onClickOkButton(View view) {
    // password を保存するとか認証に使うとか必要な処理を行う
```

```
        String password = null;

        if (mIsDummyPassword) {
            // 最後までダミーパスワード表示だった場合は前回入力パスワードを確定パスワードとする
            password = getPreviousPassword();
        } else {
            // ダミーパスワード表示じゃない場合はユーザー入力パスワードを確定パスワードとする
            password = mPasswordEdit.getText().toString();
        }

        // パスワードを Toast 表示する
        Toast.makeText(this, "password is ¥"" + password + "¥"",
                Toast.LENGTH_SHORT).show();

        // Activity を閉じる
        finish();
    }
}
```


5.1.2. ルールブック

パスワード入力画面を作る際には以下のルールを守ること。

1. パスワードを入力するときにはマスク表示(* で表示する)機能を用意する	(必須)
2. パスワードを平文表示するオプションを用意する	(必須)
3. Activity 起動時はパスワードをマスク表示にする	(必須)
4. 前回入力したパスワードを表示する場合、ダミーパスワードを表示する	(必須)

5.1.2.1. パスワードを入力するときにはマスク表示(* で表示する)機能を用意する (必須)

スマートフォンは電車やバス等の人混みで利用されることが多く、第三者にパスワードを盗み見られるリスクが大きい。パスワードをマスク表示する機能が必要である。

レイアウト XML で指定する場合

password_activity.xml

```
<!-- パスワード入力項目 -->
<!-- android:password を true に設定する -->
<EditText
    android:id="@+id/password_edit"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/hint_password"
    android:password="true" />
```

Activity 内で指定する場合

PasswordActivity.java

```
// パスワード表示タイプを設定
// InputType に TYPE_TEXT_VARIATION_PASSWORD を設定する
EditText passwordEdit = (EditText) findViewById(R.id.password_edit);
int type = InputType.TYPE_CLASS_TEXT
        | InputType.TYPE_TEXT_VARIATION_PASSWORD;
passwordEdit.setInputType(type);
```

5.1.2.2. パスワードを平文表示するオプションを用意する （必須）

スマートフォンにおけるパスワード入力はタッチパネルでの入力となるため、PC でキーボード入力する場合と比較するとパスワードの誤入力が生じやすい。マスク表示を解除する機能がない場合、ユーザーは単純なパスワードを利用してしまい、かえって危険である。背後からの覗き見への注意を促しつつ、パスワードを平文表示できるオプションを用意することで、タッチパネル入力の不便を補いつつ、安全なパスワードを利用してもらえるようになる。

EditText の InputType 指定で、マスク表示と平文表示を切り替えることができる

PasswordActivity.java

```
    /**
     * パスワードの表示オプションチェックを変更した場合の処理
     */
    private class OnPasswordDisplayCheckedChangeListener implements
            OnCheckedChangeListener {

        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
            // ★ポイント 5★ ダミー表示時は空表示にする
            if (mIsDummyPassword && isChecked) {
                // ダミーパスワードフラグを設定する
                mIsDummyPassword = false;
                // パスワードを空表示にする
                mPasswordEdit.setText(null);
            }

            // カーソル位置が最初に戻るので今のカーソル位置を記憶する
            int pos = mPasswordEdit.getSelectionStart();

            // ★ポイント 2★ チェックに応じてパスワードを平文表示する
            // InputType の作成
            int type = InputType.TYPE_CLASS_TEXT;
            if (isChecked) {
                // チェック ON 時は平文表示
                type |= InputType.TYPE_TEXT_VARIATION_VISIBLE_PASSWORD;
```



```
            } else {
                // チェック OFF 時はマスク表示
                type |= InputType.TYPE_TEXT_VARIATION_PASSWORD;
            }

            // パスワード EditText に InputType を設定
            mPasswordEdit.setInputType(type);

            // カーソル位置を設定する
            mPasswordEdit.setSelection(pos);
        }

    }
```

5.1.2.3. Activity 起動時はパスワードをマスク表示にする （必須）

意図せずパスワード表示してしまい、第三者に見られることを防ぐため、Activity 起動時にパスワードを表示するオプションのデフォルト値はオフにするべきである。デフォルト値は安全側に定めるのが基本である。

5.1.2.4. 前回入力したパスワードを表示する場合、ダミーパスワードを表示する （必須）

前回入力したパスワードを指定する場合、第三者にパスワードのヒントを与えないように、固定文字数のマスク文字(*など)でダミー表示するべきである。また、ダミー表示時に「パスワードを表示する」とした場合は、パスワードをクリアしてから平文表示モードにする。これにより、スマートフォンが盗難される等によって第三者の手に渡ったとしても前回入力したパスワードが盗み見られる危険性を低く抑えることができる。なお、ダミー表示時にユーザーがパスワードを入力しようとした場合には、ダミー表示を解除して通常の入力状態に戻す必要がある。

前回入力したパスワードを表示する場合、ダミーパスワードを表示する

PasswordActivity.java

```
    @Override
    public void onCreate(Bundle savedInstanceState) {

        ～ 省略 ～

        // 前回入力パスワードがあるか
        if (getPreviousPassword() != null) {
            // ★ポイント 4★ 前回入力パスワードがある場合はダミーパスワードを表示する

            // 表示はダミーパスワードにする
            mPasswordEdit.setText("**********");
            // パスワード入力時にダミーパスワードをクリアするため、テキスト変更リスナーを設定
            mPasswordEdit.addTextChangedListener(new PasswordEditTextWatcher());
            // ダミーパスワードフラグを設定する
            mIsDummyPassword = true;
        }

        ～ 省略 ～

    }

    /**
     * 前回入力パスワードを取得する
```

```
     *
     * @return 前回入力パスワード
     */
    private String getPreviousPassword() {
        // 保存パスワードを復帰させたい場合にパスワード文字列を返す
        // パスワードを保存しない用途では null を返す
        return "hirake5ma";
    }
```

ダミー表示時は、パスワードを表示するオプションをオンにすると表示内容をクリアする

PasswordActivity.java

```
    /**
     * パスワードの表示オプションチェックを変更した場合の処理
     */
    private class OnPasswordDisplayCheckedChangeListener implements
            OnCheckedChangeListener {

        public void onCheckedChanged(CompoundButton buttonView,
                boolean isChecked) {
            // ★ポイント 5★ ダミー表示時は空表示にする
            if (mIsDummyPassword && isChecked) {
                // ダミーパスワードフラグを設定する
                mIsDummyPassword = false;
                // パスワードを空表示にする
                mPasswordEdit.setText(null);
            }

            ～ 省略 ～

        }

    }
```

ダミー表示時にユーザーがパスワードを入力した場合には、ダミー表示を解除する

PasswordActivity.java

```
    // 状態保存用のキー
    private static final String KEY_DUMMY_PASSWORD = "KEY_DUMMY_PASSWORD";

    ～ 省略 ～

    // パスワードがダミー表示かを表すフラグ
    private boolean mIsDummyPassword;

    @Override
    public void onCreate(Bundle savedInstanceState) {

        ～ 省略 ～

        // 前回入力パスワードがあるか
        if (getPreviousPassword() != null) {
            // ★ポイント 4★ 前回入力パスワードがある場合はダミーパスワードを表示する

            // 表示はダミーパスワードにする
            mPasswordEdit.setText("**********");
            // パスワード入力時にダミーパスワードをクリアするため、テキスト変更リスナーを設定
            mPasswordEdit.addTextChangedListener(new PasswordEditTextWatcher());
            // ダミーパスワードフラグを設定する
            mIsDummyPassword = true;
```



```
        }

        ～ 省略 ～

    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);

        // TODO 画面の縦横変更で Activity が再生成されないよう指定した場合には不要
        // Activity の状態保存
        outState.putBoolean(KEY_DUMMY_PASSWORD, mIsDummyPassword);
    }

    @Override
    public void onRestoreInstanceState(Bundle savedInstanceState) {
        super.onRestoreInstanceState(savedInstanceState);

        // TODO 画面の縦横変更で Activity が再生成されないよう指定した場合には不要
        // Activity の状態の復元
        mIsDummyPassword = savedInstanceState.getBoolean(KEY_DUMMY_PASSWORD);
    }

    /**
     * パスワードを入力した場合の処理
     */
    private class PasswordEditTextWatcher implements TextWatcher {

        public void beforeTextChanged(CharSequence s, int start, int count,
                int after) {
            // 未使用
        }

        public void onTextChanged(CharSequence s, int start, int before,
                int count) {
            // ★ポイント 6★ ダミー表示時にパスワードを再入力した場合は入力内容に応じた表示にする
            if (mIsDummyPassword) {
                // ダミーパスワードフラグを設定する
                mIsDummyPassword = false;
                // パスワードを入力した文字だけにする
                CharSequence work = s.subSequence(start, start + count);
                mPasswordEdit.setText(work);
                // カーソル位置が最初に戻るので最後にする
                mPasswordEdit.setSelection(work.length());
            }
        }

        public void afterTextChanged(Editable s) {
            // 未使用
        }

    }
```

5.1.3. アドバンスト

5.1.3.1. ログイン処理について

パスワード入力が求められる場面の代表例はログイン処理である。ログイン処理で気を付けるポイントをいくつか紹介する。

ログイン失敗時のエラーメッセージ

ログイン処理では ID（アカウント）とパスワードの 2 つの情報を入力する。ログイン失敗時には ID が存在しない場合と、ID は存在するがパスワードが間違えている場合の 2 つがある。ログイン失敗のメッセージでこの 2 つの場合を区別して表示すると、攻撃者は「指定した ID が存在するか否か」を推測できてしまう。このような推測を許さないためにも、ログイン失敗時のメッセージでは、上記 2 つの場合を区別せずに下記のように表示すべきである。

メッセージ例: ログイン ID または パスワード が間違っています。

自動ログイン機能

一度、ログイン処理が成功すると次回以降はログイン ID とパスワードの入力を省略して、自動的にログインを行う機能がある。自動ログイン機能は煩わしい入力が省略できるので利便性が高まるが、その反面スマートフォンが盗難された場合に第三者に悪用されるリスクが伴う。

第三者に悪用された場合の被害が受け入れられる用途にのみ、自動ログインは利用すべきである。例えば、オンラインバンキングアプリの場合、第三者に端末を操作されると金銭的な被害が出るので自動ログイン機能をつけるのは望ましくない。利用に当たっては、利便性とリスクを勘案して、十分に検討する必要がある。

なお、パスワード変更後の初回ログイン処理では自動ログインを行わず、ユーザーのパスワード入力を求めるべきである。

5.1.3.2. パスワード変更について

一度設定したパスワードを別のパスワードに変更する場合、以下の入力項目を画面上に用意すべきである。

- 現在のパスワード
- 新しいパスワード
- 新しいパスワード（入力確認用）

自動ログイン機能がついている場合、第三者がアプリを利用できる可能性がある。その場合、勝手にパスワードを変更されないよう、現在のパスワードの入力を求める必要がある。また、新しいパスワードが入力ミスで使用不能に陥る危険を減らすため、新しいパスワードは 2 回、入力を求める必要がある。

5.1.3.3. システムの「パスワードを表示」設定メニューについて

Android の設定メニューの中に「パスワードを表示」という設定がある。Android 2.3.3 の場合は以下にある。

設定 ＞ 位置情報とセキュリティ ＞ パスワードを表示

「パスワードを表示」設定をオンにすると最後に入力した1文字が平文表示となる。一定時間（2 秒程度）経過後、または次の文字が入力されると平文表示されていた文字はマスク表示される。オフにすると、入力直後からマスク表示となる。これはシステム全体に影響する設定であり、EditText のパスワード表示機能を使用しているすべてのアプリに適用される。

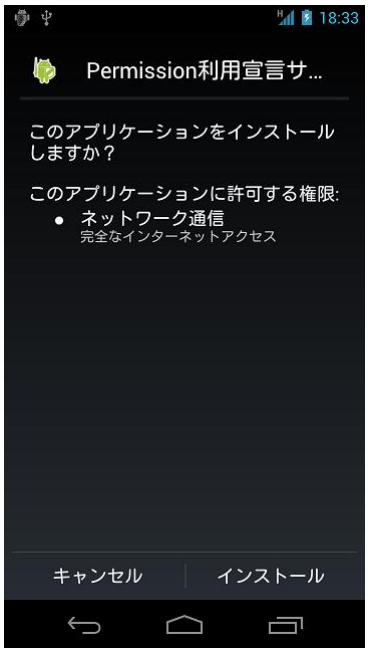
5.2. Permission と Protection Level

Permission の Protection Level には normal, dangerous, signature, signatureOrSystem の 4 種類がある。Permission はどの Protection Level であるかによってそれぞれ、Normal Permission, Dangerous Permission, Signature Permission, SignatureOrSystem Permission と呼ばれる。以下、このような名称を使う。

5.2.1. サンプルコード

5.2.1.1. Android OS 既定の Permission を利用する方法

Android OS は電話帳や GPS などのユーザー資産をマルウェアから保護するための Permission というセキュリティの仕組みがある。Android OS が保護対象としている、こうした情報や機能にアクセスするアプリは、明示的にそれらにアクセスするための権限（Permission）を利用宣言しなければならない。ユーザー確認が必要な Permission では、その Permission を利用宣言したアプリがインストールされるときに次のようなユーザー確認画面が表示される。



この確認画面により、ユーザーはそのアプリがどのような機能や情報にアクセスしようとしているのかを知ることができる。もし、アプリの動作に明らかに不必要な機能や情報にアクセスしようとしている場合は、そのアプリはマルウェアである可能性が高い。ゆえに自分のアプリがマルウェアであると疑われないためにも、利用宣言する Permission は最小限にしなければならない。

ポイント:

1. 利用する Permission を AndroidManifest.xml に uses-permission で利用宣言する
2. 不必要な Permission は利用宣言しない

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

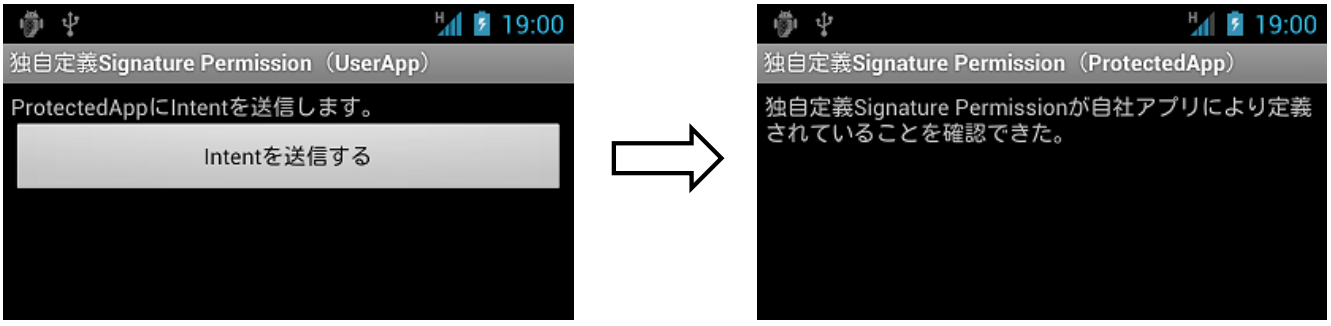


```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.usespermission"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント 1★　アプリで利用する Permission を利用宣言する -->
    <!-- インターネットにアクセスする Permission -->
    <uses-permission android:name="android.permission.INTERNET"/>

    <!-- ★ポイント 2★　不必要な Permission は利用宣言しない -->
    <!-- アプリの動作に不必要な Permission を利用宣言していると、ユーザーに不信感を与えてしまう -->

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

5.2.1.2. 独自定義の Signature Permission で自社アプリ連携する方法

Android OS が定義する既定の Permission の他に、アプリケーションが独自に Permission を定義することができる。独自定義の Permission（正確には独自定義の Signature Permission）を使えば、自社アプリだけが連携できる仕組みを作ることができる。複数の自社製アプリをインストールした場合に、それぞれのアプリの単機能に加え、アプリ間連携による複合機能を提供することで、複数の自社製アプリをシリーズ販売して収益を上げる、といった用途がある。

サンプルプログラム「独自定義 Signature Permission（UserApp）」はサンプルプログラム「独自定義 Signature Permission（ProtectedApp）」に startActivity()する。両アプリは同じ開発者鍵で署名されている必要がある。もし署名した開発者鍵が異なる場合は、UserApp は Intent を送信せず、ProtectedApp は受信した Intent を処理しない。またアドバンストセクションで説明しているインストール順序による Signature Permission 回避の問題にも対処している。



Componentを利用するアプリ　　Componentを提供するアプリ

ポイント：Component を提供するアプリ

1. 独自 Permission を protectionLevel=”signature”で定義する
2. Component には permission 属性で独自 Permission 名を指定する
3. Component が Activity の場合には intent-filter を定義しない
4. ソースコード上で、独自定義 Signature Permission が自社アプリにより定義されていることを確認する
5. Component を利用するアプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.protectedapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント 1★ 独自 Permission を protectionLevel="signature"で定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />
```



```
    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >

        <!-- ★ポイント 2★ permission 属性で独自 Permission 名を指定する -->
        <activity
            android:name=".ProtectedActivity"
            android:exported="true"
            android:label="@string/app_name"
            android:permission="org.jssec.android.permission.MY_PERMISSION" >

            <!-- ★ポイント 3★ intent-filter を定義しない -->
        </activity>
    </application>
</manifest>
```

ProtectedActivity.java

```
package org.jssec.android.permission.protectedapp;

import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.widget.TextView;

public class ProtectedActivity extends Activity {

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    private TextView mMessageView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mMessageView = (TextView) findViewById(R.id.messageView);

        // ★ポイント 4★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            // ★ポイント 4★ 証明書が一致しない場合は、Intent の内容は一切利用しない
```

```
            mMessageView.setText("独自定義 Signature Permission が自社アプリにより定義されていない。");
            return;
        }

        // ★ポイント 4★ 証明書が一致する場合にのみ、処理を続行する
        mMessageView.setText("独自定義 Signature Permission が自社アプリにより定義されていることを確認できた。");
    }
}
```

SigPerm.java

```
package org.jssec.android.shared;

import android.content.Context;
import android.content.pm.PackageManager;
import android.content.pm.PackageManager.NameNotFoundException;
import android.content.pm.PermissionInfo;

public class SigPerm {

    public static boolean test(Context ctx, String sigPermName, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, sigPermName));
    }

    public static String hash(Context ctx, String sigPermName) {
        if (sigPermName == null) return null;
        try {
            // sigPermName を定義したアプリのパッケージ名を取得する
            PackageManager pm = ctx.getPackageManager();
            PermissionInfo pi;
            pi = pm.getPermissionInfo(sigPermName, PackageManager.GET_META_DATA);
            String pkgname = pi.packageName;

            // 非 Signature Permission の場合は失敗扱い
            if (pi.protectionLevel != PermissionInfo.PROTECTION_SIGNATURE) return null;

            // sigPermName を定義したアプリの証明書のハッシュ値を返す
            return PkgCert.hash(ctx, pkgname);

        } catch (NameNotFoundException e) {
            return null;
        }
    }
}
```

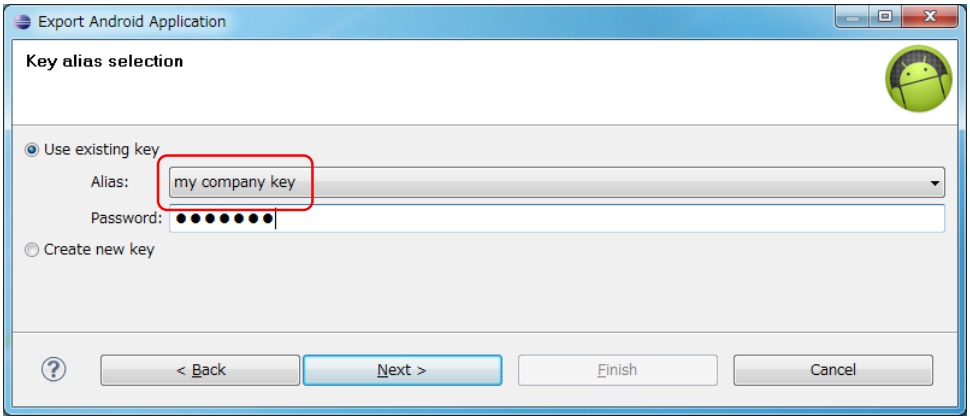
PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;
```



```
public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

★ポイント 5★ eclipse から APK を Export するときに、Component を利用するアプリと同じ開発者鍵で署名する。



ポイント:Component を利用するアプリ

1. 独自 Permission を protectionLevel=”signature”で定義する
2. uses-permission により独自 Permission を利用宣言する
3. ソースコード上で、独自定義 Signature Permission が自社アプリにより定義されていることを確認する
4. 利用先アプリが自社アプリであることを確認する
5. 利用先 Component が Activity の場合、明示的 Intent を使う
6. Component を提供するアプリと同じ開発者鍵で APK を署名する

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.jssec.android.permission.userapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" />

    <!-- ★ポイント 1★ 独自 Permission を protectionLevel=” signature” で定義する -->
    <permission
        android:name="org.jssec.android.permission.MY_PERMISSION"
        android:protectionLevel="signature" />

    <!-- ★ポイント 2★ uses-permission により独自 Permission を利用宣言する -->
    <uses-permission
        android:name="org.jssec.android.permission.MY_PERMISSION" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".UserActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```


UserActivity.java

```
package org.jssec.android.permission.userapp;

import org.jssec.android.shared.PkgCert;
import org.jssec.android.shared.SigPerm;
import org.jssec.android.shared.Utils;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class UserActivity extends Activity {

    // 利用先の Activity 情報
    private static final String TARGET_PACKAGE  = "org.jssec.android.permission.protectedapp";
    private static final String TARGET_ACTIVITY = "org.jssec.android.permission.protectedapp.ProtectedActivity";

    // 自社の Signature Permission
    private static final String MY_PERMISSION = "org.jssec.android.permission.MY_PERMISSION";

    // 自社の証明書のハッシュ値
    private static String sMyCertHash = null;
    private static String myCertHash(Context context) {
        if (sMyCertHash == null) {
            if (Utils.isDebuggable(context)) {
                // debug.keystore の"androiddebugkey"の証明書ハッシュ値
                sMyCertHash = "0EFB7236 328348A9 89718BAD DF57F544 D5CCB4AE B9DB34BC 1E29DD26 F77C8255";
            } else {
                // keystore の"my company key"の証明書ハッシュ値
                sMyCertHash = "D397D343 A5CBC10F 4EDDEB7C A10062DE 5690984F 1FB9E88B D7B3A7C2 42E142CA";
            }
        }
        return sMyCertHash;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    public void onSendButtonClicked(View view) {

        // ★ポイント 3★ 独自定義 Signature Permission が自社アプリにより定義されていることを確認する
        if (!SigPerm.test(this, MY_PERMISSION, myCertHash(this))) {
            Toast.makeText(this, "独自定義 Signature Permission が自社アプリにより定義されていない。", Toast.LENGT
H_LONG).show();
            return;
        }

        // ★ポイント 4★ 利用先アプリの証明書が自社の証明書であることを確認する
        if (!PkgCert.test(this, TARGET_PACKAGE, myCertHash(this))) {
            Toast.makeText(this, "利用先アプリは自社アプリではない。", Toast.LENGTH_LONG).show();
            return;
        }
```

```
        // ★ポイント 5★ 利用先 Component が Activity の場合、明示的 Intent を使う
        try {
            Intent intent = new Intent();
            intent.setClassName(TARGET_PACKAGE, TARGET_ACTIVITY);
            startActivity(intent);
        } catch(Exception e) {
            Toast.makeText(this,
                    String.format("例外発生:%s", e.getMessage()),
                    Toast.LENGTH_LONG).show();
        }
    }
}
```

PkgCertWhitelists.java

```
package org.jssec.android.shared;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;

public class PkgCertWhitelists {
    private Map<String, String> mWhitelists = new HashMap<String, String>();

    public boolean add(String pkgname, String sha256) {
        if (pkgname == null) return false;
        if (sha256 == null) return false;

        sha256 = sha256.replaceAll(" ", "");
        if (sha256.length() != 64) return false;    // SHA-256 は 32 バイト
        sha256 = sha256.toUpperCase();
        if (sha256.replaceAll("[0-9A-F]+", "").length() != 0) return false; // 0-9A-F 以外の文字がある

        mWhitelists.put(pkgname, sha256);
        return true;
    }

    public boolean test(Context ctx, String pkgname) {
        // pkgname に対応する正解のハッシュ値を取得する
        String correctHash = mWhitelists.get(pkgname);

        // pkgname の実際のハッシュ値と正解のハッシュ値を比較する
        return PkgCert.test(ctx, pkgname, correctHash);
    }
}
```

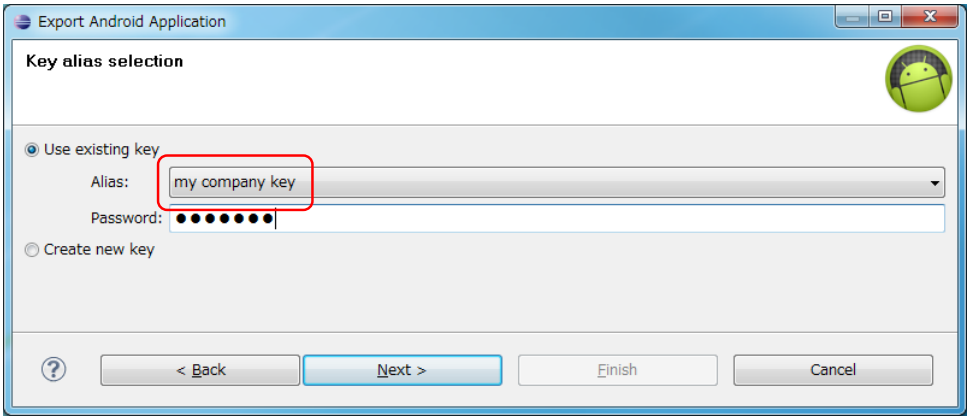
PkgCert.java

```
package org.jssec.android.shared;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import android.content.Context;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.content.pm.PackageManager.NameNotFoundException;
```



```
public class PkgCert {

    public static boolean test(Context ctx, String pkgname, String correctHash) {
        if (correctHash == null) return false;
        correctHash = correctHash.replaceAll(" ", "");
        return correctHash.equals(hash(ctx, pkgname));
    }

    public static String hash(Context ctx, String pkgname) {
        if (pkgname == null) return null;
        try {
            PackageManager pm = ctx.getPackageManager();
            PackageInfo pkginfo = pm.getPackageInfo(pkgname, PackageManager.GET_SIGNATURES);
            if (pkginfo.signatures.length != 1) return null;    // 複数署名はおかしい
            Signature sig = pkginfo.signatures[0];
            byte[] cert = sig.toByteArray();
            byte[] sha256 = computeSha256(cert);
            return byte2hex(sha256);
        } catch (NameNotFoundException e) {
            return null;
        }
    }

    private static byte[] computeSha256(byte[] data) {
        try {
            return MessageDigest.getInstance("SHA-256").digest(data);
        } catch (NoSuchAlgorithmException e) {
            return null;
        }
    }

    private static String byte2hex(byte[] data) {
        if (data == null) return null;
        final String digit = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        for (byte b : data) {
            int h = (b >> 4) & 15;
            int l = b & 15;
            sb.append(digit.charAt(h));
            sb.append(digit.charAt(l));
        }
        return sb.toString();
    }
}
```

★ポイント 6★ eclipse から APK を Export するときに、Component を提供するアプリと同じ開発者鍵で署名する。



5.2.1.3. アプリの証明書のハッシュ値を確認する方法

このガイド文書の各所で出てくるアプリの証明書のハッシュ値を確認する方法を紹介する。厳密には「APK を署名するときに使った開発者鍵の公開鍵証明書の SHA256 ハッシュ値」を確認する方法である。

Keytool により確認する方法

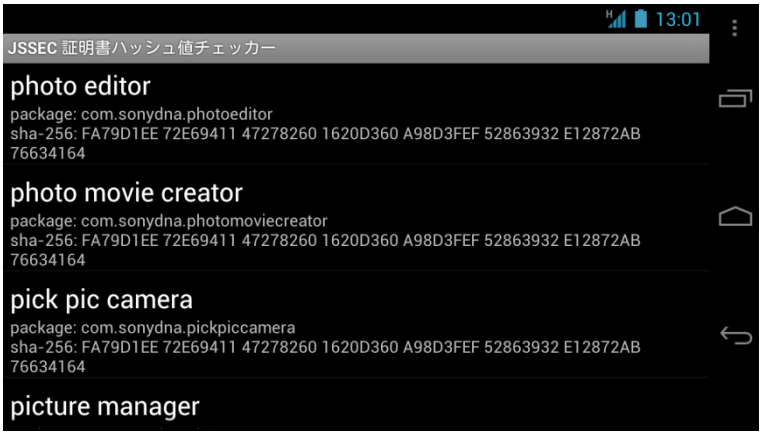
JDK に付属する keytool というプログラムを利用すると開発者鍵の公開鍵証明書のハッシュ値（証明書のフィンガープリントとも言う）を求めることができる。ハッシュ値にはハッシュアルゴリズムの違いにより MD5 や SHA1、SHA256 など様々なものがあるが、このガイド文書では暗号ビット長の安全性を考慮して SHA256 の利用を推奨している。残念なことに Android SDK で利用されている JDK6 に付属する keytool は SHA256 でのハッシュ値出力に対応しておらず、JDK7 に付属する keytool を使う必要がある。

Android のデバッグ証明書の内容を keytool で出力する例

```
> keytool -list -v -keystore <キーストアファイル> -storepass <パスワード>

キーストアのタイプ: JKS
キーストア・プロバイダ: SUN

キーストアには 1 エントリが含まれます

別名: androiddebugkey
作成日: 2012/01/11
エントリ・タイプ: PrivateKeyEntry
証明書チェーンの長さ: 1
証明書[1]:
所有者: CN=Android Debug, O=Android, C=US
発行者: CN=Android Debug, O=Android, C=US
シリアル番号: 4f0cef98
有効期間の開始日: Wed Jan 11 11:10:32 JST 2012 終了日: Fri Jan 03 11:10:32 JST 2042
証明書のフィンガプリント:
         MD5:  9E:89:53:18:06:B2:E3:AC:B4:24:CD:6A:56:BF:1E:A1
         SHA1: A8:1E:5D:E5:68:24:FD:F6:F1:ED:2F:C3:6E:0F:09:A3:07:F8:5C:0C
         SHA256: FB:75:E9:B9:2E:9E:6B:4D:AB:3F:94:B2:EC:A1:F0:33:09:74:D8:7A:CF:42:58:22:A2:56:85:1B:0F:85:C6:35
         署名アルゴリズム名: SHA1withRSA
         バージョン: 3


*******************************************
*******************************************
```

JSSEC 証明書ハッシュ値チェッカーにより確認する方法

JDK7 をインストールしなくても、JSSEC 証明書ハッシュ値チェッカーを使えば簡単に証明書ハッシュ値を確認できる。



これは端末にインストールされているアプリの証明書ハッシュ値を一覧表示する Android アプリである。上図中、「sha-256」の右に表示されている 16 進数文字列 64 文字が証明書ハッシュ値である。このガイド文書と一緒に配布しているサンプルコードの「JSSEC CertHash Checker」フォルダがそのソースコード一式である。ビルドして活用していただきたい。

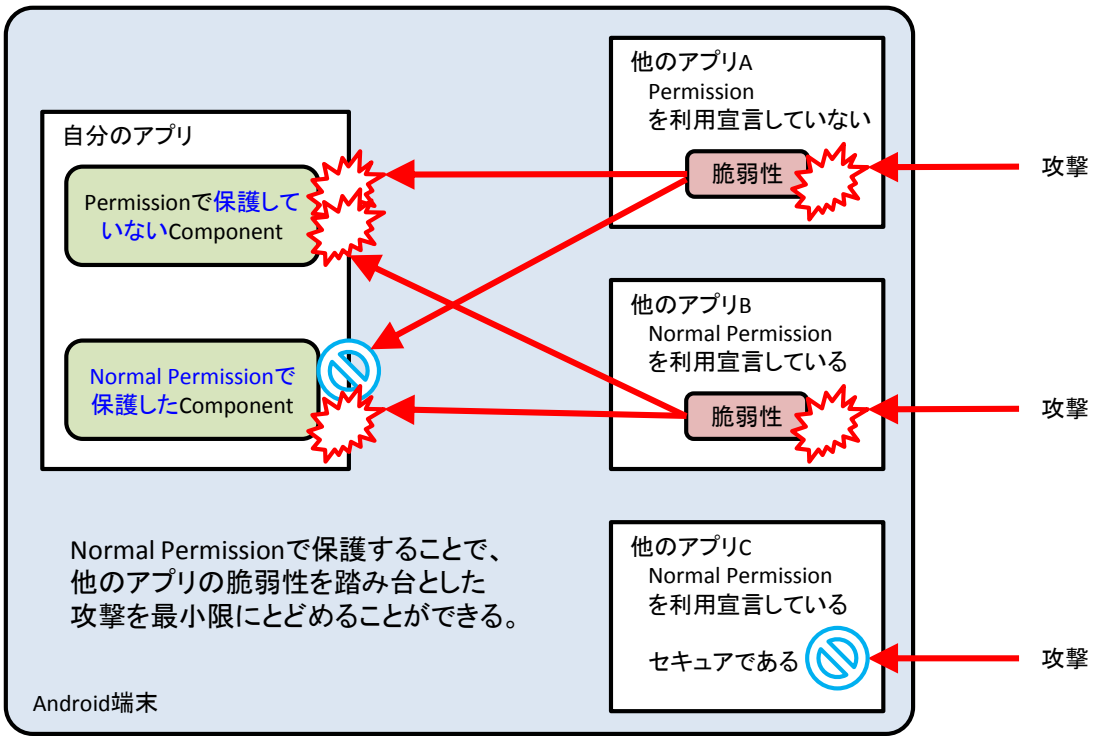
5.2.2. ルールブック

独自 Permission 利用時には以下のルールを守ること。

1.	公開 Component は Normal Permission で保護する	（推奨）
2.	Dangerous Permission はユーザーの資産を保護するためにだけに利用する	（必須）
3.	独自定義の Dangerous Permission は利用してはならない	（必須）
4.	独自定義 Permission は Component の提供側アプリだけでなく利用側アプリでも定義する	（必須）
5.	独自定義 Signature Permission は自社アプリにより定義されていることを確認する	（必須）
6.	独自定義の Permission 名はアプリのパッケージ名を拡張した文字列にする	（推奨）

5.2.2.1. 公開 Component は Normal Permission で保護する （推奨）

Normal Permission を利用するアプリは AndroidManifest.xml に uses-permission で利用宣言するだけでその権限を得ることができる。そのため一度インストールされてしまったマルウェアから Component を保護するような目的には Normal Permission は利用できない。しかし他のアプリの脆弱性を踏み台とした攻撃を最小限にとどめるためには有効に利用できる。なお、以下の説明は Android OS 既定の Normal Permission だけでなく、独自定義の Normal Permission にも同様に適用できる。



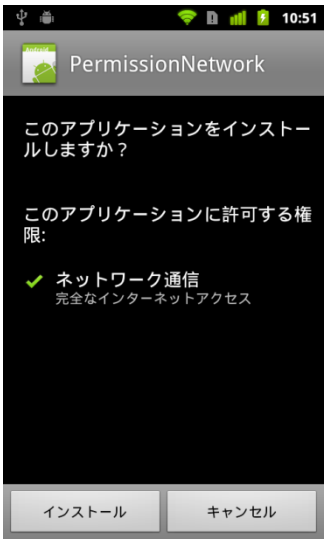
上の図では、1 つの Android 端末内に自分のアプリと他のアプリが複数存在している状況を表している。自分のアプリには他のアプリに公開している Component が 2 つあり、1 つは Permission による保護がなく、もう 1 つは Normal Permission により Component のアクセスを制限しているという設定である。Permission による保護がない Component は、脆弱性のある他のすべてのアプリを踏み台として、攻撃者にアクセスされてしまう危険がある。一方、Normal Permission により保護した Component は、他の脆弱な Normal Permission を利用宣言してい

ないアプリを踏み台とした攻撃を防御することができる。これが Normal Permission の利用価値である。

他のアプリに公開する Component には Normal Permission による保護を検討すべきである。その Component を利用する他のアプリの開発者には、自分のアプリで定義した Normal Permission を uses-permission により利用宣言してもらい、さらに脆弱性を作り込まないようにセキュアコーディングしてもらえるよう依頼すべきである。

5.2.2.2. Dangerous Permission はユーザーの資産を保護するためにだけに利用する （必須）

Dangerous Permission は他の 3 つの Permission と異なり、アプリにその権限を付与するかどうかをユーザーに判断を求める機能がある。Dangerous Permission を利用宣言しているアプリを端末にインストールするとき、次のような画面が表示される。これにより、そのアプリがどのような権限（Dangerous Permission および Normal Permission）を利用しようとしているのかをユーザーが知ることができる。ユーザーが「インストール」をタップすることで、そのアプリは利用宣言した権限が付与され、インストールされるようになっている。



Dangerous Permission で保護できるのはユーザーの資産だけであることに注意が必要である。Dangerous Permission によってユーザー以外の資産は保護できない。これだけでは意図が伝わりにくいので、次に具体例を挙げて説明する。

自社アプリだけと連携する Component は他社アプリからのアクセスを禁止したいことがある。このような Component を Dangerous Permission により保護したとする。他社アプリがインストールされるとき、ユーザーに確認画面が表示される。ユーザーが自社の都合を考えるはずもなく、他社アプリと自社アプリを連携させるためにアクセスを許可してしまう。自社アプリだけで連携したいのは自社の都合であって、この都合こそ自社の資産である。この場合、この資産を狙う攻撃者は他社アプリベンダーとユーザーである。これらの攻撃者から自社資産を保護するためには独自定義の Signature Permission を使うとよい。

5.2.2.3. 独自定義の Dangerous Permission は利用してはならない （必須）

独自定義の Dangerous Permission を作ることはできるが、ユーザーに判断を求める画面が表示されずに無条件に権限が付与されてしまう場合がある。つまり Dangerous Permission の特徴である「ユーザーに許可を求める」機能が働かない場合があるのだ。

まず説明のために 2 つのアプリを想定する。1 つは Dangerous Permission により保護した Component を公開するアプリでこれを ProtectedApp とする。もう 1 つは ProtectedApp の Component を利用する別のアプリでこれを UserApp とする。これら 2 つのインストール順序に依存して、うまくいくケースとうまくいかないケースがある。

うまくいくケース	うまくいかないケース
1. 先に ProtectedApp をインストールする 2. 次に UserApp をインストールしようとすると、Dangerous Permission の利用許可を求める画面が表示され、ユーザーが許可することで UserApp はインストールされる 3. その後、ユーザーが UserApp を起動すると、UserApp は ProtectedApp の Component にアクセスできる	1. 先に UserApp をインストールすると、Dangerous Permission の利用許可を求める画面は表示されずに、そのままインストールできてしまう 2. 次に ProtectedApp をインストールすると普通にインストールできてしまう 3. その後、ユーザーが UserApp を起動すると、UserApp は ProtectedApp の Component にアクセスできてしまう
ユーザーが明示的に許可しなければ UserApp は ProtectedApp の Component にアクセスできないので、正しく保護できている。	ユーザーが許可していないにも関わらず、UserApp は ProtectedApp の Component を利用できてしまうという問題がある。

うまくいかないケースの原因は次の通りである。先に UserApp をインストールしようとすると uses-permission により利用宣言された Permission はまだその端末上では定義されていない。このとき Android OS はエラーとすることもなくインストールを続行してしまう。Dangerous Permission のユーザー確認はインストール時だけしか実施されないため、一度インストールされたアプリは権限を許可されたものとして扱われる。したがって後からインストールされるアプリの Component を同名の Dangerous Permission で保護していた場合、ユーザーの許可なく先にインストールされたアプリからその Component が利用できてしまうのである。

なお、Android OS 既定の Dangerous Permission はアプリがインストールされるときにはその存在が保証されているので、uses-permission しているアプリがインストールされるときには必ずユーザー確認画面が表示される。独自定義の Dangerous Permission の場合にだけこの問題は生じる。

現在、「うまくいかないケース」でも保護するよい方法は見つかっていない。したがって、独自定義の Dangerous Permission は利用してはならない。

5.2.2.4. 独自定義 Permission は Component の提供側アプリだけでなく利用側アプリでも定義する （必須）

同じ名前の Permission が複数のアプリで定義される場合、最初にインストールされたアプリの Permission 定義が優先される。つまり、Permission 付きのコンポーネントは、先にインストールされたアプリの Permission 定義が持つ Protection Level で保護される。

この仕組みを知っている人は、連携する全てのアプリに対し同じ Permission を定義することが無駄に感じられるかもしれない。しかし、最初にインストールするアプリのみに Permission を定義しておいて、他のアプリでは定義しないという実装をしてはいけない。なぜなら、アプリのインストール順が、開発時に想定していた順序と異なった場合に、登録されていない Permission でコンポーネントを保護しているという状況が発生してしまうためである。未登録の Permission は、Protection Level が normal として扱われる。そのため、uses-permission を宣言した他のアプリからアクセスされてしまう。

このような状況を作らないために、自社アプリ間連携するすべてのアプリにおいて、Protection Level が signature の Permission を AndroidManifest.xml に定義しておくことが必要である。

5.2.2.5. 独自定義 Signature Permission は自社アプリにより定義されていることを確認する （必須）

AnroidManifest.xml で Signature Permission を宣言し、コンポーネントをその Permission で保護しただけでは、実は保護が十分ではない。この詳細はアドバンストセクションの「独自定義 Signature Permission を回避できる Android OS の」を参照すること。

以下、独自定義 Signature Permission を安全に正しく使う手順である。

まず、AndroidManifest.xml にて次を行う。

1. 連携するすべてのアプリの AndroidManifest.xml にて、同名の独自 Signature Permission を定義する（Permission の定義）
   例: <permission android:name="xxx" android:protectionLevel="signature" />
2. 保護したい Component のある AndroidManifest.xml にて、その Component の定義タグの permission 属性で、独自定義 Signature Permission を指定する（Permission の要求宣言）
   例: <Activity android:permission="xxx" … >…</Activity>
3. 保護したい Component にアクセスする連携アプリの AndroidManifest.xml にて、uses-permission タグに独自定義 Signature Permission を指定する（Permission の利用宣言）
   例: <uses-permission android:name="xxx" />

次にソースコード上にて次を行う。

4. 保護したい Component でリクエストを処理するソースコードにおいて、独自定義した Signature Permission が自社アプリにより定義されたものかどうかを確認し、そうでなければリクエストを無視する（Component 提供側による保護）
5. 保護したい Component にアクセスするソースコードにおいて、独自定義した Signature Permission が自社アプリにより定義されたものかどうかを確認し、そうでなければ Component にアクセスしない（Component 利用側による保護）

最後に eclipse の Export 機能にて次を行う。

6. 連携するすべてのアプリの APK を同じ開発者鍵で署名する

ここで「独自定義した Signature Permission が、自社アプリにより定義されたものかどうかを確認」する必要があるが、具体的な実装方法についてはサンプルコードセクションの「5.2.1.2 独自定義の Signature Permission で自社アプリ連携する方法」を参照すること。

5.2.2.6. 独自定義の Permission 名はアプリのパッケージ名を拡張した文字列にする （推奨）

複数のアプリが同じ名前で Permission を定義する場合、先にインストールされたアプリが定義する Protection Level が適用される。先にインストールされたアプリが Normal Permission を定義し、後にインストールされたアプリが同じ名前で Signature Permission を定義した場合、Signature Permission による保護がまったく効かない。悪意がない場合でも、複数のアプリにおいて Permission 名が衝突して意図しない Protection Level で動作する可能性がある。このような事故を防ぐため、Permission 名にはアプリのパッケージ名を入れた方が良い。

```
（パッケージ名）.permission.（識別する文字列）
```

例えば、org.jssec.android.sample というパッケージに READ アクセスの Permission を定義するならば、次の様な命名が好ましい。

```
org.jssec.android.sample.permission.READ
```

5.2.3. アドバンスト

5.2.3.1. 独自定義 Signature Permission を回避できる Android OS の特性

独自定義 Signature Permission は、同じ開発者鍵で署名されたアプリ間だけでアプリ間連携を実現する Permission である。開発者鍵はプライベート鍵であり絶対に公開してはならないものであるため、Signature Permission による保護は自社アプリだけで連携する場合に使われることが多い。

Android の Dev Guide（http://developer.android.com/guide/topics/security/security.html）で説明されている独自定義 Signature Permission の基本的な使い方は次のとおりである。しかし、後述するようにこれには問題がある。

1. 連携するすべてのアプリの AndroidManifest.xml にて独自 Signature Permission を定義する
   例: <permission android:name="xxx" android:protectionLevel="signature" />
2. 保護したい Component のある AndroidManifest.xml にて、保護したい Component の定義タグの permission 属性で、独自定義 Signature Permission を要求する
   例: <Activity android:permission="xxx" … >…</Activity>
3. 保護したい Component にアクセスしたい連携アプリの AndroidManifest.xml にて、独自定義 Signature Permission を利用宣言する
   例: <uses-permission android:name="xxx" />
4. 連携するすべてのアプリの APK を同じ開発者鍵で署名する

実はこの基本的な使い方だけであると、Signature Permission を回避できる抜け道がある。説明のために独自定義の Signature Permission で保護したアプリを ProtectedApp とし、ProtectedApp とは異なる開発者鍵で署名したアプリを AttackerApp とする。次の条件が成立するとき、AttackerApp は署名が一致していないにもかかわらず、ProtectedApp の Component にアクセスできてしまう。

1. AttackerApp も ProtectedApp が独自定義した Signature Permission と同じ名前で Normal Permission を定義する（厳密には Signature Permission でも構わない）
2. AttackerApp は独自定義した Normal Permission を uses-permission で利用宣言する
3. AttackerApp を ProtectedApp より先に端末にインストールする

ProtectedApp が独自定義している Permission の名前は APK ファイルから AndroidManifest.xml を取り出せば容易に知ることができる。上記の条件 1 および条件 2 は攻撃者には何ら障害とならない。条件 3 はユーザーを騙すなどの方法により攻撃者もある程度制御できる条件である。

このように独自定義の Signature Permission は基本的な使い方では保護を回避されてしまう抜け道が存在する。正しい使い方については、ルールセクションの「独自定義 Signature Permission は自社アプリにより定義されていることを確認する （必須）」を参照すること。

5.2.3.2. ユーザーが AndroidManifest.xml を改ざんする

独自 Permission の Protection Level が意図しないものになるケースは既に説明した。そのことによる不具合を防ぐために、Java のソースコード側で何らかの対応を実施する必要があった。ここでは、AndroidManifest.xml が改ざんされるという視点から、ソースコード側の対応について述べる。改ざんを検知する簡易な実装例を提示するが、犯罪意識をもって改ざんを行うプロのハッカーに対してはほとんど効果がない方法であることに注意すること。

この節はアプリの改ざんに関するものであり、ユーザー自身が悪意を持っているケースである。本来はガイドラインの範囲外であるが、Permission に関する事、これを行うツールがアプリとして公開されている事、から「プロでないハッカーに対する簡易な対策」として述べておくことにした。

Google Play からインストールできるアプリは、root 権限無しに改ざんできるアプリであることを頭に置いておく必要がある。なぜなら、AndroidManifest.xml を変更して APK ファイルを再生成、署名するアプリケーションが配布されているためである。このアプリを使用する事で、誰でも任意のアプリから Permission を削除することが可能になっている。

事例としては INTERNET Permission を取り除いた AndroidManifest.xml から別署名の APK を生成し、アプリに組み込まれた広告モジュールが動作しないようにするケースが多いようである。個人情報がどこかに送信されているかもしれない等の不安が払拭されるということで、この種のツールの存在を評価しているユーザーも存在する。このような行為は、アプリケーションに組み込まれた広告が機能しなくなるため、広告収入を期待している開発者に対して金銭的被害を与える行動であるとも言える。ユーザーのほとんどは罪の意識無くこれらの行為を行っていると思われる。

インターネット Permission を uses-permission で宣言しているアプリが、実行時に自身の AndroidManifest.xml に記載されている Permission を確認する実装例を次に示す。

```
public class CheckPermissionActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // AndroidManifest.xml に定義した Permission を取得
        List<String> list = getDefinedPermissionList();

        // 改竄を検知する
        if( checkPermissions(list) ){
            // OK
            Log.d("dbg", "OK.");
        }else{
            Log.d("dbg", "manifest file is stale.");
            finish();
        }
    }

    /**
```

```
     * AndroidManifest.xml に定義した Permission をリストで取得する
     * @return
     */
    private List<String> getDefinedPermissionList(){
        List<String> list = new ArrayList<String>();
        list.add("android.permission.INTERNET");
        return list;
    }

    /**
     * Permission が変更されていないことを確認する。
     * @param permissionList
     * @return
     */
    private boolean checkPermissions(List<String> permissionList){
        try {
            PackageInfo packageInfo = getPackageManager().getPackageInfo(
                    getPackageName(), PackageManager.GET_PERMISSIONS);
            String[] permissionArray = packageInfo.requestedPermissions;
            if (permissionArray != null) {
                for (String permission : permissionArray) {
                    if(! permissionList.remove(permission)){
                        // 意図しない Permission が付加されている
                        return false;
                    }
                }
            }

            if(permissionList.size() == 0){
                // OK
                return true;
            }

        } catch (NameNotFoundException e) {
        }

        return false;
    }
}
```

